군 사 과 학 연 구 지
제16권 제1호(2023년 6월)

사례를 통해 살펴본 M&S 원시자료변환(Data Wrangling) 방안

Case Study Approach to Data Wrangling Techniques for Modeling and Simulation

윤봉규[1)]
Bongkyoo Yoon

ABSTRACT

This paper provides an overview of necessary data wrangling techniques in the defense area, where Modeling and Simulation (M&S) approaches are increasingly popular. The exponential growth in computing power across various domains has had a profound impact on the use of M&S in defense. However, there is a dearth of necessary techniques for dealing with M&S model data in the defense domain. In response to this gap, the present study explores data wrangling techniques leveraging the R tidyverse package, using two case studies from a Markovian model and an ABM simulation model that are relevant to defense-related analyses. The techniques introduced in the study have important implications for improving data manipulation practices for insightful M&S analyses among researchers and practitioners in the defense sector.

Keyword: R Tidyverse, Data Wrangling, Modeling and Simulation(M&S)

논문접수일 : 2023년 4월 30일, 심사일 : 2023년 5월 31일, 게재확정일 : 2023년 6월 8일
1) 국방대학교 관리대학원 운영분석전공

1. 서론

최근 주목 받고 있는 AI 분야의 발전은 컴퓨팅 파워의 급격한 증가에 의해 가능해졌으며, 날씨 예측, 석유 탐사, 신약 개발 등의 다양한 영역에서 컴퓨팅 파워의 증가는 획기적인 변화를 가져왔다. 이런 분야에서 49%에서 94%의 개선이 컴퓨팅 파워의 증가에 따른 성과라는 주장 있을 정도로 컴퓨팅 파워 즉, 계산 능력의 증가는 사회 전반의 변화를 주도하고 있다(Zewe, 2022).

컴퓨팅 파워의 급격한 증가는 연구방법론에도 변화를 가져왔다. 기존에는 계량적 연구 방법으로 방정식을 세우고 해를 구하는 방법에 기반한 분석적 방법(Analytic Approach)이 시뮬레이션과 같은 계산적 접근법(Computational Approach)에 비해 높은 가치를 가진 것으로 여겨졌으나, 이제는 시뮬레이션으로 대표되는 계산적 방법의 활용 빈도와 중요성이 점점 증가하고 있다. 이 현상은 국방 분야에서도 동일하게 나타나고 있다.

계산적 방법론은 분석 결과가 깔끔한 해(Solution)의 형태로 나오는 것이 아니라 분석적 접근에 비해서 다양한 잡음이 섞인 원시 자료의 형태로 제공될 가능성이 높다. 문제는 '자료 분석의 80%는 자료를 정제하고 전처리하는 것(Dasu & Johnson, 2003; Wickham, 2014)'이라는 말이 있을 정도로 자료 분석은 태생적으로 잡음을 제거하고 활용 가능한 형태로 만드는, 시간과 노력이 많이 드는 과정임에도 이해 대한 체계적인 소개나 연구가 많지 않다는 점이다.

컴퓨팅 파워의 증가에 따른 데이터 증가와 빅데이터, 데이터 과학자 등으로 대표되는 데이터 분석의 가치 증가는 원시 자료 전처리에 대한 관심과 필요성을 증가 시켰으며, 그 결과 데이터 전문가뿐만 아니라 현장 실무자나 연구자들이 데이터 전처리에 관심을 가지게 되었다. 이런 변화에도 불구하고 원시자료 전처리에 대한 체계적인 연구는 비교적 최근까지도 거의 없었다(Wickham, 2014).

원시 자료의 전처리에 대한 체계적인 연구가 거의 없었던 이유는 데이터 전처리는 과거에는 주로 데이터베이스 분야 전문가들만 관심을 가지는 주제였고, 해당 분야 전문가가 아닌 연구자나 실무자가 활용하기에는 내용이 너무 방대했기 때문이다. 이에 반해 분석을 위한 자료 전처리는 경험이나 배경 지식에 따라 소요되는 시간의 편차가 크기는 하지만, 시행착오를 거치면서 수행할 수 있기 때문이기도 하다. 결과적으로 특별한 경우가 아니라면 원시 자료의 전처리는 연구자나 실무자에게 시간은 많이 들지만 따로 체계화할만한 가치가 있어 보이지는 않은 분야였다.

또한, 자료 전처리 과정은 사용하는 언어와 컴퓨터에 따라서 다양한 문제를 노출하고, 논리적으로 설명되지 않는 자연어와 유사한 불규칙성이 많아, 표준화나 체계화가 가능한 과학이라기 보다 임기응변으로 상황 상황에 따라 대응하는 기술의 영역으로 치부하는 경우가 많았다. 그 결과 자료 전처리는 숙련된 연구자라고 하더라도 원시 자료의 특성이나 분석 분야에 따라서 전처리 절차나 기법이 차이가 나는 경우가 많다. 이런 이유로 자료 처리 수요가 높아지고 저변이 넓어지는 상황에서도 이에 대한 표준화나 전처리 과정의 효율성을 높이는 방안에 대한 연구가 미흡했다.

최근에는 자료 처리의 관심과 저변 확대에 대한 반향으로 효율적인 자료 전처리와 표준화에 대한 연구가 급속하게 발전하고 있으며, R의 'tidyverse' 패키지는 이런 노력의 일환이다.

'tidyverse' 패키지는 자료의 입력을 위한 'readr' 패키지, 문자열처리를 위한 'stringr', 데이터 정제 및 변형을 위한 'dplyr', 'tidyr' 등의 패키지를 묶어 놓은 R 통합 패키지이다. 이를 주도한 Wickham은 자료의 시각화에 대한 표준

적인 절차가 존재한다는 Wilkinson(1999)의 주장을 'ggplot2'로 구현할 정도로 시각화(Visualization)를 포함한 자료 처리의 표준화가 가능하다는 생각이 확고한(Wickham, 2010) 데이터 과학자이다.

한편, 'Wrangling'은 원시자료를 시각화(Visualization)와 모형구축(Modeling)에 활용할 수 있는 형태로 정리하는 것을 의미한다(Wickham and Grolemund, 2017). 'Wrangling'은 원시 자료를 처리해서 통찰력 있는 분석 결과를 얻기 위해, 시각화에 바로 활용할 수 있는 형태로 변환하는 것을 포괄하는 개념으로 '원시자료 전처리와 자료 정제'를 포함하는 개념으로 이해할 수 있다. 본 연구에서는 '원시자료변환'이라는 용어를 전처리와 변환을 포함하는 'Wrangling'의 의미로 사용한다.

본 연구에서는 최근의 R의 'tidyverse' 패키지를 중심으로 진행되고 있는 원시자료 변환 방안(Wrangling)을 기초로, 국방분야에서 M&S를 활용한 분석을 수행할 때 자주 활용되는 원시자료변환 기법을 소개하고자 한다. 특히 문자열을 활용하여 다양한 조건에 부합하는 자료를 추출하는 방식과 최근의 개선된 R 명령어를 활용하여 원시자료변환 과정에서 엑셀로 작업하는 것과 같은 직관적인 방식을 소개함으로써 숙련된 전문가가 아니더라도 복잡한 조건을 만족하는 데이터를 추출하여 원하는 형태로 만드는 방법을 제시하고자 한다.

본 연구는 데이터 전문가가 아닌 국방분야 M&S 연구자나 실무자가 R에 대한 기본적인 이해만 가지고 있더라도 원시자료변환을 용이하게 수행할 수 있도록 사용빈도가 높은 핵심적 원시자료변환 기법을 제시하여, 현장에서 바로 활용할 수 있도록 하는 것을 목표로 한다.

이를 위해 2장에서는 M&S 분석과 관련하여 활용될 사례를 소개하고, 3장에서는 원시자료변환을 위해서 필요한 R의 자료형을 개별 자료와 자료의 집합으로 구분하여 소개하고, 이를 다루는 법에 대해서 살펴본다. 4장에서는 'tidyverse' 패키지를 중심으로 원시자료변환에 자주 활용되는 기법을 사례를 통해 살펴봄으로써 M&S 자료 전처리에 대한 이해의 깊이를 더하고자 한다.

2. 원시자료변환 분석 사례

본 연구는 이해의 편의를 위해서 국방분야 M&S 사례를 통해 도출된 원시자료를 중심으로 원시자료변환 기법을 살펴본다. 본 연구에 활용할 사례는 모델링 분야에서 자주 활용되는 방법론인 마코프체인을 활용한 모델링과, 시뮬레이션 방법인 에이전트기반 시뮬레이션 모델링(Agent Based Modeling)을 통해 얻은 다음과 같은 원시자료를 바탕으로 한다.

① 단계형 정비 모형(마코프체인 모델링)

사전검사, 정비, 출고검사의 단계를 거치는 정비 설비의 운영 성과를 비교하기 위해서 마코프체인으로 모델링하는 경우 원시 자료는 보통 아래 <그림 2-1>과 같이 두 테이블로 나타난다.

A data.frame: 396 × 4

S1	S2	S3	Index
<int>	<int>	<int>	<int>
0	0	0	1
0	0	1	2
0	0	2	3
0	0	3	4
0	0	4	5
0	0	5	6
0	1	0	7
0	1	1	8
0	1	2	9
0	1	3	10

A data.frame: 396 × 2

Index	Prob
<int>	<dbl>
1	0.0008670620
2	0.0018991894
3	0.0033288712
4	0.0001890095
5	0.0055546257
6	0.0009027354
7	0.0003486690
8	0.0009073764
9	0.0003500549
10	0.0003542620

<그림 2-1> 단계형 정비 모형 원시자료(왼쪽: 상태대응표, 오른쪽: 안정상태확률)

<그림 2-1>은 정비 설비의 사전 검사팀, 정비팀, 출고검사팀이 각각 5, 10, 5팀으로 구성되어 있을 때, 각 정비 단계별 팀별 안배의 적정성을 평가하고자 하는 목적으로 모형을 구축하여 분석한 결과이다. S1은 사전검사 단계에서 현재 업무를 수행중인 팀의 수, S2는 정비업무 수행중인 팀의 수, S3은 출고검사를 수행중인 정비팀의 수를 나타낸다. (S1, S2, S3)가 모여 마코프체인의 상태를 구성하며, Index는 (S1, S2, S3)의 일련 번호이다. <그림 2-1>의 오른쪽 테이블은 마코프체인 모형을 분석한 결과로 도출한 각 상태에 머문 시간의 상대적인 비율(안정상태확률)을 나타낸다.

마코프체인을 활용한 모델링 분석 결과가 이렇게 두 개의 표로 주어지는 이유는 마코프체인으로 분석할 때 안정상태확률 계산을 위해 행렬을 활용하기 때문이다. 행렬의 각 행과 열은 (S1, S2, S3)과 일대일로 대응되지만 행렬의 행과 열은 자연수 값을 가지므로 세 개의 숫자 조합으로 이루어진 (S1, S2, S3) 상태를 직접 표현하기 어렵다. 따라서 행렬의 인덱스와 실제 상태의 대응관계를 나타내는 상태 대응표와 분석 결과값인 안정상태확률을 잘 조합해야만 원하는 성과지표를 산출할 수 있다. 예를 들어, 정비팀은 전원 업무중인데 사전검사와 출고검사는 절반 이하로 일을 하는 비율이 어느정도인지를 알기 위해서는 상태 대응표에서 S2는 10이고 S1과 S3는 0, 1, 2인 (0, 10, 0), (1, 10, 2) …(2,10,2) 상태의 Index를 확인해서 그 확률값을 더해야 한다. 즉 두 표를 Index를 기준으로 연결하고 연결된 값을 더하는 과정이 필요하다.

② 미사일 방어시스템 배치 (에이전트기반 시뮬레이션)

다음 <그림 2-2>는 에이전트기반 시뮬레이션 언어인 NetLogo를 활용하여 왼쪽에 위치한 홍군이 쏜 미사일을 오른쪽에 배치한 청군의 미사일 방어체계가 방어하는 시뮬레이션 결과이다. who는 에이전트(미사일)의 일련번호이며, color는 홍군(15), 청군(105)을 의미한다. xcor, ycor는 에이전트(미사일 또는 미사일 방어체계)의 현재 위치 좌표이며, Time은 시뮬레이션 시간이다. 미사일 방어 시뮬레이션 결과가 각 에이전트(미사일)의 시간대별 위치로 주어진 것이다.

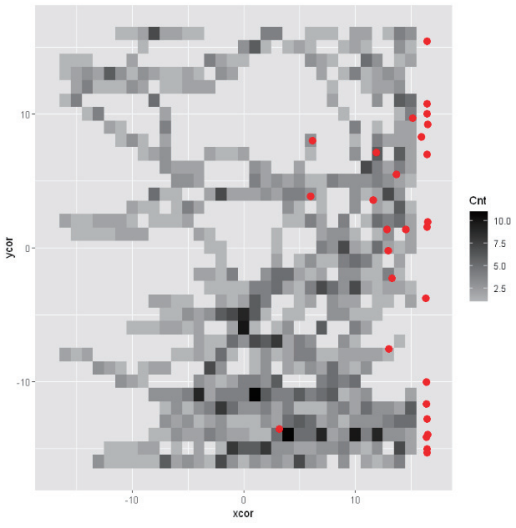
A spec_tbl_df: 4972 × 5

who	color	xcor	ycor	Time
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
0	15	-15	-6.1850115	0
1	15	-4	5.3104675	0
2	15	-9	-11.4228603	0
3	15	-2	14.9425713	0
4	15	-2	-6.8643591	0
5	15	-13	5.4798548	0
⋮	⋮	⋮	⋮	⋮
85	15	15.3058922	9.9813362	61
91	15	16.4316808	15.4047195	61
118	105	5.5171209	-9.6121732	61

<그림 2-2> NetLogo 시뮬레이션 결과

다음 <그림 2-3>은 격추되지 않고 청군의 미사일 방어 시스템을 통과해서 피해를 준 홍군 미사일의 이동 경로이다. x축 0을 기준으로 왼쪽이 홍군, 오른쪽이 청군의 미사일 방어시스템이 배치되어 있는 모형이므로 홍군의 공격이 왼쪽에서 오른쪽으로 진행되었다. 빨간색 원은 살아남은 홍군 미사일이 타격한 지점을 나타내며, 경로의 색깔이 진할수록 많은 미사일이 해당 지점(좌표)을 지나갔다는 것을 의미한다. 결과적으로 격추를 피한 홍군의 미사일이 아래쪽을 통과한 경우가 많았다는 것을 알 수 있다.

아래 <그림 2-3>은 앞선 <그림 2-2>의 원시자료를 격자형으로 그릴 수 있도록 xcor, ycor를 격자의 좌표를 나타낼 수 있도록 실수에서 정수로 위치좌표를 바꾸고, 각 위치좌표를 지나간 미사일의 수를 포함하는 표가 있어야 그릴 수 있다.



<그림 2-3> 격추되지 않은 홍군 미사일의 이동경로와 타격지점

3. 자료의 입출력과 R의 자료형

통찰력 있는 분석을 위해서는 앞선 <그림 2-1, 2-2>의 원시자료를 원하는 형태로 변환해야 한다는 것을 앞서 사례를 통해서 확인할 수 있었다. 이제 이를 수행하기 위해서 필요한 배경 지식인 자료의 종류를 나타내는 자료형(Data Type)에 대해서 살펴보자.

자료형을 원시자료처리에 앞서 파악해야 하는 이유를 먼저 살펴보자. 원시자료변환 과정에서 많은 오류의 원인이 잘못된 자료형의 사용임에도 불구하고, 자료형에 대한 설명은 대부분의 문헌에서 간단히 언급되거나 두꺼운 참고 매뉴얼(Reference Manual)과 같이 지나치게 상세히 설명하는 극단적인 방식인 경우가 많다. 그 결과 피상적인 내용만 알거나 필요한 내용만 빠르게 파악하기 어렵다. 이런 이유로 원시자료입력이나 변환시 자료형과 관련된 문제가 생겨도 문제를 파악하기도 어렵고, 파악하더라도 다시 매뉴얼을 들여다봐야 하는 상황이 생긴다. 그 결과 알고 보면 간단한 문제임에도 문제 식별과 해결에 많은 시간이 걸리는 경우가 많다. 여기에서는 이런 점을 해결할 수 있도록 자료형과 관련해서 문제 식별과 해결에 꼭 필요한 내용을 중심으로 R의 자료형에 대해서 살펴본다.

자료형에 대한 설명 이전에 자료를 R에서 분석을 위해서 데이터를 불러오는 자료 입/출력을 간단히 살펴보자.

원시자료는 다양한 형태의 파일로 제공이 되겠지만, 많이 쓰는 형태가 쉼표로 자료의 값을 분리한 CSV파일이다. CSV는 'Comma Splitted Value'의 약자로 말 그대로 쉼표로 분리한 값을 나타낸다. CSV파일은 다음과 같은 명령어로 불러올 수 있다.

```
> read_csv(prob, "result.csv")2)
```

위 명령은 파일명이 result.csv인 CSV파일을 R에 불러와서 prob이라는 변수에 저장하는 명령이다. 파일을 출력하는 명령어는 write_csv이며, 쉼표가 아니라 탭이나 다른 문자로 구분된 파일은 read_delim을 활용하면 불러 올 수 있다.

원시자료를 불러오면 대부분은 잘못된 값이나 결측값이 포함되어 있기 마련이다. 이를 R에서는 NA(Not Available)로 표기하며, 결측값 여부를 판단하는 명령어는 is.na이다. 한편, R의 대부분의 계산 명령어는 na.rm이라는 옵션[3]을 포함하고 있다. 이 명령어를 활용하면

2) '>'는 R 명령창의 프롬프트이며, 그 뒤에 오는 것이 R의 명령어이다. 한편, R 명령어 실행전에 'tidyverse' 패키지를 다음과 같이 설치하고 활성화해야 여기에서 나오는 명령어를 사용할 수 있다.

```
> install.packages("tidyverse") # tidyverse 설치
> library(tidyverse) # tidyverse 활성화
```

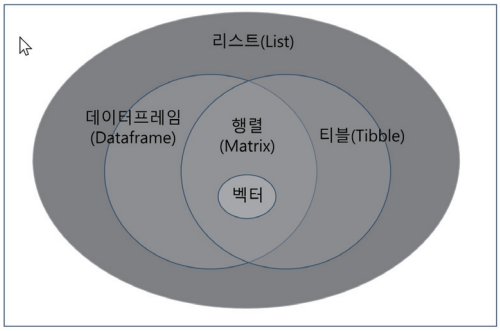
3) 옵션은 명령어 안에서 설정하는 명령어의 상세 내용이라고 할 수 있다. 예를 들어 sum(c(1,NA,3))을 실행

결측값을 제외하고 계산을 수행할 수 있다.

다시 자료형으로 돌아와서, R의 자료형을 살펴보자. R의 자료형은 벡터(Vector), 행렬(Matrix), 요인(Factor), 배열(Array), 문자(Character), 데이터프레임(Dataframe) 등 다양하지만, 기본 자료형은 벡터이다. (Vernables et al., 2023; 노만 매트로프, 2014).

벡터는 원소의 조합으로 이루어져 있으며, 한 줄로 연결된 상자 안에 숫자나 문자 또는 문장(문자열)이 하나씩 들어가 있는 것으로 비유적으로 생각할 수 있다. 각 상자에 들어가 있는 원소들의 연결이라는 의미로 R에서 벡터는 '이어붙이다'는 의미를 가지는 'concatenate'의 첫자 c를 활용하여 만든다. 예를 들어 c(1,2,3)은 숫자 1, 2, 3이 연결된 벡터이고, c("I am", "You are", "He is")는 세 개의 문자열로 이어진 벡터이다. 벡터는 원소들이 한 줄로 이어진 집합이므로 위치를 나타내는 방식으로 불러올 수 있다. prob[3]은 prob라는 변수의 세 번째 방에 들어가 있는 원소를 불러오라는 명령어이다.

R의 자료형이 혼동을 주는 이유는 원소의 자료형과 원소들의 집합을 나타내는 자료형을 섞어서 사용하기 때문이다. 데이터 처리와 시각화를 위해 만든 언어의 특성상 R은 원소의 자료형보다는 원소들의 집합과 관련된 자료형에 초점을 맞추고 있음에도 기존의 컴퓨터 언어의 자료형을 계승한 측면이 있어 둘이 혼재되어 혼선이 생긴 것이다. 여기에서는 이 둘을 구분하여 원소의 자료형과 집합의 자료형을 구분해서 살펴보자.



<그림 3-1> 집합의 자료형

먼저 R의 원소의 자료형을 살펴보면, 정수(Integer), 실수(Double), 복소수(Complex), 문자(Character)로 나눌 수 있다. 원소의 자료형을 모아둔 집합의 자료형은 원소를 모아둔 포장지나 상자와 같다. 집합의 자료형은 <그림 3-1>과 같이 벡터(Vector)를 기본 단위로 해서 벡터를 여러 개 모은 행렬(Matrix), 행렬에 이름을 붙이고, 데이터처리에 알맞게 만든 데이터프레임(Dataframe)과 데이터프레임과 유사하지만 tidyverse 패키지에 맞게 약간 수정한 티블(Tibble), 마지막으로 여러 자료형을 모아서 묶어 놓기만한 리스트(List)로 이루어져 있다.

우선 행렬을 살펴보기 위해 벡터를 행렬로 다음과 같이 만들어 보자.

```
> matrix(c(1,2,3,4),ncol=2, byrow=T)    (3.1)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

식 (3.1)의 명령어 아래쪽은 실행결과이다. 벡터는 특별한 설명이 없으면 아래로 이어져 있는 열벡터를 의미하므로 차례로 불러서 행으로 입력하라는 옵션(byrow=T)을 지정해야 식 (3.1)과 같은 결과를 얻을 수 있다. 식 (3.1) 행렬의 원소는 행과 열의 주소를 통해 불러올 수

하면 결측값 NA를 포함하고 있어 결과 또한 NA가 된다. 대신에 sum(c(1,NA,3), na.rm =T)를 실행하면 결과 4를 얻을 수 있다.

있다. 식 (3.1)의 원소를 변수 ex에 저장했다고 하면 ex[1,2]의 값은 1행 2열의 원소인 2를 불러온다. 원소 호출에 관한 상세한 사항은 Vernables et al.(2023)과 노만 매트로프(2014)를 참조하기 바란다.

데이터프레임과 티블은 자료 처리에 특화된 형태로 둘은 앞선 <그림 2-1, 2-2>와 같은 형식으로 구성되어 있다. <그림 2-1, 2-2>의 자료에서 열은 특성을 나타내고, 행은 개별 관측값을 나타낸다. <그림 2-2>를 기준으로 보면 열은 에이전트의 고유식별번호(who), 색깔(color), x좌표(xcor), y좌표(ycor), 시간(Time)이라는 특성을 나타내고, 행은 이런 특성을 가진 관측값을 나타낸다.

데이터프레임과 티블은 변수명을 정하는 규칙이나, 열의 길이가 다를 때 처리하는 방식, 처음 자료를 만들 때 앞서 지정한 열이름을 그대로 사용할 수 있느냐 여부 등 몇 가지 차이가 있지만(Wickham, 2019), 특별하게 복잡한 자료가 아니라면 처리할 때 큰 차이가 나지 않으므로 여기서는 데이터프레임을 기준으로 설명한다. 둘은 거의 완전히 변환 가능하므로 편리한 방식을 선택해서 사용하면 된다.

데이터프레임은 각 열의 이름(변수명)이 있다. 앞선 <그림 2-1>의 왼쪽에 있는 자료는 열의 이름이 S1, S2, S3, Index이고, 오른쪽 자료는 열의 이름이 Index, Prob이다. 열의 이름(또는 변수명)을 불러오는 명령어는 colnames이고 이름을 새롭게 바꾸는 명령어는 renames이다.

아래 식 (3.2a)는 숫자 1,2,3의 원소를 가지는 이름이 x인 자료를 첫 열로, 두 번째 열은 문자 kim, yoon, jung을 원소로 가지는, 이름이 y인 데이터프레임을 ex1이라는 변수에 저장하는 명령어이다. 식 (3.2b)는 데이터프레임 ex1의 이름을 불러오는 명령어이고, 식 (3.2c)는 ex1의 이름을 x, y 대신 Index, Name으로 바꾸라는 명령어이다. 식 (3.2)의 실행 결과는 다음 <그림 3-2>에서 확인할 수 있다.

```
> ex1 <- data.frame(x=1:3, y=c("kim", "yoon", "jung") )   (3.2a)
> colnames(ex1)                                            (3.2b)
> renames(ex1, Index = x, Name = y)                        (3.2c)
```

A data.frame: 3 × 2

x <int>	y <chr>
1	kim
2	yoon
3	jung

A data.frame: 3 × 2

Index <int>	Name <chr>
1	kim
2	yoon
3	jung

<그림 3-2> ex1의 최초 모습(왼쪽)과 변수명(열이름) 변환후(오른쪽)

데이터프레임의 원소는 행렬과 같이 주소로 불러올 수도 있다. ex1[3,2]는 jung이라는 문자열을 불러온다. 한편 ex1[,2]은 ex1의 2열을 불러오는 명령어이며, 이는 행렬의 주소를 활용한 것이다. ex1[,2]는 kim, yoon, jung의 원소를 가지는 벡터를 불러온다. 이는 데이터프레임이 벡터들을 모아놓은 것이라는 점을 명확하게 보여준다. 한편 ex1의 2열의 이름인 Name을 활용해서 ex1$Name을 통해 불러올 수도 있다. ex1$Name은 kim, yoon, jung을 원소로 갖는 벡터의 변수명이다. $는 '속에 들어 있는'의 의미로 읽을 수 있다. 즉, ex1$Name은 ex1 속에 들어 있는 Name이라는 변수를 의미한다.

ex1에 들어 있는 변수 Index와 Name은 ex1이라는 데이터프레임의 포장을 제거할 수도 있는데, attach(ex1)이라는 명령어를 실행하면 데이터프레임 ex1 안에 있는 변수들이 각각의 변수명을 가진 벡터로 풀어 헤쳐진다. 이를 다시 묶어서 데이터프레임 안으로 넣으려면 detach(ex1, Index, Name)을 실행하면 된다.

이제 가장 큰 집합의 자료형인 리스트에 대해서 살펴보자. 리스트는 형태와 무관하게 어떤 집합이라도 모을 수 있는 자료형이며 다음 식 (3.3)은 이를 잘 보여준다.

```
> ex2 <- list(x1 = ex1, x2=c(1,2,3), x3=1000)
```
(3.3)

식 (3.3)을 실행하면 ex2에 x1, x2, x3를 원소로 갖는 아래 <그림 3-3>과 같은 리스트가 저장된다.

$x1

A data.frame: 3 × 2

x <int>	y <chr>
1	kim
2	yoon
3	jung

$x2 1 · 2 · 3

$x3 1000

<그림 3-3> 리스트 ex2

x1에는 ex1에 있던 데이터프레임이, x2에는 숫자 1,2,3으로 이루어진 벡터가, x3에는 숫자 1000이 저장되어 있다. 데이터프레임과 달리 행렬의 형태가 아니며 다양한 형태를 가진 자료를 동시에 포함하고 있다. 여기서 주목할 것은 ex2의 변수명 앞에 $가 붙어 있다는 점이다. 데이터프레임에서 $를 '속에 들어 있는'이라고 부를 수 있었던 이유는 데이터프레임이 앞선 <그림 3-1>에서와 같이 리스트의 부분집합으로 리스트의 특징을 가지고 있기 때문이었다. ex2$x1은 ex1과 동일한 데이터프레임이고, ex2$x2는 1,2,3으로 이루어진 벡터, ex2$x3은 1000이라는 숫자이다.

리스트는 특이한 성질을 하나 가지고 있는데 원소를 담고 있는 포장지가 두 겹이라는 점이다. 그 결과 원소를 불러오는 방식에 따라서 리스트 변수형을 유지할 수도 있고, 리스트에 저장된 원소의 변수형으로 바뀔 수도 있다. 예를 들어 식 (3.4a)는 데이터프레임을 담고 있는 리스트이지만, 식 (3.4b)는 ex1과 동일한 그냥 데이터프레임이다.

```
> ex2[1]
```
(3.4a)

```
> ex2[[1]]
```
(3.4b)

식 (3.4b)는 ex2$x1과 동일하지만 ex2[1]은 담겨 있는 내용은 비슷하지만 리스트 포장지가 아직 남아 있다. 이 차이가 문제를 만들 수 있다. ex[1][,2]는 데이터프레임의 두번째 열에 있는 원소인 kim, yoon, jung으로 구성된 벡터를 불러올 것 같지만, 실행되지 않는다. 반면 ex[[1]][,2]는 kim, yoon, jung으로 구성된 벡터를 불러온다. 포장지가 두 겹인 리스트의 특징으로 인해 한 겹만 풀어서는 그 안에 있는 원소들을 끄집어낼 수 없는 것이다.

원시자료변환 과정에서 의도하지 않아도 사용하는 명령어가 리스트의 형태로 자료를 변환하는 경우가 있다. 이때 리스트와 관련해서 포장지가 두 겹이라는 특징을 이해하지 못한다면 에러를 찾기 어려울 수 있다. 이런 점을 피하기 위해서 리스트를 완전히 벡터로 풀어 일렬로 세우는 unlist 명령이 있다. 데이터프레임의 attach 명령도 일종의 데이터프레임의 포장지를 제거하고 벡터로 만들었던 점을 상기해 보면 데이터프레임이 리스트의 부분집합임을 다시 한번 확인할 수 있다.

이제 자료형들을 변환하는 방법을 알아보자. 먼저 원소의 자료형인 정수, 실수, 복소수는 as.integer, as.double, as.complex 명령어로 변환할 수 있다. 변환할 때 정수, 실수, 복소수의 순서나 역순으로 차례로 변환할 수 있다. 이를 건너뛰어 변환할 수는 없다. 문자형은

as.character 명령어로 변환할 수 있으며 문자를 정수, 실수, 복소수로 변환할 수 있다. 아래 식 (3.5)은 문자 1을 실수로 변환한다. 큰따옴표 안에 있는 "1"은 숫자가 아니며 문자이다. 이 문자는 사칙연산을 하면 에러가 난다. 연산을 위해서는 식 (3.5)와 같이 계산이 가능한 자료형으로 변환해 주어야 한다.

```
> as.double("1")                    (3.5)
```

A data.frame: 3 × 4

x1.x	x1.y	x2	x3
<int>	<chr>	<dbl>	<dbl>
1	kim	1	1000
2	yoon	2	1000
3	jung	3	1000

<그림 3-4> as.data.frame(ex2) 변환 결과

집합의 자료형은 벡터, 행렬, 데이터프레임, 티블, 리스트로 as.vector, as.matrix, as.data.frame, as_tibble, as.list 명령어로 변환할 수 있다. 그러나 자료형의 변환은 순서가 없지만 자료의 형태에 따라서 변환이 안되는 경우도 있고, 예상하지 못한 결과가 나타날 수도 있으므로 세심한 주의가 필요하다. 예를 들어 as.data.frame(ex2)는 데이터프레임은 행의 수가 일정해야 하므로 차이가 나는 값은 알아서 채우는 형태로 앞선 <그림 3-4>와 같이 변환된다. 이런 변환이 의도한 결과라면 괜찮지만 그렇지 않으면 에러를 만들지는 않더라도 분석자료가 예상과 완전히 달라질 수 있으므로 유의해야 한다.

4. Tidyverse를 활용한 전처리 기법

원시자료를 전처리하고 정제하는 원시자료 변환(Data Wrangling)은 자료를 나누고, 나눠진 자료를 다시 변환하고, 합치는 과정을 포함한다(Rattenburry et al., 2017). 이런 맥락에서 여기에서는 Tidyverse 패키지의 명령어를 활용하여 자료를 나누고, 변환하고, 합치는 방법을 살펴본다.

원시자료변환 과정은 변수 이름을 만들거나 이름을 바꾸고 특정 문자를 찾아내는 등의 문자열과 관련된 작업을 반드시 포함한다. 따라서 원시자료변환 과정의 사례를 본격적으로 살펴보기 앞서 공통적으로 필요한 문자열과 변수명을 다루는 법을 먼저 알아보자.

문자열을 다룰 때 가장 중요한 명령어인 paste는 문자를 합치는 명령어이다. paste("a","1")은 "a 1"을 결과로 출력한다. paste는 문자를 합치되 두 문자 사이에 공백을 둔다. 이 공백을 없애기 위해서는 paste0라는 명령어를 활용하면 된다. 이를 응용해서 var1, var1, …, var9, var10로 명명되는 10개의 변수명은 paste0("var",1:10)을 활용하면 만들 수 있다.

var1에 1,2,…,10을 저장하는 명령어는 var1 = 1:10 이다. 그런데 이 방식이 아니라 다음 명령어를 활용해도 동일한 작업을 수행할 수 있다.

```
> assign("var1", 1:10)              (4.1)
```

assign은 문자열을 변수 이름으로 해서 자료를 저장하는 명령어다. 식 (4.1)을 수행하면 R에서는 1,2,…,10이 저장된 var1이라는 변수가 생긴다. 여기에서 문자열 "var1"과 변수 var1이 같지 않다는 점을 주목할 필요가 있다. 문자열 "var1"과 변수명 var1의 차이는 문자열 "1"과 숫자 1의 차이와 유사하다. assign은 문자열을 변수 이름을 나타내는 자료형으로 변환하고, 그 안에 자료를 할당(저장)하라는 명령어이다. 문자열을 변수이름으로 바꾸기 위해서는 str2expression이라는 명령어가 필요하다. str2expression은 as.integer나 as.double과 유사

한 변환 명령어이다. 한편, 변수 이름에 들어있는 값을 불러오는 것은 eval이라는 명령어가 필요하다.[4] 프롬프트 창에 var1이라는 변수 이름을 실행하면 변수에 저장된 자료나 값이 호출되는데, 이 과정은 프롬프트에서 eval을 보이지 않게 수행한다고 생각해도 무방하다. 그러나 문자열에서 변환한 변수 이름은 eval이라는 명령어를 명시적으로 수행해야 저장된 값을 불러온다. 식 (4.2)는 문자열 "var1"에 저장되어 있는 자료를 불러오는 명령어이다. 즉, var1을 프롬프트에서 실행하는 것과 동일하다.

```
> eval(str2expression("var1")          (4.2)
```

식 (4.1)과 (4.2)는 이해하기 쉽지 않고, 심지어 식 (4.2)는 비효율적으로 보이기도 하지만 여러 변수에 여러 자료를 할당하거나 불러올 때 반복문(for loop)를 통해 작업을 자동화할 수 있는 중요한 명령어이다. 이를 확인하기 위해 var1에 10에서 20, var2에는 20에서 30 ,…,var10에는 100에서 110을 저장하는 명령을 만들어 보면 다음 식 (4.3)과 같이 같다.

```
> val_list = paste0("var",1:10)
> for (i in 1:10){
    assign(var_list[i],(10*i):(10*i+10))
  }                                    (4.3)
```

이 자료를 var_new1에서 var_new10에 복사하는 방법은 다음 식 (4.4)와 같다.

```
> val_new_list = paste0("var_new",1:10)
> for(i in 1:10){
assign(var_new_list[i],eval(str2expression
(var_list[i])))
}                                      (4.4)
```

텍스트를 처리할 때 paste와 더불어 사용빈도가 높은 명령어는 특정 조건을 만족하는 문자를 찾아내는 str_detect이다. 다음 명령식 (4.5)는 "who", "color", "xcor", "ycor", "Time"으로 이루어진 앞선 <그림 2-2>의 NetLogo 시뮬레이션 결과에서 cor이라는 문자열이 들어간 변수 이름을 찾는 명령이다.

```
> str_detect(c("who", "color", "xcor", "ycor",
"Time"), "cor")                        (4.5)
```

str_detect의 두번째 인수인 "cor"은 정규표현식(Regular Expression)으로 문자열에서 특정 패턴을 찾는 방식으로 널리 활용되는 방식이다. 상세한 내용은 Wickham(2021)과 DataCamp(2022)를 참조하기 바라며, 원시자료 변환에 활용도가 높은 정규 표현식은 <표 4-1>과 같다.

<표 4-1> 간단한 정규표현식

정규표현식	찾는 패턴
^a	첫문자가 a
a$	끝문자가 a
.a.	중간에 a를 포함한 세 개의 문자
^ak.*b$	처음에는 ak로 시작하고 마지막이 b로 끝남

식 (4.5)에서 두번째 인수 "cor" 대신에 "cor$"을 넣으면 끝이 cor로 끝나는 xcor, ycor

4) expression은 R 명령어 표현을 일컫는 자료형이다. 'a <- c(1,2,3)'은 프롬프트에서 입력되면 1,2,3을 a에 저장하라는 R의 명령어이지만, 문자열로 볼 수도 있다. 사람은 상황을 고려하여 이를 쉽게 구분하지만, 컴퓨터는 이를 명확하게 정해주어야 한다. 문자 '1'과 숫자 1을 구분하는 것과 같이 문자열은 문자(텍스트)에 불과하고, expression은 프롬프트에 입력되는 R의 명령을 나타낸다. 이런 맥락에서 eval 명령어는 R 명령어를 입력하고 엔터를 눌러 실행하는 것과 동일한 명령어이다.

을 찾는다. 실제로는 False, False, True, True, False라는 결과를 제시한다.

이제 Tidyverse를 활용해서 본격적으로 원시자료를 변환하는 과정을 살펴보자. 앞서 설명한 바와 같이 원시자료 변환은 자료를 쪼개고, 변환하고, 다시 붙이는 과정으로 이루어져 있다. 이를 위해 우선 자료를 쪼개고 붙이는 과정을 살펴보고, 변환하는 과정은 R4.2.3 버전이후 새롭게 추가된 reframe이라는 명령어를 중심으로 살펴본다.

Tidyverse는 원시자료변환 과정에서 파이프 연산(Pipe Operation)을 사용한다. 파이프 연산은 원하는 데이터를 얻을 때까지 원시자료변환 과정을 직관적으로 이해하기 쉽게 한 단계 한 단계씩 진행하며 앞 단계의 결과를 그대로 활용하는 방식으로 진행되는 자료 변환 기법이다.

이런 과정은 물을 원하는 압력으로 가정에 공급하기 위해서 큰 수도관에서 점차 작고 가는 수도관으로 바꾸면서 압력을 낮추는 과정과 유사하다. 이런 맥락에서 데이터 변환을 차례로 연속적으로 수행하는 방법을 수도(?) 파이프(Pipe) 연산이라고 한다(윤봉규, 2021).

R에서 파이프 연산은 '%>%'를 통해 수행한다. 앞선 <그림 3-2>의 ex1은 열이름이 Index와 Name으로 구성된 데이터프레임이다. ex1에서 Index가 2 이상인 경우만 뽑는 명령어는 다음과 같다.

```
> ex1 %>% filter(Index >= 2)          (4.6)
```

식 (4.6) 뒤에 '%>% filter(Name == "yoon")'을 추가하면 ex1에서 Index가 2 이상인 자료를 뽑고, 이 자료를 바탕으로 Name이 yoon인 자료를 뽑으라는 명령이 된다. 이렇듯 파이프 연산은 순차적으로 자료를 변화시켜 가는 과정을 코드에 반영할 수 있다.

Tidyverse에서 자료를 분리하는 방법은 행을 기준으로 분리하는 방법과 열을 기준으로 분리하는 방법이 있다. 행을 기준으로 분리하는 방법은 식 (4.6)과 같이 filter 명령어를 사용한다. 열을 기준으로 분리하기 위해서는 R의 주소체계를 활용한다. 즉 ex1의 첫 열은 ex1[1], 두번째 열을 ex1[2]를 활용하면 호출할 수 있다. 이를 파이프 연산으로 활용하면 다음과 같다.

```
> ex1 %>% .[1]                        (4.7a)
> ex1 %>% .[2]                        (4.7b)
```

식 (4.7)의 [1], [2] 앞에 붙어 있는 '.'은 앞의 변수명 ex1을 부득이하게 사용해야 하는 경우 쓰는 약어이다. R의 주소를 부르는 방식은 Tidyverse가 아니라 기본 패키지(Base)에 포함된 명령어이므로 파이프 연산을 완벽하게 구현하고 있지 못하다. 그렇다고 기본 패키지를 바꾸기에는 여러 문제가 발생할 가능성이 있어 타협한 결과로 나타난 것이 이런 표현이다. 주소 체계와 문자열 정규표현식을 응용하면 다음과 같이 ex1에서 N으로 시작하는 열만 뽑아낼 수 있다.

```
> ex1 %>%  .[str_detect(colnames(.),"^N")]
                                        (4.8)
```

str_detect(colnames(.),"^N")는 열이름(colnames) 중 N으로 시작하는 열을 찾으라는 명령어이다. colnames도 기본패키지에 포함되어 있어 '.'를 사용하고 있음을 주목할 필요가 있다. R은 오랜 기간에 걸쳐 여러 사람과 단체에 의해서 개발되어 온 역사성으로 인해서 이런 식의 타협의 결과가 자주 나타난다.

이제 자료를 합치는 방법을 앞선 <그림 2-1>의 단계형 정비 모형 원시자료 사례를 통해 살펴보자. <그림 2-1>의 왼쪽에 있는 상태참조표를 먼저 만들어 dat1_1에 저장하는 명령은 다음 식 (4.9)와 같다.

```
> dat1_1 <- data.frame(S1 = 0:5) %>% cross_join(data.frame(S2 = 0:10)) %>% cross_join(data.frame(S3 = 0:5)) %>% cbind(data.frame(Index=1:(6*11*6)))   (4.9)
```

cross_join은 원소의 조합으로 순서쌍을 만드는 명령어로 S1의 원소 0,1,…,5와 S2의 원소 0,1,…,10으로 이차원 순서쌍을 만들고 다시, S3의 원소들을 포함한 순서쌍을 만들고 이를 일련번호를 나타내는 Index 변수와 열로 붙이는(cbind) 코드이다.

식 (4.9)의 코드는 파이프연산을 통해 자료변환을 순차적으로 진행하면서 변환 과정의 가시성을 높인다는 점을 주목할 필요가 있다. 파이프 연산은 자료의 변환을 블랙박스처럼 결과만 얻는 방법이 아니라 그 과정을 순차적으로 진행함으로써 가시성을 높여 변환 과정의 오류를 줄이는 이점을 제공한다. 한편 두 자료를 옆으로 붙이는 것은 cbind (Column Bind), 아래 위로 붙이는 것은 rbind (Row Bind)를 사용한다.

dat1_2에 <그림 2-1>의 오른쪽 표인 안정상태확률이 저장되어 있다면, 다음 <그림 4-1>과 같이 두 자료를 Index를 기준으로 합치는 방법은 두 가지가 있다. 두 표의 Index가 순서대로 정렬되어 있는 경우에는 식 (4.10a)를 실행하면 되고, 정렬되어 있지 않은 일반적인 경우에는 식 (4.10b)를 실행하면 된다.

```
> dat1_3 <- dat1_1 %>% cbind(dat_1_2[2])   (4.10a)
> dat1_3 <- dat1_1 %>% full_join(dat1_2, by="Index")   (4.10b)
```

A data.frame: 396 × 5

S1	S2	S3	Index	Prob
<int>	<int>	<int>	<dbl>	<dbl>
0	0	0	1	0.0008670620
0	0	1	2	0.0018991894
0	0	2	3	0.0033288712
0	0	3	4	0.0001890095
0	0	4	5	0.0055546257
0	0	5	6	0.0009027354
0	1	0	7	0.0003486690
0	1	1	8	0.0009073764
0	1	2	9	0.0003500549
0	1	3	10	0.0003542620

<그림 4-1> 단계형 정비 모형 최종 결과(dat1_3)

이제 2장의 사례 설명에서 예시로 들었던 상황인 정비팀은 전원 업무중인데 사전검사와 출고검사는 절반 이하로 일하는 비율 계산해 보자. 이를 구하기 위해서는 <그림 4-1>의 자료가 저장되어 있는 dat1_2에서 S2는 10이고 S1과 S3는 0, 1, 2인 (0, 10, 0), (0, 10, 1) …(2, 10, 2) 상태를 뽑아서(filter), Prob 변수명을 가진 열을 선택한 후(select) 그 확률값을 다음과 같이 더하면(sum) 된다.

```
> dat1_3 %>% filter(S1 <= 2 & S2 == 10 & S3 <= 2) %>% select(Prob) %>% sum
```

이제 앞선 <그림 2-2>의 미사일 방어와 관련된 NetLogo 시뮬레이션 결과 자료를 앞선 <그림 2-3>과 같이 홍군의 미사일이 격추되지 않고 지나간 경로를 그리기 위한 자료로 변환해 보자.

A tibble: 544 × 3

xcor	ycor	Cnt
<int>	<int>	<int>
-15	11	4
-13	-16	3
-10	16	1
-3	-13	2
⋮	⋮	⋮
15	0	2
8	-1	1
1	-8	1
10	7	3
11	-9	3

<그림 4-2> 격추되지 않은 미사일의 경로(dat2_4)

이를 위해서는 1) 격추되지 않은 홍군의 미사일을 파악해서, 2) 그 미사일의 경로 자료만 뽑아서, 3) 미사일이 지나간 경로의 자료를 격자에 적합하게 정수 좌표로 변환하고, 4) 각 격자 좌표를 지나간 미사일의 수를 세어 이를 포함하는 앞선 <그림 4-2>와 같은 표를 만들어야 한다. <그림 4-2>의 Cnt는 각 좌표를 지나간 미사일의 수를 나타낸다. 앞선 <그림 2-2>와 달리 xcor, ycor의 자료형이 실수형을 나타내는 <dbl>에서 정수형을 나타내는 <int>로 변했음을 알 수 있다.

원시자료가 dat2_1[5]에 저장되어 있다고 가정하고, 앞선 <그림 4-2>와 같은 표를 만들기 위해 먼저 격추되지 않은 미사일을 파악해 보자. 격추되지 않은 미사일은 청군의 미사일 방어시스템의 작동시간인 60분 이후에도 살아남아 있는 홍군의 미사일이라고 가정하면, NetLogo 시뮬레이션의 원시자료에서 Time이 60보다 크고 홍군의 색깔 코드인 color가 15인 자료를 뽑아내면 된다. 이는 아래 식 (4.10)과 같이 수행할 수 있다.

> missed <- dat2_1 %>% filter(color == 15 & Time > 60) (4.10)

이제 살아남은 홍군의 미사일은 데이터프레임 missed의 who라는 변수 즉, missed$who에 저장되어 있다. 이를 활용하여 살아남은 미사일의 경로만 추출하는 방법은 다음과 같다.

> dat2_2 <- dat2_1 %>% filter(who %in% missed$who) (4.11)

식 (4.11)의 filter(who %in% missed$who)는 data2_1의 who 변수가 missed$who 안에 (%in%) 포함되어 있느냐 여부를 판단해서 뽑아내는 명령어이다.

좌표별로 미사일이 지나간 횟수를 파악하기 위해서는 좌표를 격자형에 맞게 실수에서 정수로 변환해 주어야 한다. 이는 as.integer(round(xcor)) 명령을 사용해서 다음 식 (4.12)와 같이 수행할 수 있다.

> dat2_3 <- data2_2 %>% reframe(who, xcor = as.integer(round(xcor)), ycor = as.integer(round(ycor)), Time) (4.12)

식 (4.12)의 reframe 명령은 Tidyverse에서 자료를 변환하는 새로운 기법이다. Tidyverse에서 자료를 변환하는 방식은 자료를 추출하고(filter), 변형하고(mutate), 그룹핑하고(group_by), 변수를 선택해서(select), 요약(summarize)하는 과정으로 이루어진다(윤봉규,

5) 원시자료는 저자에게 요청시 csv파일로 제공되며 >dat2_1 <- read_csv("ex2.csv")를 실행하면 dat2_1에 원시자료가 저장된다.

2021). reframe은 mutate, group_by, select, summarize를 한 번에 수행할 수 있는 방법이다. 식 (4.12)를 기존 방식으로 수행하기 위해서는 reframe 대신 mutate명령어를 사용하면 동일한 결과를 얻을 수 있다.

마지막으로 각 격자 좌표별로 지나간 미사일의 수를 세는 방법은 다음식 (4.13)과 같다.

```
> dat2_4 <- dat2_3 %>% reframe(xcor, ycor, Cnt = n(), .by=c(xcor,ycor)) %>% unique
```
(4.13)

위 식 (4.13)의 'Cnt = n(), .by=c(xcor,ycor)'은 xcor, ycor를 기준으로 그룹을 만들어 그 그룹의 빈도를 세어서 Cnt에 저장하라는 옵션이다. 빈도가 아니라 그룹의 평균을 구하고 싶으면, n() 대신 mean()을, 최대값은 max() 등 그룹의 다양한 특성을 구할 수 있다. 한편, reframe의 옵션 by 앞에 마침표 점이 포함되어 있음을 주목해야 한다. 이는 앞서 설명한 기술적 타협의 문제이니 여기에서 상세히 설명하지는 않지만 마침표를 포함하지 않으면 명령이 실행되지 않는다는 점은 주목할 필요가 있다. 마지막에 있는 unique 명령어는 자료가 중복되는 것을 방지하기 위한 명령어이다. xcor, ycor는 같아도 who가 다르면 다른 자료로 인식되었는데, who를 제외하면 동일한 자료의 중복이 된다. 이를 보완하기 위한 명령어로 그룹을 만들어서 자료를 변환한 경우에는 unique를 써주는 것이 좋다. 식 (4.13)은 reframe 대신 group_by와 summarize 명령어를 활용해서 다음과 같이 수행할 수도 있다.

```
> dat2_4 <- dat2_3 %>% group_by(xcor,ycor) %>% summarize(Cnt = n()) %>% unique
```

식 (4.11)-(4.13)는 NetLogo 시뮬레이션의 원시자료인 dar2_1를 출발점으로 해서, 미사일 방어시스템에서 처리하지 못하고 놓친 홍군의 미사일 자료를 뽑아 dat2_2에 저장하고, 미사일이 지나간 경로의 좌표를 정수 좌표로 변환하여 dat2_3에 저장하고, 각 좌표 격자에 지나간 미사일의 수를 세어 dat2_4에 저장한 세 절차를 수행하는 명령이다. 그런데 각 연산을 차례로 수행하는 파이프 연산의 장점을 활용하여 이를 다음과 같이 한 번에 수행할 수도 있다. 이렇게 함으로써 매 연산을 따로 저장하지 않으므로 연산을 효율적으로 수행하면서도, 세 절차가 어떻게 수행되고 있다는 것을 명시적으로 파악할 수 있는 코드를 작성할 수 있다.

```
> dat2_4 <- dat2_1 %>% filter(who %in% missed$who) %>%
  reframe(who, xcor = as.integer(round(xcor)), ycor = as.integer(round(ycor)), Time) %>% reframe(xcor, ycor, Cnt = n(), .by=c(xcor,ycor)) %>% unique
```
(4.14)

한편, dat2_4에 저장된 데이터를 활용하여 <그림 2-2>를 그리는 명령어는 다음과 같다. ggplot을 활용해서 R 그래프를 그리는 방법은 윤봉규(2021)에 상세하게 소개되어 있으니 참고하기 바란다.

```
> ggplot() + geom_tile(data=dat2_3,aes(x=xcor,y=ycor,fill=Cnt)) + scale_fill_gradient(low = "grey", high = "black")+ geom_point(data=missed,aes(xcor,ycor), color="red",size=3) + xlim(-17,17)
```
(4.15)

5. 결론 및 시사점

지금까지 모델링과 시뮬레이션 분야에서 활용도가 높은 방법론인 마코프체인과 에이전트 기반 시뮬레이션 모형에서 자주 나오는 원시자

료를 하나씩 사례로 뽑아, R로 원시자료변환(Wrangling)을 수행하는 기법을 살펴봤다. 그 과정에서 원시자료변환에 필요한 배경지식인 자료형과 자료형의 변환, 정규표현식을 포함한 문자형 자료를 처리하는 방법과 이를 R에서 활용하는 방법 또한 살펴봤다.

본 연구를 통해 국방분야 M&S 연구자와 실무자들이 많은 노력을 들이지 않고 원시자료변환을 위한 기본적 기법을 이해하고, 원시자료변환의 의의를 파악함으로써 향후 이 분야의 표준화를 통한 효율성 제고를 통해 국방분야 M&S 발전에 기여할 수 있기를 기대한다.

참 고 문 헌

[1] 윤봉규, "간소화된 그래픽 문법(Grammer of Graphics) 기반 데이터 시각화," 군사과학연구지 14(1): 29-41, 2021.

[2] 매트로프, 노만 (권정민 역), 빅데이터 분석 도구 R 프로그래밍, 에이콘, 2014.

[3] DataCamp, "A Guide to R Regular Expressions With Examples," DataCamp, 2022. available: https://www.datacamp.com/tutorial/regex-r-regular-expressions-guide.

[4] Dasu T, Johnson T, Exploratory Data Mining and Data Cleaning, John Wiley & Sons, 2003.

[5] Rattenbury, T., J.M. Hellerstein, J. Heer, S. Kandel, and C Carreras, Principles of Data Wrangling: Practical Techniques for Data Preparation, O'Reilly, 2017.

[6] Venables, W.N., D. M. Smith and the R Core Team, Introduction to R:version 4.2.3, 2023.

[7] Wickham, H., "stringr: Simple, Consistent Wrappers for Common String Operations: Vignettes: Regular Expressions (R package version 1.5.0), 2021. available: https://cran.r-project.org/web/packages/stringr/vignettes/regular-expressions.html.

[8] Wickham, H., Advanced R, Taylor & Francis, 2019.

[9] Wickham, H. and G. Grolemund, R for Data Science, O'REILLY, 2017.

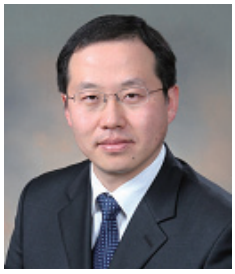
[10] Wickham, H., "Tidy Data," Journal of Statistical Software, 50(10), 2014.

[11] Wickham, H.. "A Layered Grammar of Graphics." Journal of Computational and Graphical Statistics 19(1): 3-28, 2010.

[12] Wilkinson, L., The Grammar of Graphics, Springer, 1999.

[13] Zewe, A., "Q&A: Neil Thompson on computing power and innovation," MIT News June 24, 2022. available: https://news.mit.edu/2022/neil-thompson-computing-power-innovation-0624.

저 자 소 개



윤봉규(E-mail: bkyoon1@gmail.com)
1996 연세대학교 경영학사
1998 한국과학기술원 산업공학 석사
2002 한국과학기술원 산업공학 박사
현재 국방대학교 운영분석전공 교수
관심분야 : Agent Based Modeling, Stochastic Models in Military O.R, Biz. Performance Optimization & Innovation.