Basic Statistics and Data Analysis: Korean Version

기초 통계와 데이터 관리: 한글판

Sanghoon Park

2019-09-10

Chapter 1

Prerequisites

원래 저 같은 경우에는 통계분석을 할 때 주력 툴(tool)로 STATA를 써왔습니다. 하지만 한국에서 STATA로 대학원생이 연구를 수행하는 데 있어서 한 가지 걸림돌이 되는 문제는 바로 'Copyrights'에 있습니다.

STATA는 유저들 사이에서 온라인/오프라인 등을 통하여 여러가지 매뉴얼들과 피드백이 이루어져 강력한 이점을 지닌 도구인 것은 사실이지만 그 가격이 만만치가 않습니다. 적어도 제가 석사까지 마쳤던 학교에서는 STATA를 설치하는 데 있어서 학교 차원의 지원 등은 없었기 때문에 높은 가격을 감수하고 구매하던가 혹은 어떻게든(?) 구해서 사용하는 방법밖에는 없었습니다.

그에 비해서 **R**은 오픈소스라 어느 정도 자유롭게 접근할 수 있습니다. 또한 STATA 못지 않게 폭넓게 유저들 간의 소통을 통해 매뉴얼이 제공되며, 피드백이 이루어지는 강력한 통계패키지입니다.[1]

정치학을 공부하는 입장에서 STATA에서 R로 갈아타는 데 가장 어려웠던 점은 초입의 진입장벽이었다고 할 수 있습니다. 일종의 Trade-offs 관계로까지 느껴졌던 것이 **R**의 유연함은 곧 어떠한 결과를 얻기 위해서는 하나하나의 요소를 유저가 직접 조합할 수 있어야 하고, 각 요소의 특성을 파악해야 한다는 것이었습니다. 예를 들어, STATA에서는 변수의 특성 등 통계학에서 일반적으로 고려하는 부분들에 집중하면 되었지만 **R**에서는 객체(objects)의 특성들(List인지, Character인지, Factor인지, Vector인지 등)을 살펴보아야 했습니다. 만약 객체 특성을 고려하지 않을 경우에 결과가 크게 달라질 수 있기 때문입니다. 이러한 진입장벽의 문제가 종종 **R**을 시작하자마자 포기하게끔 만드는 결과로 이어지는 경우를 종종 보았습니다.

이 Git은 데이터 분석의 기본을 소개하는 데 목적을 가지고 있습니다. Git에 올라오는 자료들을 이해하는 데 있어서 수학적인 배경지식은 거의 요구되지 않지만 간단한 응용수학들에 대해서는 다루게 됩니다. 이 Git의 자료들은 통계분석을 수행하는 데 필수적인 개념적 지식들을 익숙하게 하는 것에 있습니다. 또한 **R**을 이용하여 데이터 관리 및 가시화(visualization)에 필요한 실용적인 기법들을 익히는 것을 기대합니다. 나아가 정량연구(quantitative research)에서 요구되는 재현가능성(replicability)와 투명성(transparency)을 제고할 수 있는 일련의 작업환경들을 만드는 훈련을 같이 합니다. 이 Git의 내용은 어디까지나 기본적인 내용들을 담고 있기 때문에 정량연구

[1] 피드백이 잘 이루어진다는 것은 실제 연구분석에 적용되는 패키지의 활용에 대해 커뮤니티에서의 소통이 용이하고, 분석에 필요한 새로운 패키지들에 대한 접근성이 높다는 것을 의미합니다.

방법을 마스터하는 첫 걸음으로써는 적절한 내용들을 담고 있을 것이라고 기대합니다. 정리하면, 이 Git의 자료들을 통해 기대되는 성과는 다음과 같습니다.

- 데이터에서 변수들의 관계를 서술하고 평가하는 데 필요한 통계방법을 이용하고 사용할 수 있을 것입니다.
- 데이터를 보여주기 위해 유용한 그래픽을 활용할 수 있게 될 것입니다.
- 데이터 접근성과 연구 투명성 원칙들을 자신의 연구에 충분히 적용할 수 있게 될 것입니다.
- **R**을 이용하여 데이터를 관리 및 분석할 수 있게 될 것입니다.
- LaTex를 이용하여 과학적 연구의 결과를 페이퍼와 발표자료 등으로 구성해낼 수 있을 것입니다.

이 Git의 주요 자료들은 다음과 같은 자료에 기초하여 작성되었습니다.

- Diez, David D., Christopher D. Barr, and Mine Çetinkaya-Rundel. 2019. OpenIntro Statistics. Fourth Edition. 무료로 여기에 공개되어 있습니다.
- Lander, Jared P. 2013. R for Everyone: Advanced Analytics and Graphics. ISBN-13: 978-0321888037

이외에도 추가로 다음과 같은 홈페이지의 코스들로부터 도움을 받으실 수 있습니다.

- 통계분석에 필요한 수학적 지식의 기본적인 내용들을 다시 복습하시기에 좋은 자료입니다: Havard's Math Prefresher
- 주제별로 **R**을 활용한 분석에 대한 다양한 자료를 제공합니다: DataCamp

R은 일단 하나의 언어라고 생각할 수 있습니다. **R** 자체로도 함수들을 이용해 우리가 원하는 분석을 할 수 있겠지만, 굉장히 편리하고 유용한 플랫폼을 통해서 보다 용이하게, 편리하게 **R**을 사용할 수 있습니다. 바로 RStudio입니다. RStudio는 Graphical User Interface(GUI)로 **R**을 좀 더 직관적으로 사용하는 데 도움을 줍니다.

- **R**을 다운로드 하시려면 여기
- **R**을 설치하고 나서 플랫폼으로 설치할 RStudio

Chapter 2

Introduction to R

R은 프로그래밍 언어에 가까운데 통계분석에도 꽤나 특화되어 있습니다. 요즘 Python 등도 주목받고 있지만 사회과학적 정량연구에 적용하는 정도로 통계분석을 수행하는 유관 학과들에서는 **R**을 많이 사용하는 것 같습니다. **R**은 단순하지만 동시에 복합적인 조합을 통해서 여러가지 분석을 수행할 수 있기 때문입니다.[1]

다음 포스팅에서는 RStudio를 이용한 **R**의 기본적인 요소들을 소개할 것입니다. RStudio에 대한 간략한 소개는 다음의 RStudio 공식 링크에서 살펴볼 수 있습니다.

기본 동작에 관한 소개

일단 **R**에서 일반적으로 사용하는 코드를 제외하고 코멘트를 달고 싶을 때에는 **#** 를 사용합니다. **#** 기호 뒤에 쓴 글들은 **RStudio**에서 작동되지 않스니다. 코멘트를 작성하는 습관을 들여놓는 게 좋은데, 본인이 만들어놓은 코드와 데이터셋 이름을 까먹어서 고생하지 않도록 해주고 다른 사람이 코드를 읽을 때에도 꽤나 유용하기 때문입니다.

- 작동코드를 작성하기 전에 쓰는 설명은 보통 **##**로, 작동코드 옆에 병기하는 인라인 코드(in-line code)는 **#**로 구분하여 코멘트를 작성해줍니다.
    - 즉, **##**은 한 줄이 아예 전체 코멘트일 때, **#**은 **R** 코드 옆에 해당 코드의 내용에 대해 코멘트할 때 사용합니다.
- 예를 들면, 아래와 같은 코드에서 첫 번째 줄은 단순히 어떠한 분석을 수행할지 알려주는 역할만 하고, 아래의 코드가 실제로 작동될 것입니다.

```
## 1+1을 계산해봅시다.
1 + 1  # 답은 2
```

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

[1] 그에 관련된 포스팅은 링크를 참조하시기 바랍니다.

R을 배우면 배울수록, 여러가지 노하우가 체화되고 보다 효율적인 코딩을 시도하게 되는데, 일단 간단한 팁들은 여기에서 살펴볼 수 있습니다. 이러한 팁과 노하우들에 익숙해지면 코드를 읽고, 공유하고, 짜는 게 조금 더 쉬워질 것이라고 생각합니다.

R의 연산자(operations)

일단 사회과학을 연구하고자 하는 입장에서 R을 사용하고자 하니, R이 가진 단순한 수학연산자들이 어떻게 기능하는지를 살펴볼 필요가 있습니다.

- R 안에서 어떠한 수학연산자들이 있는지 살펴보고 싶다면 RStudio 좌측 하단 콘솔에 ?"*" 라고 입력해보면 됩니다.

- 사실 코드블럭의 왼쪽 기호는 몰랐을 수도 있는데, 오른쪽의 의미를 모를 것이라고는 생각하지 않습니다. 따라서 자세한 설명은 넘어가고… 이 중에서 마지막 기호는 조금 익숙해질 필요는 있습니다. 저도 아직 연구에서 변수 조작(manipulation)할 때 사용해본 적은 없는 기호입니다만, 평소에는 크게 쓸 일이 없기 때문에 까먹지 않도록 하는 것이 중요할 것 같습니다.

- 마지막 기호를 이용해서 할 수 있는 대표적인 작업이 루프에서 특정 변수가 홀수인지 짝수인지 구분하게 하는 것 등입니다. 만약 2로 나누어 나머지가 남으면 홀수, 남지 않으면 짝수라고 생각해볼 수 있습니다.

```
## 제곱해보기: a^b라고 할 때, a를 b만큼 곱해주는 것
3^2 #9
```

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

```
2^3 #8
```

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

```
## 나머지 구하기 : a%%b라고 할 때, a를 b로 나누고 몫이 아닌 나머지를 보여줍니다.
27 %% 7 #6
```

```
## [1] 6
```

```
## 마지막으로 연산자를 가지고 계산할 때, 그리고 연산자 뿐 아니라 코딩 전체에 있어서
## 순서를 잘 고려하여 코딩해야 합니다. 아래 두 계산은 완전히 결과가 다릅니다.
3 + 4 / 7
```

```
## [1] 3.571429
```

```
(3 + 4) / 7
```

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

R의 기본적인 자료 유형

R은 여러 가지 자료 유형을 제공합니다. 예를 들어 바로 위에서 계산할 때 사용하였던 숫자는 말 그대로 R에서 숫자형(numeric)으로 간주됩니다. 기본적으로 반드시 알아두어야 할 자료 유형은 다음과 같습니다.

- 숫자형(numeric): 1.111 같이 소수점 값을 가지는 자료 유형입니다.
- 정수형(integer): 2와 같은 자연수를 말하는 데, 소수점 값을 갖지 않는 것. 숫자형이랑 비슷합니다.
- 논리형(logical): 부울리안 값(Boolean values), 참(TRUE)/거짓(FALSE)을 가지는 자료 유형입니다.
- 문자형(character): 문자열(text or string)의 값을 지니는 것으로 인용자("")를 사용하여 입력합니다.

자료 유형을 살펴보았으니, 잠깐 변수 배정(variable assignment)에 대해 얘기해보겠습니다. 뭐랄까, 걍 어떤 변수를 만드는 거라고 생각하면 됩니다. 변수는 통계학에서 가장 기본적인 개념 중 하나이니 따로 설명할 필요는 없을 것 같습니다.

R에서 우리는 어떤 값(values)을 객체(object)에 저장합니다. 이때 객체는 함수(functions)일 수도 있고 그래프(plots)이거나 혹은 데이터셋(datasets)일 수도 있습니다. 즉, 그냥 콘솔에다가 2를 치면 결과 창에 2가 나오기는 하겠지만 그거 자체를 바로 분석에 사용할 수는 없고, 사용한다고 하더라도 지속적이지 않습니다. 계속 그 값을 써먹으려면 우리도 그 값에 이름을 붙여줘야 합니다. 그 값의 성격에 따라서 그것이 저장된 용기(container)의 이름도 바뀐다고 생각하시면 편합니다. 어떤 식 자체를 저장했다면 함수의 형태로, 혹은 숫자만 넣어놨다면 걍 숫자 하나가 담긴 객체가 될 것입니다. R에서 변수를 배정하기 위해서는 <- 나 =의 기호를 사용하여야 한다. 사용해본 경험으로는 <-를 더 추천합니다.[2]

```
## 사과라는 변수에다가 값을 집어넣어 보겠습니다
apples <- 4
## 사과에 담긴 값을 출력합니다.
apples
```

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

이제는 객체/변수와 연산자를 같이 사용하여 계산을 해보겠습니다.

```
## 오렌지라는 변수에다가 값을 집어넣어 보겠습니다.
oranges <- 6
## 오렌지에는 6이 들어가 있고 사과에는 4가 들어가 있습니다. 두 개를 더해 보겠습니다.
apples + oranges
```

```
## [1] 10
```

다음으로는 숫자가 아니라 다른 형태의 자료를 담아보겠습니다. 주의할 점은 자료 형태가 서로 다른 변수들 끼리는 연산자를 통해 계산할 수 없다는 점입니다.

```
## 오렌지 객체에 문자열 자료를 다시 저장해봅니다.
oranges <- "six"
```

[2] 참고로 R은 한글로 변수를 입력하는 기능을 제공하지 않습니다.

```
## 오렌지와 사과를 더해 보겠습니다.
apples + oranges # 에러메세지를 확인할 수 있습니다.
```

```
## Error in apples + oranges: non-numeric argument to binary operator
```

논리형 연산자 (Logical operators)

앞서 수학연산자를 간단하게 살펴보았는데, 이번에는 논리형 연산자를 한 번 살펴 보겠습니다. 논리형 연산자는 부울리안 값(TRUE or FALSE)을 나타냅니다. 논리형 연산자는 아래와 같고, 좀 더 구체적인 내용은 여기에서 확인할 수 있습니다.

```
a < b    # a가 b보다 작다는 것을 보여줍니다.
a <= b   # a가 b보다 작거나 같다는 것을 보여줍니다.
a > b    # a가 b보다 크다는 것을 보여줍니다.
a >= b   # a가 b보다 크거나 같다는 것을 보여줍니다.
a == b   # a와 b가 같다/동일하다는 것을 보여줍니다.
!a       # a가 아니라는 의미입니다.
```

위의 연산자를 가지고 아래의 연습을 해보겠습니다.

```
## 1이 2보다 작을까?
1 < 2 # TRUE, 사실이라는 결과를 얻을 것입니다.
```

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

```
## 1 더하기 1이 3일까?
1 + 1 == 3 # FALSE, 거짓이라는 결과를 얻을 것입니다.
```

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

R에서 TRUE는 1과 같고, FALSE는 0과 같습니다. 그렇다면 다음의 연습을 해보겠습니다.

```
apples <- 4
oranges <- TRUE
apples + oranges
```

```
## [1] 5
```

R은 매우 까다롭습니다. 하나라도 다르면 기대한대로 결과가 나오지 않거나 작동하지 않고 에러메세지를 띄우기도 다반사입니다. 아래의 경우를 살펴보겠습니다.

```
## 대문자와 소문자를 가리는 R
oranges <- "six"
Oranges <- "Six"
oranges == Oranges # six와 Six는 앞의 문자가 하나 다르기 때문에
```

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

```
                    # FALSE라는 결과를 얻을 것입니다.
```

벡터 (Vectors)

앞서 변수에 대해서 얘기했는데, 이번에는 벡터에 대해서 살펴볼 것입니다. 벡터는 원하는 만큼 많은 데이터를 일차원(one-dimension)에 배열할 수 있는 형태의 자료로, **R**에서 벡터를 만들기 위해서는 `c()` 형태의 함수를 이용합니다. 이 함수의 괄호 내부에 원하는 요소들을 콤마(`,`)를 이용해 배열하면 하나의 벡터에 담을 수 있습니다.

```
## 숫자형 자료들이 담긴 벡터를 만들어 보겠습니다.
num_vec <- c(1, 2, 3)
```

이번에는 문자형 자료가 담긴 벡터와 논리형(부울리안) 값을 가진 벡터를 만들어 보겠습니다. 만들고 난 이후에 `class()`나 `typeof()` 함수를 이용하여 벡터에 어떠한 형태의 자료가 담겼는지를 확인할 수 있습니다.

```
## 여러 자료 유형을 이용하여 벡터를 만들어 보겠습니다.
mix_vec <- c(1, "Hi", TRUE)
class(mix_vec)  # Character라는 답을 얻게 됩니다.
```

```
## [1] "character"
```

```
typeof(mix_vec)
```

```
## [1] "character"
```

하나라도 다른 유형의 자료가 벡터 안에 포함되면 기대한 결과를 얻지 못할 수 있기 때문에 항상 `class()` 함수로 확인해주는 것이 필요합니다. 그리고 벡터도 수학연산자들을 이용해 일종의 계산이 가능한데, 단, **R**은 굉장히 '까다롭다'는 것을 기억하셔야 합니다. 벡터의 경우에는 요소 하나하나를 구별해서 인식하기 때문입니다. 다음의 예를 살펴 보겠습니다.

```
c(1, 2, 3) + c(4, 5, 6)
```

```
## [1] 5 7 9
```

```
c(1 + 4, 2 + 5, 3 + 6)
```

```
## [1] 5 7 9
```

위의 두 결과는 동일합니다. 정확히는 위의 식을 아래와 같은 식으로 **R**이 계산하여 결과를 보여준다고 하는 것이 맞을 것입니다. 다른 연산자들은 어떨까요? 그리고 만약 벡터의 요소 개수가 서로 다르면(보통 길이가 다르다고 한다) 어떻게 될까요? 다음 코드를 통해 한 번 살펴 보겠습니다.

```
c(1, 2, 3) * c(4, 5, 6)      # 4, 10, 18의 결과값을 얻게 될 것입니다
```

```
## [1]  4 10 18
```

```
c(1, 2) + c(4, 5, 6, 7, 8)  # 5, 7, 7, 9, 9의 결과를 얻게 되고, 두 벡터의 길이가
```

```
## Warning in c(1, 2) + c(4, 5, 6, 7, 8): longer object length is not a
## multiple of shorter object length
```

```
## [1] 5 7 7 9 9
```

```
                                # 다르다는 경고 메시지를 보게 될 것입니다.
c(1, 2) * c(4, 5, 6, 7, 8)  # 4, 10, 6, 14, 8
```

```
## Warning in c(1, 2) * c(4, 5, 6, 7, 8): longer object length is not a
## multiple of shorter object length
```

```
## [1]  4 10  6 14  8
```

아래의 두 코드를 살펴보면 짧은 길이의 벡터가 긴 길이의 벡터에 반복해서 계산되는 것을 알 수 있습니다. 벡터는 여러 개의 요소 값(element value)을 가질 수 있는데, 그 중에서 하나의 값을 원할 경우에는 대괄호를 이용합니다.

```
num_vec <- c(11, 21, 63, 44, 95, 86)
num_vec[3]          # 63이라는 값, 벡터의 세 번째 값을 얻게 됩니다.
```

```
## [1] 63
```

```
num_vec[c(1,4)]    # c(1, 4)는 첫 번째와 네 번째의 값을 산출하라는 뜻으로 11, 44라는
```

```
## [1] 11 44
```

```
                   # 결과를 얻게될 것입니다.
```

매트릭스 (Matrices)

이번에는 매트릭스를 살펴보겠습니다. 매트릭스는 일정한 수의 열과 행으로 이루어진 두 차원(two dimension)의 집합이라고 할 수 있습니다. 이때, 매트릭스를 이루는 요소들은 같은 유형의 자료들이어야 합니다. 매트릭스는 다음과 같은 함수를 통해 만들어볼 수 있습니다.

```
matrix(1:12, byrow=TRUE, nrow=3)
```

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

매트릭스 함수를 이용하지 않고서라도 벡터들을 `cbind()`, 즉 열(column) 결합 또는 `rbind()`, 행(row) 결합 함수를 이용하여 합쳐 매트릭스를 만들 수 있습니다.

```
c1 <- 1:3  # 1, 2, 3
c2 <- 4:6  # 4, 5, 6
c3 <- 7:9  # 7, 8, 9
cbind(c1,c2,c3)
```

```
##      c1 c2 c3
## [1,]  1  4  7
## [2,]  2  5  8
## [3,]  3  6  9
```

```
rbind(c1,c2,c3)
```

```
##    [,1] [,2] [,3]
## c1    1    2    3
## c2    4    5    6
## c3    7    8    9
```

매트릭스를 구성하는 요소 중 하나만 선택하기 위해서 우리는 대괄호(square brackets)를 사용합니다. 대괄호라고 하지만 이 과정은 R 프로그래밍에서 인덱싱(indexing)이라고 하는 것입니다. 즉, 행과 열의 목록에서 필요한 요소만을 지정해서 꺼낼 수 있도록 하는 기능입니다. 매트릭스는 두 차원으로 이루어져 있기 때문에, 특정 요소 하나만을 뽑아내기 위해서는 각각 차원에 배정된 숫자, 열 번호와 행 번호가 모두 필요합니다.

```
matrix <- matrix(1:12, byrow=TRUE, nrow=3) # matrix라는 객체에 결과를 저장합니다.
matrix[1, 2]
```

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

```
matrix[1:2, 2:3]   # 더 작은 형태의 매트릭스로 추출되는 것을 확인할 수 있습니다.
```

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    6    7
```

기본적인 수학연산자를 이용하여 매트릭스도 요소들 간 계산을 할 수 있습니다.

```
11 + matrix #아까 저장해 둔 matrix 객체에 11을 더하면 모든 요소에 11이 더해집니다.
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   12   13   14   15
## [2,]   16   17   18   19
## [3,]   20   21   22   23
```

매트릭스를 이용한 계산을 자세하게 알고 싶다면 다음의 링크를 참조하면 좋을 듯합니다.

데이터프레임 (Data frame)

아마 사회과학 연구를 하게 되면 가장 많이 다루게 되는 자료 유형 중 하나일 것입니다. 일반적으로 우리가 사용하는 데이터셋은 거의 데이터 프레임 형태로 불러오게 됩니다. 데이터셋하면 일반적으로 행과 열이 있는 엑셀이 생각나 매트릭스랑 뭐가 다르지? 할 수 있는데, 아까도 말했다시피 매트릭스의 모든 요소는 동일한 유형의 자료여야 합니다. 정치학에서 많이 쓰는 자료 중 COW 데이터를 예로 들어보자면 COW 국가 코드(ccode)는 숫자형인 반면에, 국가 이름(cname)은 문자형입니다. 하나의 데이터셋에 서로 다른 유형의 자료가 담기게 되는 것입니다.

- 국가명: 문자형
- 국가의 GDP: 숫자형
- 어떤 국가가 민주주의인지 여부: 논리형

데이터 프레임은 다양한 유형의 자료를 열과 행의 틀 안에서 저장할 수 있도록 돕습니다.

```
# R에 내장되어 있는 데이터 프레임을 불러들여 보겠습니다
mtcars
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

데이터 프레임을 분석하는 데에는 여러 가지 이용가능한 함수들이 있습니다. 대표적인 것은 데이터셋의 상위 행 일부로 자료를 간략하게 보여주는 `head()`, 반대로 아래의 행들을 보여주는 `tail()`, 데이터셋이 몇개의 관측치(obs.)와 변수들로 이루어져 있는지 그 구조(structure)를 보여주는 `str()`, 요약통계치들을 제시하는 `summary()` 등이 대표적입니다.

만약 어떤 함수였는지, 어떤 자료 유형이었는지 헷갈린다면 **R** 콘솔 창에다가 물음표 뒤에 함수 이름을 쳐보면 자세한 정보를 확인할 수 있습니다.

그러나 데이터 프레임보다 자료를 불러올 때, `tidyverse` 패키지에 속한 티블 유형으로 불러올 것을 추천합니다.

티블 (Tibbles)

티블은 tidyverse 패키지의 속한 함수로 티블로 저장한 자료의 유형은 데이터프레임과는 약간 차이가 있습니다. 우선 티블의 장점은 더 유저 친화적이라는 것입니다. 예를 들어, 티블은 R 콘솔창에서 한 눈에 확인할 수 있을 정도로 데이터의 구조를 출력해주고, 각 열의 변수들이 가지는 자료 유형이 어떤 것인지를 보여줍니다. 종종 데이터프레임으로 구성된 자료 유형을 티블로 강제 변환해야 할 경우가 있는데, 이때는 as_tibble() 함수를 사용하면 된다.

일단 tidyverse 패키지는 R에 내장된 것이 아니라 Hadley Wickham이 개발한 것이기 때문에 별도로 불러와야 합니다. 패키지를 설치할 때는 install.packages(), 설치된 패키지를 불러올 때는 library() 혹은 require() 함수를 사용합니다.

```
## install.packages("tidyverse") # 저는 이미 설치되어 있는 상태라 코멘트 처리합니다.
library(tidyverse)
```

```
## -- Attaching packages ------ tidyverse 1.2.1 --

## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts --------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
mtcars <- as_tibble(mtcars)
mtcars
```

```
## # A tibble: 32 x 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
##  2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
##  3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
##  4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
##  5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
##  6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
##  7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
##  8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
##  9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
## 10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
## # ... with 22 more rows
```

패키지 (Packages)

위의 `tidyverse` 패키지에 대한 설명이 사실 여기 들어와야 하는데, 티블을 설명하느라 조금 당겨서 적었습니다. 이 섹션에서 패키지에 대한 내용을 조금 더 자세하게 들여다 보도록 하겠습니다.

R의 패키지는 함수와 객체의 모음(collection of functions and objects)이라고 할 수 있습니다. RStudio나 R을 열때마다, 여러 개의 패키지들이 자동으로 로드(load)됩니다. 어떤 패키지들이 로드되어 있는지 확인하고 싶으면 `sessionInfo()` 라는 함수를 사용하면 됩니다.

좀 더 복잡한 문제를 해결하기 위해서는 R에 기본적으로 탑재된 함수/패키지 이외에 추가적인 패키지를 필요로 할 때가 있습니다. 이 경우에는 다음과 같은 함수를 사용합니다.: `install.package()`

그리고 R에서는 패키지의 설치와 사용은 별개의 작업으로 설치된 패키지를 사용하기 위해서는 그 패키지를 로딩해야 하는데 이때는 `library()` 함수를 사용하시면 됩니다. 한 번 패키지를 설치하면 다시 설치할 필요는 없지만 새로운 R 세션을 시작할 때마다 매번 `library()` 함수를 이용해서 로딩을 해주어야 합니다.

```
## R 패키지 설치 예시
## foreign 함수는 version 12 이하의 STATA 파일(.dta)을 로딩할 수 있게 도와줍니다.
## install.packages("foreign")
## plyr 함수는 좀 더 복잡하고 고급스러운 자료 조작(manipulation)을 가능하게 합니다.
## install.packages("plyr")
## ggplot2 함수는 함수 가시화(visualization)를 돕습니다.
## install.packages("ggplot2")

# 설치한 패키지들을 사용하기 위하여 라이브러리(libraries)를 로드한다.
library(foreign)
library(plyr)
library(ggplot2)
```

이렇게 로드된 패키지들의 상태와 작업창을 한 번에 저장하고 다음에 불러오고 싶을 때, 다음과 같은 패키지를 사용해 패키지들을 관리할 수도 있습니다.

```
## install.packages("session")
library(session)
save.session(file="test.Rda") # 현재까지 불러온 패키지와 객체들이 R 스크립트가 저장된
                              # 디렉토리에 test.Rda라는 이름으로 저장됩니다.
## 나중에 Rstudio 종료 후 다시 켰을 때,
restore.session(file="test.Rda") # 기존에 저장되었던 test.Rda를 불러옵니다.
## 이때, 주의해야할 점은 R 스크립트가 저장된 디렉토리가 세션 정보를 담은 Rda가 저장된
## 디렉토리와 같아야 한다는 점입니다. 만약 다르다면 file="다른 디렉토리/file.Rda"로
## 별도로 지정해주어야 합니다.
```

패키지들에 대한 더 자세한 내용을 알고 싶다면 다음의 링크를 참고하면 좋습니다.

그리고 R이 여러 패키지를 설치하는 데 제약이 없다고는 하지만 로드는 개별 패키지별로 해야합니다. 마지막으로 한 번에 여러 개의 패키지들을 설치 및 로드할 수 있게 도와주는 패키지(패키지의

패키지…)가 있는데, 그건 여기에서 살펴볼 수 있습니다.[3]

디렉토리 생성 코드

R 스크립트의 작성을 시작하기에 앞서, 디렉토리 생성 코드를 살펴보는 이유는 코딩하는 데 있어서 깔끔한 파일 구조를 설정하는 법을 숙지해야 효율적인 작업이 가능하기 때문입니다. 논문을 쓰는 입장이기 때문에 제 경우는 다음과 같이 폴더 구조를 정리합니다.

- `Main project` directory ← 예를 들어, [2018_FALL_Regime_Growth]
  - 폴더명 `code` subdirectory ← R 스크립트를 여기다 저장합니다.
  - 폴더명 `tables` subdirectory ← R에서 만든 표를 저장합니다.
  - 폴더명 `figures` subdirectory ← R에서 만든 그래프 등을 저장합니다.
  - 폴더명 `tex file` subdirectory ← 논문 본문을 작성하는 `tex` 파일을 저장합니다.[4]

하나 하나 윈도우 폴더 탐색기에서 만들 수도 있는데, R을 가지고도 편하게 만들 수 있습니다. 사실 익숙해져야 편하고 익숙해지기 전에는 약간 노가다 느낌나서 뭐하러 이짓하나 싶기도 합니다. 근데 익숙해지면 구조화된 폴더 트리 속에서 규칙적으로 네이밍되는 각 가지들로 이름만 바꾸면 되기 때문에 굉장히 편하다는 것을 알게 되실 겁니다.

- 예를 들어서, 원래 하던 프로젝트가 `Main`이라는 폴더의 `P1`이라면 그 다음은 `P2`니까 `P1`의 R 스크립트에서 디렉토리를 `P2`로 주소를 바꾸기만 하면 됩니다.

```
## 현재 R 콘솔에 저장된 모든 값, 모델 등을 제거하는 코드
rm(list=ls())

## 현재 작업중인 디렉토리가 어딘지 확인하는 코드
getwd()

## 새롭게 작업 디렉토리를 설정하는 코드
## 작업하고자 하는 폴더 우클릭 후 경로보기 하면 나옴
setwd("/Users/Documents")

## 표와 그래프를 위한 폴더를 만들기
dir.create("./tables")
dir.create("./figures")
```

용례 (A working example)

아래는 실제로 코딩을 이용해서 자료를 요약하거나 가시화하는 사례들입니다. 앞으로는 다음과 같은 내용들을 차근차근 다루어볼 것입니다.

[3] Github는 알아두면 굉장히 유용한데, 이 내용은 나중에 차차 업로드하도록 하겠습니다.

[4] `.tex`의 확장자를 갖는 `LaTex` 혹은 `knitr` 패키지와 같은 맥락으로 문서작업에 유용한 `Rmarkdown` 등에 관한 정보는 나중에 따로 업로드하도록 하겠습니다.

```
## 먼저 깔끔하게 R-콘솔 창을 정리합니다.
rm(list = ls())

## diamonds라는 데이터셋을 로드합니다. 이 데이터셋을 불러오려면 먼저 ggplot2를 설치하고
## 로드해야 합니다. ggplot2라는 패키지에 포함된 예제 데이터셋이기 때문입니다.
## install.packages("ggplot2") # 저는 이미 설치되어 있습니다.
library(ggplot2)
data(diamonds)
names(diamonds) # 데이터셋에 포함된 변수들의 이름을 확인할 수 있습니다.
```

```
##  [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
##  [8] "x"       "y"       "z"
```

```
head(diamonds) # 맨 위 몇 개 행의 특성을 간략하게 보여줍니다.
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
str(diamonds) # 데이터셋의 구조 (관측치의 수, 변수의 수, 자료유형 등)를 보여줍니다.
```

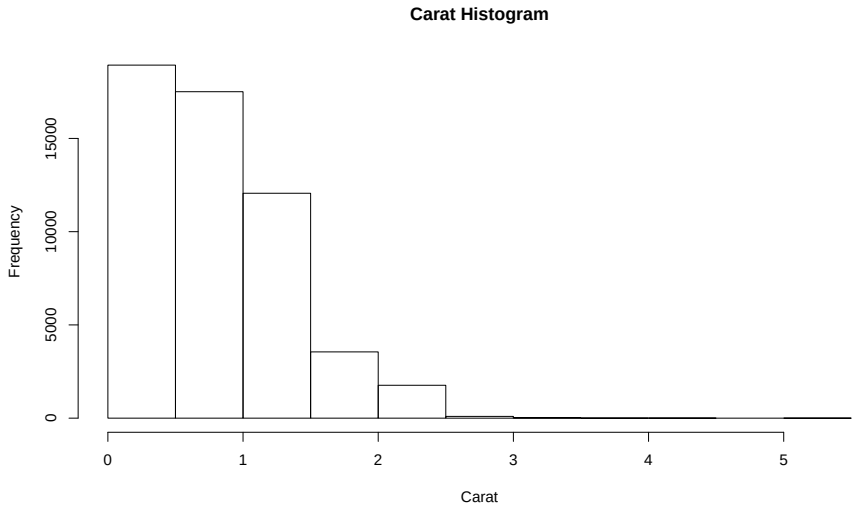
```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  10 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
summary(diamonds) # 데이터셋의 요약통계치 (평균, 중간값, 분위수 등)를 보여줍니다.
```

```
##      carat               cut        color        clarity
##  Min.   :0.2000   Fair      : 1610   D: 6775   SI1    :13065
##  1st Qu.:0.4000   Good      : 4906   E: 9797   VS2    :12258
##  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194
##  Mean   :0.7979   Premium   :13791   G:11292   VS1    : 8171
##  3rd Qu.:1.0400   Ideal     :21551   H: 8304   VVS2   : 5066
##  Max.   :5.0100                     I: 5422   VVS1   : 3655
```

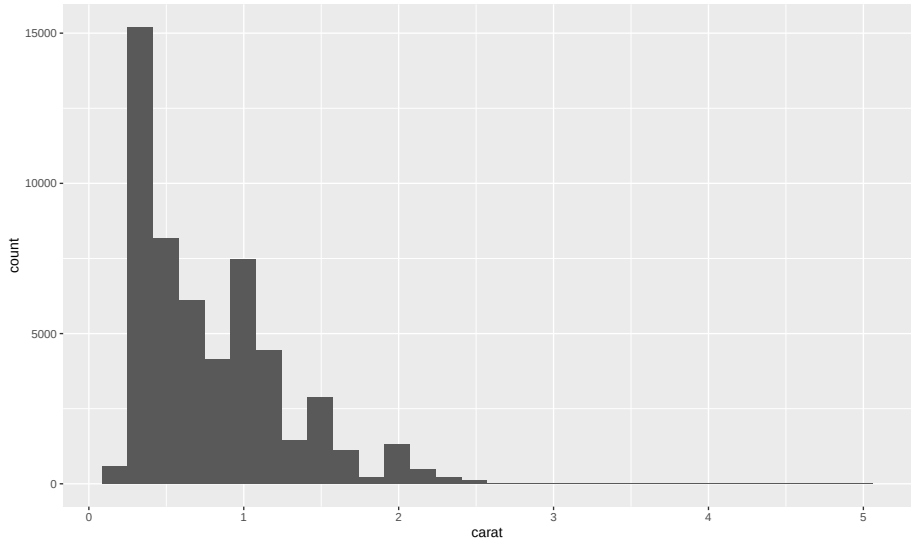
```
##                                   J: 2808   (Other): 2531
##      depth           table           price             x
##  Min.   :43.00   Min.   :43.00   Min.   :  326   Min.   : 0.000
##  1st Qu.:61.00   1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710
##  Median :61.80   Median :57.00   Median : 2401   Median : 5.700
##  Mean   :61.75   Mean   :57.46   Mean   : 3933   Mean   : 5.731
##  3rd Qu.:62.50   3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540
##  Max.   :79.00   Max.   :95.00   Max.   :18823   Max.   :10.740
##
##        y               z
##  Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 4.720   1st Qu.: 2.910
##  Median : 5.710   Median : 3.530
##  Mean   : 5.735   Mean   : 3.539
##  3rd Qu.: 6.540   3rd Qu.: 4.040
##  Max.   :58.900   Max.   :31.800
##
```

```
## 라벨을 포함한 R 히스토그램
hist(diamonds$carat, main = "Carat Histogram", xlab = "Carat")
```



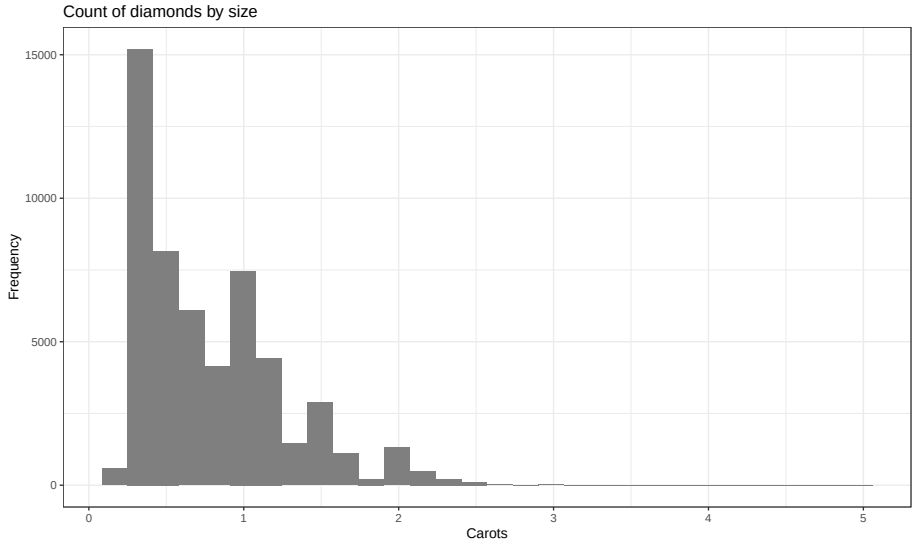

```
## R애 내장된 기본 함수가 아니라 ggplot2를 이용해서 똑같은 히스토그램 만들어 보겠습니다.
ggplot(data = diamonds) + geom_histogram(aes(x = carat))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
## ggplot2는 "+"를 이용해서 다양한 형태의 추가적인 정보를 레이어 형식으로 더할 수 있습니다.
ggplot(data = diamonds) +
  geom_histogram(aes(x = carat), fill = "grey50") + # 히스토그램 막대색 변경
  ylab("Frequency") + xlab("Carots") +
  ggtitle("Count of diamonds by size") +
  theme_bw() # 그래프 배경색 변경
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
## 아까 만들었던 그래프 폴더에 그래프를 저장할 수 있습니다.
# ggsave(file="./figures/figure1.pdf", width=6.5, height=5)
# ggsave(file="./figures/figure1a.png", width=6.5, height=5, device = "png")
```

```
## 표 폴더 만든 것에다가 요약통계표를 저장하기
library(stargazer) # 통계표를 작성하는 데 특화된 패키지입니다.

## 세 가지 변수에 대해 요약통계치를 확인하기
diamonds <- subset(diamonds, select = c("carat", "depth", "price"))
## 몇몇 R 예제 데이터들은 티블로 저장되어 있지 않을 수 있습니다.
## 이 경우에는 자료를 먼저 티블 유형으로 바꿔주고 시작하는 게 좋습니다.
## 자료 유형을 확인하는 함수는 class(), 혹은 typeof()입니다.
## library(tidyverse) # 아까 불러왔지만, 여기서는 로드하지 않았다고 가정합시다.
class(diamonds)
diamonds <- as_tibble(diamonds)
sum.table1 <- stargazer(diamonds,
                        covariate.labels=c("Size (carats)",
                                           "Cut", "Color",
                                           "Clarity"),
                                           title = "Summary stats for diamond data",
                                           label = "table:summary1")
# write(x=sum.table1, file="./tables/Summary1.tex") # LaTex로 열고 편집할 수 있습니다.
```

기타 참고자료

여기서 정리한 Introduction to R의 내용은 모두 DataCamp와 R for Everyone이라는 자료의 내용을 요약, 정리한 것입니다. 책(R for Everyone; R4E1)이야 구매할 수밖에 없지만 DataCamp 사이트의 강의들 중에는 무료강의가 많으니까 한 번쯤 확인해보는 것도 큰 도움이 될 거라고 생각합니다.

RStudio 홈페이지도 두 개의 기초강의를 제공하는데, RStudio Cloud에 가입하여 Primers를 클릭하면 됩니다. Primers에서 Basics를 선택하면 두 개 강의를 볼 수 있는데, R-coding에 도움이 되는 건 Programming Basics Course입니다.

Chapter 3

Introduction to Data

여기에서는 자료를 관리하고 다루는 방법들에 대해 간단한 소개를 하고자 합니다. 구체적으로 다룰 내용들은 다음과 같습니다.

- 무작위 추출 (Random draws)
- 루프 (Loops)
- 히스토그램 (Histograms)
- 표 (Tables)
- 벡터에서 요소들을 만들고, 계산하고, 추출하는 법 (Creating, summing, pulling elements from vectors)
- 분포와 확률에 따라 사고하는 법

먼저, 시작하기에 앞서서 작업 디렉토리 (Working Directory)를 설정할 필요가 있습니다.

작업 디렉토리 설정

기본적으로 R은 설치할 때, 기본 설정된 폴더에 이어서 작업을 진행합니다. 이 경우 기존에 만들어놓은 `R.data` (R 파일 확장자)들을 불러들여서 Global Environment에 사용하지 않을 데이터들이 쌓이게 됩니다. 그러면 현재 작업하여 저장한 새로운 객체들과 기존 디렉토리에 상존하는 데이터들이 혼재되어 헷갈릴 수 있습니다.

그리고 어차피 연구를 진행하면 해당 프로젝트에 따르는 폴더들을 만들어서 별개로 진행해야할 필요가 있기 때문에, 애초에 연구에 필요한 R.data를 만들 때 그 폴더에 만들고 작업 디렉토리도 설정하는 게 마음 편합니다.

그럼 일단 작업 디렉토리를 확인하고, 변경하는 코드를 살펴보겠습니다.

```
getwd() #현재 R이 인식하고 있는 작업 디렉토리가 어딘지 알려준다.
setwd("C:/Users/phere") #원하는 장소로 디렉토리를 변경한다.
```

기서 주의해야 할 것은, 디렉토리 주소를 적을 때에는 반드시 "" 기호를 사용해야 한다는 것입니다.

그리고 아마 윈도우 PC는 /로 디렉토리를 구분하는 반면에 MAC OS의 경우는 \(back slash)를 사용하는 것으로 알고 있습니다.

나중에 한 번 다루긴 하겠지만 몇 가지 함수들의 경우에는 MAC에서는 오류가 안 생기는데 윈도우에서는 오류가 발생하는 것들이 있습니다. 그런 경우가 조금 번거로울 때가 있기는 한데, 그 이외에는 뭐 다 같은 컴퓨터에 같은 **R**이니 큰 불편함은 없다고 할 수 있습니다 (하지만 MAC 구매 뽐뿌가 오는 것은 사실입니다).

```
dir.create(path= "figures")
dir.create(path= "tables")
dir.create(path= "datasets")
dir.create(path= "references")
dir.create(path= "tex")
```

저 같은 경우에는 어떤 연구 프로젝트를 시작할 때, 제일 먼저 R-script 빈 것을 만들어두고 작업 디렉토리를 설정합니다.

- 이후에 위의 코드와 같이 세부 폴더들을 만듭니다.
- **R**로 만든 그래프들을 저장할 `figures`, 표를 저장할 `tables`, 기타 다운받은 데이터들을 저장할 `datasets`, 그리고 참고문헌을 저장할 `references`와 본문 작성을 위한 `tex` 폴더로 구성됩니다.

무작위 추출(Random Draws)

무작위 추출을 실제로 해보기 위해서 한 가지 시뮬레이션을 돌려 보겠습니다. 어떤 사업장에서 사람을 승진시키는 데 있어서 성차별(gender discrimination)이 있을 수 있다고 가정하는 것입니다. 간단하게 100명의 승진 후보자들이 있다고 가정하고, 남녀가 동일한 비율로 나뉘어져 있다고 생각해 보겠습니다. 그리고 일반적으로 승진할 확률은 70% (0.7)라고 가정합니다. R-code로 남녀 각각 50명을 대상으로, 승진할 경우 1로 코딩하고 승진하지 못할 경우를 0으로 코딩하는 일종의 더미변수를 만들겠습니다. 그리고 승진 확률은 70%로 설정합니다.

- 즉, 무작위로 추출한 50명 중 승진확률이 그대로 반영된다면 우리는 35개의 1과 15개의 0을 확인할 수 있을 것입니다.
- 그러나 표본추출과 무작위화의 본연의 속성 상, 항상 확률대로 정확하게 그러한 비율의 결과를 갖는 것은 불가능합니다.
- 따라서 우리는 무작위로 추출할 때마다 약 70% 승진확률에 근거한 그 언저리의 값들을 얻게 될 것입니다.

```
MP <- rbinom(50, 1, .7) #70%의 승진확률로 무작위로 추출한 50명의 남성
WP <- rbinom(50, 1, .7) #70%의 승진확률로 무작위로 추출한 50명의 여성
sum(MP) #총 승진한 남성의 수
```

```
## [1] 41
```

```
sum(WP) #총 승진한 여성의 수
```

```
## [1] 35
```

```
sum(MP)/50 #전체 남성 승진후보자에 대한 승진한 남성의 비율
```

```
## [1] 0.82
```

```
sum(WP)/50 #전체 여성 승진후보자에 대한 승진한 여성의 비율
```

```
## [1] 0.7
```

MP와 WP는 각각 Mem Promotion과 Women Promotion의 약자입니다. `rbinom()`은 randomly draw as binomial로 이해하면 됩니다. 따라서 이 함수의 뜻은 "50개의 이항변수를 만드는데 1의 값을 가질 확률을 0.7로 해서 추출해라"고 할 수 있겠습니다.[1]

이렇게 만든 MP와 WP에 1, 즉 승진자가 각각 몇명인지 살펴보려면 단순하게 `sum()`, 합계를 나타내는 함수를 이용하면 됩니다. MP와 WP는 각각 벡터 자료로 값을 가지는데, `sum()`을 이용하면 이 벡터 자료의 각 요소들(elements)의 합을 계산할 수 있습니다. 만약 실제로 관측된 50명 각각에 대한 승진확률을 구하려면 그 집단의 총 인원수로 나누면 됩니다.

우리가 알고 싶은 것은 과연 이 사업장에서 승진하는 데 남녀의 성차별이 있느냐는 것입니다.

- 다시 말하면 남자와 여자의 승진 결과에 있어서 어떤 차이가 나타날 수는 있는데, 과연 그 차이가 진짜로 차별이 있어서 나타나는지를 확인하자는 것입니다.
- 이를 위해서 우리는 주어진 표본(남녀 각 50명, 총 100명에 있어서의)에서의 승진자 수의 차이를 계산해야 합니다.

```
DP <- sum(MP) - sum(WP) #DP는 Difference in Promotion, 승진자 수의 차이입니다.
```

DP의 값이 양수(positive)라면 남성 승진자의 수가 더 많다는 의미일 것이고, 음수(negative)라면 여성 승진자의 수가 더 많다고 볼 수 있습니다. 만약 DP가 0이면 두 성별에서의 승진자의 수가 동일하다는 것입니다다. 우리는 R을 이용하여 이와 같은 표집(sampling)을 여러 번 시뮬레이션 해볼 수 있습니다.

앞선 100명(남50 여50)을 추출한 것을 말 그대로 여러 번 반복할 수 있다는 것인데, 50명에 대한 확률은 0.7로 동일하더라도 무작위 추출이기 때문에 매번 추출할 때마다 그 결과는 달라질 것입니다.

- 우리가 알고 싶은 것은 이렇게 여러 번 추출해서 돌리더라도 만약 성차별이 실제로 존재한다면 꾸준하게, 그리고 일정한 차이로 DP가 나타날 것이라는 점입니다.
- 만약 DP가 있다가 없다가 한다면 통계적 관점으로 "평균적으로" (on average) 그 효과는 체계적이지 않을(non-systematic) 가능성이 높습니다.

먼저, 10개의 결측치를 가진 벡터를 만들어보겠습니다. 이게 무슨 말이냐면 아무 값도 들어있지 않은 10개 열자리 1개 행의 표를 만들라는 얘기로 이해할 수 있습니다. 완전 같은 말은 아닌데, 뭐 이렇게 이해하면 편할 거 같습니다. 그리고 그 각각의 칸에 이제부터 총 10번 시뮬레이팅하여 남50 여50에 대한 승진자 수의 차이 값 10개를 채워넣겠습니다.

```
trial.size <- 10 #시뮬레이션을 시도할 횟수
Test10 <- rep(NA, trial.size) ##rep는 replicate, 즉 반복하라는 함수입니다.
                              ##즉, 이 함수는 trial.size의 수만큼 NA를 반복해서
```

[1] 자세한 내용은 Kachitvichyanukul, V. and Schmeiser, B. W. (1988) "Binomial random variate generation." Communications of the ACM, 31, 216-222.를 참조. `?rbinom`이라고 R-console에 입력하면 우측 하단의 창에서 함수들에 대한 자세한 설명을 볼 수 있습니다.

```
                                    ##Test10이라는 벡터에 담으라는 의미입니다.
Test10 #위의 함수를 통해 얻게 되는 Test10 벡터의 값은 아래와 같습니다.
```

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

그리고 나서 우리는 이 10칸의 NA, 결측치(missing values)에 10번에 시뮬레이션을 통해 얻은 승진자 수의 차이(총 10개의 차이)를 담을 것입니다. 귀찮게 함수를 10번 반복할 수도 있겠지만, 만약 시뮬레이션을 해야 하는 수가 10번이 아니라 100만 번이라면 그 짓하다가 하루? 일주일이 다 갈 것이기 때문에 여기서부터 루프 (Loops)를 알아봅시다.

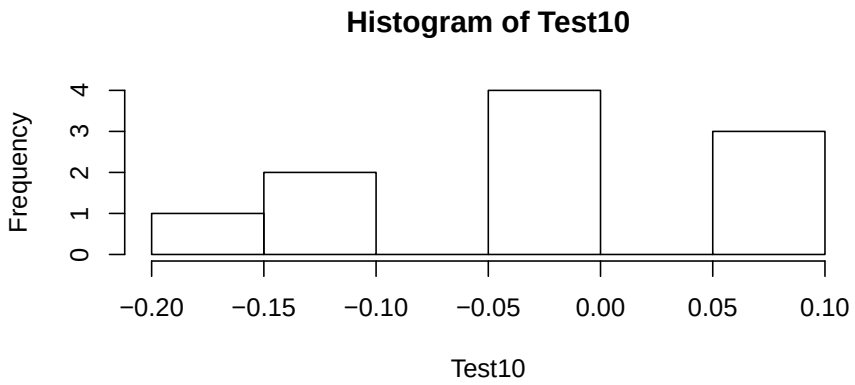
루프-반복(Loops)

자, 10번 반복하기를 해보겠습니다. 함수는 아래와 같습니다.

```
for (a in 1:trial.size) { # a라는 벡터에 1부터 trial.size까지의 수를 넣으라는 명령
  MP <- rbinom(50, 1, .7) # MP를 계산하라, 총 50명이 0.7의 확률로 1을 가질 것
  WP <- rbinom(50, 1, .7) # WP를 계산하라, 총 50명이 0.7의 확률로 1을 가질 것
  Test10[a] <- (sum(MP) - sum(WP))/50
}
```

위의 함수를 통해서 우리는 총 10개의 DP값을 50으로 나눈, 승진자 수의 차이가 한 개 집단에(남 or 녀) 대해 비율로 계산된 벡터의 형식으로 Test10에 가지게 됩니다. Test10[1]은 첫 번째 시뮬레이션의 DP 확률이고 Test10[4]는 네 번째 시뮬레이션의 DP 확률을 가지게 될 것입니다. 벡터 자료 뒤에 [n]는 그 벡터의 n번째 요소를 보여달라는 명령입니다.

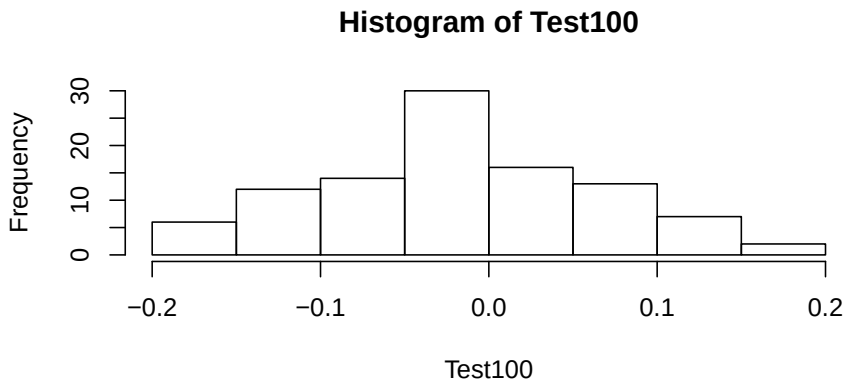
```
hist(Test10)
```



총 10개의 Test10의 값을 구해 히스토그램을 구한 것입니다. 즉, 10개 DP 확률의 분포를 나타낸 것이라고 할 수 있습니다. 10개의 표본으로는 뚜렷한 경향을 보기가 힘듭니다. 한 번 백

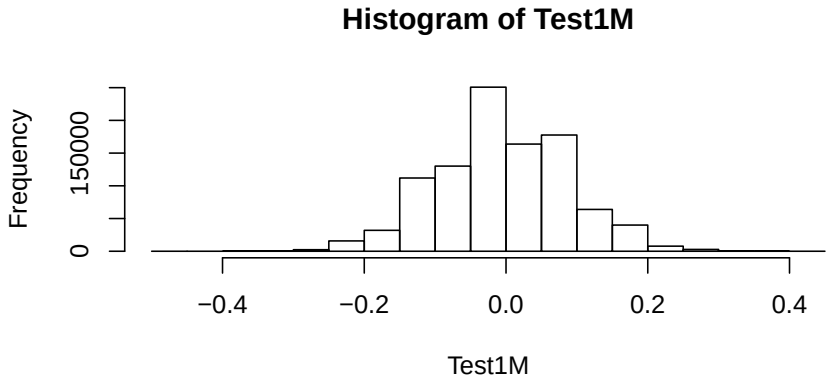
개의 시뮬레이션을 진행해보겠습니다. 진행과정은 10번의 시뮬레이션이랑 동일합니다. 단지 trial.size가 100으로 늘어났다는 것과 반복횟수가 b로 a랑 구분한다는 것만 다릅니다.

```
trial.size <- 100
Test100 <- rep(NA, trial.size)
for (b in 1:trial.size) {
  MP <- rbinom(50, 1, .7)
  WP <- rbinom(50, 1, .7)
  Test100[b] <- (sum(MP) - sum(WP))/50
}
hist(Test100)
```



매우 눈에 익숙하고 우리가 사랑하는(?) 분포의 형태로 변해가는 것을 볼 수 있습니다. 이제 극단적으로 백 만번의 시뮬레이션을 진행해보겠습니다.

```
trial.size3 <- 1000000
Test1M <- rep(NA, trial.size3)
for (c in 1:trial.size3) {
  MP <- rbinom(50, 1, .7)
  WP <- rbinom(50, 1, .7)
  Test1M[c] <- (sum(MP) - sum(WP))/50
}
hist(Test1M)
```



여기서 한 가지 알 수 있는 것은 시뮬레이션 시도 횟수가 늘어날수록(무한에 가까워질수록), 우리가 알고 싶어하는 승진에서의 차이(결과)의 진짜 확률(true probabilities)이 드러난다는 것입니다.

- 이는 나중에 표집(sampling)과 표집오차(sampling errors), 그리고 중심극한정리(Central Limit Theorem)을 언급할 일이 있을 때 다시 살펴보도록 하겠습니다.

위에서 추출한 100만번의 시뮬레이션 결과를 토대로 남성과 여성의 승진에 있어서의 차이가 과연 0.3보다 큰지 살펴보겠습니다.

- 조금 더 문제를 단순화하기 위해서 남자가 여성보다 더 승진할 가능성, 승진자 수의 차이가 전체 후보자에 대한 비율에 있어서 +0.3보다 큰지 작은지를 살펴보려는 것입니다.
- 먼저 각각(남성과 여성)에 있어서의 승진자 수의 차이를 비율로 구하여 그 비율이 0.3보다 크거나 같은 경우가 전체 시뮬레이션으로 나타난 승진자 수의 차이 비율 전체의 몇 퍼센트를 차지하는지 구합니다.

```
length(Test1M[Test1M >= 0.3]) / length(Test1M)
```

```
## [1] 0.000765
```

- 처음에 R을 할 때 잘 외우지 못했던 함수 중 하나인데, `length()`는 뭐랄까… `count`라고 이해하면 조금 더 쉬울 것 같습니다.[2]
- 즉, 아래의 함수는 `Test1M`, 100만개의 승진자 수가 전체 집단에서 차지하는 비율이 0.3보다 큰 경우만을 구해서 그걸 전체 승진차이 확률 100만개에 대한 비율로 구하라는 것입니다.
    - 남자가 여자에 비해서 30% 이상 승진을 많이 할 확률이 나타난 것이 총 100만번의 시뮬레이션 가운데 몇 번이었냐는 것입니다.
- 아래 함수는 그 결과는 0.000711, 즉 약 0.07%의 확률로 남자가 여자에 비해 30% 이상 승진자가 많을 확률이 나타난다는 것을 알 수 있습니다.
- 적어도 이 예제에 한하여 해석은 개인의 몫으로 남겨 놓겠습니다.

[2] 나중에 가면 알아볼 `dplyr` 패키지에는 그룹별 관측치의 개수를 셀 수 있는 `count()` 함수가 존재합니다.

Rplots를 파일의 형태로 저장하기

이번 섹션에는 조금 기술적인(technical) 소개를 하고자 합니다. 바로 위에서 만든 히스토그램 등을 pdf나 png와 같은 형태의 파일로 저장하는 함수입니다. 먼저 pdf로 저장하는 것을 살펴보겠습니다. pdf로 저장하는 것은 추천할만한 방식입니다. LaTeX를 이용하는 경우에는 이렇게 저장한 pdf를 깔끔하게 LaTeX로 생산하는 pdf 문서에 삽입할 수 있습니다.

```
pdf("figures/histogram.pdf", width=8, height=6)
## 아까 만든 figures에 histogram.pdf라는 이름으로 저장하게 합니다.
hist(Test1M, xlab = "Margin", main = NULL)  #표 전체 제목은 NULL, 없습니다
##Test1M, 100만번 시뮬레이팅한 히스토그램 X축에는 Margin이라고 레이블을 달 것입니다.
dev.off()
```

dev.off는 device off로 위의 함수는 여기서 끝!이라고 이해하시면 되겠습니다다.자세한 내용은 ?dev.off로 알아보시길… 그리고 pdf로 저장하기 어려운 경우에는 png로 저장할 경우 해상도 자체는 조금 떨어지지만 유용하게 쓸 수 있는 그림파일로 동일한 Rplots를 저장할 수 있습니다. pdf는 인치(Inches)로 저장되는데, png는 픽셀(Pixels)로 저장됩니다. 그래서 아래 코드에서의 너비랑 높이를 지정하는 방법이 조금 다릅니다.

```
png("figures/histogram.png", width=720, height=480)
hist(Test1M, xlab = "Margin", main = NULL)
dev.off()
```

데이터 다루기 기본

이번 섹션부터는 데이터에 대해 조금 더 깊이 살펴보고자 합니다. 구체적으로는,

- 데이터 불러오기 (loading)
- 서로 다른 분석수준, 분석단위를 가지고 작업하기 (working with different levels of analysis/units of observation)
- 작업 흐름(workflow)

순으로 진행하고자 합니다.

먼저 기존에 존재하던 데이터셋을 불러오기에 앞서 개략적으로 데이터라는 것이 어떻게 생겼는지를 살펴볼 필요가 있습니다. 데이터프레임은 열에 변수(variables), 행에 관측치들을 갖는 형태로 이루어져 있습니다. 하지만 앞서 언급했던 바와 같이, 데이터프레임 유형보다는 티블을 사용하는 것이 앞으로 배울 함수들을 적용하기에 좀 더 효율적입니다. 따라서 티블 함수를 이용해 가상의 데이터셋을 직접 만들어 보겠습니다.

```
library(tidyverse) # 티블을 사용하기 위해서는 tidyverse 패키지를 불러와줘야 합니다.
data1 <- tibble(name = c("Jane", "John", "Jen", "James"),
                height = c(60, 70, 65, 68),
                eye.color = c("blue", "blue", "brown", "brown"),
                gender = c("female", "male", "female", "male"),
                highest.degree = c("college",
                                   "high school",
```

```
                                    "post graduate",
                                    "college"))
glimpse(data1)
```

```
## Observations: 4
## Variables: 5
## $ name           <chr> "Jane", "John", "Jen", "James"
## $ height         <dbl> 60, 70, 65, 68
## $ eye.color      <chr> "blue", "blue", "brown", "brown"
## $ gender         <chr> "female", "male", "female", "male"
## $ highest.degree <chr> "college", "high school", "post graduate", "col...
```

만약 티블이 아니라 데이터프레임으로 저장하고 싶으시다면 `tibble()` 대신 `data.frame()` 함수를 사용하시면 됩니다. 이렇게 생성된 데이터는 열(column)에 변수를 갖습니다.

- `data1`에서는 `name, height, eye.color, gender, highest.degree`가 변수명이 됩니다.
- 그리고 각 변수의 하위에 행마다 관측치들이 주어집니다다.
  - `name`이라는 변수에는 `Jane, John, Jen, James` 라는 각 개인을 관측한 결과가 입력됩니다다.
  - `c()`는 안의 요소들을 벡터의 형태로 묶으라는 것입니다(이전 포스팅에서 벡터에 대한 설명 참조).
- 그리고 이렇게 만들어진 데이터의 구조를 확인하기 위해서는 `glimpse()` 함수를 사용하면 된다.
  - `str()` 함수도 있는데, `glimpse()`가 좀 더 깔끔하게 데이터의 구조를 보여주는 것 같습니다.

데이터에 새로운 변수 & 더미변수 만들기

기존 데이터에 새로운 변수를 추가하는 방법은 매우 간단합니다. 그냥 새로운 변수명을 $ 표시를 이용해 데이터에 써주고 거기에 배정(assign)을 의미하는 <- 표시로 넣어주면 됩니다. 백문이 불여일견이니 한 번 해보겠습니다.

```
data1$female <- NA
```

위의 코드는 `female`이라는 변수를 새롭게 만들되 `female`의 모든 관측치는 결측치(missing values)로 생성하라는 것을 의미합니다.우측 항에 단순한 값을 적는다면 데이터의 새로운 변수, 일종의 벡터에는 모두 그 값으로 채워질 것입니다. 반면, 우리가 원하는 체계적인 형태의 변수(variable)로 대체하고 싶다면 함수(function)를 이용하면 됩니다.

여기서 살펴볼 더미변수란 말 그대로 더미(dummy), 바보 변수입니다. 더미변수는 존부(存否)만을 나타내는 변수인데 1일 경우에는 있음, 0일 경우에는 없음을 나타냅니다. 예를 들어, 바로 아래에서 만들 `female`변수는 여성일 경우에 1, 남성일 경우에 0을 나타낼 것입니다. 아까 만든 데이터(`data1`)의 변수 중 `gender`는 남성(`male`)과 여성(`female`)이라는 두 문자열 변수로 구성되어 있는데, 이를 숫자형(numerical)으로 바꿔주고자 하는 것입니다.

더미변수는 어떤 점에서는 유용하지만 존부 이외의 자세한 정보를 제공하지 못한다는 점에서 더미라고 불립니다. 일단, 더미변수(숫자형)으로서의 여성(사실상 성별) 변수를 추가해보겠습니다.

```
data1$female <- ifelse(data1$gender == "female", 1, 0)
## 해석: data1에 female이라는 변수에 우측 함수에 따른 값을 배정하라.
##       ifelse(만약 ~ 면, A를, ~가 아니라면, B를) 배정하라.
## 따라서 위의 함수는 data1의 gender 변수가 "female"이라는 문자일 경우 새로운 female
## 변수에 1을, "female"이 아닌 경우에는 0을 주어라.
data1$female # 더미 변수의 이름을 지을 때에는 기준값(reference value)이 헷갈리지 않게
```

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

```
              # 1의 값을 갖는 라벨(label)로 변수 이름을 짓는게 좋다 (TIP)
data1$gender <- NULL # 이제 사용하지 않을 gender 변수는 결측치로 변경.
```

그리고 한 가지 짚고 넘어갈 것은 R에서는 요인형(factor)과 문자형(character) 유형이 다르다는 것입니다. 요인형 자료를 문자형 자료로 변환하거나 혹은 그 역도 가능하지만 요인형에서 문자형으로 변환하는 것은 별로 추천드리고 싶지 않습니다. 일단 기본적으로 이 내용은 일반적인 통계분석을 다루고 있기 때문에 문자형 그 자체를 가지고 뭔가를 분석하는 텍스트 분석이 아닌 이상 문자열은 그저 고유값, 이상도 이하도 아니기 때문입니다. 하나 밖에 없는 값으로 무언가를 일반화하거나 설명하기란 쉽지 않습니다.

```
data1$name <- as.character(data1$name)
# name 변수의 자료들은 문자형(STATA에서 string이라고 하는)으로 이루어져 있는데,
# 이를 요인형으로 바꾸는 것이다.
glimpse(data1)
```

```
## Observations: 4
## Variables: 5
## $ name            <chr> "Jane", "John", "Jen", "James"
## $ height          <dbl> 60, 70, 65, 68
## $ eye.color       <chr> "blue", "blue", "brown", "brown"
## $ highest.degree <chr> "college", "high school", "post graduate", "col...
## $ female          <dbl> 1, 0, 1, 0
```

```
knitr::kable(summary(data1))
```

name	height	eye.color	highest.degree	female
Length:4	Min. :60.00	Length:4	Length:4	Min. :0.0
Class :character	1st Qu.:63.75	Class :character	Class :character	1st Qu.:0.0
Mode :character	Median :66.50	Mode :character	Mode :character	Median :0.5
NA	Mean :65.75	NA	NA	Mean :0.5
NA	3rd Qu.:68.50	NA	NA	3rd Qu.:1.0
NA	Max. :70.00	NA	NA	Max. :1.0

위에서 요인형을 문자형으로 바꾸는 함수는 바로 `as.factor()`입니다. 직관적인 함수인데, 괄호 안의 변수를 요인으로써(as factor) 취급하여 다시 저장하라는 의미라고 볼 수 있습니다.

요인을 문자형으로 바꿀 수 있는 것처럼 요인형 변수를 숫자형 변수로 바꿀 수도 있습니다. 요인형 → 숫자형 변환 과정은 두 단계로 이루어집니다. 일단 예제 데이터를 만들어보겠습니다.

```
GPA <- c("3.0", "4.0", "3.8", "2.2")
## 만약 "" 인용부호를 제외하고 벡터로 입력하면 GPA는 숫자형 자료가 될 것입니다.
## 기존의 data1 데이터프레임에다가 방금 만든 GPA를 새로운 열로 추가해보겠습니다.
data1 <- cbind(data1, GPA) #cbind는 열로 묶으라는 것입니다, 행으로 묶는 것은 rbind()
glimpse(data1)
```

```
## Observations: 4
## Variables: 6
## $ name            <chr> "Jane", "John", "Jen", "James"
## $ height          <dbl> 60, 70, 65, 68
## $ eye.color       <chr> "blue", "blue", "brown", "brown"
## $ highest.degree <chr> "college", "high school", "post graduate", "col...
## $ female          <dbl> 1, 0, 1, 0
## $ GPA             <fct> 3.0, 4.0, 3.8, 2.2
```

```
knitr::kable(summary(data1))
```

name	height	eye.color	highest.degree	female	GPA
Length:4	Min. :60.00	Length:4	Length:4	Min. :0.0	2.2:1
Class :character	1st Qu.:63.75	Class :character	Class :character	1st Qu.:0.0	3.0:1
Mode :character	Median :66.50	Mode :character	Mode :character	Median :0.5	3.8:1
NA	Mean :65.75	NA	NA	Mean :0.5	4.0:1
NA	3rd Qu.:68.50	NA	NA	3rd Qu.:1.0	NA
NA	Max. :70.00	NA	NA	Max. :1.0	NA

요인형 변수(GPA)를 만들고 기존 데이터에 추가했으니, 이제 이 변수를 숫자형으로 바꿔겠습니다. 앞서 언급했다시피 이 과정에는 두 가지 단계가 요구됩니다.

```
data1$GPA <- as.numeric(as.character(data1$GPA))
names(data1)[6] <- "GPA.num" # 기존 요인형 GPA랑 새롭게 만든 숫자형 GPA 비교를 위해
                             # .num(numeric 약자)을 붙여 새로운 변수로 만듭니다.
table(data1$GPA.num)
```

```
##
## 2.2   3 3.8   4
##   1   1   1   1
```

names(data1)[6]은 데이터프레임의 여섯 번째 열에 이름을 지어라(names)라는 코드입니다. 그 이름을 GPA.num으로 하기 위해 <- "GPA.num"이 지정되었습니다. 만약 요인변수를 직접적으로 숫자형으로 바꾸고자 시도할 경우에는 문제가 생길 수 있습니다. 아래의 코드를 보겠습니다.

```
data1 <- cbind(data1, GPA)
data1$GPA <- as.numeric(data1$GPA) # 이렇게 하면 문제가 생김
glimpse(data1$GPA)
```

```
##  num [1:4] 2 4 3 1
```

두 방법의 차이를 알시겠나요? GPA 변수의 소수점이 다 사라지고 정수형으로 바뀌어버렸습니다. 이것이 요인형을 숫자형으로 바꾸는 데 두 단계가 필요한 이유입니다.

다른 유형의 데이터 불러오기 (Loading data in different formats)

간략한 데이터셋을 직접 만들어보았으니, 이번에는 다른 연구자/기관이 구축한 데이터를 불러오는 방법을 살펴보겠습니다. 이 포스팅에서 사용할 데이터셋은 2016년도 기준으로 측정된 국가 단위의 자료이다. 이 자료의 원출처는 다음의 링크에서 확인할 수 있으며, 본 포스팅에서 사용할 자료는 미리 분석을 용이하게 하기 위하여 일정 변수들만을 선별한 자료입니다. STATA 파일로 저장된 자료를 사용합니다.

한 가지 말해두자면, R은 여러 가지 과정과 방법들로 동일한 결과를 얻을 수 있기 때문에, 자신에게 보다 효율적인 방법을 찾아가는 것이 중요합니다.

```
## STATA 파일을 불러오기 위해서는 "foreign" 패키지가 필요합니다.
## install.packages("foreign") # 저는 이미 설치가 되어 있습니다.
library(foreign) # 설치만 해서는 안되고 패키지를 불러와야 합니다.
## STATA 파일을 불러와 보겠습니다.
here::here() %>% setwd()
QOG <- read.dta(file = "example.data/qog_std_cs_jan19_ver13.dta",
                convert.underscore = TRUE)
## foreign 함수로는 STATA 버전 13 이전의 자료만 불러들일 수 있습니다. 즉, 아래 코드는 불가.
QOG <- read.dta(file = "example.data/qog_std_cs_jan19_ver15.dta",
                convert.underscore = TRUE)
## 그렇다면 버전 13 이후는 무슨 패키지를 사용해야 할까요?
## 버전 13 이후는 "readstata13" 로 불러올 수 있습니다.
# install.packages("readstata13")
library(readstata13)
QOG.v2 <- read.dta13(file = "example.data/qog_std_cs_jan19_ver15.dta",
                     convert.underscore = TRUE)

## 또 다른 방법이 있다. 바로 "haven" 패키지를 이용하는 것입니다.
## install.packages("haven")
library(haven)
QOG.v3 <- read_stata("example.data/qog_std_cs_jan19_ver15.dta")

## 근데 저는 foreign 이나 haven 패키지 모두 안 씁니다.
## 더 효율적인 패키지를 찾았거든요. 바로 ezpickr 입니다.
## install.packages("ezpickr")
library(ezpickr)
QOG.v4 <- pick("example.data/qog_std_cs_jan19_ver15.dta")
```

자료 머징하기 (Using Data Merging)

자료 머지(merge)에도 여러 가지 유형이 있는데, 오늘 간단하게 살펴볼 것은 기존 데이터을 다른 데이터의 변수들을 이용해 확장하는 유형의 머징입니다. 두 국가 간의 인구 차이를 측정하는 국가쌍(dyadic) 변수를 코드하고자 한다고 해보겠습니다. 먼저 QOG 데이터의 하위 셋(subset)을 만듭니다.

```
## names(QOG) # QOG 데이터프레임의 변수명을 나열하라는 함수입니다.
## 변수가 엄청 많습니다.
length(names(QOG)) # 1983개의 변수
```

```
## [1] 1983
```

```
QOG.tomerge <- subset(QOG, select = c(ccodecow, wdi.pop))
## QOG.tomerge라는 하위 셋을 만들라는 명령입니다.
## QOG라는 자료에서 ccodecow와 wdi.pop라는 두 변수만을 선택(select)하여 만듭니다.
QOG.tomerge <- subset(QOG, ccodecow > 100, select = c(ccodecow, wdi.pop))
## QOG.tomerge라는 하위 셋을 만들어라. 이 경우에는 앞의 QOG.tomerge를 대체(replace)합니다.
## QOG라는 자료에서 ccodecode가 100보다 큰 경우에 한하여(조건)
## ccodecow와 wdi.pop라는 변수를 선택하여 하위 셋을 만듭니다.
## 결과적으로 QOG.tomerge는 cowcode가 100보다 큰 국가들의 세계발전지표 상의 인구
## 지표들을 가지게 됩니다.
```

교차사례 데이터 QOG.tomerge를 중복하여 머지해보겠습니다. 이렇게 하면 우리는 총 국가1과 국가2, 그리고 연도에 따른 동일한 변수들을 갖는 결합된 데이터를 갖게 될 것입니다. 각각의 관측치들은 모든 관측치와 매칭(matching)이 되기 때문에 우리는 동일한 국가 쌍의 자료들을 갖게 됩니다. 말이 더 어렵군요… 백문이 불여일견.

```
QOG.dyad <- merge(x = QOG.tomerge, y = QOG.tomerge, by = NULL)
```

이 경우에 동일한 국가쌍(self-dyads)은 제거해야 두 국가 간의 관계를 살펴볼 수 있을 것입니다. 동일한 국가의 인구 지표는 차이가 없으니까요!

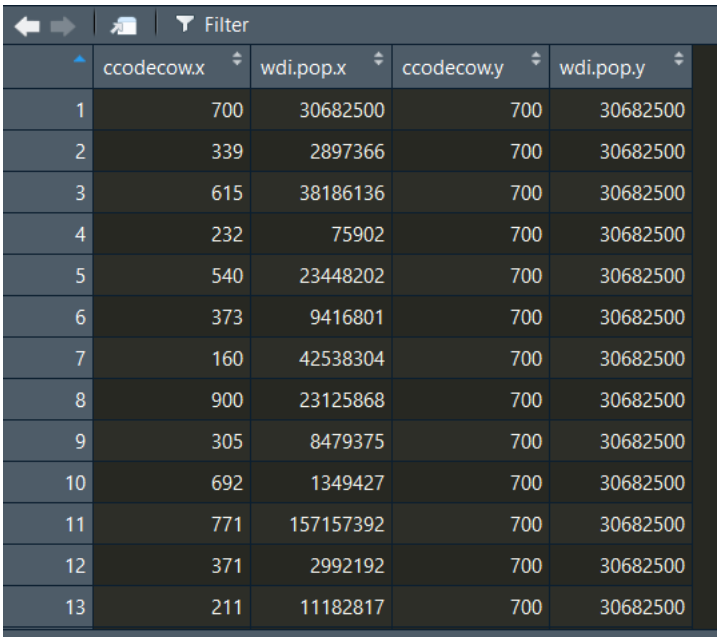
```
names(QOG.dyad)
```

```
## [1] "ccodecow.x" "wdi.pop.x"  "ccodecow.y" "wdi.pop.y"
```

```
QOG.dyad <- subset(QOG.dyad, ccodecow.x != ccodecow.y)
## QOG.dyad 자료에서 ccodecow.x가 ccodecow.y와 다른 경우만 다시 저장합니다.
QOG.dyad$pop.dif <- QOG.dyad$wdi.pop.x - QOG.dyad$wdi.pop.y
## QOG.dyad 데이터프레임에 pop.dif, 인구차이라는 변수를 새로 만듭니다.
## 인구 차이는 x국가의 wdi.pop.x에서 wdi.pop.y 를 감한 값입니다.
## 이 변수는 x국가와 y국가 간 인구 차이, 즉 두 국가 간의 역동적 관계를 보여줍니다.
```

위와 같이 wdi.dif라는 인구 차이 변수는 사실 x에서 y를 빼나 y에서 x를 빼나 부호를 제외하고 그 크기는 동일합니다. 따라서 우리는 따로 방향(direction)을 고려할 필요가 없습니다. 아래의 코드는 방향성을 제거한 국가쌍 자료(non-directed dyads)를 만드는 것입니다.

```
QOG.nddyad <- subset(QOG.dyad, ccodecow.x < ccodecow.y)
QOG.nddyad$pop.dif.nd <- abs(QOG.nddyad$wdi.pop.x - QOG.nddyad$wdi.pop.y)
```



	ccodecow.x	wdi.pop.x	ccodecow.y	wdi.pop.y
1	700	30682500	700	30682500
2	339	2897366	700	30682500
3	615	38186136	700	30682500
4	232	75902	700	30682500
5	540	23448202	700	30682500
6	373	9416801	700	30682500
7	160	42538304	700	30682500
8	900	23125868	700	30682500
9	305	8479375	700	30682500
10	692	1349427	700	30682500
11	771	157157392	700	30682500
12	371	2992192	700	30682500
13	211	11182817	700	30682500

Figure 3.1: 이렇게 머지하면, x 국가군의 cowcode와 인구지표, 그리고 y 국가군의 cowcode를 가지게 됩니다.

```
## abs는 absolute value, 부호를 고려하지 않기 위해서 절대값으로 만들라는 것입니다.
```

이번에는 국가-연도(country-year) 자료를 국가쌍-연도(dyad-year) 자료로 전환하는 사례를 살펴보겠습니다. 이를 위해서 일단 교차사례 시계열 데이터셋을 웹에서 다운받아 열어봅니다(time-series cross-sectional dataset). 이렇게 불러온 데이터셋은 1816년부터 2016년 사이의 국가들의 자료를 구축한 국가-연도를 분석단위로 한 자료입니다.

```
## COW 국가-연도 목록을 불러왔습니다.
stateyear <-
  read.csv("http://correlatesofwar.org/data-sets/state-system-membership/system2016",
                    head = TRUE, sep = ",")# Look at country codes
unique(stateyear$ccode) # 중복되지 않는 국가코드만 보이게 했습니다.
```

```
##   [1]   2 200 210 220 225 230 235 245 255 267 269 271 273 275 300 325 327
##  [18] 329 337 365 380 390 640 140 616 350 211  70 100 240 135 155 101 160
##  [35] 332 280 150 600 145 335 130 630 651  41 710 740  90  92 345 360 165
##  [52] 730 800  42 530  91  93  40  95 385 355 339 375 290 310 315 366 367
##  [69] 368 305 700  20  94 212 450 560 790 900 920 712 205 678 670 645 395
##  [86] 652 660 663 840 750 770 666 731 775 780 713 732 850 620 811 812 265
## [103] 816 817 260 625 452 820 438 352 432 433 434 435 436 437 439 461 471
## [120] 475 481 482 483 484 490 520 580 451 510 690  51  52 500 516 517 615
## [137] 501 511 338 551 553 420 552 781 830  53 110 570 571 680 411 572 590
```

```
## [154] 950 692 694 696 698 760 771  31  55 404 115 402 403 540 541 581 910
## [171] 591 990 522  54 940  56  57  58  80 935  60 835 223 565 679 359 369
## [188] 370 371 372 373 701 702 703 704 705 983 987 331 344 346 349 221 232
## [205] 316 317 343 531 986 946 955 970 947 860 341 347 626
```

```
table(stateyear$ccode)  # 각 국가코드가 몇 개의 관측치를 가지는지를 보여줍니다.
```

```
##
##   2  20  31  40  41  42  51  52  53  54  55  56  57  58  60  70  80  90
## 201  97  44 113 140 116  55  55  51  39  43  38  38  36  34 186  36 149
##  91  92  93  94  95 100 101 110 115 130 135 140 145 150 155 160 165 200
## 118 142 117  97 114 186 176  51  42 163 178 195 169 166 178 176 135 201
## 205 210 211 212 220 221 223 225 230 232 235 240 245 255 260 265 267 269
##  95 197 183  94 200  24  27 201 201  24 201  30  56 157  36  37  56  52
## 271 273 275 280 290 300 305 310 315 316 317 325 327 329 331 332 335 337
##  56  51  52  25  94 103  82  99  70  24  24 201  45  46  25  19  10  45
## 338 339 341 343 344 345 346 347 349 350 352 355 359 360 365 366 367 368
##  53  99  11  24  25 137  25   9  25 187  57 109  26 139 201  49  49  49
## 369 370 371 372 373 375 380 385 390 395 402 403 404 411 420 432 433 434
##  26  26  26  26  26 100 201 108 197  73  42  42  43  49  52  57  57  57
## 435 436 437 438 439 450 451 452 461 471 475 481 482 483 484 490 500 501
##  57  57  57  59  57  97  56  60  57  57  57  57  57  57  57  57  55  54
## 510 511 516 517 520 522 530 531 540 541 551 552 553 560 565 570 571 572
##  56   2  55  55  57  40 115  24  42  42  53  52  53  97  27  51  51  49
## 580 581 590 591 600 615 616 620 625 626 630 640 645 651 652 660 663 666
##  57  42  49  41 127  55 118  66  61   6 162 201  85 108  69  71  71  69
## 670 678 679 680 690 692 694 696 698 700 701 702 703 704 705 710 712 713
##  90  65  27  24  56  46  46  46  46  98  26  26  26  26  26 157  96  68
## 730 731 732 740 750 760 770 771 775 780 781 790 800 811 812 816 817 820
##  19  69  68 151  70  46  70  46  69  69  52  97 130  64  64  63  22  60
## 830 835 840 850 860 900 910 920 935 940 946 947 950 955 970 983 986 987
##  52  33  71  68  15  97  42  97  36  39  18  17  47  18  18  26  23  26
## 990
##  41
```

불러온 목록에서 숫자형으로 저장된 COW국가 코드만 따로 떼어서 볼 수 있고, 코드별로 연도별 자료가 얼마나 관측되어 있는지 확인할 수 있습니다. 이제 여기서 연도와 ccode 변수만 따로 떼어보겠습니다.

```
stateyear <- subset(stateyear, select = c(year, ccode))
```

그리고 새로운 패키지, countrycode()를 이용하여 국가 이름 변수를 새롭게 만들어 데이터에 추가해보겠습니다.

```
## install.packages("countrycode")
library(countrycode)
stateyear$countryname <- countrycode(stateyear$ccode, "cown", "country.name")
## stateyear 자료에서 ccode 변수를 숫자형("cown")에서 문자형("country.name")으로 변경합니다.
```

```
## stateyear 자료에 countryname 이라는 새로운 이름 변수를 만들어 바꾼 자료를 배정합니다.

## 이 자료에서 결측치(missing values)를 제거해보겠습니다.
## complete.cases는 결측치를 제외한 변수들 짝이 완전하게 맞는 사례들만을 선별하라는 옵션입니다.
stateyear <- subset(stateyear, complete.cases(stateyear))
head(stateyear)
```

```
##   year ccode    countryname
## 1 1816     2  United States
## 2 1816   200 United Kingdom
## 3 1816   210    Netherlands
## 4 1816   220         France
## 5 1816   225    Switzerland
## 6 1816   230          Spain
```

연도 변수를 이용해서 이번에는 다시 국가쌍 자료로 머징해보겠습니다.

```
dyadyear <- merge(x=stateyear, y=stateyear, by.x=c("year"), by.y=c("year"))
head(dyadyear)
```

```
##   year ccode.x countryname.x ccode.y  countryname.y
## 1 1816       2 United States       2  United States
## 2 1816       2 United States     200 United Kingdom
## 3 1816       2 United States     210    Netherlands
## 4 1816       2 United States     220         France
## 5 1816       2 United States     225    Switzerland
## 6 1816       2 United States     230          Spain
```

아까처럼 1행의 자기자신의 쌍을 구성하는 경우를 제거해보겠습니다. 이번에는 `subset()` 함수가 아니라 조건(condition, if)를 의미하는 대괄호를 이용해보겠습니다.

```
dyadyear <- dyadyear[dyadyear$countryname.x != dyadyear$countryname.y, ]
head(dyadyear)
```

```
##   year ccode.x countryname.x ccode.y  countryname.y
## 2 1816       2 United States     200 United Kingdom
## 3 1816       2 United States     210    Netherlands
## 4 1816       2 United States     220         France
## 5 1816       2 United States     225    Switzerland
## 6 1816       2 United States     230          Spain
## 7 1816       2 United States     235       Portugal
```

과연 이렇게 만든 게 자기쌍(self-dyads)을 잘 제거했는지 확인해볼 필요가 있습니다. 논리형 연산자(==)를 이용하기 때문에 결과는 TRUE 또는 FALSE로 나타날 것입니다.

```
table(dyadyear$countryname.x == dyadyear$countryname.y)
```

```
##
##   FALSE
```

```
## 1881490
```

이번에는 1980년 이후의 자료들만 가지고 국가쌍의 하위 데이터셋을 구해보겠습니다. 대괄호([])와 subset() 함수를 이용한 방법 모두를 살펴보겠습니다.

```
## 대괄호 [] 를 이용한 방법
dyadyearp80 <- dyadyear[dyadyear$year >= 1980,]
glimpse(dyadyearp80)
```

```
## Observations: 1,205,276
## Variables: 5
## $ year          <int> 1980, 1980, 1980, 1980, 1980, 1980, 1980, 1980, ...
## $ ccode.x       <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
## $ countryname.x <chr> "United States", "United States", "United States...
## $ ccode.y       <int> 20, 31, 40, 41, 42, 51, 52, 53, 54, 55, 56, 57, ...
## $ countryname.y <chr> "Canada", "Bahamas", "Cuba", "Haiti", "Dominican...
```

```
## subset 함수를 이용한 방법
dyadyearp80 <- subset(dyadyear, dyadyear$year >= 1980)
glimpse(dyadyearp80)
```

```
## Observations: 1,205,276
## Variables: 5
## $ year          <int> 1980, 1980, 1980, 1980, 1980, 1980, 1980, 1980, ...
## $ ccode.x       <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
## $ countryname.x <chr> "United States", "United States", "United States...
## $ ccode.y       <int> 20, 31, 40, 41, 42, 51, 52, 53, 54, 55, 56, 57, ...
## $ countryname.y <chr> "Canada", "Bahamas", "Cuba", "Haiti", "Dominican...
```

이번에는 변수들의 이름을 바꿔보겠습니다. 첫 번째는 기본 함수를 이용하고, 두 번째는 plyr 패키지를 이용해서 동일한 결과를 만들어 보겠습니다.

```
## 기본 함수를 이용해 변수명 바꾸기
## names(dyadyearp80)[변수 순서] <- "바꿀 변수 이름"
## 2번째 변수부터 5번째 변수들의 이름을 바꿔보자.
names(dyadyearp80)[2:5] <- c("ccode1", "countryname1",
                             "ccode2", "countryname2")
head(dyadyearp80)
```

```
##        year ccode1  countryname1 ccode2       countryname2
## 685375 1980      2 United States     20             Canada
## 685376 1980      2 United States     31            Bahamas
## 685377 1980      2 United States     40               Cuba
## 685378 1980      2 United States     41              Haiti
## 685379 1980      2 United States     42 Dominican Republic
## 685380 1980      2 United States     51            Jamaica
```

```
## install.packages("plyr")
library(plyr)
```

```
dyadyearp80 <- rename(dyadyearp80, c("ccode.x" = "ccode1",
                                     "ccode.y" = "ccode2",
                                     "countryname.x" = "countryname1",
                                     "countryname.y" = "countryname2"))
```

예제: correlatesofwar.org에서 capabilities 데이터셋을 불러오기[3]

```
cinc.link <-
  "http://correlatesofwar.org/data-sets/national-material-capabilities/nmc-v4-data"
cinc <-
  read.csv(file = cinc.link,
           head = TRUE,
           sep = ",",
           na = c(-9))
```

COW 홈페이지에서 CINC 데이터셋을 로드해보겠습니다. CINC (Composite Index of National Capability) Score는 국가-연도 별로 국가의 물질적 능력 (national materials capabilities)에 대한 구성요소로서 측정된 여섯 가지 개별 지표들을 종합하여 한 지표로 만든 결과입니다 (Singer, Bremer and Stuckey, 1972).

- CINC는 각 구성요소를 동등하게 가중치를 부여하여 종합한 개별 연도마다의 능력 (capabilities)의 평균을 체계 전체 (total system)에서의 몫 (share)으로 나타낸 것이다.
- 결과적으로 CINC는 0부터 1 사이의 값을 가지고 0은 해당 년도 체계 내에서 그 국가가 전체의 0%의 능력을 가지고 있다는 것을 의미합니다.
- 반대로 1은 주어진 연도에서의 100%의 역량을 보여줍니다 (Correlates of War Project National Material Capabilities (NMC) Data Documentation Version 5.0. Codebook 참조)

CINC 데이터셋을 ID와 CINC score의 두 변수만 갖도록 분할해보겠습니다.

```
cinc.cut <- cinc[c("ccode", "year", "cinc")]
```

CINC 값은 주어진 연도에서 국가쌍이 아닌, **한 국가**의 힘을 측정한 결과입니다. 그러나 우리는 이 CINC score를 이용해서 국가쌍의 변수로 만들 것이기 때문에, 먼저, 첫 번째 국가군 (country1 of ccode1)에 대한 CINC 자료들을 머지해보겠습니다.

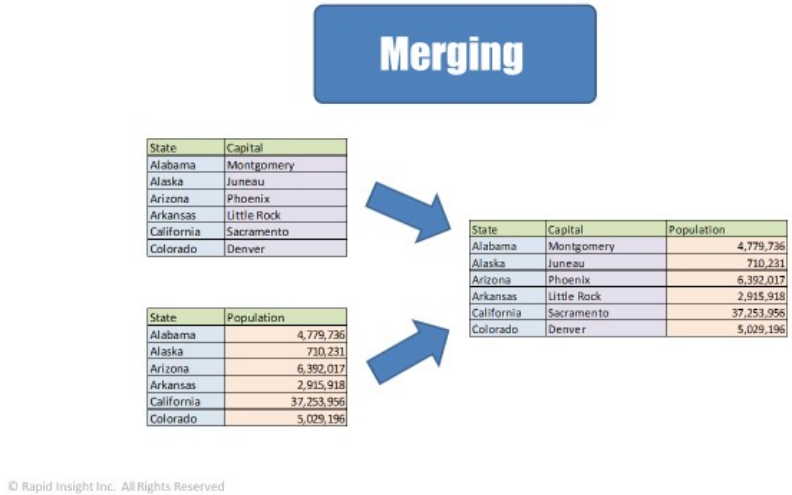
```
dyadcap <- merge(x = dyadyearp80,
                 y = cinc.cut,
                 by.x = c("ccode1", "year"),
                 by.y = c("ccode", "year"))
dyadcap <- rename(dyadcap, c("cinc" = "cinc1"))
head(dyadcap)
```

```
##   ccode1 year countryname1 ccode2 countryname2     cinc1
## 1    100 1980     Colombia    572    Swaziland 0.0036558
```

[3] 단, 결측치를 의미하는 -9를 모두 NA로 바꿔서 불러 들여올 것입니다.

```
## 2    100 1980     Colombia    581      Comoros 0.0036558
## 3    100 1980     Colombia    590    Mauritius 0.0036558
## 4    100 1980     Colombia    591   Seychelles 0.0036558
## 5    100 1980     Colombia    580   Madagascar 0.0036558
## 6    100 1980     Colombia    694        Qatar 0.0036558
```

국가쌍의 역량을 보여주는 변수(dyad capabilities)라는 의미로 dyadcap이라는 데이터를 만들어았습니다. 머지할 첫 번째 데이터는 이전에 만들어 둔 dyadyearp80이고 두 번째 머지 대상 데이터는 cinc.cut입니다. merge() 함수는 나중에 따로 구체적으로 다루겠지만 아래의 그림에서 개념을 개략적으로 파악할 수 있습니다.



그림을 보면 State 변수를 기준으로 같은 State에 Capital과 Population이 묶인 것을 볼 수 있습니다다. 위의 R-code에서는 첫 번째 데이터에서는 ccode1과 year 변수를 기준으로, 두 번째 데이터에서는 ccode와 year를 기준으로 dyadyearp80의 ccode1과 year를 cinc.cut의 ccode와 year를 동일한 것으로 간주하여 두 변수를 하나의 데이터셋 안에 머지하라는 의미입니다.

국가쌍의 CINC 변수를 만든다는 것은 국가1과 국가2의 CINC 변수 두 개가 필요하다는 것입니다. 그렇다면 이후에 또 머지하여 추가될 변수와 기존의 변수가 충돌하지 않기 위해서 이름을 바꾸어줄 필요가 있습니다. 그래서 첫 번째 CINC 변수를 CINC1로 바꾸어주었습니다.

이제 국가2의 CINC score를 머지해보겠습니다. 앞서와의 동일한 과정을 진행하되 ccode2에 대한 CINC score를 머지해야 하니까 기준을 바꿔주면 됩니다.

```
dyadcap <- merge(x = dyadcap,
                 y = cinc.cut,
                 by.x = c("ccode2", "year"),
                 by.y = c("ccode", "year"))
head(dyadcap)
```

```
##   ccode2 year ccode1           countryname1 countryname2      cinc1
```

```
## 1    100 1980    541              Mozambique     Colombia 0.00097980
## 2    100 1980    680 Yemen People's Republic     Colombia 0.00035290
## 3    100 1980    490        Congo - Kinshasa     Colombia 0.00238030
## 4    100 1980    900               Australia     Colombia 0.00721210
## 5    100 1980     90               Guatemala     Colombia 0.00061660
## 6    100 1980    403     São Tomé & Príncipe     Colombia 0.00000384
##        cinc
## 1 0.0036558
## 2 0.0036558
## 3 0.0036558
## 4 0.0036558
## 5 0.0036558
## 6 0.0036558
```

```
dyadcap <- rename(dyadcap, c("cinc" = "cinc2"))
```

이렇게 만들어진 두 개의 CINC score를 이용하여 국가1과 국가2의 상대적 국력을 측정하는 변수를 새롭게 만들어 보겠습니다.

```
dyadcap$caprat <- dyadcap$cinc1/dyadcap$cinc2
```

이렇게 만들어진 변수는(데이터는) 방향성을 가진 국가쌍의 분석 수준의 자료입니다. 예를 들어, 이 자료에는 미국-캐나다 간의 국력과 캐나다-미국 간의 국력이 순서만 바뀐 채로 동일한 값 (동일한 caprat 변수값)을 가지고 있습니다.

이 중복된 값은 불필요하기 때문에 방향성을 제거할 필요가 있습니다(non-directed). 아래의 변수는 ccode1이 ccode2보다 작은 경우만을 남기라는 코드입니다. 즉, ccode 값이 2인 미국과 ccode 값이 731(맞나…? 이거 쓸 때는 지금 코드북을 안보고 있숩… 아마 우리나라 맞을 것인디…)인 한국의 상대적 국력 변수를 보면,

```
dyadcap %>% select(ccode1, ccode2, year, caprat) %>%
  dplyr::filter(ccode1 %in% c(2, 731),
                ccode2 %in% c(2, 731),
                year == 2000) %>%
  knitr::kable()
```

ccode1	ccode2	year	caprat
731	2	2000	0.0712893
2	731	2000	14.0273479

로 두 국가 간의 국력비교 변수가 중복되어 있음을 확인할 수 있습니다. 이를

```
dyadcap %>%
  dplyr::filter(ccode1 %in% 2,
                ccode2 %in% 731,
                year == 2000) %>% knitr::kable()
```

ccode2	year	ccode1	countryname1	countryname2	cinc1	cinc2	caprat
731	2000	2	United States	North Korea	0.1429513	0.0101909	14.02735

의 결과로 바꾸어주는 것입니다.

```
dyadcapnd <- dyadcap[dyadcap$ccode1 < dyadcap$ccode2,]
```

이번에는 방향성을 제거한 국가쌍의 상대적 국력 지표를 만들어보겠습니다. 두 국가의 국력 중 큰 국력을 작은 국력으로 나누어서 상대적 국력 변수를 만들 것입니다.

```
dyadcapnd$caprat <- pmax(dyadcapnd$cinc1, dyadcapnd$cinc2)/
  pmin(dyadcapnd$cinc1, dyadcapnd$cinc2)
summary(dyadcapnd$caprat)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##      1.0      2.7      9.1    506.8     52.9 775694.9
```

상대적 국력 변수라고 이름짓는 이유는 두 `cinc1`과 `cinc2`의 값이 같다면 (상대적 국력이 같다면) 이 변수는 1의 값을 가질 것이고, 만약 `cinc1`이 `cinc2`에 비해 상대적으로 큰 국력을 가진다면 1보다 큰 값을 가질 것이기 때문입니다. 다만 `pmax()`와 `pmin()` 함수는 뒤에 포함된 괄호 안의 두 값 중 최대값과 최소값을 뽑아내라는 의미이므로 두 국가쌍의 값은 항상 최대값/최소값을 갖습니다.

만약 이 함수를 이용하지 않았더라면 `cinc1 < cinc2` 인 상황에서 `cinc1/cinc2`로 계산되어, 1보다 작은 분수 값을 가지게 되었을 테지만, `pmax & pmin` 조합을 통해서 우리는 1을 최소값으로 갖는 변수로 상대적 지표를 조작하게 된 것입니다.

요약통계표를 .tex 파일로 저장하기

```
library(stargazer)
dyadcapnd <- subset(dyadcapnd,
                    select = -c(countryname1, countryname2))
glimpse(dyadcapnd)
```

```
## Observations: 434,728
## Variables: 6
## $ ccode2 <int> 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, ...
## $ year   <int> 1980, 1980, 1980, 1980, 1980, 1980, 1980, 1980, 1980, 1...
## $ ccode1 <int> 90, 70, 94, 55, 92, 54, 53, 31, 56, 20, 2, 40, 93, 95, ...
## $ cinc1  <dbl> 0.00061660, 0.01232570, 0.00020330, 0.00001250, 0.00048...
## $ cinc2  <dbl> 0.0036558, 0.0036558, 0.0036558, 0.0036558, 0.0036558, ...
## $ caprat <dbl> 5.928965, 3.371547, 17.982292, 292.464000, 7.545511, 10...
```

`= -c` 표시는 이 기호 뒤의 벡터를 제외한 모든 변수들을 선택하여 하위 데이터셋을 만들라는 명령입니다. 혹은 아래와 같이 포함하고 싶은 변수들을 벡터로 하나하나 다 적어도 됩니다.

stargazer() 함수를 이용해서 요약통계치를 만들 때에는 굳이 국가명이 필요하진 않으니 제외하였습니다.

```
stat.table <- stargazer(dyadcapnd,
                        covariate.labels =
                          c("Country code 1",
                            "Country code 2", "Year",
                            "CINC 1", "CINC 2",
                            "Capability ratio"),
                        title = "Summary Statistics",
                        label = "stat.table")
write(x = stat.table, file = "tables/table1.tex")
```

.tex 파일에 대한 소개는 다른 탭의 포스팅을 통해 진행하도록 하겠습니다. 일단 여기서는 코드만 알아두고 써먹을 일 있으면 copy & paste 해보시기 바랍니다.

서로 다른 분석수준(lower & higher) 통합하기

일단 COW의 양자간 무역 데이터(Correlates of War trade bilateral data version 3.0)를 불러오겠습니다자. 이 변수는 csv 또는 dta 파일과는 달리 온라인에 게재되어 있기 때문에 바로 객체로 불러와 열 수는 없습니다. 따라서 먼저 압축파일(zip)의 형식으로 다운받고 난 뒤에 그 압축파일에서 필요한 자료를 추출해내야 합니다.

```
link <- "http://correlatesofwar.org/data-sets/bilateral-trade/cow_trade_3.0"
download.file(url = link,
              destfile = "COWTrade3.0.zip", mode="wb")
unzip("COWTrade3.0.zip", exdir = getwd())
```

압축을 푼 파일을 R의 객체로 불러들여 열고 분석할 것입니다. 데이터는 압축이 풀린 폴더의 하위에 있기 때문에 그 디렉토리를 정확하게 반영하여 코드를 짜야 합니다.[4]

```
btrade <- read.csv(file = "COW_Trade_3.0/dyadic_trade_3.0.csv")
names(btrade)
```

```
##  [1] "ccode1"            "ccode2"            "year"
##  [4] "importer1"         "importer2"         "flow1"
##  [7] "flow2"             "source1"           "source2"
## [10] "bel_lux_alt_flow1" "bel_lux_alt_flow2" "china_alt_flow1"
## [13] "china_alt_flow2"   "version"
```

```
glimpse(btrade)
```

```
## Observations: 791,490
## Variables: 14
## $ ccode1            <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,...
## $ ccode2            <int> 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, ...
```

[4] 정말 짜증나는군요. 이런 데이터는 써주지 말아야 합니다. COW 불매운동…

```
## $ year               <int> 1920, 1921, 1922, 1923, 1924, 1925, 1926, 19...
## $ importer1          <fct> United States of America, United States of A...
## $ importer2          <fct> Canada, Canada, Canada, Canada, Canada, Cana...
## $ flow1              <dbl> 611.86, 335.44, 364.02, 416.00, 399.14, 473....
## $ flow2              <dbl> 735.48, 442.99, 502.84, 598.14, 496.32, 608....
## $ source1            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -9, 1, 1, 1, 1...
## $ source2            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ bel_lux_alt_flow1 <dbl> -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, ...
## $ bel_lux_alt_flow2 <dbl> -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, ...
## $ china_alt_flow1   <dbl> -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, ...
## $ china_alt_flow2   <dbl> -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, -9, ...
## $ version            <dbl> 2.01, 2.01, 2.01, 2.01, 2.01, 2.01, 2.01, 2....
```

이렇게 불러들여온 양자간 무역 자료의 분석수준은 무엇일까요? 그리고 각 변수들은 무엇을 의미할까요? 이러한 내용은 **R**로 불러들여온 자료만으로는 알 수 없습니다. 따라서 항상 데이터셋을 다운받을 때는 코드북을 같이 다운 받아서 숙지하는 습관을 들일 필요가 있습니다. 일단 당장의 분석에 필요없는 변수들을 날려버리겠습니다.

```
btrade$source2 <-
  btrade$bel_lux_alt_flow1 <-
  btrade$bel_lux_alt_flow2 <-
  btrade$china_alt_flow1 <-
  btrade$china_alt_flow2 <-
  btrade$version <- NULL
glimpse(btrade)
```

```
## Observations: 791,490
## Variables: 8
## $ ccode1    <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2...
## $ ccode2    <int> 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, ...
## $ year      <int> 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928...
## $ importer1 <fct> United States of America, United States of America, ...
## $ importer2 <fct> Canada, Canada, Canada, Canada, Canada, Canada, Cana...
## $ flow1     <dbl> 611.86, 335.44, 364.02, 416.00, 399.14, 473.72, 974....
## $ flow2     <dbl> 735.48, 442.99, 502.84, 598.14, 496.32, 608.35, 609....
## $ source1   <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -9, 1, 1, 1, 1, 1, 1, ...
```

중요한 것은 이 데이터셋에서 결측치는 NA가 아니라 -9로 코딩되어 있다는 사실입니다. 만약 이 -9들을 가만히 놔뒀다가는 요약통계와 추론통계를 부정확하게 만들 것이기 때문에 flow1과 flow2의 결측치들을 NA로 바꾸어 주어야 합니다. 사실 csv 파일을 불러들여올 때 했을 수도 있지만 여기서는 불러들인 상황에서 뒤늦게 -9를 발견하고 바꾸는 대처 방법을 배워봅시다.

```
btrade$flow1[btrade$flow1 == -9] <- NA # 대괄호는 조건(condition)을 의미합니다.
btrade$flow2[btrade$flow2 == -9] <- NA
summary(btrade$flow1) # -9이 사라졌습니다.
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
##      0.0      0.0      0.1    158.4      6.0 363989.6   224291
```

개별 국가들의 연간 총무역량을 계산해보겠습니다. 먼저 국가쌍의 수입(imports)과 수출(exports)을 더합니다. 이때 만약 flow1나 flow2 둘 중 하나가 결측치일 경우 총무역량 또한 결측치로 나타날 것입니다. 따라서 단순 더하기 코드로는 결과가 왜곡될 수 있습니다.

```
btrade$tottrade1 <- btrade$flow1 + btrade$flow2 # 잘못된 결과를 얻게 됩니다.
summary(btrade$tottrade1)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
##       0.0       0.0       1.0     315.1      18.9 1198528.8    250983
```

```
btrade$tottrade2 <- ifelse(is.na(btrade$flow1), btrade$flow2,
                           ifelse(is.na(btrade$flow2), btrade$flow1,
                                  btrade$flow1 + btrade$flow2))
summary(btrade$tottrade2)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
##       0.0       0.0       0.6     289.0      14.6 1198528.8    202128
```

첫 번째 코드는 flow1과 flow2 (수입과 수출)을 더하여 총무역량1이라는 변수를 만들라는 것입니다. 이때, 둘 중 하나에 결측치가 있으면 그 합 역시 결측치가 됩니다.

이 문제를 피하기 위한 두 번째 코드는 만약 flow1이 결측치라면 총무역량2라는 변수에 flow2의 값을 부여하라는 얘기이고 만약 flow1이 결측치가 아니며, 동시에 flow2는 결측치라면 그 자리에 flow1 값을 부여하라는 얘기입니다. 만약 둘 다 결측치가 아닐 경우에는 마지막의 조건, flow1과 flow2를 결합한 값을 넣으라는 얘기입니다.

이와 같은 결과는 plyr 패키지를 이용해서도 얻을 수 있습니다. plyr패키지의 summarise() 함수를 이용하여 새로운 데이터프레임을 만들 수 있습니다. 물론 기존 데이터프레임으로 지정하면 새롭게 통합(aggregate)된 자료로 대체됩니다.

이렇게 얻은 styrtrade1이라는 변수는 ccode1과 year 단위로 결측치를 제외하고(na.rm = TRUE) 총무역값을 전부 더한 결과입니다. 그리고 그 양자간 무역 총액 변수를 머지하여 btrade 데이터에 붙여줍니다. 이제 국가1의 특정연도 하에서의 양자간 무역과 그 해의 총액을 확인할 수 있습니다.

아래의 코드로 만든 share1은 국가1(state1)의 총무역량에서 양자간 무역이 차지하는 비율을 보여줍니다.

```
library(plyr)
btrade <- ddply(btrade, .(ccode1, year),
                transform, styrtrade1 = sum(tottrade2, na.rm = TRUE))
btrade$share1 <- btrade$tottrade2 / btrade$styrtrade1
```

오늘의 포스팅에서는 꽤나 여러 가지 자료들을 다루어 보았는데, 좀 복잡할 수도 있습니다. 사실 저도 IR 쪽 자료는 잘 사용하지 않고 교차사례 시계열로만 쓰는데 IR은 그걸 국가쌍의 자료로 (dyadic) 재구성해서 국가들 간의 관계를 살펴보는 연구들도 합니다. 여하튼 이번 포스팅에서는 plyr패키지의 사용을 확인하고 숙지하는 것만 건져도 승리하는 것이라고 할 수 있겠습니다.

Chapter 4

Probability into R

확률 관련 포스팅에서는 몇 가지 간단한 확률을 R을 통해서 증명하고 (Some simple probability demonstrations), 정규분포 (normal distributions) 와 이항분포 (binomial distributions) 로부터 분위 (quantiles) 를 얻어내는 방법을 살펴보고자 합니다.

여기서는 일단 간단하게 정규분포랑 이항분포, 그리고 독립사건과 종속사건만 간단하게 살펴보겠습니다.

- 분위 (Quantiles) 에 대해 어떻게 설명할까 고민을 좀 했는데 좋은 블로그 포스팅을 찾아서 참조용으로 링크하겠습니다.
- 링크는 여기를 보시면 됩니다. Q-Q Plot 설명하면서 Quantile의 정의도 잘 정리해놓으셨습니다.

그럼 본격적으로 포스팅을 시작하기에 앞서서 항상 그렇듯이 작업 디렉토리를 설정해보겠습니다.

```
rm(list=ls()) # 현재 콘솔 창에 저장되어 있는 모든 값과 모델 등을 삭제
```

```
library(here)
library(knitr)
library(dplyr)
library(tidyverse)
library(kableExtra)
here::here() %>% setwd()
```

주사위굴리기 게임!

확률을 공부할 때, 지겹도록 등장하는 놈들이 총 셋이 있습니다. 동전, 카드, 그리고 주사위입니다. 인류는 아마도 이 셋을 만듦으로써 스스로를 괴롭히는 통계학을 발전시켰는지도 모르겠습니다… 일단 주사위 굴리기는 직관적으로 확률과 통계를 이해하기 좋은 방식입니다. 먼저 주사위 하나를 한 번 굴리는 것을 시뮬레이팅하는 함수를 코딩해보겠습니다.

```
die <- as.integer(runif(1, min=1, max=7))
die
```

```
## [1] 3
```

die는 ?runif 라고 입력하여 살펴보면 generates random deviates.라고 기술되어 있는 것을 확인할 수 있습니다. 이어지는 함수를 풀어서 설명하면 다음과 같습니다.: 다음의 결과를 정수의 형태로 저장하라(as.integer) → 무작위로 다음의 범주 내에서 다른 값을 1번 추출하라 → 최소값은 1, 최대값은 6 (1 이상 7미만)을 갖게하라.

그러면 이번에는 두 개의 주사위를 굴려보도록 하겠습니다. 굴리는 횟수는 한 번입니다.

```
dice <- (as.integer(runif(1, min=1, max=7))) +
  (as.integer(runif(1, min=1, max=7)))
dice
```

```
## [1] 12
```

여기서 + 연산자는 부울리안 논리에 따르면 OR를 의미합니다. 즉 각 주사위를 한 번씩 랜덤으로 굴려 얻는 값을 더한 결과를 dice에 저장하라는 명령입니다. 그럼 이번에는 100번, 1000번, 그리고 100만번을 돌려겠습니다.

```
# 주사위 두 개를 100번 던져보기
dice100 <-  (as.integer(runif(100, min=1, max=7))) +
  (as.integer(runif(100, min=1, max=7)))
dice100 %>% table() %>% t() %>% knitr::kable() %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"),
                            full_width = F, position = "float_right")
```

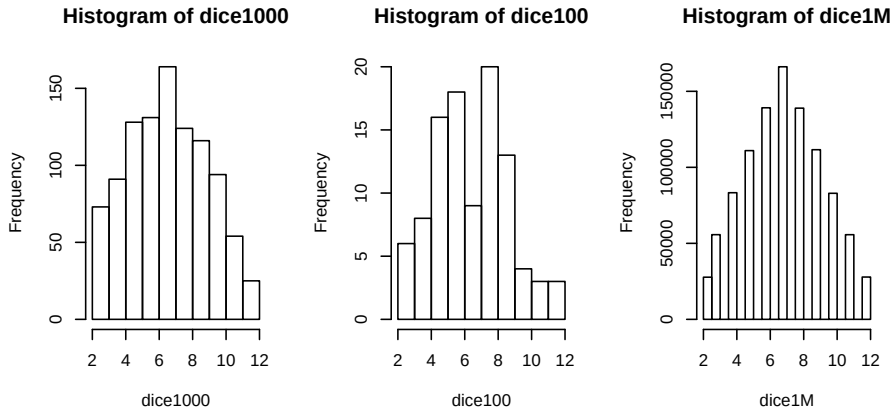
```
# 주사위 두 개를 1,000번 던져보기
dice1000 <-  (as.integer(runif(1000, min=1, max=7))) +
  (as.integer(runif(1000, min=1, max=7)))

dice1000 %>% table() %>% t() %>% knitr::kable() %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"),
                            full_width = F, position = "left")
```

2	3	4	5	6	7	8	9	10	11	12
25	48	91	128	131	164	124	116	94	54	25

```
# 주사위 두 개를 100만 번 던져보기
dice1M <-  (as.integer(runif(1000000, min=1, max=7))) +
  (as.integer(runif(1000000, min=1, max=7)))
dice1M %>% table() %>% t() %>% knitr::kable() %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"),
                full_width = F, position = "float_right")
```

```
par(mfrow = c(1, 3))
hist(dice1000)
hist(dice100)
```

```
hist(dice1M)
```



2	3	4	5	6	7	8	9	10	11	12
27733	55703	83292	110990	139165	166151	138927	111597	82927	55688	27827

와! 단 시적으로 살펴볼 수 있게 각 코드 이후에 그 결과값의 빈도를 표로 나타내보았습니다. 그리고 그 표를 히스토그램으로 재구성해보았습니다. 역시 N이 늘어날 수록 우리(?)가 사랑하는 그 녀석의 모습이 드러나기 시작합니다.

주사위는 1에서 6까지의 한정된 값을 가지고, 두 개를 합쳐서 굴려봐야 2부터 12까지의 한정된(bounded) 값이긴 하지만 이 주사위 굴리기를 통해서 우리는 지난 번 포스팅에서 살펴보았던 것처럼 정규분포(normal distribution)와 표본 크기(n), 혹은 표집(sampling)의 관계를 간접적으로 다시 한 번 살펴볼 수 있습니다. 동전 던지기

Chapter 5

Applications

Some significant applications are demonstrated in this chapter.

Example one

Example two

Chapter 6

Final Words

We have finished a nice book.