개방형 데이터 저널리즘 교재 (초고)

박찬경

2023-06-26

목차

서문

추후 작성

이 웹사이트는 지역 언론인을 위한 개발 사이트 입니다. donorschoose.org data와 스포츠, 공공데이터 이용. 언어로는 R을 이용할 것입니다. - 상대적으로 쉽고, - 공짜고, - 해외 데이터 저널리즘 커뮤니티에서 사실상 표준으로 사용되고 있습니다.

https://github.com/fivethirtyeight https://github.com/TheEconomist

- 좋은 IDE를 가지고 있고
- 바로 온라인으로 publish할 수 있는 기술이 잘 발달되어 있습니다 (공동 프로젝트 가능)

    - 이 책 역시 같은 방식을 이용해 만들어지고 있습니다.

복잡한 것들을 최대한 늦게 필요할 때 가르치는 것을 목표로 합니다. 빅데이터 또는 주어진 관찰 데이터를 예시로 삼습니다.

저널리즘 - 시각화/데이터 스토리텔링 이론 등등

기초와 고급 토픽 구분

모든 단원 시작할 때 필요한 패키지부터.

서문

To-do's

- 데이터 시각화 하기 2 (Optional): Storytelling
  - 웹 프로그래밍의 응용
  - HTML, CSS 이해
- 공공 데이터 이용하기
  - API 이해하기
  - XML, JSON 이해하기
  - For-loop, function 이해하기
- 스크래이핑
  - 고급 스크레이핑에 대한 이야기.
- 고급 토픽: 공간 데이터 이용하기 + 시각화.
- 고급 토픽: 예측을 이용한 보도
  - 주식 집값이 보편적인 예시이지만,
  - 스포츠가 더 plausible한 예시.
- 고급 토픽: Interactive Design
- 데이터 배포하기에 대한 이해
  - HTML
  - Markdown
  - Pandoc
  - Quarto
  - Github

빠진 부분

ifelse() -> Boolean

1 데이터 저널리즘이란

1.1 데이터 저널리즘의 사례와 요소

데이터 저널리즘이라는 이야기를 많이 들어보셨을 것입니다. 그러나 데이터 저널리즘이 무엇인지 명확한 정의를 가지고 있는 분은 많지 않을 것이라고 생각합니다. 아마도, '데이터를 이용한 저널리즘'이 좋은 정의가 아니라는 것은 명백한 것 같습니다. 저널리즘이 '사실'을 전달하는 일인 이상, 그 사실의 증거가 되는 어떠한 데이터를 이용하지 않을 방법은 없으니까요. 그것이 통계가 되었든, 인터뷰가 되었든, '관계자'로 부터 비공식적으로 듣게된 증언이든, 그러한 증거들은 모두 데이터 입니다. 따라서, 데이터를 사용한다는 것 그 자체가 데이터 저널리즘(그런 것이 따로 존재한다면)을 정의하지는 않는 것 같습니다.

어떠한 대상을 정의하기 어려울 때, 우리가 흔히 취하는 방식 중 하나는 그 대상의 '외연(外延)'을 나열하는 것입니다. 즉, 데이터 저널리즘은 이러한 것이다, 라고 말하기 어려우니, 우리가 데이터 저널리즘으라고 여기는 사례를 죽 검토해보고, 아, 우리는 이런 것이 있어야 데이터 저널리즘이라고 여기는구나, 라고 경협적으로 접근해 보는 것이지요. 이는 데이터 저널리즘의 '방법' 문제를 넘어서 (정의 없이) 어떤 데이터 저널리즘이 좋은 데이터 저널리즘인가를 생각해 보는데 도움이 됩니다. 그러니 몇 가지 사례들을 검토해보도록 하지요.

다음 그림은 코로나 19가 여전히 맹위를 떨치던 2021년 7월 파이낸셜 타임즈에선 낸 기사입니다.

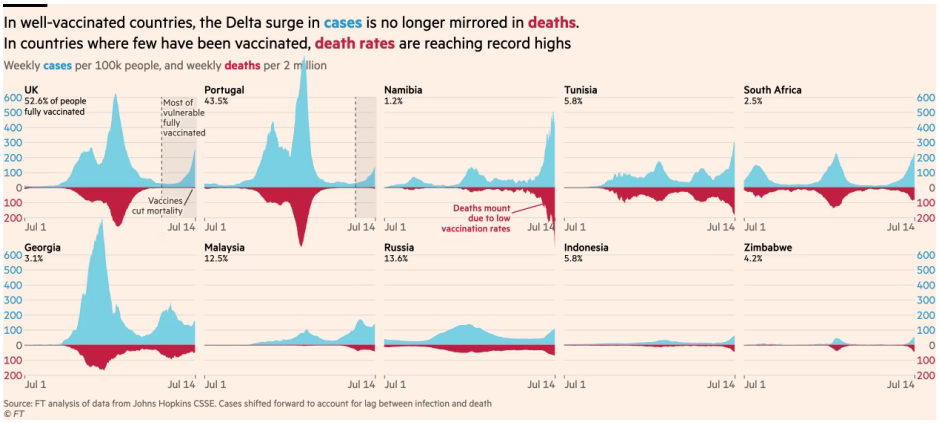
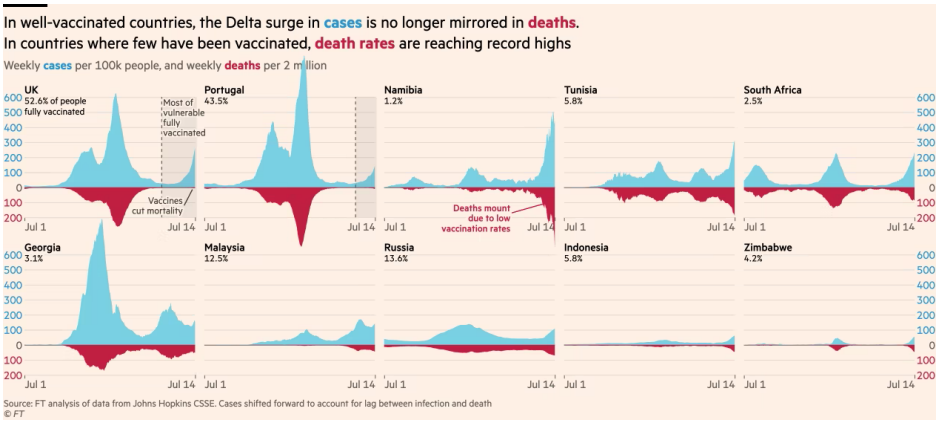


그림 1.1: 파이낸셜 타임즈의 코로나 백신 보도



이 보도에서 푸른 색으로 표현된 그래프는 10개 국가의 일자별 코로나 감염자수를 표현하고 있습니다. 반면, 빨간색은 그래프는 사망자수를 (거꾸로) 표현하지요. 만약 백신이 없다면, 파란색 그래프와 빨간색 그래프는 대칭이어야 할 겁니다. 감염자가 많아지면 사망자가 많아지는 것은 당연한 이치이니까요. 실제로 그래프로 표현된 10개의 국가들에서 대부분의 기간동안 그러한 대칭성이 보입니다. 그런데, 예외가 있습니다. 제일 처음 표현된 영국(UK)와 포르투갈의 7월15일 이후 그래프 이지요. 7

월15일에 대부분의 위험군에게 백신 접종이 완료되었고, 그 이후에는 더 이상 파란 그래프와 빨간 그래프가 대칭적이지 않게 되었습니다. 백신이 광범위하게 접종되지 않은 다른 나라들은 물론 그런 일이 일어나지 않았지요. 즉, 영국과 포르투갈은 위험군에 대한 백신 접정 덕분에 코로나 감염자가 늘더라도, 사망자가 그만큼 늘지는 않게 되었다는 것입니다.

왜 이런 보도가 등장했을까요? 이 때는 '안티백서'라고 코로나19 백신 접종에 반대하는 사람들의 주장이 특히 서양에서 기승을 부리던 때입니다. 이들의 주장 중 하나가 코로나19 백신은 사실 효과가 없다는 것이었죠. 사실 어떤 신약이 질병에 '인과적인' 효과가 있다는 것을 증명하는 것은 대단히 어렵고, 시간이 오래 드는 일입니다. 그리고 그러한 과정을 통해 밝혀낸 효과의 근거를 대중들에게 설명하는 것은 더욱 더 어려운 일이지요. 위의 그래프 역시 사실 뜯어보면 대단히 많은 정보들이 포함되어 있습니다. 하지만, 두 그래프를 반전시켜서 대칭성이라는 시각 요소를 사용하고, 그래프 색깔과 글자의 '깔맞춤'(파란색으로 표시된 'cases'라는 글자와 빨간색으로 표시된 'deaths'라는 글자를 주목해보세요)을 통해, 많은 양의 정보를 효율적으로 표현하여, 백신의 효과를 상당히 설득력 있게 전달하고 있지요.

다음은, 데이터 저널리즘의 선구자라고 할 수 있는 미국 언론사 FiveThirtyEight의 스포츠 보도 사례입니다. 스포츠 보도야 말로, 데이터 저널리즘의 힘을 제대로 느낄 수 있는 분야 중 하나죠. 첫번째 그림은 이미 미국프로농구(NBA)의 레전드라고 할 수 있는 르브론 제임스와 앤소니 데이비스라는 선수가 LA레이커스에서 29경기(y축) 각 48분(x축) 동안 언제 함께 뛰었고, 언제 혼자 뛰었는지를 시각화한 것입니다. 짙은 파란색은 두 선수가 같이 뛴 시간, 하늘색은 앤소니 데이비스, 분홍색은 르브론 제임스가 혼자 뛴 시간 입니다. 회색은 두 선수 모두 뛰지 않은 시간이라고 하네요. 여기서 보면 알 수 있듯이, 생각보다 한 경기 안에서 두 명이 같이 뛰는 시간이 길지 않다는 것을 알 수 있습니다. 기사에 따르면 실제로 레이커스는 전략적인 이유로 두 선수를 번갈아 기용하는 이른바 '로테이션'을 해야할 필요가 있었다고 하네요.

다음 그림은 NBA의 또 다른 레전드, 스테픈 커리와 드레이몬드 그린이 골든스테이트 워리어스에서 경기중 언제 함께 뛰고 언제 따로 뛰었는지를 시각화한 것입니다. LA 레이커스와는 한 눈에 보아도 확연한 차이를

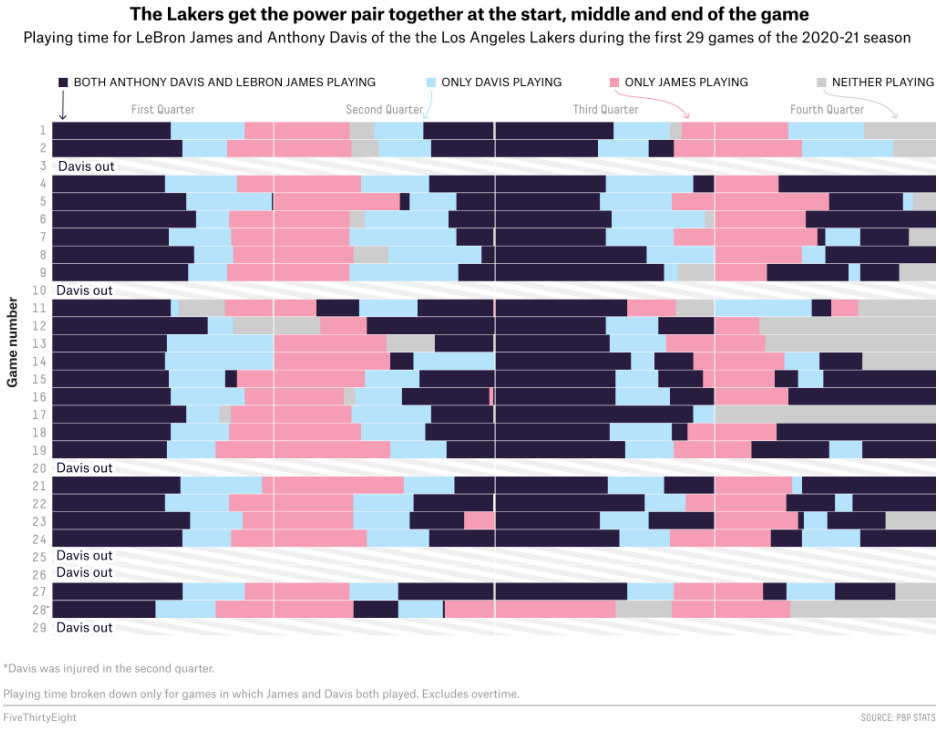

The Lakers get the power pair together at the start, middle and end of the game
Playing time for LeBron James and Anthony Davis of the the Los Angeles Lakers during the first 29 games of the 2020-21 season
BOTH ANTHONY DAVIS AND LEBRON JAMES PLAYING
ONLY DAVIS PLAYING
ONLY JAMES PLAYING
NEITHER PLAYING
First Quarter
Second Quarter
Third Quarter
Fourth Quarter
Game number
Davis out
*Davis was injured in the second quarter.
Playing time broken down only for games in which James and Davis both played. Excludes overtime.
FiveThirtyEight
SOURCE: PBP STATS

그림 1.2: LA레이커스

느낄 수 있지요? 워리어스의 두 선수는 뛰면 함께 뛰고, 안 뛸 때는 같이 빠지고 있네요.

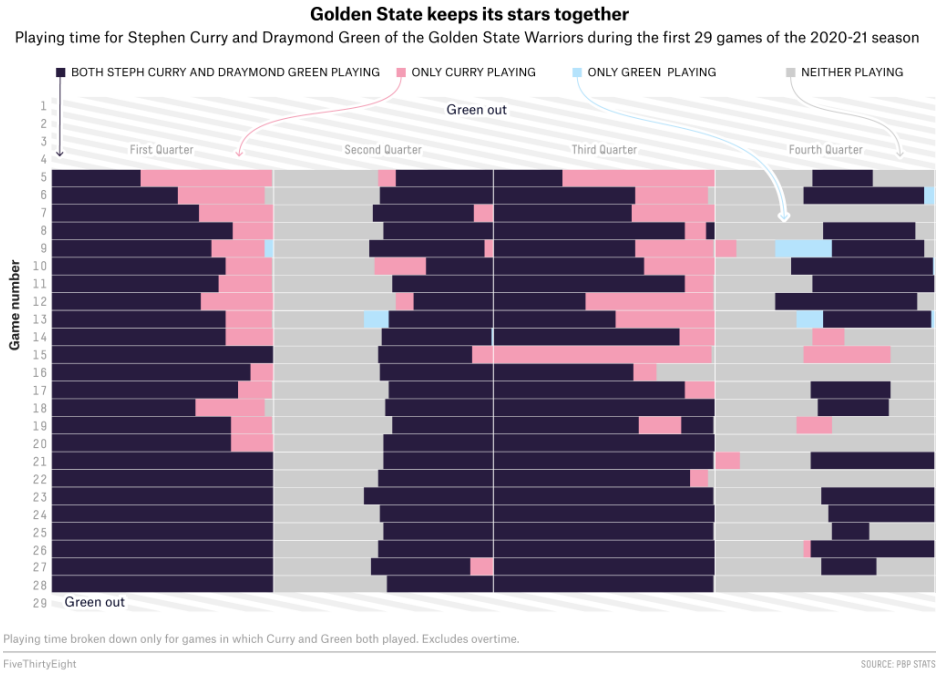


그림 1.3: 글든스테이트 워리어스

이 기사는 기존의 일반적인 기사들과 다음 두 가지 점에서 차이를 보입니다. 첫째, 이전에는 사용하지 않았던, 혹은 사용하지 못했던 데이터를 사용합니다. 기존의 보도가 이용했던 데이터는 이미 전문가들에 의해 가공된 데이터들이었습니다. 통계청이 되었던, 기업이 되었던, 미리 어느 정도 이해할 수 있는 형태로 재구성해서 비교적 작은 테이블에 담아둔 데이터를 마이크로소프트 엑셀과 같은 스프레드시트로 열어 그룹별 평균을 구하고, 그래프를 그리는 것이 일반적인 보도를 위한 데이터 처리 방식이었죠. 하지만 위와 같은 데이터는 가공된 데이터가 아닙니다. 선수들의 코트위에서의 행동이 그대로 기록된 '행동 데이터'이죠. 선수가 코트에 나와서 뛰고 있다는 사실은 그 자체로 데이터가 됩니다. 이러한 데이

터는 센서를 이용한 데이터 생성과 분석이 스포츠에서 광범위하게 사용되면서 이전에 비해 훨씬 흔해졌습니다. 지난 번 2022년 카타르 월드컵 포르투갈전에서 황희찬 선수가 상의를 탈의했을 때, 브라톱 형태의 센서가 부착된 조끼를 입고 있어서 화제가 된 적이 있었죠? 그러한 장치들이 운동선수의 실시간 행위를 데이터로 만들어내고 있는 것이지요. 사실 데이터의 크기와 상관 없이 이렇게 가공되지 않은 채, 유입되는 실시간 비정형 데이터가 '빅데이터'의 정의에 가깝습니다. 이러한 데이터들은 이전에는 존재하지 않았거나, 존재했더라도 가공/분석 방법이 없어서, 컴퓨터의 성능이 모자라서, 등의 이유로 분석되지 않았습니다. 하지만, 위의 보도 사례를 보면, 과감하게 두 팀, 네 명의 선수들이 29경기 각48분 동안 언제 뛰었는지를 모조리 보여주고 있네요. 빅데이터의 이용이 보도로부터 그리 멀지 않다는 것을 보여주는 사례 입니다.

두 번째 차이는 시각화 입니다. 이는 앞서 이야기한대로, 이전에는 데이터가 존재했더라도, 어떻게 가공/분석할 줄을 몰라서 사용하지 않았다는 지적과 강한 관련이 있죠. 전통적인 언론 보도들이 데이터를 통한 분석을 제시하는 경우는 많았지만, 그 결과는 작은 표나, 막대 그래프, 선 그래프 등을 이용해 평균 등의 몇 가지 수로 요약해 표현하는 것이 대부분이었습니다. 그에 반에 위의 시각화는 상당히 창의적인 시각화 방식을 동원해서 데이터를 거의 있는 그대로 보여주고 있지요. 그럼에도 불구하고, LA 레이커스와 골든스테이트 워리어스의 차이는 명확했죠? 이것이 뒤에서 이야기 하게 될 '데이터 스토리텔링'의 힘입니다. 좋은 시각화 전략은 훨씬 더 많은 정보를 독자들에게 전달할 수 있게 해 줍니다. 그렇게 독자들에게 전달할 수 있어야, '빅데이터'를 입수했다는 사실이 비로소 의미가 있는 것이겠지요.

다음 사례는 KBS의 손흥민 선수에 대한 보도입니다. 축구에 큰 관심이 있지 않았던 분들도 손흥민 선수에 대한 보도에서 '기대득점'이라는 용어를 사용하는 것을 들어보셨을 것입니다. 예컨대, 손흥민 선수가 골을 자주 넣는 '손흥민존'에서 기대득점이 예를들면 0.03에 불과했는데도 불구하고 골을 넣었다, 그런 식의 보도 말이죠. 그런데, '기대득점'이 뭘까요? 사실 이건 위에서 이야기한 스포츠 빅데이터와 관련이 있습니다. 이제 워낙 많은 실시간 데이터가 쌓이다 보니, 어떤 선수가, 경기 중 어떤 시점에, 어떤 위치에서, 어떤 팀을 상대로 슛을 시도했을 때 골이 들어갈 확률이 몇%다, 이런 예측을 통계 모형을 통해 할 수가 있게 되었습니다. 즉,

앞서의 시나리오에서라면, 손흥민 선수가 골을 넣은 곳이 과거 데이터에 따른 예측 모형에 따르면 100번 슛을 해야 고작 3번 골이 들어갈 수 있는 지점이었는데, 그럼에도 골을 넣었다는 것이죠. 더 간단히 말해, 어지간하면 골을 넣지 못할 상황에서 골을 넣었으니, 손흥민 선수가 대단하다, 그런 의미인 것이지요.

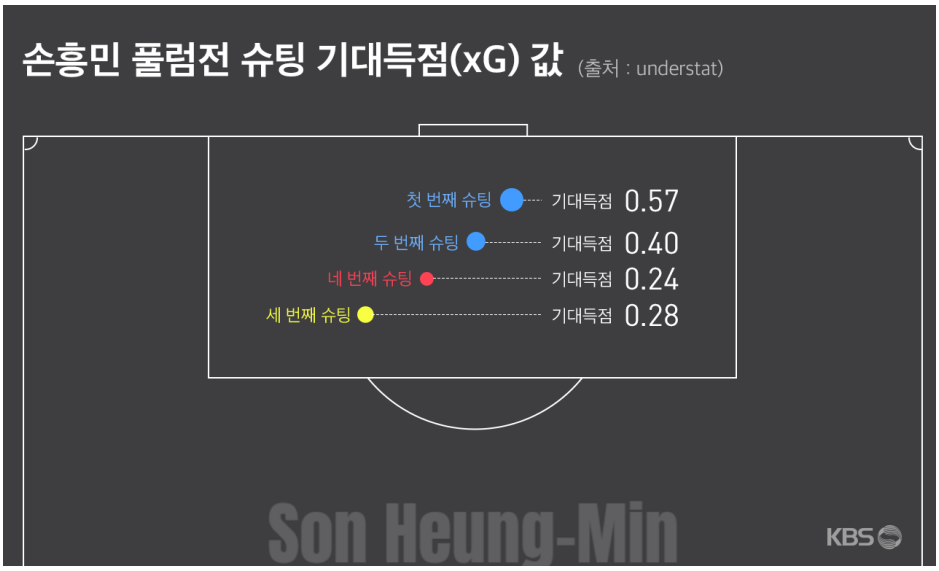


그림 1.4: 손흥민 기대득점

여기서는 통계 모형과 예측이 이용되었다는 점이 중요합니다. 과거 보도에서도 예측은 사용되기도 했지만, 다소 조심스럽게 사용되었습니다. 예컨대 어떤 증권사의 주가 예측이라던지, 부동산 전문가 여러명의 부동산 가격 추이 예측을 '전문가 의견'으로 참조한다든지 하는 식이죠. 데이터 저널리즘에서는 예측이 조금 더 적극적으로, 객관적 전거로 사용됩니다. 행동 데이터가 다양해지고, 이를 이용한 예측 모형이 발달하면서, 모형의 성능이 훨씬 좋아졌기 때문이죠. 다만, 모든 분야에서 그런 것은 아닙니다. 안타깝게도 주가와 부동산 가격 예측은 보도에서 객관적 전거로 사용할 수 있을 정도로 정확하지 못합니다.

1 데이터 저널리즘이란

통계 모형 이야기가 나왔으니, 관련된 데이터 저널리즘 보도를 하나 더 보도록 하지요. 다음은 서울대 국제정치데이터센터와 MBC가 협업해서 운영하고 있는 여론M이라는 웹사이트 입니다.

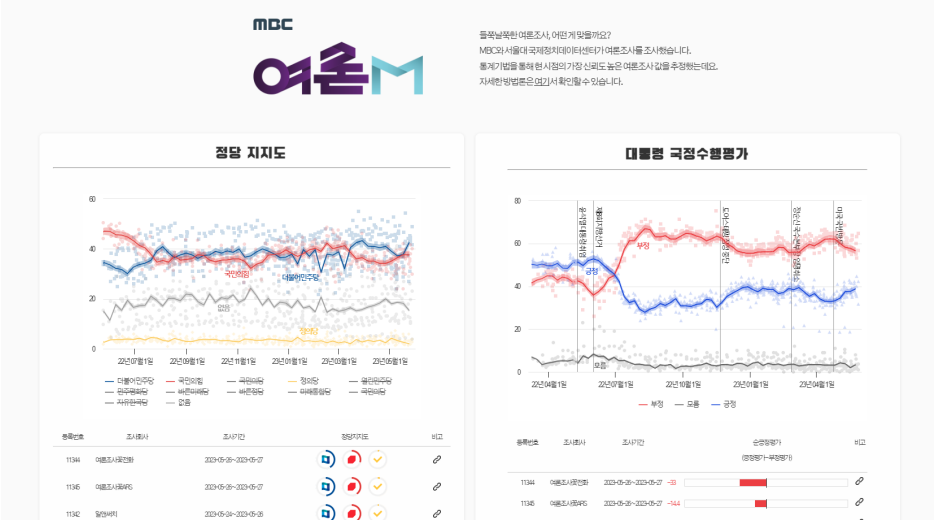


그림 1.5: 여론M

이 시각화 자료들은 정당 지지도와 대통령 국정수행평가에 대한 여론조사 결과를 보여주고 있습니다. 이런 여론조사는 너무도 흔한데, 왜 이러한 보도를 데이터 저널리즘이라고 하는 것일까요? 왜냐하면, 위의 선 그래프가 나타내고 있는 것은 실제 여론조사 결과가 아니라, 여러 여론조사 결과를 통계모형을 통해 합산한 결과이기 때문입니다. 왜 그런 합산을 하냐고요? 독자들이 더 이상 여론조사 결과를 믿지 않기 때문이죠. 정치적 양극화가 극심해지면서, 꽤 많은 독자들이 자신의 정치적 성향에 따라, '대통령의 지지율이 낮게 나온 것은 설문조사 기관이 좌파라서', 또는 '민주당 지지율이 낮게 나온 것은 설문조사 기관이 우파라서' 그렇다고 믿는 것을 여러분들도 생활에서 느끼고 있을 것입니다. 위의 보도는 통계 모형을 이용해 그러한 설문조사 기관의 '편향'(이 이른바 편향은 사실 꼭 기관의 정치적 성향 때문에 나타난다고만은 할 수 없습니다. 응답자를 모으는 방법의 차이 때문에 발생하는 경우도 많기 때문이죠)을 감안한 '진짜' 여

론을 추정해서 보여주는 것입니다. 물론 이 추정된 '진짜' 여론이 '진짜로 진짜' 여론인 것은 아닙니다. 모든 모형은 100% 정확할 수 없기 때문지요. 하지만, 이런 말도 있지요. "모든 모형은 틀렸다. 하지만 어떤 모형은 유용하다".

이렇게 모형을 이용한 보도를 '유용'하게 만들기 위해서 이 프로젝트는 몇가지 툴을 동원했습니다. 첫째, 불확실성을 시각화하는 것입니다. 위의 그래프를 보면 추정된 여론이 선으로 표현되어 있지만, 그 선 주위로 투명한 색깔로 범위가 표현되어 있지요? 이는 추정 모형이 가지고 있는 불확실성을 계산하여 보여주는 것입니다. 만약 국민의힘과 더불어민주당 지지도 사이에 약간의 차이가 있더라도, 그 차이가 저 불확실성의 범위 안에 있다면, 단적으로 어떤 정당의 지지율이 더 높다, 라고 할 수는 없다는 단서조항 같은 역할을 한다고 생각하면 됩니다. 둘째, 로고 옆에 보면 추정 모형의 세부사항에 대해 알 수 있는 링크가 있습니다. 이 링크를 따라가 보면, 전문용어로 가득한 문서가 하나 나오는데요, 모형 자체가 복잡하다보니, 그 설명도 복잡합니다. 다만, 이 문서를 이해할 수 있는 전문 지식이 있다면, 웹사이트에서 보여주는 것과 동일한 추정을 할 수 있지요. 물론, 모형의 검증도 가능합니다. 이렇게, 분석 결과만 보여주는 것이 아니라, 분석 방법까지 공개해서 투명성을 확보하는 것입니다. 더 나아가, 해외 데이터 저널리즘 프로젝트 다수는 분석 방법 뿐 아니라, 원 데이터를 공유하기도 하고, 수식 뿐 아니라, 분석, 시각화 프로그램 코드까지 공개해서 바로 재사용할 수 있는 정도로까지 공유하기도 합니다. 마치 컴퓨터 개발자들이 오픈소스 소프트웨어를 개발하는 것처럼요.

마지막으로 그래프 아래에서 추정된 여론이 아닌 원래 여론조사를 결과를 확인하는 것도 가능합니다. 대부분의 사람은 위의 그래프를 보고 전반적인 여론 추세를 아는데 만족하겠지만, 좀 더 관심이 많은 사람은 각각의 여론조사 결과를 알고 싶은 경우도 있겠지요. 그럴 때는, 각 설문조사에 해당하는 링크를 클릭하면, 중앙선거관리위원회가 확보하고 있는 각 여론조사의 구체적인 정보, 예컨대, 조사인원, 조사기간, 조사지역, 조사일시, 응답자 구성까지 알 수 있습니다. 이 모든 정보를 한 화면에 보여주면 너무도 복잡하겠지만, 월드와이드웹의 기능을 이용해 관심있는 사람은 원데이터의 세부사항을 볼 수 있도록 해 주는 것이지요.

자, 지금까지 본 사례들에서 기존 언론 보도와 다소 다르게 느껴지는 것

들을 꼽아보자면 다음과 같습니다.

- 새로운 데이터
- 시각화
- 공개 데이터
- 상호작용성
- 공유
- 통계 모형과 예측
- 데이터 스토리텔링
- Surprise!

우리가 데이터 저널리즘이 무엇인가를 정의하는 것은 너무도 어렵지만, 위의 요소들을 포함한 보도를 대체로 데이터 저널리즘이라고 부르는 것 같습니다. 이전에는 사용하지 않았던, 행동 데이터, 빅데이터를 분석한 결과를 보도합니다. 그러한 데이터를 이용할 수 있게 된 것은 여러 과학 기술의 발달 때문이기도 하지만, 그것을 분석할 통계/기계학습 모형과 시각화 도구가 생겼기 때문이기도 하지요. 새로운 데이터들은 기자가 노력을 통해 단독으로 입수하는 것도 있지만, 더 많은 경우는 이미 어디엔가 공개되어 있는 것들인 경우가 더 많습니다. 데이터저널리스트에게는 나만 아는 정보를 찾아내는 것보다 이곳 저곳에 존재하는 데이터들을 연결해서 새로운 통찰을 이끌어 내는 능력이 더욱 중요합니다. 하지만, 복잡한 데이터의 분석 결과는 그 자체로 복잡한 경우가 많습니다. 따라서 효율적인 시각화와 글쓰기를 통해 분석 결과로부터 찾아낸 스토리에 집중할 수 있도록 하는 스토리텔링 능력 역시 중요합니다. 그러한 과정에서 생략해야만 했던 세부사항들은 상호작용적 인터페이스를 통해 독자에게 따로 제공하거나, 공유플랫폼을 통해 제공하는 것이지요. 하지만 제일 중요한 것은, 이 모든 요소들을 종합해 독자들을 놀라게 하는 능력이라고 할 수 있겠습니다. 몰랐던 충격적 사실을 알아서가 아니라, 독자도 접근할 수 있었지만, 연결될 줄은 몰랐던 것들의 연결을 깨달음으로써 써 세상에 대한 이해가 넓어질 때 터져나오는 경탄, 그것이 아마도 데이터 저널리즘의 중요한 목표 중요한 목표 중 하나일 것입니다.

1.2 데이터 사이언스로서의 데이터 저널리즘

사실 데이터 저널리즘에 대한 명확한 정의는 없다고 보아도 무방합니다. 2010년대 후반 이후로 많은 학자들이 데이터 저널리즘을 어떻게 정의할 것인가에 대해서 많은 논쟁들을 해 왔지만, '데이터'라는 말과 '저널리즘'이라는 말 모두가 매우 모호한 말이다 보니, 그 합성어인 데이터 저널리즘이라는 말 역시 많은 사람들이 저마다의 방식으로 이해할 뿐입니다. 하지만, 우리가 경험적으로 '데이터 저널리즘'은 대충 이런 것이라고 느끼고 있는 바는 있습니다. 그것을 저의 방식대로 요약하자면, '데이터 사이언스를 이용한 보도 방식'이라고 할 수 있을 것 같습니다. 기자들의 전통적인 보도에서는 정보원과의 긴밀한 소통이 강조되었습니다. 그것이 직접 만남을 통한 것이든, 잠입을 통한 것이든, 전화 인터뷰를 통한 것이든, 기자들이 몸으로 뛰어 다니며 정보, 또는 정보원을 캐내는 그 물리적 과정이 참된 저널리스트로서의 중요한 미덕으로 여겨졌던 것이지요. 몸으로 뛰어서 정보를 캐내지 않고, 다른 기자나 다른 누군가가 생산한 정보를 가공해서 만들어낸 2차 보도를 낮추어 보는 것에는 아마 그런 이유도 있었을 것입니다.

하지만, 우리의 막연한 상상 속에서 데이터 저널리스트는 어떻게 보도를 하나요? 이들은 사무실에서 앉아서 글자와 숫자만 가득한 여러개의 모니터 앞에서 그래프를 만들어 내거나, 커다란 화이트 보드 앞에서 멋들어진 분석을 해내기 위한 토론을 하고 있지 않은가요? 사실 그것은 우리가 데이터 사이언티스트들에게 가지고 있는 스테레오타입 입니다. 그리고 이러한 상상은 다소 과장된 것이기는 하지만, 어느 정도는 사실입니다. 데이터 저널리즘이 막연하지만 무언가 대단히 다르게 느껴지는 이유는 정보를 얻는 방식도, 보도할 내용을 만들어 내는 방식도 전통적인 저널리즘의 그것보다는 데이터 사이언스의 그것에 대단히 가깝기 때문일 것입니다. 그리고 전통적인 저널리즘의 시각에서 보기에 이러한 방식은 '저널리즘' 그 자체와 배치되는 것처럼 보일 수도 있고, 윤리적으로 옳지 않게 보일 수도 있고, 또 지나치게 '한가해' 보일 수도 있겠죠. 전통적인 조직에서 저널리즘이 잘 받아들여지지 않는 이유이기도 합니다.

위에서, 데이터 저널리즘이 '정보를 얻는 방식'과 '기사를 생산하는 방식'에서 데이터 사이언스를 차용한다고 말했습니다. 사실 여러분들이 이

교재에서 배우게 될 핵심이죠. 그러면 데이터 사이언스에서 이 두 가지는 전통적인 보도와 어떻게 다를까요? 첫째, 데이터 사이언스는 이미 대중에게 공개되어 있는 정보를 얻습니다. 이것은 기자이기에 접할 수 있었던 정보를 수집하는 전통적인 보도 행위와 다르지요. 따라서, 데이터 저널리스트를 '게이트키퍼'라고 부르기는 쉽지 않습니다. 하지만, 이미 공개되어있는 데이터를 이용한다고 그 데이터를 수집하는 것이 마냥 쉬운 것은 아닙니다. 어떤 경우에는 데이터가 분석하기 어려운 형태나 위치에 있거나, 너무 크거나(빅데이터), 또는 이 데이터와 저 데이터가 함께 연결되었을 때 그 전에는 알지 못했던 함의를 도출할 수 있는 경우도 있습니다. 따라서 필요한 정보를 '찾아서', '분석할 수 있는 형태로 저장/가공하고', '여러 정보를 잇는' 작업이 필요합니다. 데이터 사이언스에서는 이를 information retrieval이라고 부릅니다. Information retrieval의 일반적인 번역어는 '정보검색'인데요, 사실 이는 좋은 번역이라고는 할 수 없습니다. 물론 정보검색 하면 생각나는 검색창과도 관련이 있지만, 그것을 위해 필요한 웹스크레이핑, 크롤링, 인덱싱, 데이터베이싱, 추천 알고리즘 등을 모두 포괄하는 개념이기 때문이지요. 자세한 이야기는 나중에 하기로 하고, 일단 information retrieval이 전통적인 보도에서는 취재 과정에 해당한다고 볼 수 있고, 이것이 발로 뛰는 정보 수집은 아니지만, 그 나름대로 여러 난관과 그것들을 뚫어내기 위한 기술과 노력을 필요로 한다는 것 정도를 이해하면 될 것 같습니다.

둘째, 데이터 사이언스에서 전달할 내용을 생산하는 방식은 '시각화'와 '데이터스토리텔링'에 의존합니다. 이는 '글쓰기'가 중요하지 않다는 것을 의미하지는 않습니다. 대부분의 데이터 저널리즘 기반 보도에는 글과 스토리가 있어야만 합니다. 다만, 글의 핵심은 데이터 또는 데이터 분석의 의미를 잘 전달하는데 있다는 것, 그리고 그것을 달성하는데 필요한 것 이상의 텍스트는 최대한 생략하는 것이 좋다는 것이 '데이터스토리텔링'에서 대단히 중요합니다. 데이터스토리텔링이 그러한 간결성을 추구하는 이유는 데이터 분석을 이용해 전달하려고 하는 정보가 이미 상당히 복잡하기 때문입니다. 여러분들도 기자이면서 독자이기에 느끼시겠지만, 독자는, 혹은 인간은 정말이지 한정된 집중력을 가지고 있습니다. 기사를 보겠다고 클릭해 놓았으면서도 마치 조그만 틈만 보이면 도망가기 위해 최선을 다하는 것처럼 보일 지경이라, 짧은 기사 조차 처음부터 끝까지 보도록 만드는 것이 대단히 어렵다는 것은 누구나 공감하실 것입니

다. 그런 독자들에게 빅데이터를 분석해서 쏟아낸다면 어떨까요? 그 기사를 처음부터 끝까지 만드는 것은 불가능하게 보이기도 합니다. 데이터 사이언스의 분석 결과는 그 결과조차 복잡한 경우가 허다하거든요. 그렇다면, 글쓰기가 기존의 기사 쓰기와 같아져서는 안 됩니다. 애써 크고 복잡한 데이터로 훌륭한 분석을 도출해 내었다면, 그 분석 결과를 온전히 전달하는데 초점이 맞춰져야 하는 것이지요. 그리고, 글쓰기로 해결되지 않는 부분을 해결해 주는 것이 바로 '그림' 입니다. 데이터 시각화란 언제든지 달아나려는 준비가 된 예민한 독자들을 달래고 붙잡아 효과적으로 분석 결과를 전달하기 위한 그림 입니다. 따라서, 데이터 시각화에도 기술이 필요합니다. 물론 더 좋은 것은 간명한 글과 시각화가 잘 결합되어 복잡한 내용을 쉽게 전달하는 것입니다. 이를 '데이터 스토리텔링'이라고 하는 것이고요.

따라서, 데이터 저널리즘은 데이터 사이언스를 이용한 저널리즘이다, 라는 정의를 조금 더 구체화하자면, 기존의 취재와 기사 쓰기를 정보검색(더 정확하게는 information retrieval)과 데이터 스토리텔링으로 대체한 저널리즘이라고 보아도 좋습니다. 다만, 이러한 저널리즘이 우리가 '저널리즘'이라는 직종의 가치에 적합한 것인가를 따질 수는 있습니다. 여기서는 그런 복잡한 논의는 건너뛰기로 하지요. 아무튼 그러한 이유로 이 교재에서는 정보검색과 데이터 스토리텔링에 필요한 다양한 개념과 기술들을 배우게 될 것입니다.

1.3 데이터 저널리즘은 왜 필요한가

앞서 설명한 데이터 저널리즘이 갖추고 있는 요소들과 '데이터 사이언스를 활용한 저널리즘'이라는 일종의 정의가 데이터 저널리즘이 무엇인지를 이해하는 데에는 어느 정도 도움을 주지만, 아주 만족스러운 것은 아닙니다. 왜냐하면 그런 설명들은 데이터 저널리즘을 방법의 관점에서 바라보고 있기 때문에, 그로부터 어떤 데이터 저널리즘이 '좋은' 데이터 저널리즘인지 이해하는 데에는 한계가 있기 때문이지요. 물론, 이 역시 정답이 있는 문제는 아니지만, 그것을 판단하기 위해서는 이제 데이터 저널리즘이 사회적 필요성에 대해서 조금 이야기해 볼 필요가 있습니다.

1 데이터 저널리즘이란

왜 '좋은' 데이터 저널리즘이 무엇인지 판단하기 위해서 그것의 '필요성'에 대해 이야기 하는 것일까요? 만약 데이터 저널리즘이라는 것이 사회적으로 그다지 필요하지 않은 일정의 기술적 유희에 불과하다면, 좋은 데이터 저널리즘이란 것 자체가 형용 모순이기 때문입니다. 이미 기존의 저널리즘으로 충분하다면, 기자들이 데이터 사이언스를 배워야 한다는 구호는 한낱 유행이거나, 심지어 한정된 자원을 엉뚱한 곳에 쓰게 만드는 사회악에 가까울 것입니다. 만약, 그렇지 않고 데이터 사이언스가 세상의 변화로 인해 발생한 어떤 구멍을 메꾸기 위해 '필요해진' 것이라면, 우리는 그러한 사회적 필요성에 적합한 데이터 저널리즘을 '좋은' 데이터 저널리즘이라고 부를 수도 있겠지요.

데이터 저널리즘의 필요성으로 두 가지를 들 수 있을 것 같습니다. 첫째, 현대의 독자들은 '정보의 부족'이 아니라 '정보의 과잉' 때문에 양질의 정보를 소비하지 못하고 있다는 문제입니다. 전통적인 저널리즘은 시민들의 '정보의 부족' 문제를 해결하기 위해 시민사회와 시장이 만들어낸 일종의 솔루션이었다고 볼 수 있습니다. 시민들은 접할 수 없지만, 나름의 조직과 자본을 갖추었을 때 접할 수 있는 정보, 즉, 기자만 접할 수 있는 정보가 있었던 것이지요. 따라서, 기자들은 그러한 정보들 중 독자들이 '읽을만 하다'라고 여기는 정보를 전달해 주는 '게이트키퍼' 역할을 했던 것입니다. 물론, 이러한 상황이 지금은 사라졌다고 볼 수는 없겠지마는, 최근의 독자들은 기자들이 접할 수 있는 정보에 똑같이 접할 수 있는 경우가 많습니다. 심지어 개인 유튜버들이 기자들보다 더 빨리 정보를 캐내고 기자가 이를 후속 보도하는 경우도 많지요. 하지만, 독자 입장에서 더 큰 문제는 쏟아지는 정보 중에서 어떤 정보가 의미있는 것인지 알기 어렵다는데 있습니다. 다들 먹고 사느라 바쁜데, 그 많은 정보들이 '접근가능'하다고 해서 개개인에게 '처리가능'한 것은 아니지요. 데이터 저널리즘이 필요한 것은 바로 이 지점 입니다. 데이터 저널리즘은 이미 접근 가능한 수많은 정보들 중 의미 있는 것들을 찾아내고, 연결해서, 숨겨져 있는 연관을 찾아내고, 그로부터 시민들이 주목해야 할 통찰을 끌어내는데 특화되어 있다는 점에서, 전통적 저널리즘과 다른 사회적 효용을 갖는 것입니다. 즉, 데이터 저널리즘은 지난 밤에 어떤 일이 일어났는지를 보도했을 때 좋은 저널리즘이 되는 것이 아니라, 주어진 정보들이 어떤 의미를 갖는 것인지를 발견하고 이를 효과적으로 전달할 때 좋은 저널리즘이 되는 것이라고 할 수 있습니다.

두 번째 데이터 저널리즘의 필요성은 과학 커뮤니케이션의 중요성이라고 할 수 있습니다. 따지고 보면, 세상이 참 많이 변해서 정치권에서 보수와 진보가 싸우는 주제가 참 많이 바뀌었습니다. 예전에는 노동정책, 복지정책, 큰 정부와 작은 정부 이런 것들이 보수와 진보를 나누는 키워드였죠. 물론 이런 주제는 여전히 중요합니다만, 최근에는 예전에는 과학만의 영역이라고 여겨졌던 주제들이 정치권에서 매우 중요한 논쟁 거리가 됩니다. 예컨대, 기후변화, 핵에너지, 환경 오염, 코로나19, 백신, 이런 것들이죠. 과학적 사실에는 옳고 그름이 있어 정쟁의 대상이 되지 않을 것 같지만, 사실은 그 반대입니다. 많은 사람들이 과학은 옳고 그름의 문제라고 생각하기 때문에, 자신이 믿는 '옳은' 과학적 사실에 반대하는 상대방은 단순히 의견을 달리 하는 사람이라기 보다는 '바보'라고 믿기 때문이지요. 의견이 다르면 논쟁을 하면 되겠지만, 많은 사람들이 '바보'와 논쟁하려 하지 않습니다. 단지 배척할 뿐이지요. 이렇게 과학은 정쟁은 주요 대상으로 떠올랐습니다. 하지만, 문제는 과학적 '사실' 또는 '발견'을 대중들에게 전달하는 것은 생각보다 어렵습니다. 백신의 효과에 대한 논쟁을 떠올려 보세요. 많은 사람들은 '백신을 맞은 사람들 중에 코로나에 걸릴 확률'과 '코로나에 걸린 사람들 중 백신을 맞은 사람들의 비율'의 차이를 잘 이해하지 못합니다. 이는 그것을 이해하지 못하는 사람들의 잘못이 아니라, 원래 인간의 두뇌는 과학을 이해하기 위한 그런 중요한 차이를 이해하는데 취약합니다. 이는 노벨 경제학상을 받기도 한 심리학자 다니엘 카네만이 밝혀낸 사실이기도 하지요. 물론, 가만히 앉아서 설명을 듣고 5분, 10분 고민하면 누구나 이해할 수 있습니다. 하지만, 백신의 효과를 설명하는 여러분의 기사를 읽는 독자들 중에 몇이나 그런 시간과 공을 들여줄까요? 앞서 설명한 시각화, 데이터 스토리텔링 방법들은 복잡한 과학적 사실을 가능한 적은 자원을 이용하면서 이해할 수 있는 방식으로 독자들에게 전달하는 것을 목적으로 하고 있습니다. 앞서 본 파이낸셜 타이즘의 백신 보도가 그런 예이지요. 그러니, 좋은 데이터 저널리즘은 단지 멋진 시각화를 이용한 보도라기 보다는 그런 수단을 이용해 복잡한 사실을 쉽게 이해할 수 있는 방식으로 전달하는 것이라고 할 수 있겠습니다.

버스 -> 사모펀드 (데이터 저널리즘?)

하지만, 가치는 저널리즘.

1.4 데이터 저널리즘의 전략

1. 새로움
2. 예외성
3. 전형성
4. 추세
5. 예측
6. 폭로/반박

https://fivethirtyeight.com/features/lionel-messi-is-impossible/

https://fivethirtyeight.com/features/ferguson-missouri/

https:// fivethirtyeight.com/ features/ you-cant-trust-what-you-read-about-nutrition/

https:// blog.devgenius.io/ create-an-expected-goals-model-for-any-league-in-minutes-in-python-972b2423dc4b

그림 1.6: 새로움 1


검색위치 대구광역시 동구
우리동네 의원님
맛집 리스트
대구광역시 동구의회 의장님들이 2년 간 쓴 업무추진비는 총 1억 2,698만 8,550원. 전국에서 100번째로 많이 썼네요? 참고로 전국 226개 기초의회 의장단 업무추진비 평균 사용 금액은 7,000만 원이에요. 그 중 평균 93%가 식비로 지출된다는데 동구의회 의장님들이 가장 많이 찾은 동네 맛집을 공개합니다!
대구광역시 동구 의원님 픽
Top5 맛집
1 아양교식육식당 6,788,000원
2 뜰누리 4,028,400원
3 선돌 3,613,000원
4 삼천포생아구찜 2,763,000원
5 공산한우식육식당 2,704,740원

그림 1.7: 새로움 1

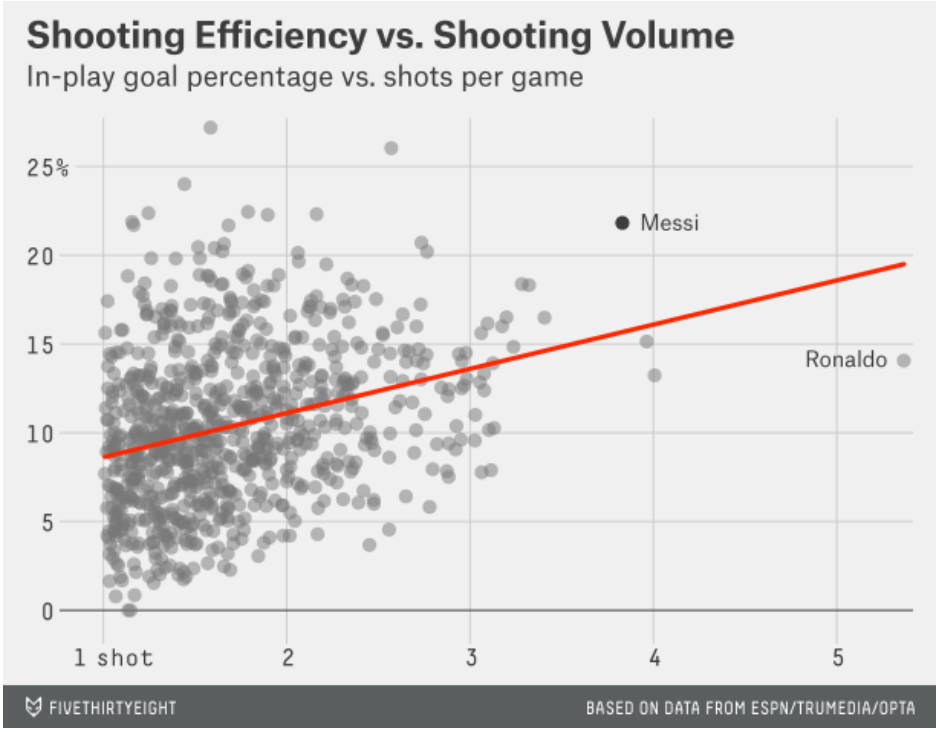

Shooting Efficiency vs. Shooting Volume
In-play goal percentage vs. shots per game
25%
20
15
10
5
0
1 shot
2
3
4
5
Messi
Ronaldo
FIVETHIRTYEIGHT
BASED ON DATA FROM ESPN/TRUMEDIA/OPTA

그림 1.8: 예외성

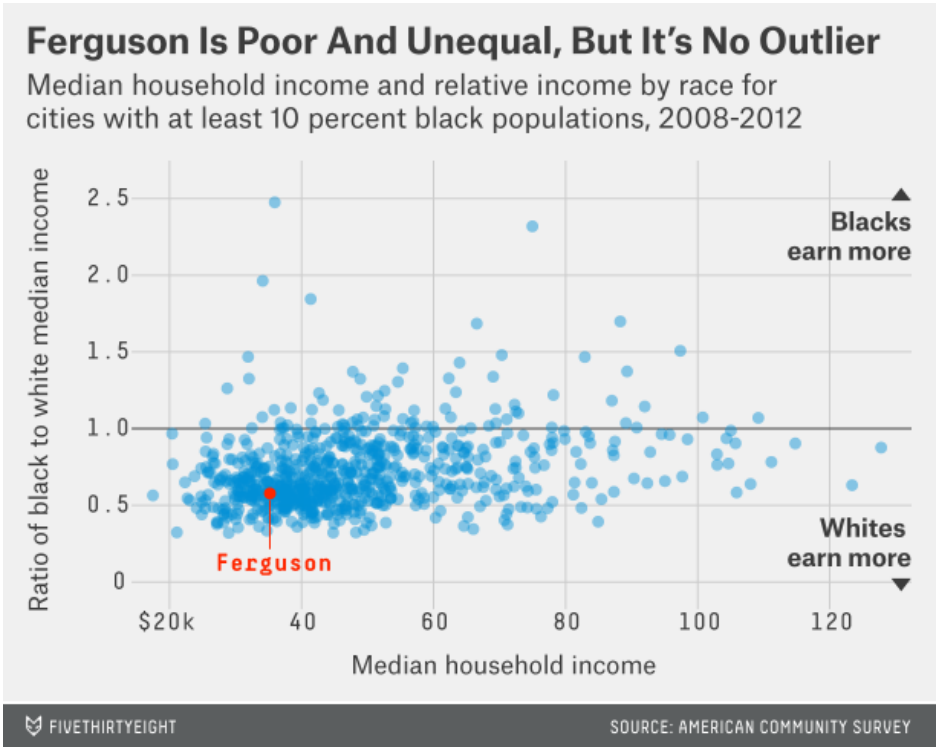

Ferguson Is Poor And Unequal, But It's No Outlier
Median household income and relative income by race for cities with at least 10 percent black populations, 2008-2012
Ratio of black to white median income
2.5
2.0
1.5
1.0
0.5
0
Blacks earn more
Whites earn more
Ferguson
$20k
40
60
80
100
120
Median household income
FIVETHIRTYEIGHT
SOURCE: AMERICAN COMMUNITY SURVEY

그림 1.9: 전형성

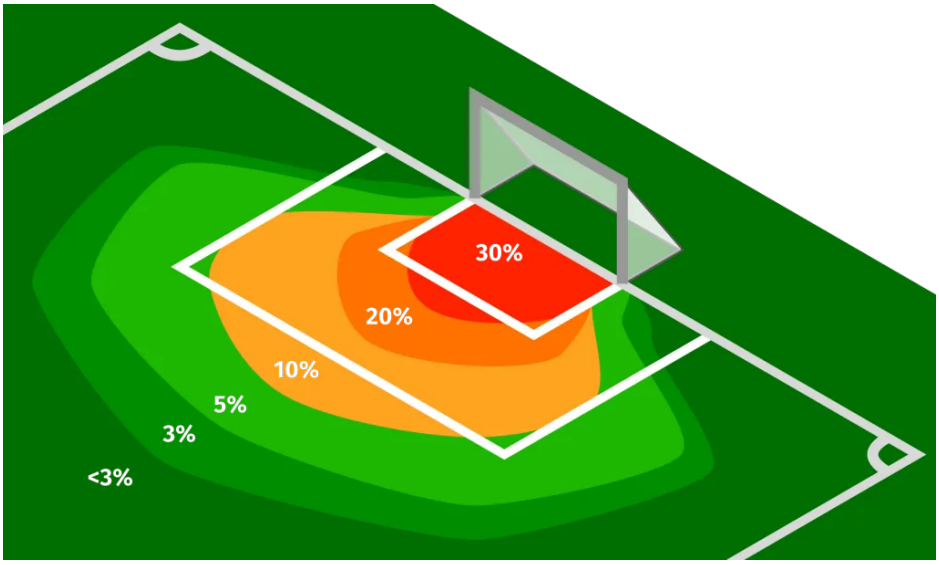


그림 1.10: 예측


It's hard to remember what you ate last July
Calories eaten per day according to a food diary vs. the food frequency questionnaire
1,000 cal.
1,500
2,000
2,500
3,000
Food diary
FFQ
Difference
Christie
Walt
Anna
FIVETHIRTYEIGHT

그림 1.11: 폭로/반박

2 보도를 위한 데이터과학 기초

작성 예정

이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.

이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.

이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.
이 페이지는 목차 생성을 위해 임시로 만들어졌습니다.

3 R과 Rstudio 인스톨

여기서는 R과 Rstudio를 설명하고 기본적인 용어법, 개념을 설명합니다. 다소 지루할 수 있는 내용이지만,꼭 알아야 하니 정독해 주세요!

R은 설치도 쉽고, 간단한 이용환경을 가지고 있습니다. 프로그래밍에서 이는 통합개발환경(IDE)라고 하는데, 이는 Posithttps://posit.co/이라는 회사가 개발 및 배포하고 있습니다.

R과 Rstudio의 관계에 대한 그림

3.1 R의 역사와 이용.

R은 통계학자들이 만든 언어입니다. 원래는 S라는 유명한 Bell Lab에서 만든 상용 언어였다가, 이를 Ross Ihaka와 Robert Gentleman이라는 두 명의 뉴질랜드 교수가 뉴질랜드 오클랜드 대학에서 통계학 수업 시간에 이용할 언어로 다시 implement한 버전입니다. Implement라는 말이 이상하게 들릴 수도 있는데, 내부가 어떻게 작동하는지와는 상관 없이 같은 문법으로 작동할 수 있도록 만들었다고 생각하면 됩니다. 즉, 자동차의 부속품은 완전히 다르지만, 운전하는 방식은 완전하게 같은 두 대의 자동차를 생각하면 되겠습니다. 내부 작동 방식은 소스 코드, 이용자의 운전 방식은 implemetation이라고 간단히 생각하면 되겠습니다.

이 두 교수는 S와는 달리 이 내부가 어떻게 작동하는지 그 설계까지 공개해 버렸습니다. 이것을 보통 컴퓨터 공학자들은 '오픈소스로 풀었다'고 표현합니다. 더 나아가 이 설계를 조금 변경해서 자기만의 R을 만들거나 그것을 파는 것도 이론적으로 가능합니다.

R이 '통계학자들의 보편 언어'라고 하는 데에는 여러가지 뜻이 숨어있습니다. 먼저 통계학자들이 만든 언어라는 것은 중요합니다. 통계학자들은 수학을 그들의 언어로 사용하고 이는 중고등 교육을 받은 우리의 직관과 유사하기 때문에, 통계학자들의 머리에서 나온 이 언어는 우리의 직관에서 크게 벗어나지 않습니다. 당장, 우리가 계산기 두드리듯 식을 넣고 엔터를 치면 동작합니다.

두번째로 '보편언어'라는 말도 중요합니다. 어지간한 것은 다 된다고 봐도 무방합니다.

3.2 R 설치

이렇게 하면 설치가 완료됩니다.

3.3 Rstudio 설치

3.4 Rstudio 환경 설명

Text, Qmd, 콘솔

3.5 변수, 함수

서영씨에게 그림을 좀 그려달라고 해야겠다. 명사 동사 값 화살표 등등에 대한 그림.

R의 작동 방식을 이해하기 위해서는 다음과 같은 것들을 알아야 합니다.

먼저 컴퓨터가 작동하는 방식에 대해서 간단하게만 알아봅시다. 컴퓨터에 데이터를 저장되는 곳은 크게 세 군데가 있습니다.

연산장치(CPU, GPU) - 메모리 - 저장장치(HDD, SSD)

이는 '악마의 두뇌'를 가졌다고 일컬어지기도 하는 저 유명한 수학자 존 폰 노이만(John von Neumann; 1903-1957)이 고안해 낸 컴퓨터의 구조입니다. 여기에 대해 알고 있는 것은 앞으로 불필요한 에러를 피하는데 큰 도움이 됩니다.

아직 소개하지 않았지만, 여러분이 사용할 데이터들은 HDD, SSD와 같은 저장 장치에 살고 있습니다. 파일을 저장한다는 행위는 여러분들도 익숙하실 것이라고 믿습니다. 여러분들이 데이터를 이용해서 어떤 계산을 하고 싶다면, 그 계산은 '연산장치'에서 일어납니다. CPU도 아마 많이 들어보셨을테고, 최근 인공신경망(ANN)이나, 암호화폐 채굴이 CPU 대신 GPU라는 연산장치 위에서 돌아간다는 이야기도 들어보셨을 것입니다. CPU건, GPU건, 컴퓨팅이라는 것은 결국 그들이 어떤 계산을 해 주기를 바라는 것이지요.

- 메모리 위에 쓰여진 값은 휘발한다 (전원이 꺼지면 없어진다)
- 메모리 위에 쓰여진 값은 이름이 없으면 없는 것이나 다름 없다

문제는 저장장치에 있는 데이터를 연산장치가 바로 사용하는 것이 아니라는 것입니다. 연산장치가 계산을 하기 위해서는 중간 단계에 해당하는 메모리 위에 값이 기록되어야 합니다. 그리고 아마도 여러분들이 알고 계실 것처럼, 메모리에 기록된 데이터는 휘발합니다. 즉, 컴퓨터 전원이 꺼지면 없어지는 것이죠. 이 때문에 저장장치들이 필요합니다.

그런데, 메모리에 대해서는 그것 말고도 알아야 하는 것이 있습니다. 메모리 위에 기록된 값은 '이름'이 없으면 존재하지 않는 것이나 다름 없다는 사실입니다. 이게 무슨 말인지 알기 위해 다음과 같은 예를 살펴 봅시다.

```
3+ 4
```

```
[1] 7
```

이러한 연산이 이루어지기 위해서는 3이라는 '값'과 4라는 '값'이 메모리에 먼저 기록되어야 합니다. 그 다음에 연산 장치가 메모리에 저장된 두 값을 더하는 '연산'을 해 준 후, 그 결과 값이 7을 메모리 위에 기록합니다. 우리는 메모리 위에 기록된 결과를 모니터를 통해 보는 셈입니다. 그런데, 문제는 3, 4, 7이라는 모든 값에 별도의 이름을 붙이지 않았다는 것이지요. 앞서 말했듯이 '이름'이 없는 값은 존재하지 않는 것이나 다름 없습니다. 따라서, 우리는 메모리에 쓰여진 이 값들을 다시 사용할 수 없습니다. 모니터에서 한 번 확인하고 날려보낸 것입니다.

이는 반대로 이야기 하면, 이름을 부여한다면, 적어도 이름을 부여하는 프로그램, 즉 우리의 경우 R을 켜 놓은 동안은 재사용할 수 있다는 것입니다. 위의 프로그램을 다시 써 보죠.

```
a <- 3
b <- 4
c <- a + b
print(c)
```

```
[1] 7
```

모니터에서 확인하는 결과는 같습니다. 그러나 우리는 몇 가지 작업을 더 했는데요, 3과 4를 각각 a와 b라는 이름에 할당(assign) 했습니다. 이름을 부여한 것이지요. 또 a에 해당하는 값과 b에 해당하는 값을 더한 결과 역시 c라는 이름에 할당했습니다. 이제 이 값들은 이름이 있으니 R을 켜 둔 동안은 다시 불러 사용할 수 있는 것입니다.

```
a * b
```

```
[1] 12
```

이제 앞으로 사용할 몇 가지 용어를 정의하도록 하죠. - 값(value): 데이터 그 자체 - 변수(variable): 거기에 붙은 이름 - 할당(assign): 값을 변

수로 만드는 행위. R에서는 <- 부호를 사용함. (사실은 =을 사용할 수도 있지만, 구분하겠습니다.)

그림

사람의 언어에 비유해서 설명하자면, 값과 변수는 명사, 또는 목적어에 해당합니다. 많은 경우에는 이렇게 주어진 대상에 어떤 행위를 하고 싶어 하지요. 우리는 그것을 연산이라고 합니다. 그리고 그러한 연산을 문법으로 표현한 것을 함수(function)라고 하지요. 인간의 언어에서라면 함수는 동사에 해당합니다.

그림

우리는 위의 예에서 이미 함수를 보았습니다. print()가 그것입니다. 이것은 주어진 값을 콘솔에 출력하는 동작을 의미하는 것이니 동사라고 할 수 있습니다. 그러면 동사와 명사를 구분하듯, R에서 변수와 함수를 구분할 수 있을까요? 구분할 수 있습니다. 여러 방법이 있지만, 가장 간단한 방법은 문자열 뒤에 괄호가 있는지 보는 것입니다. 변수는 괄호가 없습니다. 하지만, 함수는 print()처럼 괄호가 있지요.

괄호는 왜 있는 것일까요? 괄호 안에 무언가를 써넣어야 하기 때문입니다. 즉, 함수가 표현하는 행위의 대상이 되는 목적어를 집어 넣어야 하기 때문이죠. 예컨대 print(c)에서 c라는 변수는 바로 그 목적어에 해당하요. 앞서 값이나 변수가 목적어의 역할을 하게 될 것이라고 했던 것을 기억할 것입니다. print(c)를 사람의 언어로 표현하면 다음과 같습니다.

> c를 콘솔에 print()하라.

그림 (변수, 함수)

이렇게 값이나 변수가 함수가 하는 연산의 대상이 되면, 즉, 함수의 괄호 안에 들어가면, 이를 입력값(input), 또는 인수(argument)라고 합니다. 입력값을 받았으니 함수는 연산의 결과로 출력값(output)을 내어놓겠지요. 사실 더 정확한 표현은 다음과 같습니다.

```
print(x=c)
```

```
[1] 7
```

여기서 x는 print()라는 함수의 매개변수(parameter), c는 그 매개변수에 집어넣은 인수, 또는 입력값입니다. 매개변수는 print()라는 함수가 그 내부에서 사용하는 변수입니다. print()라는 함수는 내부에서 x라는 변수를 이용해 행위를 하지, c라는 함수 밖에 존재하는 변수에 대해서는 알지 못합니다. 그런데 x=c라고 하는 순간, 이용자는 함수에게 "네가 이용할 x가 바로 바로 c라는 변수에 들어있는 값이야"라고 연결해 주는 것입니다.

그림 (x=c)

그러면 print()는 자신이 이용하는 x라는 그릇에 c에 이미 연결되어 있는 값을 담고 그것을 출력하는 행위를 하게 됩니다.

그런데 왜 첫번째 예에서는 x=을 생략하고 print(c)라고 했는데도 작동했을까요? 그것은 print() 함수 자체가 첫번째로 입력한 숫자를 x에 대응하는 인수로 자동으로 인식하도록 프로그램 되어 있기 때문입니다. 무언가를 출력하라고 하는 명령은 수없이 사용하게 될텐대, 매번 매개변수를 반복행 하면 너무 귀찮겠지요.

그런데, 모든 함수가 인수를 필요로 하는 것은 아닙니다. 자연언어에서 동사 역시 자동사와 타동사로 구분되는 것과 같은 이치입니다. 예컨대 다음과 같은 함수는 인수 없이 실행 됩니다.

```
getwd()
```

getwd()라는 함수가 working directory 즉, 현재 R이 작업을 하고 있는 컴퓨터의 경로를 표시하는 동작을 의미하니, 인수가 따로 필요하지는 않겠지요. 어떤 경우에는 함수가 인수를 필요로 하지만 (즉, 타동사 이지만), 인수를 쓰지 않아도 작동하는 경우가 있습니다. 그런 경우에는 함수가 자신이 필요로 하는 인수에 대해 디폴트 값을 가지고 있는 경우 입니다. 즉, 사용자가 아무 인수도 주지 않으면 자동으로 인수로 가정하는 값이 있는 경우도 있다는 것입니다. 우리가 "밥 먹어"하는 대신 상대방이 알아들을만 한 상황에서는 "먹어"라고 하는 것과 비슷한 이치입니다.

3.6 경로

앞서 작업폴더(working directory) 이야기를 했는데요, 이것은 무슨 의미일까요? 작업폴더는 현재 R이 자신이 위치하고 있다고 생각하는 저장장치 안의 폴더를 의미합니다. 보통은 지금 작성하고 있는 코드가 저장되어 있는 곳이 작업폴더가 되지만, 이는 경우에 따라 다릅니다. 그래서 앞에서처럼 getwd()를 이용해 현재 작업 폴더가 어디인지 확인하는 것이지요.

작업폴더가 중요한 이유는, 저장장치로부터 R이 데이터를 불러올 때, 특별한 이야기가 없으면 작업폴더로부터 불러와야 한다고 생각하기 때문입니다. 하지만, R이 현재 작업폴더로 이용하고 있는 장소와 이용자가 작업폴더라고 믿고 있는 장소 사이에 차이가 있는 경우가 종종 있습니다. 프로그래밍을 처음 해 보시는 분들이 처음에 제일 많이 겪는 에러의 원인입니다. 만약 이런 차이가 발생한다면, 해법은 세 가지가 있습니다. 첫째, R에게 작업폴더를 바꿀 것을 지시하면 됩니다.

```
setwd()
```

둘째, 불러올 데이터가 저장되어 있는 곳을 자세하게 알려주면 됩니다. 나중에 더 자세히 설명하겠지만, 다음과 같은 명령어를 볼까요?

```
read.csv("")
```

이는 저 파일(csv라는 확장자를 처음 보신 분들은 나중에 배우게 될테니, 일단 엑셀 파일보다 간단한 형태의 테이블 데이터 파일이라고 생각해 주세요)이 있을 경우에 파일을 불러들여서 콘솔에 출력할 것입니다 (quiz: 그렇다면 이 데이터는 우리가 앞으로 분석에 사용할 수 있는 것일까요? 아까 이름과 메모리에 대해 이야기 했던 것을 떠올려 보세요).

하지만, 해당 작업폴더에 저장되어있지 않은 데이터 파일을 불러오려고 하면, R은 에러를 뱉어낼 것입니다.

```
read.csv("")
```

이 때, 이렇게 써 보세요.

```
read.csv("/")
```

이것이 작동한 이유는 현재 작업폴더 아래에 있는 폴더 안에 파일이 저장되어 있기 때문입니다. 이렇게 파일로 가는 길을 경로(path)라고 표현하고, 특히 위의 예에서처럼 현재 작업폴더를 기준으로 경로를 설정해 주는 것을 상대경로라고 표현합니다. 상대경로가 있으면, 절대경로가 있겠지요? 다음과 같은 경우를 절대경로라고 합니다.

```
read.csv("C:/")
```

즉, 가장 높은 폴더, 흔히 루트 디렉토리라고 부르는 장소에서부터 원하는 폴더까지 이동하는 경로를 완전히 알려주는 방식이지요.

사실 가장 권장하는 방식은 세번째 방식입니다. 프로젝트라고 부르는 이 방식은 R의 기능을 활용하는 것이라기 보다는 Rstudio의 기능을 활용한 방식인데요, 가장 에러를 효과적으로 최소화하는 방식이라고 할 수 있습니다.

3.7 프로젝트

Rstudio에서 프로젝트라고 부르는 것의 개념은 정말 단순합니다. 프로젝트는 특정한 작업을 할 때는 항상 사용하기로 미리 정해놓은 폴더 그 이상도 이하도 아닙니다. 예컨대, 지금 저는 교과서를 쓰기 위해 Rstudio를 사용하고 있는데요, 이 교과서를 쓸 때는 Book이라는 프로젝트를 만들고, 해당 폴더 안에 책의 원고, 코드, 데이터 등을 모두 저장해 둡니다. 교과서를 쓰기 위해 필요한 모든 자료를 하나의 폴더 안에 모두 저장해 두고, Rstudio에게 “난 책을 쓸 때는 Book 폴더만 사용할거야”, 라고 미리

말해두기만 하면, Rstudio에게 “나 이제부터 교과서 쓴다”라고 알려주면 Rstudio는 알아서 작업경로를 바꿔줍니다.

물론 저는 이 교과서를 쓸 때 말고도, 다양한 이유로 Rstudio를 사용합니다. 예컨대 연구를 위해 Research라는 프로젝트를 만들어 두었다고 해 보죠. 그러면 Rstudio에게 “나 이제부터 연구한다” 라고 말해주면 Rstudio는 Research 폴더로 작업 경로를 바꿔줍니다. 그러면 교과서를 쓰다가 연구를 하다가 반복한다고 하더라도 작업경로 때문에 골치아픈 에러를 겪을 일이 없어지겠지요. 여러 개의 기획 기사를 동시에 쓰는 상황이라면, 기사1, 기사2, 기사3에 대한 프로젝트를 따로 만들어 두고 해당 기사를 작성할 때는 해당 프로젝트 안에서 활동하면 헷갈릴 일이 없습니다.

이렇게만 이야기하면, 프로젝트가 큰 쓸모가 없어보이지만, 동시에 하는 작업의 종류가 늘어날 수록 경로로 인해 발생하는 에러의 빈도는 그야말로 '기하급수'적으로 늘어납니다. Rstudio의 기능 중에 프로젝트 보다 에러를 줄여주는 기능은 없다고 감히 단언할 수 있으니, R을 처음 배우는 지금부터 꼭 프로젝트를 사용하는 버릇을 들이길 바랍니다.

그러면, 프로젝트를 어떻게 만들고, 어떻게 프로젝트 사이를 이동하는지를 살펴보겠습니다.

3.7.1 새 프로젝트 만들기

새 프로젝트를 만드는 방법은 간단합니다. Rstudio의 File 메뉴를 선택해보세요. 그러면 드롭다운 메뉴에서 New Proejct라는 기능을 찾을 수 있습니다. 이를 누르면, 다음과 같은 화면이 등장할 것입니다.

캡처

여기서 프로젝트 명은 새로 만들어질 폴더의 이름이고, 경로는 그 폴더가 만들어질 상위 폴더라고 생각하시면 됩니다. 예컨대 위의 예에서, 저는 Spring2023 폴더 아래 Book이라는 새로운 폴더를 만들어 바로 그 폴더를 프로젝트 폴더로 사용하려는 것이지요.

OK를 누르고 나면, 몇 초에 걸쳐 프로젝터(=폴더)가 만들어집니다. 프로젝트가 폴더에 불과하다는 것을 확인하기 위해서 해당 폴더를 찾아보지요. 저는 지금 윈도우를 이용하고 있기 때문에 탐색기를 이용하겠습니다.

캡처

새로 폴더가 하나 만들어져 있지요? 그 안에는 Book.Rproj라는 파일도 자동으로 만들어져 있는데, 이 파일은 단지 이 폴더가 그냥 폴더가 아니고 R을 이용한 프로젝트를 수행하기 위한 폴더임을 표시하는 것입니다. 이것은 조금 있다가 서로 다른 폴더 사이를 이동할 때 이용할 것입니다. 자, 이제 우리는 이제 책을 쓰기 위해서는 이 폴더만 사용하게 될 것입니다. 새로 작성한 코드도, 사용할 데이터도 이 폴더 안에 모두 저장하는 것이지요.

3.7.2 프로젝트 바꾸기

그런데 이미 복수의 프로젝트를 가지고 있었다면, 프로젝트 사이는 어떻게 왔다갔다 할 수 있을까요? 간단합니다. 다시 File 메뉴를 선택해 보세요. 그러면 Open Project라는 기능이 있을 것입니다. 이것을 선택하면, 일반 탐색기처럼 폴더를 선택할 수 있습니다. 만약 Research라는 프로젝트로 이동하고 싶다고 해당 프로젝트 폴더를 찾아가면 됩니다. Research 프로젝트가 이미 만들어져 있었다면 해당 폴더에는 Research.Rproj라는 파일이 저장되어 있겠지요. 그 파일을 선택한 후, OK를 누르면 이제 Rstudio는 Research 프로젝트로 이동합니다. 이제 저는 연구를 하는 것입니다.

캡처

그런데, 사실 이렇게 복잡한 프로젝트 사이를 이동하는 경우는 많지 않습니다. 더 간단한 방법이 있거든요. 아까 이용한 Open Project 메뉴 아래에는 Recent Projects라는 메뉴도 있습니다. 그 위에 마우스 포인트를 올리면 최근에 이용한 프로젝트의 목록이 나타납니다. 그 목록 중 내가 이동하고 싶은 프로젝트를 클릭하면, 아주 간단하게 프로젝트 사이를 이동할 수 있습니다.

이제 여러분이 Rstudio를 켜고 작업을 하려고 할 때 처음으로 해야 할 일은 적절한 프로젝트를 이용하는 것입니다. 새로운 작업을 시작한다면? 프로젝트를 만드세요. 만약 프로젝트를 이미 만들어 놓은 작업을 계속하고 싶다면? 지금 Rstudio가 그 작업에 해당하는 프로젝트를 이용하고 있는지 확인하세요. 만약 그렇지 않다면 Recent Projects나 Open Project를 이용해 해당 프로젝트로 이동하면 됩니다. 항상 작업은 그 이후에 시작합니다.

3.8 그 외 권장사항

프로젝트 말고도 앞으로의 학습을 위해 지키면 좋을 권장사항들이 있습니다. ### 한글 설정 R은 다른 많은 프로그래밍 언어들과 마찬가지로 영어권에서 개발되고 오랫동안 사용되어 왔기 때문에, 한글을 포함한 데이터를 다룰 때 자잘한 문제들을 발생시키게 됩니다. 앞서 프로젝트를 이용해서 제어하기로 한 경로 문제만큼이나 초보자에게 많은 에러를 안겨주는 이유 중 하나가 바로 이 언어의 문제 입니다. 이는 정확한 표현으로는 인코딩 문제라고 하는데, 인코딩은 사람이 읽을 수 있는 자연 언어를 컴퓨터가 이해하는 유일한 정보인 0,1로 이루어진 이진수로 바꾸는 과정을 의미합니다. 문제는 이러한 인코딩 방식이 무수히 많다는 것입니다. 예컨대 예전에 주로 사용하던 인코딩 방식은 알파벳 이외의 문자를 이진수로 바꾸는 룰을 포함하지 않고 있기도 했습니다. 여기서 발생하는 문제를 최소화하기 위해 다음과 같은 설정을 해 줄 것을 권합니다.

```
Sys.setlocale("LC_ALL", "Korean")
```

```
[1] "LC_COLLATE=Korean_Korea.949;LC_CTYPE=Korean_Korea.949;LC_MONETARY=Korean_Kor
```

이렇게 하면 R의 언어 설정이 바뀌면서 한글을 올바르게 출력하게 됩니다. 덤으로 에러 메시지도 한글로 바뀌지요. 그런데 에러 메시지가 한글로 나오는 것은 장점이 되기도, 단점이 되기도 합니다. 왜냐하면, 여러분이 이해할 수 없는 에러가 발생하면, 인터넷에서 해결책을 검색해야 하

는데, 영어로 된 정보의 질이 압도적으로 좋기 때문입니다. 영어 에러 메시지를 이용해 해법을 검색하면 영어 이용자가 작성한 답변을 검색할 수 있지만, 한글 에러 메시지를 이용해 검색하게 되면 이용할 수 있는 정보의 풀이 크게 줄어들게 됩니다. 이 경우, 언어 세팅을 다시 영어로 돌려놓고 싶을 수 있겠지요. 그럴 때는 다음과 같은 명령어를 실행시킵니다.

```
Sys.setlocale("LC_ALL", "C")
```

[1] "C"

또 하나 주의해야 할 것은, 이 혼경설정은 여러분이 코딩을 마치고 Rstudio를 끄는 순간 날아가버린다는 것입니다. 따라서 한글 데이터를 다룰 때에는 코딩을 시작하기 전에 이 세팅을 먼저 해 주는 것이 좋습니다.

3.8.1 숫자 표시 설정

R은 애초에 우리와 같은 문과생들이 이용하라고 만든 언어가 아닙니다. 그래서 우리의 머리로써는 이해하기 힘든 초기 설정을 가지고 있는 경우가 종종 있는데요, 그 중 대표적인 것이 숫자 표시 설정입니다. 많은 분들이 아시겠지만, 공학에서는 숫자를 과학적 기수법 이라는 것을 통해 표시하는 경우가 많습니다.

그림

이는 공학에서 너무 작거나, 너무 큰 수를 다루어야 하는 경우가 많다보니, 숫자를 길게 쓰는 수고를 줄이고자 도입된 것입니다. 그러나 이러한 표기법에 익숙한 우리나, 우리가 작성한 기사를 읽을 독자 중 다수는 이러한 표기법에 익숙하지 못할 것입니다. 따라서 다음과 같은 명령어를 실행시켜 숫자 표시 설정을 바꾸어주는 것이 좋습니다.

```
options(scipen=999)
```

사실 위의 명령어에 들어가는 숫자는 999자리 이상이 아니면 과학적 기수법을 사용하지 말라는 것이니꼭 999이어야 하는 것은 아닙니다. 충분히 큰 수이면 됩니다. 이 설정 역시 Rstudio를 끄는 순간 메모리에서 지워지니, 코딩을 시작하실 때마다 다시 설정해주시는 것이 좋습니다.

3.8.2 에러 메시지 읽기

사실 에러 메시지는 매우 중요한 정보들을 담고 있습니다. 에러 메시지를 얼마나 정성들여 읽느냐에 따라 코딩 실력이 느는 속도가 천차만별로 차이가 나게 되니, 꼭 에러 메시지를 소중하게 읽기를 권합니다.

4 R 문법의 기초

R 문법은 쉽습니다. 그리고 앞서 이미 몇 가지 중요한 개념들, 특히, 값, 변수, 변수 할당, 함수, 인수 등의 개념들을 배웠습니다.

여기서는 R을 적어도 계산기처럼 사용할 수 있도록 해 주는 기초 문법들을 몇 가지 더 배울 것입니다. 사실 우리는 R 문법 그 자체를 깊게 배우지는 않을 것입니다. 왜냐하면 곧 tidyverse라는 일종의 대체문법을 이용할 것이기 때문이죠. tidyverse는 여러가지 이용의 편리함 때문에, R의 원래 문법을 사실상 대체한 표준처럼 되어 버렸기 때문에, R 고유의 문법을 깊이 배우지 않는다고 하더라도 너무 걱정할 필요는 없습니다.

그러나, 아무리 우리가 tidyverse 문법을 사용할 것이라고 하더라도 여전히 R을 사용하는 것은 변함이 없기에 몇 가지 기초 문법은 익숙하게 알고 있어야 합니다. 다행이도, 그 종류가 많지는 않습니다.

4.1 데이터의 타입과 구조

R은 여러가지 종류의 데이터를 다룰 수 있습니다. 하지만 컴퓨터 프로그램은 인간처럼 연산을 하면서 주어진 데이터가 숫자인지, 문자인지 직관적으로 결정할 능력을 가지고 있지 않습니다. 연산이 이루어지기 전에 해당 데이터가 숫자인지, 문자인지, 미리 정해놓아야 합니다. 이렇게 정해놓은 데이터의 종류를 데이터의 타입(Type)이라고 합니다. 그와는 달리, 타입을 가지고 있는 복수의 값을 엮어 놓는 방식도 여러가지가 있습니다. 우리는 그 다양한 방식들을 데이터의 구조(Structure)라고 부를 것입니다. 데이터의 타입과 구조는 매우 많은 종류가 있고, 심지어 이용자가 만

들어낼 수도 있는데, 여기서는 일반적인 데이터 분석 및 시각화를 위해 자주 사용하게 되는 것만 짚고 넘어가려고 합니다.

4.1.1 데이터 타입

- numeric: 1, 1.2, -3.42와 같이, 우리가 일상생활에서 사용하는 실수의 개념에 가깝습니다.
- character: 이는 문자를 의미합니다. “a”, “b”, “3-2”와 같이 따옴표로 둘러싸 문자를 표현합니다. 사람의 눈에는 숫자이더라도 “1”라고 쓰면 R은 이를 문자로 인식합니다. 따라서 "1"-"2"와 같은 명령어를 치면 에러가 발생합니다.
- logical: 이는 TRUE 또는 FALSE 두 개의 값을 갖는 논리 연산을 위한 데이터 타입 입니다. 따옴표를 사용하지 않고, 대문자로만 표기했다는 것에 유의하세요. R은 소문자와 대문자를 구분하기 때문에, True 또는 False라고 쓰는 순간 완전히 다른 의미를 가지게 됩니다. 또 "TRUE", "FALSE"라고 쓴다면, logical 타입이 아닌 character 타입으로 인식됩니다. logical 타입은 다른 언어에서는 Boolean 타입이라고 불리기도 합니다.

그 외에 특별한 데이터가 있습니다. NA는 값이 없음을, NaN은 계산 결과가 숫자로 표현될 수 없었음을 이야기 합니다. 예컨대, 0을을 0으로 나누려고 한다면, 그 결과값은 NaN이 됩니다.

```
0/0
```

```
[1] NaN
```

정수를 뜻하는 integer, 복소수를 뜻하는 complex 타입도 있지만 당장 다룰 일이 별로 없을 것이므로, 건너뛰도록 하겠습니다. 또 하나 중요한 데이터 타입으로는 factor라는 것이 있는데요, 이것은 바로 이해가 어려우니, 나중에 사용할 때 설명하도록 하겠습니다.

4.1.2 데이터 구조

R의 특이한 점 중 하나는 한 개의 값이 따로 존재한다는 개념이 없다는 것입니다. 이상하게 들리겠지만, 일단 이해를 위해 다음과 같은 예를 보도록 하죠.

```
2
```

```
[1] 2
```

콘솔에서 2라고 치고 엔터를 누르면, R은 이 값을 그대로 반복해서 보여주는데요, 그 옆에 [1]이라고 쓰여져 있는 것이 보일 것입니다. 그것은 2가 값들을 모아놓은 집합의 첫번째 요소라는 뜻입니다. 우리에게는 그냥 하나의 숫자 같지만, R은 여러개의 값을 담을 수 있는 그릇이 있는데, 그 안에 들어있는 값이 하필이면 한 개 였고, 그 한 개의 값이 첫번째 요소이니(물론 마지막 요소이기도 합니다) [1]이라고 표시한 것입니다. 이렇게 R은 항상 모든 값이 여러개의 값을 가질 수 있는 그릇에 담겨있다고 생각하고, 그릇이 없는 값 같은 것은 존재하지 않는다고 봅니다. 이렇게 값들을 담을 수 있는 그릇을 데이터 구조(structure)라고 부릅니다.

그릇 그림

4.1.2.1 (1) 벡터(vector)

R이 사용하는 가장 간단한 데이터 구조는 벡터 입니다. 벡터는 같은 데이터 타입을 가진 값들의 순서가 있는 집합이라고 생각하면 좋습니다. 사실 위에서 본 [1] 2라는 출력 값은 “2라는 값 하나만 가지고 있는 벡터”라는 뜻입니다. 이제 여러개의 값이 담겨 있는 벡터를 보겠습니다.

```
vecNum <- c(1,2,3,4,5)
vecNum
```

```
[1] 1 2 3 4 5
```

```
vecChar <- c('a', 'b', 'c')
vecChar
```

```
[1] "a" "b" "c"
```

위에서 보듯 c()라는 함수를 이용해서 여러 개의 값을 만들 수 있습니다. 또 값을 변수에 할당하는데 사용했던 <- 연산자를 이용해 벡터 전체를 하나의 변수에 할당할 수도 있습니다. 위의 예에서 vecNum은 numeric 데이터 타입만을 가지고 있는 벡터이지만, vecChar의 경우에는 character 타입으로만 이루어진 벡터입니다.

4.1.3 (2) 리스트(list)

리스트는 '키(key)'라고 불리는 데이터의 이름과 그에 상응하는 '값(value)' 사이의 연결로 표현되는 데이터 구조 입니다. 이렇게 말하면 조금 난해하지만, 예를 보면 간단합니다.

```
persons <- list(id = c(1, 2, 3),
                gender = c("Male", "Female", "Female"),
                height = c(173, 165, 170))
persons
```

```
$id
[1] 1 2 3

$gender
[1] "Male"   "Female" "Female"
```

```
$height
[1] 173 165 170
```

위의 예에서 c(1,2,3)과 같은 정보는 '값'에 해당하고, id, gender, height과 같은 값들은 이 값들과 연결된 '키'에 해당합니다. 이렇게 리스트를 만들어 놓으면 벡터보다 직관적으로 데이터의 관심있는 일부분을 불러올 수 있습니다. 예컨대 위에서 만든 리스트에 담긴 3명에 대한 정보 중, 성별만을 알고 싶다면 다음과 같이 명령하면 됩니다.

```
persons$gender
```

```
[1] "Male"   "Female" "Female"
```

위의 예에서 바로 알 수 있듯이 $ 연산자는 리스트에서 특정 키에 해당하는 값을 부르기 위한 역할을 합니다. 당장은 리스트 보다 벡터를 자주 사용하게 되겠지만, 곧 리스트가 유용한 경우가 자주 발생하게 됩니다.

4.1.4 (3) 데이터프레임(Data Frame)

데이터 프레임은 우리가 잘 알고 있는 표(table)를 의미합니다. 예컨대 위에서 사용한 리스트의 예를 데이터 프레임으로 만들어 볼까요?

```
df <- data.frame(id = c(1, 2, 3),
                 gender = c("Male", "Female", "Female"),
                 height = c(173, 165, 170))
df
```

```
  id gender height
1  1   Male    173
2  2 Female    165
3  3 Female    170
```

우리에게 익숙한 테이블 모양으로 출력이 된다는 것을 알 수 있습니다. 예컨대 데이터로 엑셀 파일이 있어 이를 R을 이용해 분석하려 한다면, 이렇게 데이터프레임으로 인식시키는 것이 가장 직관적이겠지요. 데이터프레임은 여러분들이 가장 자주 보게 될 데이터 구조 입니다.

4.2 서브세트(Subset)

앞서 리스트에 대해 이야기 하면서 $ 연산자를 통해 데이터의 일부만 보는 작업의 예를 보았습니다. 이렇게 데이터의 일부만 보는 행위를 서브세트라고 합니다. 서브세트를 분석 목적에 맞게 잘 하는 것은 데이터 과학에서 가장 중요하고, 자주 하게 되는 테크닉 입니다. 우리는 여기서 배우는 서브세트 기술 보다 조금 더 명료한 방법을 앞으로 사용하게 되겠지만, 이를 100% 피해갈 수는 없으므로, 가장 기초적인 것만 보도록 하겠습니다.

4.2.1 벡터

일단 벡터의 서브세트를 이해하기 위해 다음과 같은 예를 보겠습니다.

```
a <- c(4, 5, 6, 7, 8)
a[2]
```

```
[1] 5
```

이렇게 벡터를 서브세트 하기 위해서는 간단하게 [] 안에 몇 번째 값을 보고 싶은지를 써 주면 됩니다. 이런 것도 가능합니다.

```
a[2:4]
```

```
[1] 5 6 7
```

여기서 2:4는 “2에서 4까지의 정수”라는 뜻으로 c(2,3,4)라고 쓴 것과 동일한 효과를 갖습니다. 따라서, 두번째, 세번째, 네번째 값이 서브세트 된 것이지요. 또 이런 방식으로 특정 순서에 있는 값만 제외하는 것도 가능합니다.

```
a[-2]
```

```
[1] 4 6 7 8
```

4.2.2 리스트

이번엔 아까 썼던 것과 같은 리스트 예를 사용해 보겠습니다.

```
persons <- list(id = c(1, 2, 3),
                gender = c("Male", "Female", "Female"),
                height = c(173, 165, 170))
```

아까 persons$gender라고 써서 subset하는 방식으로 봤습니다. 똑같은 효과를 갖는 명령어로 다음과 같이 쓸 수 있습니다.

```
persons[["gender"]]
```

```
[1] "Male"   "Female" "Female"
```

대괄호를 두번 써 주었다는 것([[]에 유의해 주세요. 정확하게 같은 결과가 나오지만, 이 때는 키(key)를 따옴표를 써서 문자열로 제시해 주어야 합니다. 사실 persons$gender는 이 코드를 조금 더 간단하게 쓰기 위한 약어라고 생각하시면 되겠습니다. 사실 gender는 persons 리스트에서 두번째 키이기 때문에 다음과 같이 써도 결과는 같습니다.

```
persons[[2]]
```

```
[1] "Male"   "Female" "Female"
```

대괄호를 한번 쓰는 것은 조금 다른 의미를 갖습니다.

```
persons[2]
```

```
$gender
[1] "Male"   "Female" "Female"
```

결과에서 보는 것처럼, persons[[2]]는 persons라는 list의 두번째 키/값의 쌍에서 값만 되돌려달라는 의미입니다. 따라서 되돌려받는 값은 값에 해당하는 벡터, c("Male", "Female", "Female")이 됩니다. 반면, persons[2]는 두 번째 키/값의 쌍 전체를 돌려달라는 돌려달라는 의미이므로, 반환값 역시 리스트인 list(gender = c("Male", "Female", "Female")) 형태가 됩니다. 따라서, persons[2:3]과 같은 표현도 자연스럽게 정의가 되겠지요.

4.2.3 데이터프레임

사실 여러분은 데이터프레임을 서브세트 할 일이 가장 많을 것입니다. 사실 데이터프레임을 서브세트 하는 방식은 리스트와 매우 유사합니다. 아까 만든 df 데이터프레임을 생각해 보면,

```
df$gender
```

```
[1] "Male"   "Female" "Female"
```

는 리스트처럼 gender열의 값에 해당하는 벡터를 돌려주고, 이는 다음과 같습니다.

```
df[['gender']]
```

```
[1] "Male"   "Female" "Female"
```

반면 숫자를 이용한 서브세팅도 리스트와 유사합니다.

df[[2]]라고 하면 두번째 열(column)에 해당하는 벡터를, df[2]라고 하면 두번째 열을 그 값에 해당하는 벡터값만이 아니라, 열의 이름인 gender가 더해진 열 한 개짜리 데이터프레임을 돌려줍니다. 그런데, 이런 방식 말고 리스트와 다른 방식의 서브세팅도 있습니다. 이는 데이터프레임을 다룰 때에는 값을 취할 행과 열을 모두 지정해 주는 경우도 많기 때문이지요. 그 방식은 다음과 같습니다.

```
df[2,3]
```

```
[1] 165
```

이 명령어는 두 번째 행, 세 번째 열에 해당하는 값을 취하라는 뜻입니다. 그렇다면 앞에서 배운 : 연산자를 이용해 다음과 같은 표현도 가능합니다.

```
df[1:2, 2:3]
```

```
  gender height
1   Male    173
2 Female    165
```

물론, 이는 첫번째 부터 두번째 행, 두번째부터 세번째 열에 해당하는 값을 데이터프레임 형태로 되돌려달라는 의미입니다.

이렇게 연속된 숫자를 이용한 서브세팅 말고 첫번째와 세번째 열에 해당하는 값을 돌려달라고 할 수도 있겠지요? 이 때는 숫자로 이루어진 벡터를 이용합니다.

```
df[ , c(1,3)]
```

```
  id height
1  1    173
2  2    165
3  3    170
```

그런데 이번에 행을 지정하는 부분 (쉼표 앞 부분)이 비어 있습니다. 이는 행 전체에 해당하는 정보를 달라는 것을 뜻합니다.

이것 말고도 서브세팅에 이용할 수 있는 기술은 몇 가지 더 있는데요, 우리는 사실 이 방식을 사용하지 않을 예정입니다. 왜냐하면, 이렇게 R이 기본적으로 제공하는 방식은 가독성이 그리 좋지 않아, 몇 일만 지나도 내가 무엇을 하려고 했던 것인지 잊어버리는 경우가 많습니다. 예컨대, 위의 예에서 첫번째, 세번째 열이 무엇이었는지 기억하는 사람은 별로 없겠지요. 또 내가 스스로 쓴 코드를 이해하기 어렵다는 것은 다른 사람이 이해하기는 더 어렵다는 뜻입니다. 이는 협업을 자주 해야 하는 데이터 저널리즘에서는 치명적인 문제라고 할 수 있습니다. 따라서 우리는 다음 장에서 가독성과 협업 효율성이 높은 다른 방식을 이용할 것입니다.

4.3 조건(Predicate)

여기서 말하는 조건은 쉽게 말하면 질문 입니다. 코딩을 하다보면 질문을 할 일이 정말 많은데요, 특히 우리가 자주 사용하는 조건(Predicate)을 이용한 질문들은 그 대답이 logical 값, 즉, TRUE 또는 FALSE가 됩니다. R에게 참, 거짓을 물어보는 방식은 아주 많은데요, 그 중 아주 많이 사용하는 것들로는 다음과 같은 것들이 있습니다.

==, !=, %in%, is.na(), >, >=, <, <=

==는 “두 값이 같은지”를 물어보는 것입니다. 예컨대,

```
2 == 2
```

[1] TRUE

```
2 == 3
```

[1] FALSE

```
a <- 2
a == 2
```

[1] TRUE

등이 가능합니다. 마지막 예에서 알 수 있듯이, == 앞 뒤에는 값이 와도, 변수명이 와도 좋습니다. 여기서 등호가 2개라는 것에 유의하세요. R에서 =와 ==는 전혀 다른 것입니다. =는 <-와 유사하게 할당의 의미로, ==는 질문의 의미로 사용합니다.

반면, !=는 “구 값이 다른지”를 물어보는 것이겠지요. 예컨대,

```
2 != 3
```

[1] TRUE

두 값이 다르기 때문에 이에 대한 대답은 TRUE 입니다. 반면 2!=2의 답은 FALSE겠지요. !=은 등호 앞에 !를 붙여서 만드는데, !는 R에서 not을 의미합니다.

%in%은 해당 연산자 앞의 값이 연산자 뒤에 나오는 벡터에 속하는지를 물어보는 조건 입니다. 예컨대 다음과 같은 방식입니다.

```
2 %in% c(2,3,4,5)
```

[1] TRUE

```
6 %in% c(2,3,4,5)
```

[1] FALSE

마지막으로 in.na()는 결측값이 있는 곳에 TRUE를 되돌려주는 조건입니다.

```
is.na(c(2,3,NA,5))
```

[1] FALSE FALSE TRUE FALSE

is.na()는 앞에서 다룬 조건들과 몇 가지 점에서 조금 다릅니다. 먼저, 질문을 하는 방식에 두개의 다른 값을 이용하는 것이 아니라 하나의 값(위의 예에서는 벡터 c(2,3,NA,5))만을 사용합니다. 둘째, 그렇기 때문에 조건의 형태가 일반 함수 형태로 되어 있고, 괄호 안에 질문의 대상이 되는 값을 써 넣도록 되어 있습니다. 셋째, 대답이 하나의 TRUE 또는 FALSE가 아니라, 벡터의 모든 요소에 대해서 주어집니다. 세번째 요소만 결측값이니, 답이 c(FALSE, FALSE, TRUE, FALSE)가 되는 것이지요. 만약 반대로 “결측값이 아닌 곳”을 찾으려 한다면, !이 부정을 의미한다고 하였으니 다음과 같이 쓰면 되겠지요.

```
!is.na(c(2,3,NA,5))
```

```
[1]  TRUE  TRUE FALSE  TRUE
```

5 Quarto: 기사작성과 코딩을 동시에!

우리는 지금까지 R의 기초적인 기능을 살펴보면서, 콘솔이라고 하는 창을 이용해 왔습니다. 이 콘솔이라고 하는 창은 R의 엔진, 실제로 프로그램이 돌아가는 곳이라고 볼 수 있습니다. 하지만 콘솔에서는 코드를 작성하고 엔터키를 누르는 순간, 프로그램이 실행되고 맙니다. 지금까지 작성해 온 간단한 코드라면, 이런 방식으로 충분하겠지만, 만약 수십줄, 많게는 수백줄의 코드를 작성하고, 작성한 코드가 한 번에 실행되기를 바란다면, 이것은 그다지 효율적인 방법이 아니겠지요. 따라서, 대부분 프로그래머들은 컴퓨터 메모장에 글을 쓰듯, 먼저 긴 코드를 작성한 후, 이것이 차례차례 콘솔에서 작동되도록 합니다. 이를 위해 작성한 코드로 이루어진 텍스트 파일을 흔히 '스크립트'라고 부릅니다. 하지만 이런 스크립트 방식의코드 작성법은 콘솔을 이용한 방법과는 반대로 작성한 코드의 부분부분이 의도한대로 잘 작동하는지 확인하는 것을 어렵게 만듭니다.

따라서, 우리는 그 중간쯤 되는 '노트북'이라는 방법을 사용하려고 합니다. 노트북 방법을 사용하면, 코드를 작성하면서 중간중간 결과를 확인할 수도 있고, 프로그램을 모두 작성한 후에는 전체 프로그램을 한 번에 실행할 수도 있습니다.

5.1 노트북 생성하기

노트북 환경을 이용하는 방법은 여러가지가 있지만, 우리는 "Quarto Document"라는 가장 표준적인 방법을 사용할 것입니다. "Quarto Document"를 사용하려면 새로운 "Quarto Document" 파일을 하나 만들어주면 됩니다. Rstudio 상단의 File 메뉴에서 New File이라고 되어 있는 곳에 마우스 커서를 올려보세요, 그러면 오른쪽에 떠오르는 메뉴에

“Quarto Document”라는 항목이 있을 것입니다. 이를 클릭하면 다음과 같은 창이 떠오릅니다.

캡처

여기서 지시하는대로 문서의 제목, 작성자(Author)의 이름 등을 입력하세요. 그러면, 파일 하나가 생성됩니다.

그 다음 Visual이라고 되어 있는 모드를 Source로 바꿔주세요. (Visual 모드를 사용해도 되지만, 우리는 Source 모드를 이용하겠습니다.)

캡처

마지막으로 해당 파일을 저장해 주세요. ‘File’ 메뉴에서 ’Save’를 클릭한 후, 파일명을 정해주면 됩니다. 확장자는 qmd가 됩니다.

이제 여러분은 노트북 환경을 이용하기 위한 준비가 되었습니다.

5.2 노트북 이용하기

노트북 환경은 구체적으로는 다음과 같습니다.

그림

이 노트북 환경은 크게 세 부분으로 이루어집니다. 첫번째, ---로 둘러싸여 저자, 제목 등의 정보를 제공하는 부분입니다. 이는 YAML header라고 부르는데요, 꼭 있어야 하는 부분은 아닙니다. 하지만 조금 후에 굉장히 유용해 질 것이므로, 그대로 이용하실 것을 권장합니다. 제목(title)이나, 저자(author)의 내용에 해당하는 부분은 원하는대로 바꾸어도 좋습니다.

두번째 부분은 코드가 쓰여지는 chunk라고 부르는 부분입니다. chunk는 세 개의 “backtick”, 즉 “역따옴표”로 코드를 둘러싸서 표현합니다. 역따옴표에 익숙지 않다면, 키도드 숫자 1 왼쪽에 있는 키가 역따옴표를 사용하기 위한 키 입니다. 여러분들은 주로 shift 키와 함께 물결 표시를 사용

하기 위해 더 자주 사용했을 것입니다. Rstudio는 세 개의 역따옴표 있는 문자를 모두 프로그래밍 코드라고 인식합니다.

특히, 시작하는 세 개의 역따옴표 옆에 {r}이라고 쓰게되면, 이제 Rstudio는 해당 chunk 안의 모든 문자를 다른 프로그램도 아닌 R 문법으로 작성한 코드라고 생각합니다. 만약, chunk 안에서 R 문법에 어긋나는 문자가 있다면 에러가 발생하게 될 것입니다.

역따옴표와 사용할 프로그램을 표시하여 (예컨대, "'{r}이라고 표시하여) R 코드임이 명확해지면, Rstudio는 chunk가 시작하는 줄 오른쪽에 작은 플레이 버튼 같은 것을 표시합니다. 이 버튼을 클릭해 보세요. 그러면 해당 chunk에 쓴 R 코드가 실행되고, 그 결과는 해당 코드 바로 밑에 표시됩니다. 이런 식으로 스크립트를 작성하면서, 즉시 콘솔에서처럼 결과도 확인할 수 있게 되는 것이지요.

마지막 부분은 앞의 두 요소 밖에 있는 모든 문자들 입니다. 이는 물론 일반 텍스트를 뜻합니다. 이 부분은 글을 쓰듯 자유롭게 작성하면 됩니다. 사실 아무 것도 아닌 것 같지만, 이렇게 내가 쓰고 싶은 글과 R 코드, 그리고 R 코드의 결과물까지 하나의 텍스트 파일 위에서 표현할 수 있게 됩니다. 마치 다음 그림 처럼요.

그림

마치 하나의 웹페이지 화면처럼 보이지 않나요? 실제로 텍스트 파일 상단에 있는 Render 버튼을 누르기만 하면 바로 웹페이지를 만들어냅니다. 더 정확하게는 해당 qmd파일을 일종의 스크립트 삼아, 텍스트 부분은 텍스트로 표현하고, chunk에 해당하는 부분은 코드를 실행시킨 다음, 그 결과까지 모두 포함하여 하나의 HTML 파일을 만들어 냅니다. 여러분이 사용하고 있는 프로젝트 폴더에 들어가보세요. 방금 Render한 qmd파일과 같은 파일명을 가진 HTML 파일이 생성되어 있을 것입니다. 그 파일을 더블클릭하면, 여러분이 사용하는 브라우저를 통해 마치 웹페이지처럼 정돈된 결과물을 볼 수 있습니다.

방금 Render한 qmd 파일에서 일반 텍스트가 여러분이 작성한 기사 내용, 그리고 chunk가 여러분이 독자들에게 보이고자 하는 데이터 시각화를 시행하는 코드라고 생각해 보세요! 그러면 여러분은 이미 인터넷으로 배포할 수 있는 기사 웹페이지 하나를 만든 셈입니다!

5.3 마크다운 문법

Rstudio가 우리가 작성한 qmd 파일에서 코드에 해당하는 부분, 일반 텍스트에 해당하는 부분을 자동으로 구분하고 다양한 기능을 사용할 수 있게 해줄 뿐만 아니라, 심지어 웹페이지로까지 만들어 주는 것은 우리가 Rstudio가 이해할 수 있는 특정 문법을 따르기 때문입니다. 이 문법을 '마크다운(markdown)'이라고 부릅니다. 사실 확장자명 qmd에서 "md"가 마크다운(markdown)을 뜻하지요. 앞서 사용한 "—" ""'{r}" 이런 것들이 마크다운 문법 중 일부입니다. 마크다운의 문법은 물론 이것 말고도 다양합니다.

우리는 앞서 Rstudio가 chunk 안에 있는 문자는 코드로 인식해 실행하고, 그 밖의 문자는 텍스트 그대로 표현한다고 했습니다. 사실 이는 정확한 이야기가 아닙니다. 왜냐하면 Rstudio가 chunk 밖의 문자를 일반 텍스트로 인식하는 것은 맞지만, 일반 텍스트에 여러 효과를 부여하기 위한 몇 가지 간단한 문법이 있기 때문이지요. 이러한 문법들은 써도 그만, 안 써도 그만이지만 시각적으로 더 효율적인 웹페이지를 생성하기 위해 유용합니다. 예컨대 다음과 같은 것들이 그 문법에 해당합니다.

> italic bold ~~strikeout~~ code ## Headings

6 1st Level Header

6.1 2nd Level Header

6.1.1 3rd Level Header

6.2 Lists

- Bulleted list item 1
- Item 2
- Item 2a
- Item 2b

1. Numbered list item 1
2. Item 2. The numbers are incremented automatically in the output.

6.3 Links and images

http://example.com

linked phrase

6 1st Level Header

6.4 Tables

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

이러한 문법들은 Render 버튼을 누르면 웹페이지를 위한 적절한 시각 요소로 번역됩니다.

6.5 마크다운과 마크업

사실 “마크다운”이라는 용어는 일종의 말장난입니다. 원래는 “마크업(markup)”이라는 용어가 먼저 있었지요. 마크업이라는 것은 일반 텍스트에 “여러 표시를 해서(mark up)”, 텍스트에 여러 효과와 형식을 부여하는 문법을 의미합니다. 가장 대표적인 예가 바로 HTML 이죠.

```
<!DOCTYPE html>
<html>
<body>

<h1> 이것은 제목 입니다</h1>

<p> 이것은 문단 입니다 </p>

</body>
</html>
```

여기에 쓰여진 “이것은 제목 입니다”, “이것은 문단 입니다” 등의 텍스트들은 바로 우리가 전달하고 싶은 텍스트 정보 입니다. 하지만 효율적

으로 이를 독자들에게 전달하기 위해서는 약간의 양식이 있으면 좋겠죠. 따라서 제목은 제목처럼 보이게 문단은 문단처럼 보이게 하는 양식 정보를 전달하는 것이 ’

‘,’

’ 같은 HTML 문법입니다. 구글 크롬, 마이크로소프트 엣지 같은 웹브라우저는 이러한 양식을 이해해서 이용자에게 텍스트를 그대로 전달하는 것이 아니라, 양식이 부여된 텍스트 시각화해서(이를 “Render”라고 합니다) 보여주는 역할을 합니다.

사실 이렇게 텍스트에다 여러 양식정보를 부가하는(즉, 마크업하는) 문법은 HTML말고도 여러개가 있습니다. 이들을 통틀어 ’마크업 언어’라고 하지요. 그런데, 위의 HTML 문서에서 알 수 있듯이, 단 두문장을 전달하는 데에도 사용해야 할 문법이 꽤나 복잡하고 많습니다. 이 때문에 개발자들은 텍스트에 양식을 부과하되, 사람이 직관적으로 이해할 수 있는 좀 간단한 문법이 없을까? 라는 생각을 하게되었습니다. 그 결과물이 바로 ’마크다운’이지요, 즉, 마크’업’이 너무 복잡해서 간단하게 만든 새로운 마크업 문법이 마크’다운’이라는 것입니다.

이제 간단한 마크다운 문법을 알면 이용자는 HTML 같이 복잡한 마크업 문법을 몰라도 웹페이지를 간단하게 만들 수 있습니다. Rstudio가 내장하고 있는 프로그램이 마크다운 문서를 HTML로 다시 재번역해 주거든요(그것이 여러분이 Rstudio에서 이용하는 Render 버튼의 의미입니다). 여러분들이 매일 이용하는 웹브라우저는 이렇게 번역된 HTML을 이해할 수 있으니 여러분은 뉴스 작성을 위해 HTML을 따로 공부할 필요가 없습니다.

마크업 -> pandoc -> HTML -> 웹브라우저

명심하세요! 여러분들은 이 책을 통해 연습을 하는 동안 항상 qmd 파일을 하나 만들어 마크다운 문법을 이요해 코드와 텍스트를 동시에 작성하고 이를 HTML로 렌더하는 작업을 반복할 것입니다. 그 외에도 R을 사용하는 여러 방법이 있지만, 일단 가장 쉽고 기사 작성을 위해 유용한 이 방법을 반복해서 사용하세요!

7 Tidyverse의 이해

앞서 R이 기본적으로 제공하는 간단한 문법을 배워보았는데요, 그러면서도 앞으로 해당 기본 문법보다는 일종의 대체 문법을 사용할 것이라는 언급을 여러번 했습니다. 그 대체문법이 이 장에서 배울 “Tidyverse” 문법 입니다. 사실 “Tidyverse”가 대체문법이라고 하는 데에는 좀 어폐가 있습니다. “Tidyverse” 문법을 따른다고 하더라도, R이 가지고 있는 변수, 함수, 데이터 타입, 데이터 구조 등이 ‘거의’ 그대로 사용되기 때문입니다. 다만, 세브세팅이나, 새로운 변수의 생성, 긴 코드의 작성을 편하고 일관되며, 가독성 있게 만들기 위해서 도입된 일종의 코딩 ‘스타일’ 또는 ‘패러다임’ 이라고 부르는 것이 더 나을지도 모르겠습니다.

하지만, 그러한 ’스타일’을 위해서는 R이 기본적으로 제공하지 않는 몇 가지 함수가 필요합니다. R이 기본적으로 제공하지 않는다면, 어떻게 한다는 것일까요? 이를 위해서 우리는 ’패키지’라는 개념부터 알아야 합니다.

7.1 패키지 개념 이해하기

R뿐만 아니라, 많은 다른 언어에서 ’패키지’라고 하는 것은 유용한 함수들의 묶음을 의미합니다. 기본 R을 만든 사람들이 R에 내장해두지는 않았지만, 특정 목적을 위해서는 유용함 함수들이 묶여서 ’패키지’로 제공된다고 생각하면 되겠습니다. 비유하자면, 여러분들이 매일 사용하는 ’카카오톡’이 R이라고 해 보죠. 카카오톡 안에는 기본적인 채팅 기능 안에도 수많은 다른 기능들이 있지요? 선물을 주고 받을 수도 있고, 음식을 주문할 수도 있고, 쇼핑을 할 수도 있습니다. 물론 모든 이용자들이 이 기능들을 모두 사용하지는 않지요. 하지만, 어떤 사람들은 자주 사용합니다.

이러한 세부 기능에 해당 하는 일종의 카카오톡 안의 '미니 프로그램'을 R에서는 패키지라고 부른다고 생각하면 되겠습니다.

그림 카카오 미니 프로그램

R에서 패키지를 부르는 방법은 다음과 같습니다.

```
library(stats)
```

library()는 괄호가 있으니, 패키지를 부르는 '함수'일 것이고, 괄호 안에는 부르고 싶은 패키지의 이름을 따옴표 없이 쓰면 됩니다. 여기서는 R을 설치할 때 같이 설치되는 stats라는 패키지를 불렀습니다.

그림: 기본 패키지 - 공식 패키지 - 비공식 패키지

그런데 R이 설치될 때 모든 패키지가 함께 설치되는 것은 아닙니다. 사실 그렇지 않은 패키지가 훨씬 많지요. 심지어 패키지는 여러분도 만들 수 있습니다. 그렇게 개인이 여러 목적으로 만든 비공식 패키지까지 포함한다면, 이 세상에는 무한히 많은 R 패키지들이 있습니다. 하지만, 우리가 자주 사용하게될 패키지는 R개발진들의 일종의 감독 하에 있는 '공식' 패키지들입니다. 공식 패키지이지만, R과 함께 설치되지 않는 패키지를 설치하기 위해서는 다음과 같은 함수를 실행시키면 됩니다 (여기서는 따옴표가 필요합니다!).

```
install.packages("tidyverse")
```

예, 우리가 사용한다고 했던 바로 그 tidyverse 입니다. tidyverse 문법을 이용하기 위해서 몇 가지 추가적인 함수가 필요하다고 했지요? 이렇게 설치된 tidyverse 패키지 안에 우리가 사용하고자 하는 바로 그 함수들이 포함되어 있습니다.

물론, 설치만 한다고 해서 바로 이 패키지를 사용할 수 있는 것은 아닙니다. 아까 stats 패키지의 예에서처럼, tidyverse 패키지도 다음과 같이 불러주어야 합니다.

```
library(tidyverse)
```

이 함수를 실행시키면 tidyverse 문법을 사용할 준비가 된 것입니다.

7.1.1 패키지 사용시 주의사항: 설치? 로드?

방금 tidyverse 패키지를 설치하고 불러왔는데요, 여기서 패키지를 '설치'한다는 것과 '불러온다'는 것의 의미에 대해 잠깐 생각해볼 필요가 있습니다.

여러분이 일상적으로 윈도우 또는 맥OS 상에서 어떤 소프트웨어를 '설치'하면 그것은 어디에 저장되었다는 것을 의미하나요? 당연히 HDD, SSD와 같은 '저장장치' 입니다. 그래야, 컴퓨터를 껐다 다시 키더라도, 다시 '설치'할 필요가 없을테니까요. tidyverse 패키지를 설치했다는 것도 비슷한 의미입니다. 설치한 패키지는 저장장치 어디엔가 저장되어 있습니다.

> 따라서 설치는 한 번 했다면 다시 할 필요가 없습니다.

그런데 여러분 설치한 소프트웨어가 항상 켜져있나요? 물론 아닙니다. 그 소프트웨어를 실행시켜야지요. 실행시킨다는 것은 뭘까요? 여러 의미가 있지만, 그 프로그램을 컴퓨터 메모리 상에 '불러온다'는 것입니다. 가끔 여러분들 너무 많은 소프트웨어를 동시에 열어두면 컴퓨터가 느려져서 몇몇 사용하지 않는 프로그램을 끈적인 있을 것입니다. 프로그램을 껐다는 것은 메모리에서 지운다는 것을 의미합니다. 따라서 메모리에 여유 공간을 조금 확보하려고 프로그램을 끄는 것이지요. 이렇게 프로그램을 켠다는 것은 R 패키지로 치면 패키지를 로드하는 과정과 유사합니다. 아까의 예에서처럼 library(tidyverse)라는 함수를 실행시키면, 저장장치에 저장되어 있던 tidyverse 패키지가 메모리로 불러와집니다. 그런데 메모리는 어떤 특성을 가지고 있었나요? 컴퓨터를 꺼도, 해당 정보에 할당된 이름이 사라져도, 메모리에서 언제든 사라질 수 있습니다. 이것은 무슨 이야기일까요? 여러분들이 Rstudio를 끄는 순간, 또는 Rstudio 안

에서 다른 프로젝트로 이동하는 순간, 불러왔던 패키지는 메모리에 존재하지 않는 것이 됩니다. 그러니,

> 패키지를 불러오는 것은 Rstudio를 새로 켤 때마다, 또는 새로운 프로젝트로 이동할 때마다, 매번 다시 해야 합니다!

또 한 가지, 여러분들에게 당장 자주 일어나지는 않을 일이지만, 여러 개의 패키지를 로드해서 사용하다보면 간혹 생기는 문제가 있습니다. 바로 다른 두 개의 패키지가 하나의 함수명을 공유하고 있는 경우 입니다. 예컨대, MASS라는 패키지에도 select()라는 함수가 있고, tidyverse 패키지에도 select() 함수가 있습니다. 기능도, 문법도 완전히 다르지요. 따라서, 후자를 생각하고 select() 함수를 썼다가 에러가 발생하는 경우가 있습니다. 여러 방법이 있지만, 이럴 때에는 tidyverse::select()라고 써 주세요. 그러면, tidyverse 패키지에 속하는 select() 함수를 사용하게 됩니다.

7.2 Tidyverse란

이제 겨우 tidyverse 문법을 사용할 준비가 되었습니다. 물론 tidyverse 패키지의 설치와 불러오기를 잊지 마세요! 사실 tidyverse는 많은 다양한 코딩 방식, 시각화 방식, 데이터 타입과 구조에 대한 재정의 등을 포함한 광범위한 코딩 패러다임인데요, 우리는 학습 목적상 다음 두 가지에만 주목하려고 합니다.

1. 서브세트
2. 파이프(pipe)

그 외에 tidyverse의 다른 요소들은 학습이 심화되면서 차차 마주하게 될 것입니다.

첫번째, 서브세트는 우리가 앞장에서 이미 보았던 서브세트, 즉, 데이터의 일부만을 취하는 방법입니다. R은 이미 좋은 서브세트 기능을 제공하지만, 앞서 말했듯 해당 문법이 가독성이 아주 좋지는 않다고 했습니다. 따라서 tidyverse에서 제공하는 방식의 서브세트를 이용할 것입니다. 두

번째 '파이프'는 긴 코드를 연결해서 간결하게 작성하데 필요하나 일종의 작성법 입니다. 차례차례 살펴보지요.

7.2.1 (1) 서브세트

tidyverse의 서브세트는 기본적으로 R 데이터구조 중 데이터프레임에 적용되는 것입니다. 벡터나 리스트를 사용할 때에는 기본 R 문법을 사용하면 됩니다. 데이터프레임을 서브세트 하는 방법에는 행을 취하는 방법과 열을 취하는 방법이 있었습니다. 다음을 기억해 두세요. tidyverse에서는:

1. 열을 취할 때에는 항상 select()함수와 열의 '이름'을 사용합니다.
2. 행을 취할 때에는 항상 filter()함수와 행을 취하는 '조건'을 사용합니다.

이것만 이해하면 거의 모든 것을 이해한 것과 다름 없습니다. 일단, tidyverse 서브세트는 데이터프레임에 관한 것이라고 했으니, 일단 앞서 이미 이용했던 데이터 프레임을 다시 사용해 보지요.

```
df <- data.frame(id = c(1, 2, 3),
                 gender = c("Male", "Female", "Female"),
                 height = c(173, 165, 170))
df
```

```
  id gender height
1  1   Male    173
2  2 Female    165
3  3 Female    170
```

열을 취할 때는 select() 함수와 열의 이름을 사용한다고 했으므로, 다음과 같이 씁니다.

```
select(df, gender)
```

```
  gender
1   Male
2 Female
3 Female
```

즉, select() 함수의 첫번째 인수는 서브세트의 대상이 될 데이터프레임, 두번째 인수는 거기서 선택할 열의 이름입니다. 그런데, 사실 복수의 열을 선택할 수도 있습니다.

```
select(df, gender, height)
```

```
  gender height
1   Male    173
2 Female    165
3 Female    170
```

즉, 인수로 열의 이름을 그냥 나열하면 된다는 것입니다.

행을 취할 때는 filter() 함수와 행을 취할 조건을 사용한다고 했습니다. '조건'이라는 말을 앞장에서 보았지요? 바로 TRUE, FALSE를 뱉어내는 R에게 질문하는 방식이라고 설명했던 바로 그 조건 입니다. 따라서, 키가 170이 넘는 사람에 관한 행만을 취하고 싶다면, 다음과 같이 쓰면 됩니다.

```
filter(df, height >= 170)
```

```
  id gender height
1  1   Male    173
2  3 Female    170
```

filter() 함수의 첫 인수 역시 데이터프레임이고, 그 인수로는 조건이 옵니다. 그렇다면 두 개의 조건을 사용할 때는요? 그 때는 두 개의 조건이 AND 관계인지, OR 관계인지를 밝혀야 합니다. 그 관계를 밝히는 기호는 다음고 같습니다.

AND: 조건1 & 조건2 OR: 조건1 | 조건2

이제 다음과 같은 예를 보세요.

```
filter(df, height >= 170 & gender=="Female")
```

```
  id gender height
1  3 Female    170
```

이것은 여성'이고(AND)' 키가 170 이상인 사람에 해당하는 행을 돌려달라는 것이겠지요? 반면,

```
filter(df, height >= 170 | gender=="Female")
```

```
  id gender height
1  1   Male    173
2  2 Female    165
3  3 Female    170
```

이것은 여성'이거나(OR)' 키가 170 이상인 사람에 해당하는 행을 돌려달라는 것이겠지요? 이는 데이터 전체에 해당하네요.

7.2.2 (2) 파이프

파이프는 코드를 길게 써야 할 때 필요한 사용하는 코딩 스타일 입니다. 파이프를 위해서는 '파이프'로 불리는 기호가 하나 필요합니다. 바로 다음 기호 입니다.

그림: 파이프

이 간단한 기호가 어떻게 긴 코드를 작성하는데 도움을 준다는 것일까요?

위의 그림에서 볼 수 있는 것처럼, 파이프는 함수의 첫 번째 인수를 함수의 왼쪽에 쓸 수 있게 해줍니다. 이제 데이터에 함수를 두 번 연속적으로 적용해야 하는 경우를 생각해 볼까요? 일반적인 문법에서는 우리가 중학교 수학 시간에 배운 것과 같이 다음과 같이 쓰면 됩니다.

```
do_this_next(do_this_first(data))
```

이를 조금 풀어서 설명하면, 원 데이터(data)에 do_this_first() 함수를 먼저 적용하고, 그 결과에 do_this_next() 함수를 적용하는 것이지요. 물론 이렇게 써도 아무 문제가 없습니다. 하지만, 이런 방법에는 큰 문제가 있습니다. 바로 코드를 오른쪽에서 왼쪽으로 읽어야 한다는 것이지요. 이는 우리가 자연스럽게 글을 읽는 방식과 반대입니다. 사실, 이렇게 함수가 두 개 뿐이라면 큰 문제가 아니겠지만, 함수가 더 많아지면 가독성이 더욱 떨어집니다.

```
finally_do_this(then_do_this(do_this_next(do_this_first(data))))
```

현실적으로는 함수를 십수개 차례대로 적용해야 하는 경우도 많으니, 이러한 방식의 코드는 나중에 스스로가 리뷰하기도 어려울 뿐더러, 협업 상황이라면 상황은 더욱 악화됩니다. 그래서 전통적으로는 이런 방식의 코드를 많이 사용해 왔습니다.

```
data2 <- do_this_first(data)
data3 <- do_this_next(data2)
data4 <- then_do_this(data3)
finally_do_this(data4)
```

즉, 함수를 실행하고 그 결과를 변수로 저장한 다음, 다음 함수에 해당 변수를 인수로 사용하는 방식이지요. 읽기 조금 수월한가요? 어느 정도는 괜찮습니다. 하지만, 이것도 결코 읽기 쉬운 코드라고 할 수는 없습니다.

또 그 내용을 알기 어려운 data2, data3, data4 등의 불필요한 데이터들이 계속해서 생성됩니다. 이 역시 가독성을 낮출 뿐더러, 불필요하게 메모리를 차지하게 됩니다. 데이터의 이름을 조금 더 이해하기 쉽게 만들면 좋겠지만, 이렇게 데이터 처리 와중에 생긴 데이터에 모두 이름을 붙일만한 의미가 있는 것도 아니고, 데이터 이름을 자꾸 만들다보면, 더 이상 이름을 만들 아이디어도 떠오르지 않게 됩니다(이건 농담이지만, 진담이기도 합니다!).

우리가 파이프를 쓴다면 다음과 같이 쓸 수 있습니다.

```
data |>
    do_this_first() |>
    do_this_next() |>
    then_do_this() |>
    finally_do_this()
```

이 코드를 잘 보면 항상 파이프 왼쪽에서 발생한 결과가 다음 함수의 투입값(input)이 됩니다. 이 코드는 우리가 글을 읽는 방식과 동일한 순서에 따라 쓰여졌기 때문에 읽기 쉬울 뿐더러, 필요 없는 중간 단계의 데이터(data2, data3 따위)도 만들지 않았습니다. 우리는 앞으로 연속해서 함수를 적용해야 할 때, 위의 예와 같이 파이프를 사용할 것입니다. 여러분도 가능한 위의 코딩 스타일을 사용하도록 노력해 보세요!

7.2.3 (3) 서브세트 + 파이프

이제 tidyverse 문법의 주요 구성 요소라고 말한 서스세트와 파이프를 결합해 보겠습니다. 이제 앞서 사용했던 예에서 특정 열을 선택하고, 또 어떤 조건에 따라 행도 선택한다고 해 보지요. 그러면 다음과 같이 쓸 수 있을 것입니다.

```
df |>
    filter(height >= 170 & gender=="Female") |>
```

```
    select(gender, height)
```

```
  gender height
1 Female    170
```

코드를 잘 살펴보면 그 의미는 다음과 같습니다.

(1) 원데이터 df에서 키가 170이 넘고, 여성인 행의 데이터를 취하라.
(2) 그 다음 gender, height 칼럼만 표시하라.

파이프를 썼기 때문에, filter() 함수와 select() 함수 안에는 데이터가 인수로 들어가지 않는다는 것에 주목하세요! 파이프를 사용하면 서브세트할 데이터는 두 함수 앞에 사용한 파이프의 왼쪽에 있게 됩니다!

다음 내용으로 넘어가기 앞서, 위의 코드를 꼭 이해해야 합니다! 여러분은 이 책을 모두 읽을 때까지 비슷한 코드를 수십번 보고, 사용하게 될테니까요.

8 데이터 불러오기, 조작하기

8.1 Tidyverse를 이용해 데이터 로드하기

마지막으로, tidyverse에 내장된 패키지를 이용해 외부의 데이터를 불러오는 방법에 대해 알아보겠습니다. R은 애초에 데이터를 다루기 위해 만들어진 언어이므로, 당연히 외부 데이터를 불러오는 함수가 내장되어 있습니다. 그러나 예전에 만들어진 기능이기도 하고, 몇 가지 직관적이지 않은 동작 때문에 여러가지 에러가 발생하는 경우가 많습니다. 또 데이터에 로마 알파벳과 숫자 이외에 다른 정보가 들어 있으면 이를 처리하기 위한 작업을 추가적으로 해야 하는 경우도 있습니다. 엑셀, SPSS, Stata 등 다른 소프트웨어에서 사용하는 포멧에 취약하기도 하고요. 따라서, 앞으로 우리는 R이 원래 제공하는 함수가 아닌, tidyverse에 내장된 함수를 사용해서 외부 데이터를 불러올 것입니다.

이 책은 데이터 저널리스트가 되고자 하는 여러분들을 위한 것이므로, 나중에는 여러분만의 데이터를 얻는 방법을 배울 것입니다. 하지만, 그것은 조금 후의 일이니, 일단은 제가 생각하기에 연습에 사용하기 좋다고 생각하는 데이터를 사용하겠습니다.

8.1.1 donorschoose.org 데이터 소개

우리가 당분간 사용할 데이터는 donorschoose.org[https://donorschoose.org]라는 미국 크라우드펀딩 웹사이트의 로그 데이터 입니다. 여러분들도 크라우드펀딩 이라는 용어는 자주 들어봤을 것이라고 생각합니다. 못 들어보셨을 분들을 위해 짧게 설명하자면, 크라우드 펀딩은 돈을 달라고 요구하는 웹사이트 입니다. 물론 그냥 돈을 달

라는 것은 아니고, 돈이 필요한 기발한 아이디어를 일종의 '프로젝트' 형태로 올리는 경우가 많습니다. 웹사이트에 들른 잠재적 기부자들은 만약 그 아이디어가 마음에 들면 5달러, 10달러 정도의 작은 돈을 '기부'하는 것이지요. 미국에서는 'Indiegogo', 'Kickstarter', 한국에서는 '와디즈', '텀블벅' 등이 유명합니다.

크라우드펀딩은 많은 경우 소규모 사업 아이디어에 작은 규모의 투자를 받기 위한 플랫폼입니다. 하지만, 우리가 사용할 donorschoose.org는 그 성격이 조금 다릅니다. 여기에 올라오는 프로젝트들은 신기한 아이디어를 가지고 있는 창업자가 아니라, 교육에 필요한 기자재가 부족한 공립학교 선생님들이 올립니다. 그래서 프로젝트 소개에는 하고자 하는 교육의 목적과 학생들에 대한 소개, 필요한 기자재 목록 등과 함께 필요한 돈의 총액이 있습니다. 이 사연을 보고 기부자들은 몇 달러의 돈을 기부하는 것이지요. 어떻게 보면 진정한 의미의 기부 플랫폼입니다.

다른 크라우드펀딩 웹사이트들처럼 donorschoose.org도 몇 가지 룰이 있는데요, 그 중 가장 중요한 것은 다음 두가지 입니다. 첫째, 정해진 시간(보통 90일) 안에 프로젝트를 올린 선생님이 정해놓은 목표액을 모두 채우면, 해당 프로젝트는 완료되고 선생님은 돈을 받습니다. 둘째, 정해진 시간 안에 목표액을 채우지 못하면 기부된 돈은 원 기부자에게 모두 환불 됩니다(환불 정책).

이 데이터를 이용하는 이유는 현실의 온라인 환경에서 발생한 '로그 데이터'이기 때문입니다. 우리가 기사에서 흔히 보는 통계 데이터들은 상당히 정제되고 있고, 목적에 따른 디자인에 따라 생성된 데이터이지만, 이러한 데이터는 그렇지 않지요. 많은 경우, '데이터 저널리스트'들은 그런 데이터를 분석하고 싶어합니다. 그러니 처음부터 조금 어렵더라도 '리얼'한 데이터를 만지는 것이 좋지 않을까 합니다.

이 데이터는 어디서 얻을까요? kaggle이라고 하는 일종의 데이터 경진대회 웹사이트에서 얻을 것입니다. 'kaggle'에 대해서 길게 설명할 수는 없지만, 데이터 사이언스에 종사하는 사람들 또는 학생들이 자신이 만들어낸 알고리즘을 뽐내기 위해 경쟁하는 웹사이트라고 생각하시면 되겠습니다. 이렇게 발표된 알고리즘 중 유용한 것들이 있으니 기업들도 자신의 비즈니스에서 생성된 데이터를 'kaggle'을 통해 공개하기도 합니

다. donorschoose.org 데이터도 그런 이유로 'kaggle'에 공개되어 있습니다. 다음 링크를 따라가세요.

[https://www.kaggle.com/competitions/kdd-cup-2014-predicting-excitement-at-donors-choose]

kaggle에 가입을 완료하고, 약관 승인 절차들을 거치고 나면, 데이터탭 오른쪽 아래 'Download All' 버튼을 눌러 모든 파일을 zip 압축 형태로 받을 수 있습니다. 압축된 파일이 1기가 가까이 되니, 아마 여러분 대부분들이 지금까지 다루어본 어떤 데이터보다 '빅'데이터에 가까울 가능성이 크겠네요.

kaggle 그림

압축을 해제해 압축 파일 안에 있는 6개의 csv 파일을 여러분이 앞으로 사용할 프로젝트(잊지 않으셨죠?) 폴더 안에 data라는 폴더를 하나 만들고 그 안에 옮겨놓습니다. 다음 그림처럼요.

폴더 캡처

이제 데이터를 부를 준비가 되었습니다.

8.1.2 데이터 불러오기

데이터를 부르기 전에 다음을 했는지 확인 합시다.

1. 이 책의 내용을 학습하면서 사용할 R 프로젝트를 만들었다.
2. 해당 프로젝트로 이동하였다.
3. tidyverse 패키지를 설치하였다.
4. 프로젝트 안에서 donorschoose.org를 분석하는데 사용할 노트북 파일(qmd파일)을 만들었다.
5. 해당 노트북에서 tidyverse 패키지를 로드하였다.

위의 체크리스트에 대한 대답이 모두 '네'라면, 이제 다음과 같은 코드를 실행해 봅시다.

```
library(tidyverse)
```

```
projects <- read_csv("data/projects.csv")
```

이 코드를 조금 뜯어보도록 합시다. 첫째, read_csv()는 csv 포맷으로 존재하는 데이터 파일을 부르는 함수로 tidyverse 패키지에 속해있습니다. 둘째, 우리가 사용하고 있는 프로젝트 폴더 안에 data 폴더를 하나 만들고 그 안에 projects.csv 파일을 저장했으니, 파일 경로는 상대경로를 이용해 “data/projects.csv”가 됩니다. 셋째, 이렇게 데이터를 부르면 데이터는 저장장치에서 메모리로 불러와집니다. 하지만, 거기까지만 한다면, 문제가 발생합니다. 앞서 이야기한 바와 같이 이름이 없으니 존재하지 않는 것이나 마찬가지가 됩니다. 즉, 한 번 불러와진 다음, 그냥 날아가버린다고 생각하시면 됩니다. 따라서 우리는 이 데이터에 이름을 붙여주어야 합니다. 이름을 붙이기 위한 표현이 바로 projects <- 이 부분입니다. 벡터에 이름을 붙일 때, 리스트에 이름을 붙일 때와 똑같지요? R에서는 몇백 메가 크기의 데이터나 숫자 한개로 이루어진 벡터나 똑같이 변수로 처리될 뿐입니다.

8.1.2.1 csv와 여러 테이블 데이터 포맷

자, 이제 데이터를 분석할 준비가 끝났습니다. 그런데, 다음으로 넘어가기 전에에 csv 포멧을 이용해보지 않은 분들도 있지요? 그런 분들을 위해 잠깐 csv에 대해 설명해 보겠습니다.

csv는 “comma separated values”의 약자입니다. 간단히 말해서 “쉼표”를 이용해서 값들을 구분해 놓은 테이블 형태의 데이터라는 뜻입니다. 이게 무슨 뜻인지는 아래의 그림을 보면 쉽게 이해 되실 겁니다.

https:// www.exceldemy.com/ convert-excel-to-csv-comma-delimited/

아래의 예에서 컬럼을 구분하기 위해 아주 간단하게 쉼표를 써 주었습니다. 행을 구분해 주기 위해서는 엔터를 쳐서 줄을 바꿔주었다고 할 수 있

죠. csv 파일은 사실 이러한 규칙을 따르는 텍스트 파일에 불과합니다. 이를 R이나 마이크로소프트 엑셀, 구글 시트 등에서 부를 때 테이블로 여러분에게 '표현'해 줄 뿐입니다. 사실 이러한 csv 파일은 테이블 데이터를 표현하는 가장 간단한 방법이지요.

여러분은 아마도 마이크로소프트 엑셀이 사용하는 xlsx나 xls 등의 파일 포멧에 더욱 익숙할 것입니다. 그런데 사실 생각해 보면 이러한 포멧의 파일들은 테이블 형태의 데이터 말고도 다양한 정보를 포함하고 있습니다. 글꼴이나, 테이블의 폭, 테두리, 수식 등의 정보들이 그것이지요. 이러한 추가 정보들을 이용하면 여러 편리한 점들이 있지만, 무엇보다 여러분들은 엑셀에 의존할 수밖에 없고, 데이터의 크기도 커지게 됩니다. 즉, 편의성을 위해 범용성을 희생한 것이지요.

따라서, 대규모의 데이터를 다양한 플랫폼에서 다루어야 하는 데이터 사이언스를 위해서는 테이블 데이터를 주고 받을 때 안전하고, 범용적이고, 가벼운 csv 형식을 사용하는 경우가 많습니다.

그렇다면, 테이블 형태의 데이터가 아니라면요? 그것은 나중에 살펴보도록 하겠습니다.

csv가 데이터 저널리즘을 위해 제일 유용하고 흔하다고 하더라도, 여러분들이 항상 테이블 형태의 데이터로 csv 파일을 다루게 되리라는 보장은 없습니다. 그건 데이터를 주는 사람 마음이지요. 그러면 다른 포멧의 데이터들은 어떻게 R로 불러들여야 할까요? 다행이 tidyverse에는 이 문제를 해결하기 위한 다양한 함수들이 포함되어 있습니다. 이 교재에서는 csv 파일 이외의 형식을 다룰 일이 없겠지만, 앞으로를 위해 다음 표를 참조하세요.

데이터 포맷	함수
csv	read_csv()
xls, xlsx	read_excel()
sas7bdat, sas7cat (SAS)	read_xpt()
sav (SPSS)	read_sav()
dta (Stata)	read_dta()

8.2 데이터 조작하기

데이터를 불러온 다음 바로 분석을 시작하면 좋겠지만, 그럴 수 있는 경우는 많지 않습니다. 내가 분석하고자 하는 데이터가 바로 분석 가능하지 않는 타입인 경우도 있고, 기존 데이터에 특정 연산을 가해 분석하고자 하는 데이터를 도출해야 하는 경우도 있습니다. 어떤 경우에는 내가 분석하고자 하는 데이터가 여러 테이블에 흩어져 있어서, 이를 결합해야 하는 경우도 있을 것입니다.

이 중 가장 흔하게 나타나는 시나리오와 이를 수행하기 위한 tidyverse 스타일의 코딩 사례를 살펴보도록 하겠습니다.

이를 위해 먼저 donorschoose.org 데이터 중 두 개의 테이블을 R로 불러오도록 하겠습니다.

```
projects <- read_csv("data/projects.csv")
donations <- read_csv("data/donations.csv")
```

8.2.1 데이터 탐색

먼저 projects 데이터프레임을 한 번 훑어보겠습니다.

```
projects
```

```
# A tibble: 664,098 x 35
  projectid             teacher_acctid schoolid school_ncesid school_latitude
  <chr>                 <chr>          <chr>    <chr>                   <dbl>
 1 316ed8fb3b81402ff6ac8f~ 42d43fa6f3731~ c0e6ce8~ 063627006187             36
 2 90de744e368a7e4883223c~ 864eb466462bf~ d711e47~ 483702008193
 3 32943bb1063267de6ed19f~ 37f85135259ec~ 665c361~ 410327000109
 4 bb18f409abda2f264d5acd~ 2133fc46f951f~ 4f12c3f~ 360015302507             40
 5 24761b686e18e5eace6346~ 867ff478a63f5~ 10179fd~ 062271003157              3
```

```
 6 eac7d156205f1333de3887~ ff064802c18e6~ 929336e~ 040187001058          33.3
 7 5a3bfdf2e05781ccd0654d~ 085794a9e315b~ dc34b02~ 010060000256          32.8
 8 afda16eb54d8992db7bb42~ 1d94a31c2dc38~ 86fff7c~ 130129000571          33.9
 9 2ab3efb23acc84017cd789~ 5a497c425e05b~ ed170a1~ 261200001297          42.4
10 3118962680bb062323c356~ eb0855cc6ea55~ 71b4dee~ 340264001386          40.0
# i 664,088 more rows
# i 30 more variables: school_longitude <dbl>, school_city <chr>,
#   school_state <chr>, school_zip <chr>, school_metro <chr>,
#   school_district <chr>, school_county <chr>, school_charter <lgl>,
#   school_magnet <lgl>, school_year_round <lgl>, school_nlns <lgl>,
#   school_kipp <lgl>, school_charter_ready_promise <lgl>,
#   teacher_prefix <chr>, teacher_teach_for_america <lgl>, ...
```

이렇게 데이터프레임의 변수명을 그대로 실행시키면, Rstudio 노트북 환경에서는 명령어 아래 일부가 출력 됩니다. 여기서 오른쪽 화살표는 화면에 표시된 것 말고도 다른 열이 더 있다는 것을 의미합니다. 그 화살표를 클릭하면 화면이 오른쪽으로 넘어가게 됩니다. 그리고 표 아래에는 인터넷 게시판처럼, 페이지를 선택할 수 있습니다. 이는 화면에 출력된 것 말고도 행이 더 남아 있다는 것을 의미하지요. 그곳에 있는 숫자를 클릭하면 다음 페이지를 볼 수 있습니다.

그림

그것 말고도 데이터프레임을 탐색하기 위한 방법은 여러가지가 있습니다. 어떤 식으로든 변수가 생성되고 나면, Rstudio의 기본 설정상 오른쪽 상단에 있는 Environment 창에 지금 메모리에 저장되어 있는 변수들이 표시됩니다.

그림

지금 보면, 아까 우리가 불러온 donations 데이터 프레임과 projects 데이터프레임이 저장되어 있네요. projects를 클릭해 보겠습니다.

그림

그러면 우리가 엑셀에서 보는 것과 유사하게 스크롤 할 수 있는 테이블 화면이 떠오릅니다. 여기서는 조금 더 익숙한 방식으로 여러 데이터를 살펴보실 수 있을 것입니다.

데이터가 많은 경우에 별로 권장되지는 않지만 콘솔에서 탐색하는 방법도 있습니다. 콘솔에서 다음과 같은 코드를 실행시켜보세요.

```
head(projects)
```

그러면 다음과 같은 화면이 등장할 것입니다. 콘솔은 노트북과 달리 사람이 조작할 수 있는 환경을 제공하지 못하기 때문에, 한 번에 표현할 수 있는 것 이상의 데이터는 생략이 되어 표시된다는 것을 알 수 있습니다.

그림

콘솔에서는 다음과 같은 명령어가 더욱 유용합니다.

```
str(projects)
```

```
spc_tbl_ [664,098 x 35] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ projectid                    : chr [1:664098] "316ed8fb3b81402ff6ac8f721bb31
 $ teacher_acctid               : chr [1:664098] "42d43fa6f37314365d08692e08
 $ schoolid                     : chr [1:664098] "c0e6ce89b244764085691a1b8e28
 $ school_ncesid                : chr [1:664098] "063627006187" "48370200819
 $ school_latitude              : num [1:664098] 36.6 32.9 45.2 40.6 34 ...
 $ school_longitude             : num [1:664098] -119.6 -96.7 -122.4 -74 -118
 $ school_city                  : chr [1:664098] "Selma" "Dallas" "Colton" "Brookl
 $ school_state                 : chr [1:664098] "CA" "TX" "OR" "NY" ...
 $ school_zip                   : chr [1:664098] "93662" "75243" "97017" "11226"
 $ school_metro                 : chr [1:664098] NA "urban" "rural" "urban" ...
 $ school_district              : chr [1:664098] "Selma Unified Sch District" "Rich
 $ school_county                : chr [1:664098] "Fresno" "Dallas" "Clackamas" "
 $ school_charter               : logi [1:664098] FALSE FALSE FALSE FALSE F
 $ school_magnet                : logi [1:664098] FALSE FALSE FALSE TRUE F
 $ school_year_round            : logi [1:664098] FALSE FALSE FALSE FALSE
```

```
$ school_nlns                       : logi [1:664098] FALSE FALSE FALSE FALSE FALSE FALSE ...
$ school_kipp                       : logi [1:664098] FALSE FALSE FALSE FALSE FALSE FALSE ...
$ school_charter_ready_promise          : logi [1:664098] FALSE FALSE FALSE FALSE FALSE FAL
$ teacher_prefix                    : chr [1:664098] "Mrs." "Mrs." "Mr." "Mr." ...
$ teacher_teach_for_america             : logi [1:664098] FALSE FALSE FALSE TRUE FALSE FALSE
$ teacher_ny_teaching_fellow            : logi [1:664098] FALSE FALSE FALSE FALSE FALSE FALSE
$ primary_focus_subject                 : chr [1:664098] "Literature & Writing" "Literacy" "Literacy" "S
$ primary_focus_area                    : chr [1:664098] "Literacy & Language" "Literacy & Language"
$ secondary_focus_subject               : chr [1:664098] "College & Career Prep" "ESL" "Mathematics
$ secondary_focus_area                  : chr [1:664098] "Applied Learning" "Literacy & Language" "M
$ resource_type                     : chr [1:664098] "Books" "Books" "Technology" "Books" ...
$ poverty_level                     : chr [1:664098] "highest poverty" "highest poverty" "high poverty
$ grade_level                       : chr [1:664098] "Grades 6-8" "Grades PreK-2" "Grades PreK-2" "
$ fulfillment_labor_materials           : num [1:664098] 30 30 30 30 30 30 30 30 30 30 ...
$ total_price_excluding_optional_support: num [1:664098] 556 296 431 576 408 ...
$ total_price_including_optional_support: num [1:664098] 654 349 507 678 480 ...
$ students_reached                      : num [1:664098] 32 22 17 12 24 20 320 18 25 28 ...
$ eligible_double_your_impact_match     : logi [1:664098] FALSE FALSE FALSE FALSE FALSE FAL
$ eligible_almost_home_match            : logi [1:664098] FALSE FALSE FALSE FALSE FALSE FALS
$ date_posted                       : Date[1:664098], format: "2014-05-12" "2014-05-12" ...
- attr(*, "spec")=
 .. cols(
 ..   projectid = col_character(),
 ..   teacher_acctid = col_character(),
 ..   schoolid = col_character(),
 ..   school_ncesid = col_character(),
 ..   school_latitude = col_double(),
 ..   school_longitude = col_double(),
 ..   school_city = col_character(),
 ..   school_state = col_character(),
 ..   school_zip = col_character(),
 ..   school_metro = col_character(),
 ..   school_district = col_character(),
 ..   school_county = col_character(),
```

```
  ..   school_charter = col_logical(),
  ..   school_magnet = col_logical(),
  ..   school_year_round = col_logical(),
  ..   school_nlns = col_logical(),
  ..   school_kipp = col_logical(),
  ..   school_charter_ready_promise = col_logical(),
  ..   teacher_prefix = col_character(),
  ..   teacher_teach_for_america = col_logical(),
  ..   teacher_ny_teaching_fellow = col_logical(),
  ..   primary_focus_subject = col_character(),
  ..   primary_focus_area = col_character(),
  ..   secondary_focus_subject = col_character(),
  ..   secondary_focus_area = col_character(),
  ..   resource_type = col_character(),
  ..   poverty_level = col_character(),
  ..   grade_level = col_character(),
  ..   fulfillment_labor_materials = col_double(),
  ..   total_price_excluding_optional_support = col_double(),
  ..   total_price_including_optional_support = col_double(),
  ..   students_reached = col_double(),
  ..   eligible_double_your_impact_match = col_logical(),
  ..   eligible_almost_home_match = col_logical(),
  ..   date_posted = col_date(format = "")
  .. )
 - attr(*, "problems")=<externalptr>
```

str()은 structure라는 뜻으로, 각 컬럼 이름과 각 컬럼의 데이터 타입, 데이터의 갯수, 그리고 가장 상위에 있는 몇가지 값을 보여줍니다. 많은 경우 str()은 각 열의 데이터 타입을 확인하기 위해 사용합니다. 그런데, 만약 str()이 과도하게 많은 정보를 준다고 생각하신다면 다음 명령어도 유용합니다.

```
names(projects)
```

```
 [1] "projectid"
 [2] "teacher_acctid"
 [3] "schoolid"
 [4] "school_ncesid"
 [5] "school_latitude"
 [6] "school_longitude"
 [7] "school_city"
 [8] "school_state"
 [9] "school_zip"
[10] "school_metro"
[11] "school_district"
[12] "school_county"
[13] "school_charter"
[14] "school_magnet"
[15] "school_year_round"
[16] "school_nlns"
[17] "school_kipp"
[18] "school_charter_ready_promise"
[19] "teacher_prefix"
[20] "teacher_teach_for_america"
[21] "teacher_ny_teaching_fellow"
[22] "primary_focus_subject"
[23] "primary_focus_area"
[24] "secondary_focus_subject"
[25] "secondary_focus_area"
[26] "resource_type"
[27] "poverty_level"
[28] "grade_level"
[29] "fulfillment_labor_materials"
[30] "total_price_excluding_optional_support"
[31] "total_price_including_optional_support"
[32] "students_reached"
[33] "eligible_double_your_impact_match"
[34] "eligible_almost_home_match"
```

[35] "date_posted"

이 함수는 데이터프레임 각 열의 이름만을 보여줍니다.

8.2.2 데이터 필터링 (= 서브세팅)

위의 방법으로 데이터를 탐색하다보면, school_ncesid열에 결측값이 다수 존재한다는 것을 알 수 있습니다. NCES가 미국 국가교육통계센터(National Center for Education Statistics)를 의미하니, 공식 통계 상의 학교 ID라는 것을 알 수 있습니다. 그런데, 이 ID가 없다는 것은 뭔가 좀 이상하네요. 만약, 우리가 ID에 결측치가 있는 것은 정상적이지 않으므로 분석에서 제외하기로 결정했고 해 보죠. 그렇다면 지금까지 배운 것을 이용해 어떻게 제외하면 될까요?

물론 답은 세브세트 입니다. 그 중에서도 결측치가 있는 행을 삭제하는 것이니, filter() 함수를 사용해야 하겠네요. 그리고 school_ncesid의 값이 결측되지 않은 경우 만을 취하고자 하니 filter() 함수 안에 들어갈 조건은 !is.na(school_ncesid)가 되겠네요. 따라서 다음과 같습니다.

```
projects_no_na <- projects |>
    filter(!is.na(school_ncesid))
```

이번엔 학교의 위도와 경도 정보는 사용하지 않을 것이라고 해 볼까요? 다시 말해, school_latitude 열과 school_longitude 열을 제외하겠다는 뜻이겠군요. 그 때는 이렇게 해 볼 수 있습니다.

```
projects_cleaned <- projects_no_na |>
    select(!c(school_latitude, school_longitude))
```

여기서도 !은 not의 의미이므로, “해당 컬럼 두 개 빼고” 라는 뜻이 됩니다. 두개 컬럼 이름을 하나의 벡터로 묶기 위해 c()를 사용했다는 것에 주목해주세요.

물론 행과 열의 서브세팅을 파이프로 결합할 수도 있습니다 (이 방법이 더 좋습니다.).

```
projects_cleaned <- projects |>
    filter(!is.na(school_ncesid)) |>
    select(!c(school_latitude, school_longitude))
```

8.2.3 새로운 변수 만들기

주어진 데이터를 분석에 그대로 사용하는 경우는 많지 않습니다. 주어진 변수, 또는 변수들을 바탕으로 분석에 적합한 새로운 변수를 만들어 주어야 합니다. 예컨대, 야구에서 타수와 안타 수가 주어져 있을 때, 타자의 타격 능력을 분석하기 위해서 두 변수를 조합해 타율을 계산하는 경우가 그런 경우에 해당할 것입니다.

이렇게 새로운 변수를 만들기 위한 tidyverse 문법은 mutate() 함수를 사용하는 것입니다. 그 형식은 대략 다음과 같습니다.

> 원래 데이터프레임 |> mutate(새 변수 이름 = 함수(원래 변수))

이렇게 원래의 데이터프레임에 mutate()을 적용하고 나면, 그 결과물은 새로 생산된 변수가 아닙니다! mutate()의 결과물은 원래의 데이터프레임에 새로운 변수(열)이 추가된 새로운 데이터프레임 입니다.

앞서 불러온 donorschoose.org 데이터의 projects 테이블에서 각 프로젝트의 목표액이 달러로 포함되어 있는데요, 보도하는 사람이 이 목표액을 원화로 표시하고 싶었다고 해 보지요. 원달러 환율이 대략 1,200이라고 치면, 원화로 표현된 목표액은 원래 변수에 1,200을 곱한 것이 되겠네요. 사실 목표액에 해당하는 변수는 projects 테이블에 두 개 있는데, 그 중 일단 total_price_excluding_optional_support를 이용하겠습니다. 그러면, 새 변수를 만들기 위해서는 다음과 같이 써 주면 됩니다.

```
projects |>
  mutate(size_kw = total_price_excluding_optional_support * 1200)
```

```
# A tibble: 664,098 x 36
   projectid            teacher_acctid schoolid school_ncesid school_latitude
   <chr>                <chr>          <chr>    <chr>                  <dbl>
 1 316ed8fb3b81402ff6ac8f~ 42d43fa6f3731~ c0e6ce8~ 063627006187            36
 2 90de744e368a7e4883223c~ 864eb466462bf~ d711e47~ 483702008193
 3 32943bb1063267de6ed19f~ 37f85135259ec~ 665c361~ 410327000109
 4 bb18f409abda2f264d5acd~ 2133fc46f951f~ 4f12c3f~ 360015302507            40
 5 24761b686e18e5eace6346~ 867ff478a63f5~ 10179fd~ 062271003157             3
 6 eac7d156205f1333de3887~ ff064802c18e6~ 929336e~ 040187001058
 7 5a3bfdf2e05781ccd0654d~ 085794a9e315b~ dc34b02~ 010060000256
 8 afda16eb54d8992db7bb42~ 1d94a31c2dc38~ 86fff7c~ 130129000571             3
 9 2ab3efb23acc84017cd789~ 5a497c425e05b~ ed170a1~ 261200001297
10 3118962680bb062323c356~ eb0855cc6ea55~ 71b4dee~ 340264001386
# i 664,088 more rows
# i 31 more variables: school_longitude <dbl>, school_city <chr>,
#   school_state <chr>, school_zip <chr>, school_metro <chr>,
#   school_district <chr>, school_county <chr>, school_charter <lgl>,
#   school_magnet <lgl>, school_year_round <lgl>, school_nlns <lgl>,
#   school_kipp <lgl>, school_charter_ready_promise <lgl>,
#   teacher_prefix <chr>, teacher_teach_for_america <lgl>, ...
```

보시다시피, 위의 코드를 실행시키면, 원래 projects 테이블에 새로 만들어진 변수가 추가된 커다란 테이블이 출력됩니다. 하지만 이러한 코드에는 문제가 있습니다. 여러번 반복해서 말하기 때문에 눈치채신 분도 있겠지만, 이렇게 새로 만들어진 테이블은 원래의 테이블과는 다른 새로운 테이블이고, 이 새로운 테이블에는 이름이 없습니다! 이름이 없는 데이터는 존재하지 않는 데이터나 마찬가지라고 했지요? 따라서, 기껏 새로운 변수를 만들고 모니터로 한 번 확인하고는 이를 그냥 버린 것과 다름 없습니다.

그렇다면, projects 테이블에 새 변수(열)을 추가 하려면 어떻게 해야 할까요? 다음과 같이 하면 됩니다.

```
projects <- projects |>
  mutate(size_kw = total_price_excluding_optional_support * 1200)
```

앞의 예와 유일한 차이는 projects <- 이 부분 뿐입니다. 즉, mutate()을 통해 새로운 테이블을 만든 후, 그 테이블의 이름을 원래의 테이블 이름인 projects에 할당해 버린 것이지요. 엄밀히 말하면, 이제 변수를 새로 만들기 전 원래 테이블은 사라져 버리고, 새로운 테이블이 원래의 테이블을 대신해서 projects라는 이름을 대체하고 있는 것이지요.

만약, size_kw라는 새로운 변수를 만들지 않고, 원화로 표시된 숫자가 원래 total_price_excluding_optional_support 열에 기록된 숫자를 대체하기를 바란다면 어떻게 할까요? 위의 예와 비슷한 논리로 다음과 같이 하면 됩니다.

```
projects |>
  mutate(total_price_excluding_optional_support = total_price_excluding_optional_support * 1200
```

```
# A tibble: 664,098 x 36
   projectid            teacher_acctid schoolid school_ncesid school_latitude
   <chr>                <chr>          <chr>    <chr>                  <dbl>
 1 316ed8fb3b81402ff6ac8f~ 42d43fa6f3731~ c0e6ce8~ 063627006187           36.6
 2 90de744e368a7e4883223c~ 864eb466462bf~ d711e47~ 483702008193           32.9
 3 32943bb1063267de6ed19f~ 37f85135259ec~ 665c361~ 410327000109           45.2
 4 bb18f409abda2f264d5acd~ 2133fc46f951f~ 4f12c3f~ 360015302507           40.6
 5 24761b686e18e5eace6346~ 867ff478a63f5~ 10179fd~ 062271003157           34.0
 6 eac7d156205f1333de3887~ ff064802c18e6~ 929336e~ 040187001058           33.3
 7 5a3bfdf2e05781ccd0654d~ 085794a9e315b~ dc34b02~ 010060000256           32.8
 8 afda16eb54d8992db7bb42~ 1d94a31c2dc38~ 86fff7c~ 130129000571           33.9
 9 2ab3efb23acc84017cd789~ 5a497c425e05b~ ed170a1~ 261200001297           42.4
10 3118962680bb062323c356~ eb0855cc6ea55~ 71b4dee~ 340264001386           40.0
```

```
# i 664,088 more rows
# i 31 more variables: school_longitude <dbl>, school_city <chr>,
#   school_state <chr>, school_zip <chr>, school_metro <chr>,
#   school_district <chr>, school_county <chr>, school_charter <lgl>,
#   school_magnet <lgl>, school_year_round <lgl>, school_nlns <lgl>,
#   school_kipp <lgl>, school_charter_ready_promise <lgl>,
#   teacher_prefix <chr>, teacher_teach_for_america <lgl>, ...
```

즉, 원래의 값에 1200을 곱한 값에 원래의 이름을 할당함으로써, 새로운 값이 원래의 값을 대체하도록 하는 것입니다.

앞서 projects 테이블에 프로젝트 목표액에 해당하는 열이 두 개라고 했지요? 앞에서 사용한 것과 다른 하나는 total_price_including_optional_support입니다. 앞에서 사용한 변수가 “optional_support”라는 것을 제외한(excluding) 금액이었다면, 두번째 변수는 “optional_support”를 포함한다(including)는 차이가 있습니다. 사실 “optional support”는 해당 프로젝트에 지원되는 금액이 아니라, ’donorschoose.org’에 지원되는 금액을 의미합니다. ’donorschoose.org’도 서비스를 유지하기 위해 직원 월급, 홈페이지 유지비 등 다양한 비용을 지불해야 할테니까요.

이제 기자가 이 “optional_support”가 전체 프로젝트 목표액에서 차지하는 비율(%)에 관심을 가지게 되었다고 합시다. 그러면 두 개의 목표액 수치를 조합해서 이 비율을 계산해야 하겠네요. 이 역시 mutate()을 이용해 쉽게 할 수 있습니다.

```
projects |>
  mutate(ratio = (total_price_including_optional_support - total_price_including_
```

```
# A tibble: 664,098 x 37
  projectid            teacher_acctid schoolid school_ncesid school_latitude
  <chr>                <chr>          <chr>    <chr>                  <dbl>
1 316ed8fb3b81402ff6ac8f~ 42d43fa6f3731~ c0e6ce8~ 063627006187            36
2 90de744e368a7e4883223c~ 864eb466462bf~ d711e47~ 483702008193
3 32943bb1063267de6ed19f~ 37f85135259ec~ 665c361~ 410327000109
```

```
 4 bb18f409abda2f264d5acd~ 2133fc46f951f~ 4f12c3f~ 360015302507          40.6
 5 24761b686e18e5eace6346~ 867ff478a63f5~ 10179fd~ 062271003157          34.0
 6 eac7d156205f1333de3887~ ff064802c18e6~ 929336e~ 040187001058          33.3
 7 5a3bfdf2e05781ccd0654d~ 085794a9e315b~ dc34b02~ 010060000256          32.8
 8 afda16eb54d8992db7bb42~ 1d94a31c2dc38~ 86fff7c~ 130129000571          33.9
 9 2ab3efb23acc84017cd789~ 5a497c425e05b~ ed170a1~ 261200001297          42.4
10 3118962680bb062323c356~ eb0855cc6ea55~ 71b4dee~ 340264001386          40.0
# i 664,088 more rows
# i 32 more variables: school_longitude <dbl>, school_city <chr>,
#   school_state <chr>, school_zip <chr>, school_metro <chr>,
#   school_district <chr>, school_county <chr>, school_charter <lgl>,
#   school_magnet <lgl>, school_year_round <lgl>, school_nlns <lgl>,
#   school_kipp <lgl>, school_charter_ready_promise <lgl>,
#   teacher_prefix <chr>, teacher_teach_for_america <lgl>, ...
```

두 목표액의 차이가 “optional support” 금액에 해당할 것이고, 이를 optional support를 포함한 목표액으로 나눈 것이 전체 목표액에서 “optional support”가 차지하는 비율이 되겠지요.

이번에는 변수의 데이터 타입을 바꾸기 위해 mutate()을 사용하는 경우를 살펴보겠습니다. projects 테이블에는 school_zip이라는 변수가 있는데요, 이는 학교 우편번호를 뜻합니다. 그런데 데이터를 로드한 결과 이 우편번호가 문자(character)로 인식되어 있습니다. 이를 숫자로 다시 인식시켜주고 싶다고 하죠. 이에 해당하는 코드는 다음과 같습니다.

```
projects <- projects |>
   mutate(school_zip = as.numeric(school_zip))
```

as.numeric() 이라는 함수가 데이터의 타입을 숫자로 바꾸라는 뜻이니, school_zip의 데이터 타입을 숫자로 바꾼다음, 그 결과물을 원래 변수명인 school_zip에 대체해 버린 것입니다. 이렇게 새로운 데이터 타입을 가지게 된 컬럼을 포함한 새 테이블을 원래 테이블 이름인 projects에 할당해 버렸습니다(projects <- 부분). 따라서 이제 projects 테이블의 school_zip 열은 문자가 아닌 숫자 입니다.

8.2.4 join

두 테이블을 연결하고 싶은 경우도 있습니다. 앞서 본 projects와 donations의 관계가 바로 그렇습니다. donations 테이블에 기록된 것은 각각의 기부들인데, 각 기부는 특정 프로젝트를 위한 것일테니, 기부에 대한 정보와 프로젝트에 대한 정보는 연결될 수 있겠지요. donations 테이블에는 projectid라는 열이 있는데, 이것이 바로 어떤 프로젝트에 대한 기부인지를 표현해 주는 정보입니다. 그리고 똑같은 이름의 열이 projects 테이블에도 존재합니다. 이 테이블에서 projectid는 각 프로젝트의 고유번호겠지요. 따라서, donations 테이블에 있는 projectid가 일치하는 프로젝트를 projects 테이블에서 찾으면 해당 프로젝트에 대한 정보를 찾을 수 있겠지요.

더 나아가, projectid를 이용해 두 테이블을 연결하는 것도 가능합니다. 이를 데이터베이스 용어로 'join'이라고 합니다.

donorschoose.org의 큰 테이블을 join하기에 앞서, 아주 간단한 예로 연습을 해 보도록 하지요. 먼저 가상의 두 테이블을 만들겠습니다.

```
left <- data.frame(Key=c(1,2,3),
                   ColA=c("R1","R2","R3"))

right <- data.frame(Key=c(1,2,4),
                    ColB=c("F1","F2","F4"))
```

각 테이블은 출력해보면 알겠지만, 두 테이블은 다음과 같이 생겼습니다.

테이블 이름이 left, right인 이유는 앞으로 작성할 코드에서 left 테이블을 먼저, 즉, 왼쪽에 써주고, right 테이블을 나중에, 즉, 오른쪽에 써 줄 것이기 때문입니다.

이 두 테이블이 Key열을 이용해 연결 가능하다는 것은 바로 아시겠지요? 다만, 문제가 있습니다. left 테이블에는 Key의 값으로 1,2,3이 있고, 오른쪽 테이블에는 1,2,4가 있습니다. 즉, 연결이 안 되는 값이 있습니다. 이 때 왼쪽, 오른쪽 중 어느 테이블을 기준으로 두 테이블을 붙여줄 것인가에 따라, 'left join'과 'right join'이 구분됩니다.

먼저 left join 입니다.

```
left_join(left, right, by="Key")
```

```
  Key ColA ColB
1   1   R1   F1
2   2   R2   F2
3   3   R3 <NA>
```

left 테이블을 기준으로 했으니, left 테이블에 있는 모든 값에, Key에 의해 매칭이 되는 right 테이블의 정보가 연결되었습니다. Key 3에 해당하는 right 테이블의 행은 없으니 ColB는 결측값으로 남아있고요. right 테이블에서 매칭이 되지 못한 Key 4에 해당하는 값은 버려졌습니다.

위의 코드는 다음과 같이 써도 무방합니다.

```
left_join(left, right, by=c("Key"="Key"))
```

```
  Key ColA ColB
1   1   R1   F1
2   2   R2   F2
3   3   R3 <NA>
```

by=c("Key"="Key")는 왼쪽 테이블의 Key열과 오른쪽 테이블의 Key열을 매칭하라는 의미입니다. 지금은 이러한 문법이 중요하지 않지만, 경우에 따라서는 매칭할 열임에도 불구하고 왼쪽 테이블과 오른쪽 테이블에서 다른 이름을 사용하고 있는 경우가 있습니다.

또 다음과 같은 코드도 같은 의미입니다.

```
left |> left_join(right, by=c("Key"="Key"))
```

```
  Key ColA ColB
1   1   R1   F1
2   2   R2   F2
3   3   R3 <NA>
```

이는 left_join() 함수의 첫번째 인수가 파이프에 의해 함수 바깥 왼쪽으로 나간 것이니 지금까지 해온 파이핑과 같은 경우 입니다. 이렇게 하면 앞서와 마찬가지로 가독성 있게 코드를 길게 늘여 쓰는데 도움이 됩니다.

이제 'right join'을 해 보지요.

```
left |> right_join(right, by="Key")
```

```
  Key ColA ColB
1   1   R1   F1
2   2   R2   F2
3   4 <NA>   F4
```

이번엔 반대로 오른쪽 테이블 기준으로 연결된 것을 알 수 있지요?

사실 join에는 몇 가지가 더 있습니다. 그 중 자주 쓰는 것으로 'full join'이 있습니다. 이것은 매칭이 안 되더라도 한 쪽 테이블에라도 있으면 일단 연결해주는 방식입니다. 즉, 어떤 정보도 버리지 않는 것이지요.

```
left |> full_join(right, by="Key")
```

```
  Key ColA ColB
1   1   R1   F1
2   2   R2   F2
3   3   R3 <NA>
4   4 <NA>   F4
```

반대로 양쪽 테이블에 모두 정보가 있어야만 연결을 해 주는 'inner join'도 있습니다.

```
left |> inner_join(right, by="Key")
```

```
  Key ColA ColB
1   1   R1   F1
2   2   R2   F2
```

이제 donorschoose.org의 두 테이블을 연결해 보도록 하지요.

일단, donations 테이블이 projects 테이블보다 훨씬 더 큰 테이블일 수 밖에 없다는 것을 확인합시다. 왜냐하면, 하나의 프로젝트에 대해서 여러개의 기부가 이루어졌을 수 있기 때문이지요. 만약 프로젝트 A에 대해서 기부가 세번 이루어졌다면 'A'라는 프로젝트의 고유값은 donations 테이블에 세번 등장할 것입니다. 따라서, donations 테이블을 기준으로 projects 테이블을 연결하는 것이 자연스럽겠지요.

대신 projects 테이블에는 21개의 열이, donations 테이블에는 35개의 열이 있으니, 연결 후 정보를 확인하기 너무 어렵습니다. 각 테이블에서 일부 관심있는 열만 뽑아서 연결한 후, 이를 connected라는 테이블로 메모리에 저장하겠습니다..

```
connected <- donations |>
   select(projectid, donation_to_project, donation_optional_support, donation_timestamp) |>
   left_join(projects |>
               select(projectid, total_price_including_optional_support),
```

```
            by = c("projectid"="projectid"))
connected
```

```
# A tibble: 3,097,989 x 5
   projectid      donation_to_project donation_optional_su~1 donation_timestamp
   <chr>                        <dbl>                  <dbl> <dttm>
 1 ffffac55ee02a~               42.5                    7.5  2011-08-25 14:27:34
 2 ffffac55ee02a~               26.8                    4.73 2011-11-04 07:54:21
 3 ffffac55ee02a~               55.4                    0    2011-11-02 22:53:53
 4 ffffac55ee02a~                8.5                    1.5  2011-11-03 23:54:01
 5 ffffac55ee02a~               20                      0    2011-11-02 23:21:00
 6 ffffac55ee02a~                4.25                   0.75 2011-10-16 20:59:09
 7 ffffac55ee02a~               50                      0    2011-08-31 08:13:55
 8 ffff97ed93720~              212.                     37.5  2010-11-12 16:11:46
 9 ffff97ed93720~              185.                     32.6  2010-12-01 14:52:17
10 ffff97ed93720~               21.2                     3.75 2010-11-23 09:09:10
# i 3,097,979 more rows
# i abbreviated name: 1: donation_optional_support
# i 1 more variable: total_price_including_optional_support <dbl>
```

조금 길어졌지만, 잘 뜯어보면 donations 테이블에서 4개의 열을 뽑아서 그것을 왼쪽 테이블로, projects 테이블에서 2개의 열을 뽑아 오른쪽 테이블로 삼은 다음, prjectid열을 이용해 left_join() 한 것에 불과합니다. 어렵지 않지요?

단, 여기서 자주 나오는 실수중 하나는 각 테이블에서 일부 열을 select()할 때 매칭할 열을 뽑지 않는 것입니다. 즉, 위의 경우라면, 양쪽 테이블에서 projectid 테이블을 각각 select() 해 두어야 연결이 가능하겠지요.

8.2.5 피벗

데이터를 분석하다보면, 같은 정보를 그대로 유지하되, 테이블의 형태만 바꾸어야 하는 경우가 있습니다. 이게 무슨 말인가 하는 분도 있겠지만, 일단 다음 두 개의 테이블을 비교해 보도록 하지요.

```
countries_longer <- data.frame(country = c("x","x","x","y","y","y","z","z","z"),
                               year = rep(c(1960, 1970, 2010), 3),
                               ranking = c(10,13,15,20,23,25,30,33,35))
countries_longer
```

```
  country year ranking
1       x 1960      10
2       x 1970      13
3       x 2010      15
4       y 1960      20
5       y 1970      23
6       y 2010      25
7       z 1960      30
8       z 1970      33
9       z 2010      35
```

```
countries_wider <- countries_longer |>
   pivot_wider(names_from="year", names_prefix="yr", values_from="ranking")
countries_wider
```

```
# A tibble: 3 x 4
  country yr1960 yr1970 yr2010
  <chr>    <dbl>  <dbl>  <dbl>
1 x           10     13     15
2 y           20     23     25
3 z           30     33     35
```

잘 보면 아시겠지만, 두 개의 테이블은 완벽하게 같은 정보를 담고 있습니다. countries_wider 테이블에 countries_longer 테이블과 다른 점이라고는 단지 year 칼럼에 들어갔던 연도 데이터가 칼럼 이름으로 바뀌었다는 점 뿐입니다. 그 차이 때문에 첫번째 테이블은 길쭉한(long) 모양새를 하고 있고, 두번째 테이블은 넓은(wide) 형태를 하고 있습니다. 따라서 첫번째 테이블 형태를 'long 포맷', 두번째 형태를 'wide 포맷'이라고 부릅니다.

두 번째 테이블을 만들 때에는 직접 데이터를 집어 넣어서 만든 것이 아니라, 첫 번째 테이블을 변형해서 만들었습니다. 이렇게 두 개의 '포맷' 사이를 왔다 갔다 하며 테이블의 형태를 바꾸는 것을 '피벗'이라고 합니다. 그리고 long 포맷을 wide 포맷으로 바꾸는 함수가 바로 pivot_wider() 입니다. 위의 사용 예에서 pivot_wider() 함수의 인수는 4개 입니다. 하나는 형태를 바꿀 데이터 그 자체(위에서는 countries_wider 테이블 입니다), 두번째는 names_from에 들어갈 칼럼 이름, 즉, 피벗이 되고 나면, 칼럼 이름으로 바뀔 부분 입니다. 세번째는 names_prefix에 들어갈 문자 입니다. 위의 예에서는 "yr" 이라고 정해주었기 때문에 새로운 테이블에서 칼럼 이름이 "yr"로 시작하게 된 것입니다. 이 인수는 사실 꼭 사용해야 하는 것은 아니지만, 사용하지 않을 경우, 위의 예에서는 칼럼 이름이, 1960, 1970, 2010 이렇게 숫자로만 어루어져 있었겠지요? 이는 나중에 에러의 원인이 될 수 있습니다. 그래서 편의상 "yr"이라는 문자를 앞에 붙여주도록 지정해 준 것이지요. 마지막으로 사용한 인수는 values_from 입니다. 이것은 wide 포맷에서 (칼럼 이름이 아닌) 테이블 안의 값들이 long 포맷의 어느 칼럼에서 와야 하는지를 정해주는 것입니다. long 포맷의 ranking 컬럼에 속한 수들이 wide 포맷에서는 yr1960, yr1970, yr2010 칼럼의 값으로 들어간 것이 보이시지요?

물론 반대로 wide 포맷을 long 포맷으로 바꿀 수도 있습니다 (사실은 이쪽이 훨씬 더 자주 사용합니다!).

```
countries_wider |>
    pivot_longer(cols=!country, names_to="year", names_prefix="yr", values_to="r
```

```
# A tibble: 9 x 3
```

	country	year	ranking
	<chr>	<chr>	<dbl>
1	x	1960	10
2	x	1970	13
3	x	2010	15
4	y	1960	20
5	y	1970	23
6	y	2010	25
7	z	1960	30
8	z	1970	33
9	z	2010	35

pivot_wider() 함수의 인수와 pivot_longer() 함수의 인수가 미묘하게 다르다는 점을 잘 봐두세요. pivot_longer() 함수의 인수는 names_from, values_from이 아니라 names_to, values_to 입니다. 해당 인수들이 무엇을 의미하는지는 pivot_wider()를 잘 이해 했다면 이 역시 잘 이해할 수 있을 것입니다. 더 중요한 차이는 cols라는 인수입니다. 이는 내가 '길게'만들어 주고 싶은 칼럼을 정해주는 것입니다. 위의 예에서 !country라고 한 것은 country 칼럼 말고 '나머지'(not) 부분을 길게 만들어달라는 뜻입니다.

지금까지의 이야기를 요약하면 다음 그림과 같습니다.

그림 여기서 가져오기. https:// tavareshugo.github.io/ r-intro-tidyverse-gapminder/09-reshaping/index.html

그런데, 이런 피봇은 왜 하게 될까요? 여러 이유가 있겠지만, 가장 중요한 이유는 내가 분석이나 시각화를 위해 사용하고 싶은 함수가 특정 포맷으로 되어 있는 데이터를 요구하기 때문입니다. 단적으로, 앞으로 이용할 시각화 툴들은 long 포맷을 요구합니다. 하지만, 만약 여러분이 wide 포맷의 데이트를 입수했다면 피봇을 해야 하는 것이지요.

9 데이터 요약하기

R의 기본 문법과 데이터를 R로 불러들이는 방법까지 배웠으니, 이제 본격적으로 데이터 분석을 시작할 차례입니다. 이 장의 제목은 '데이터 요약하기' 이지만, 사실 데이터 분석이라는 것은 어떻게 생각하면 요약 그 이상도 이하도 하닙니다. 예컨대, '대통령의 국정 수행 지지율은 40%이다'라는 분석은 수천명에 달하는 설문 응답자의 각기 다른 응답을 지지하는 비율 이라는 하나의 척도로 요약한 것이라고 할 수 있습니다. 물론, 이 설문조사가 전체 대한민국 국민, 또는 투표권을 가진 국민의 의견을 대표하는지는 불확실하기 때문에, '표본오차' (±2%) 같은 추가적인 정보를 더하기도 합니다만, 이 역시 많은 데이터 수에 비하면 매우 간단하게 요약된 숫자이지요. 따지고 보면, 넓게는 데이터 과학, 좁게는 데이터 저널리즘이 제시하는 대부분의 분석은 큰 데이터를 몇 가지의 숫자로 요약하고, 이를 특정 그룹 간에 비교한 것에 지나지 않습니다. 이를 통계에서는 멋지게 '기술통계(descriptive statistics)'라고 부리기도 합니다만, 그러한 용어법이 중요한 것은 아닙니다.

우리가 다음 장에서 볼 시각화는 이렇게 요약된 숫자를 그래프, 인포그래픽과 같은 시각 요소로 표현한 것에 지나지 않습니다. 즉, 데이터 요약하기는 분석의 시작이자 끝이라고 할 수 있죠.

우리가 앞서 배운 tidyverse는 데이터를 요약하기 위한 간단하면서도 강력한 문법들을 제공합니다. 이제 여기에 대해서 알아보도록 하죠.

일단 이번 챕터에서 사용할 수 있도록 donorschoose.org 데이터를 불러보도록 하지요.

```
library(tidyverse)
```

```
projects <- read_csv("data/projects.csv")
```

9.1 summarise

tidyverse 문법에서 데이터를 요약할 때에는 그게 어떤 방식의 요약이든간에, 요약을 한다고 선언해 주어야 합니다. 그 선언은 간단하게 summarise()라는 함수를 사용하면 됩니다. 그리고 그 함수 안에 '어느 변수를' '어떤 방식으로' 요약할 것인지만 써주면 됩니다.

바로 다음과 같이 말이지요.

```
projects |>
    summarise(avg_size = mean(total_price_excluding_optional_support),
              avg_opt = mean(total_price_including_optional_support -
                             total_price_excluding_optional_support))
```

```
# A tibble: 1 x 2
  avg_size avg_opt
     <dbl>   <dbl>
1     542.    103.
```

첫번째 줄의 의미는, total_price_excluding_optional_support 칼럼의 평균(mean())을 낸 다음 이를 avg_size라고 부르자는 의미 입니다. 두번째는 total_price_including_optional_support에서 total_price_excluding_optional_support을 뺀 값들의 평균을 낸 다음 이를 avg_opt라고 부르자는 의미입니다. 이렇게, 여러개의 요약을 한 번에 할 수 있습니다.

어떻게 요약을 할 것인지에 해당하는 함수로 이번에는 평균, mean()을 사용했지만 그 외에도 여러가지가 있습니다.

함수	기능
mean()	평균
median()	중앙값
sd()	표준편차
n()	갯수
max()	최대값
min()	최소값

9.2 그룹별 요약

전체 데이터를 요약하는 것이 유용할 때도 많지만, 사실 데이터 분석에서 훨씬 더 자주 사용하는 것은 그룹별 요약 입니다. 예컨대, 대한민국 1인당 국민소득이 관심사일 수도 있지만, 지역별 1인당 국민소득에 관심을 가질 때도 있는 것이지요.

특히, 의미 있는 분석 결과를 도출하기 위해서는 특정 그룹 사이를 비교하는 것이 효과적이기 때문에, 이러한 그룹별 비교는 더욱 자주 사용하게 됩니다. 앞서의 예에서라면, 전라남도와 경상남도의 1인당 국민소득을 비교하는 것도 그룹별 비교이지만, 연도를 하나의 그룹이라고 생각한다면, 연도별 1인당 국민소득을 비교하는 것 역시 그룹별 비교라고 할 수 있습니다.

tidyverse 문법에서 그룹별 요약을 하기 위해서는 summarise() 함수 앞에 group_by() 함수를 사용해 주면 됩니다. 단, group_by() 함수의 인수로 요약에 사용하고 싶은 그룹 정보가 담긴 컬럼명을 표시해주면 됩니다.

```
projects |>
    group_by(school_state) |>
    summarise(avg_size = mean(total_price_excluding_optional_support))
```

```
# A tibble: 52 x 2
   school_state avg_size
   <chr>           <dbl>
 1 AK              516.
 2 AL              507.
 3 AR              524.
 4 AZ              500.
 5 CA              599.
 6 CO              505.
 7 CT              480.
 8 DC              511.
 9 DE              411.
10 FL              488.
# i 42 more rows
```

이렇게 하면, donorschoose.org에 생성된 프로젝트들의 주별 평균 목표액 크기가 구해진 것입니다.

이렇게 group_by()를 이용한 요약에서 두 가지를 이해하면 좋습니다.

첫째, group_by()까지 실행했을 때는 것으로 보기에 아무 일도 일어나지 않습니다. 이를 확인하기 위해서 위의 코드에서 일부만을 실행해 보겠습니다.

```
projects |>
  group_by(school_state)
```

```
# A tibble: 664,098 x 35
# Groups:   school_state [52]
   projectid            teacher_acctid schoolid school_ncesid school_latitude
   <chr>                <chr>          <chr>    <chr>                  <dbl>
 1 316ed8fb3b81402ff6ac8f~ 42d43fa6f3731~ c0e6ce8~ 063627006187            36
 2 90de744e368a7e4883223c~ 864eb466462bf~ d711e47~ 483702008193
 3 32943bb1063267de6ed19f~ 37f85135259ec~ 665c361~ 410327000109
```

```
 4 bb18f409abda2f264d5acd~ 2133fc46f951f~ 4f12c3f~ 360015302507          40.6
 5 24761b686e18e5eace6346~ 867ff478a63f5~ 10179fd~ 062271003157          34.0
 6 eac7d156205f1333de3887~ ff064802c18e6~ 929336e~ 040187001058          33.3
 7 5a3bfdf2e05781ccd0654d~ 085794a9e315b~ dc34b02~ 010060000256          32.8
 8 afda16eb54d8992db7bb42~ 1d94a31c2dc38~ 86fff7c~ 130129000571          33.9
 9 2ab3efb23acc84017cd789~ 5a497c425e05b~ ed170a1~ 261200001297          42.4
10 3118962680bb062323c356~ eb0855cc6ea55~ 71b4dee~ 340264001386          40.0
# i 664,088 more rows
# i 30 more variables: school_longitude <dbl>, school_city <chr>,
#   school_state <chr>, school_zip <chr>, school_metro <chr>,
#   school_district <chr>, school_county <chr>, school_charter <lgl>,
#   school_magnet <lgl>, school_year_round <lgl>, school_nlns <lgl>,
#   school_kipp <lgl>, school_charter_ready_promise <lgl>,
#   teacher_prefix <chr>, teacher_teach_for_america <lgl>, ...
```

이는 전체 projects 테이블을 출력하는 것과 동일합니다. 다만, 좌측 상단에 보면 'Groups: school_state [52]'라는 표시가 있습니다. 즉, group_by() 함수를 실시하면 이용자가 지정한대로 그룹만 만들어둔 상태가 되는 것입니다. 실제로 효과가 나타나는 것은 summarise() 함수나 다른 함수를 이용해서 그룹별 연산을 한 다음이지요.

한 가지만 더 이야기 하자면, 코드의 일부분만을 실행시키기 위해서는 위와 같이 부분을 새로 작성해서 실행해 주어도 되지만, 실행시키고 싶은 부분만으로 블록 설정한 다음 'Alt+Enter'를 눌러주어도 됩니다. 이는 사실 여러분들의 코드에서 에러가 발생했을 때 버그가 있는 부분을 찾는 좋은 방법이기도 합니다.

둘째, summarise() 함수를 이용해 요약한 결과 그 자체도 테이블 입니다. 따라서 그 결과물에도 서브세트, 요약, 피봇 등의 작업을 다시 적용할 있습니다. 바로 다음과 같은 경우가 그렇습니다.

```
projects |>
  group_by(school_state) |>
  summarise(avg_size = mean(total_price_excluding_optional_support),
```

```
            count = n()) |>
    filter(avg_size > 600)
```

```
# A tibble: 3 x 3
  school_state avg_size count
  <chr>           <dbl> <int>
1 HI              703.   2586
2 La              654.      3
3 NY               705. 73182
```

이렇게 하면, 평균 프로젝트 목표액이 600달러를 초과하는 주만 결과를 확인할 수 있네요.

9.3 날짜 정보를 이용한 그룹별 요약

모든 프로그램 언어에서 날짜는 에러를 자주 발생시키는 골치덩어리 입니다. 하지만, 날짜 정보는 데이터 분석에서 빠질 수 없는 요소이기도 하지요. 제일 흔하게 하는 분석 중 하나가 '연도별 추이' 따위를 계산하는 것인데, 이는 앞서도 언급했던 것처럼 연도에 따라 그룹별 평균을 내는 것에 지나지 않습니다.

다행히 R에는 날짜 정보를 쉽게 다루기 위한 패키지 lubridate이 있습니다. 해당 패키지는 다른 패키지들과 유사하게 다음과 같이 설치합니다.

```
install.packages("lubridate")
```

설치가 완료되면, 패키지를 불러옵니다.

```
library(lubridate)
```

우리가 지금까지 사용해 온 projects 테이블에서 날짜 정보, 즉, 프로젝트가 언제 만들어졌는가는 date_posted 컬럼에 포함되어 있습니다. 먼저, 이 칼럼의 데이터 타입을 확인해 볼까요?

```
str(projects$date_posted)
```

```
 Date[1:664098], format: "2014-05-12" "2014-05-12" "2014-05-11" "2014-05-11" "2014-05-11"
```

Date라는 데이터 타입을 가지고 있네요. 즉, R이 이미 이 컬럼에 담겨있는 값들을 날짜로 인식하고 있다는 것입니다. 하지만, 이는 운이 좋은 경우입니다. 많은 경우, R은 사람이 보기에는 날짜가 분명한 값들은 일반 문자라고 인식하는 경우가 많습니다. 이러한 "운이 덜 좋은" 경우를 가정하기 위해 다음과 같이 데이터 타입을 바꾸어 줍시다.

```
projects <- projects |>
    mutate(date_posted = as.character(date_posted))
str(projects$date_posted)
```

```
 chr [1:664098] "2014-05-12" "2014-05-12" "2014-05-11" "2014-05-11" ...
```

이제 데이터 타입이 문자(chr)로 바뀐 것이 보이지요? 이렇게 되면, 날짜 사이의 간격을 계산할 수도, 날짜에서 연도만 추출할 수도 없을것입니다. 하지만, lubridate의 기능을 이용하면 언제든지 이를 다시 날짜 타입으로 바꾸어 줄 수 있습니다.

문자를 날짜로 인식시켜주기 위한 함수는 여러개가 있는데, 이는 날짜를 표현하는 포맷에 따라 달라집니다. 위의 경우에는 "년(year)-월(month)-일(day)"의 순서대로 표시하는 방식이니 해당 함수의 이름은 ymd() 입니다.

```
projects <- projects |>
    mutate(date_posted = ymd(date_posted))
str(projects$date_posted)
```

```
 Date[1:664098], format: "2014-05-12" "2014-05-12" "2014-05-11" "2014-05-11
```

이제 다시 데이터 타입이 Date으로 돌아왔습니다. 만약 날짜를 표시하는 방식이 “2014-05-11”이 아니고, “May 11, 2014” 였다면, 해당 함수는 mdy()겠지요. 영국식으로 “11/5/2014” 였다면, dmy()였을 테고요.

R이 date_posted를 날짜로 인식하고 있기 때문에, 여기서 연도를 추출하는 것도 어렵지 않습니다.

```
projects |>
    mutate(year = year(date_posted)) |>
    select(year, date_posted)
```

```
# A tibble: 664,098 x 2
    year date_posted
   <dbl> <date>
 1  2014 2014-05-12
 2  2014 2014-05-12
 3  2014 2014-05-11
 4  2014 2014-05-11
 5  2014 2014-05-11
 6  2014 2014-05-11
 7  2014 2014-05-11
 8  2014 2014-05-11
 9  2014 2014-05-11
10  2014 2014-05-11
# i 664,088 more rows
```

이제 year 칼럼을 이용해 연도별 평균 프로젝트 목표액을 구해보도록 하죠.

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(year) |>
    summarise(avg_size = mean(total_price_excluding_optional_support))
```

```
# A tibble: 13 x 2
   year avg_size
  <dbl>    <dbl>
 1  2002     609.
 2  2003     952.
 3  2004     435.
 4  2005     641.
 5  2006     693.
 6  2007     547.
 7  2008     443.
 8  2009     625.
 9  2010     484.
10  2011     478.
11  2012     516.
12  2013     587.
13  2014     626.
```

연도별 평균을 주(state)에 따라 따로 계산할 수도 있을까요? 물론입니다. group_by()를 두 변수에 대해서 해 주면 되지요.

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(school_state, year) |>
    summarise(avg_size = mean(total_price_excluding_optional_support))
```

`summarise()` has grouped output by 'school_state'. You can override using the `.groups` argument.

```
# A tibble: 455 x 3
# Groups:   school_state [52]
   school_state  year avg_size
   <chr>        <dbl>    <dbl>
 1 AK            2007     380.
 2 AK            2008     414.
 3 AK            2009     412.
 4 AK            2010     419.
 5 AK            2011     442.
 6 AK            2012     462.
 7 AK            2013     577.
 8 AK            2014     522.
 9 AL            2003     339.
10 AL            2004     546.
# i 445 more rows
```

이렇게 다양한 방식으로 요약한 결과를 그래프로 표현하면, 그것이 바로 시각화 입니다. 드디어 시각화를 배울 준비가 되었습니다.

10 시각화

앞선 챕터에서 연습한 그룹별 요약까지 이루어지고 나면, 시각화는 단지 요약된 결과를 시각 정보로 전달하는 것에 불과합니다.

사실 R에는 tidyverse 방식의 시각화 말고도 강력한 시각화 기능이 포함되어 있는데요, tidyverse에 포함된 ggplot2라는 시각화 패키지는 그 기능이 너무 출중하여,웹 뿐 아니라 출판을 위한 시각화 자료 생성에 사실상 기본적인 R의 시각화 기능을 대체하는 표준 문법으로 자리잡았습니다. 심지어 R이 아닌 다른 언어를 쓰는 데이터 과학자들도 시각화만큼은 ggplot2를 사용하는 경우가 있을 정도 입니다.

10.1 ggplot2의 계층(layer) 개념

ggplot2는 상당히 독특한 문법 체계를 가지고 있습니다. 일단, ggplot2만의 독특한 계층(layer)체계를 이해해야 합니다. ggplot2는 시각화를 데이터로부터 최종 시각 표현까지 도달하기 위해 순서대로 계층을 쌓는 과정으로 이해합니다. 그 계층들은 다음과 같습니다.

1. Data
2. Aesthetic
3. Geometries
4. Facets
5. Statistics
6. Coordinates
7. Theme

다음 그림은 이를 표현하고 있습니다.

계층 그림

1. Data 계층은 ggplot2로 시각화하고 싶은 테이블 형태의 데이터를 의미합니다. 해당 데이터에 포함된 각 컬럼은 '변수'를 나타내는데, 시각화에 대해 이야기 할 때는 데이터에 포함된 변수를 외부변수라고 부르겠습니다.
2. 그 위에 Aesthetics 계층이 얹혀집니다. Aesthetics 계층은 Data 계층에 포함된 외부변수와 ggplot2가 스스로 이해할 수 있는 내부변수 간의 연결을 만들어 줍니다. 그러한 내부변수들로는 다음과 같은 것들이 있습니다.

    - x: x축 값으로 표현하는 변수
    - y: y축 값으로 표현하는 변수
    - color: 선 색깔로 표현하는 변수
    - fill: 면적의 색깔로 표현하는 변수
    - size: 크기로 표현하는 변수
    - linetype: 선의 모양으로 표현하는 변수
    - group: 그룹을 정의하는 변수 Aestheics 계층에서 이용자는 데이터에 포함된 컬럼, 즉, 외부 변수 중 어느 것들이 이들 내부 변수에 해당할 것인지를 지정해 주어야 합니다.

    그림: 외부변수-내부변수
3. Geometries 계층은 그래프 타입이라고 생각하면 됩니다. 선 그래프, 바 그래프, 파이 차트 등이 대표적인 그래프 타입들입니다.
4. Facets계층은 여러개의 그래프를 동시에 표현하는 것을 의미합니다. 예컨대, 한국과 미국에 대해 비교를 같은 종류의 그래프를 나란히 그려주는 것 입니다.

    그림: facets
5. Statistics 계층은, 시각화 전에 ggplot2가 자체적으로 특정 계산을 해 주는 것을 의미합니다.

6. Coordinates 계층은 말 그대로 좌표계를 선택하는 것입니다. 우리가 일반적으로 알고 있는 유클리디안 좌표계 말고도, 극 좌표계와 같이 특수한 좌표계를 이용할 수도 있고, 일반 좌표계에서도 x축의 간격과 y축의 간격을 다르게 하거나, x축과 y축을 바꾸어버릴 수도 있습니다.

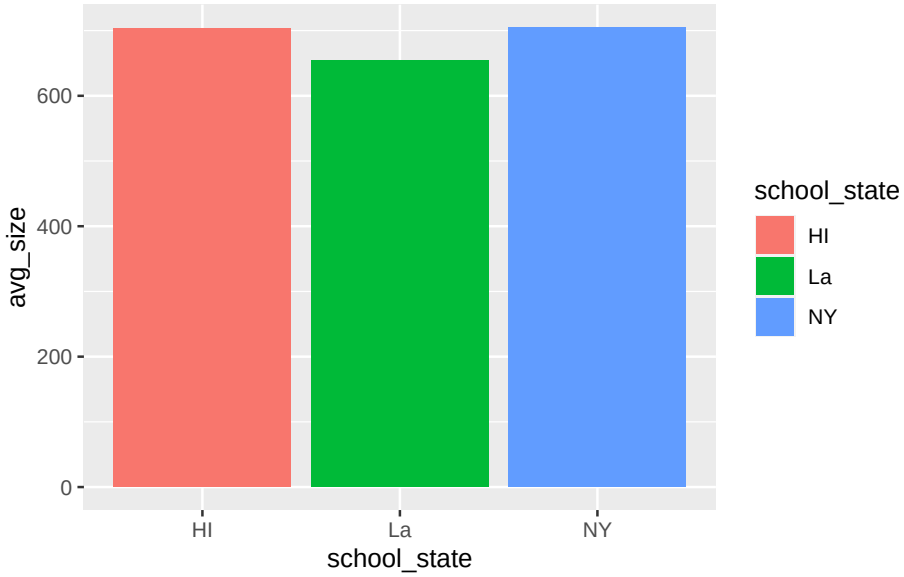
7. 마지막으로 Theme 계층은 그 외 시각화의 미적 부분을 정의합니다.

10.2 ggplot2 사용해보기

이제 바로 gglot2를 사용해 앞서 만들어낸 그룹별 요약을 바로 시각화 해보겠습니다.

```
library(tidyverse)
projects <- read_csv("data/projects.csv")

projects |>
    group_by(school_state) |>
    summarise(avg_size = mean(total_price_excluding_optional_support),
              count = n()) |>
    filter(avg_size > 600) |>
    ggplot() +
    aes(x=school_state, y=avg_size, fill=school_state) +
    geom_bar(stat="identity")
```



ggplot()으로 시작하는 코드가 나오기 전까지의 코드는 이전 챕터에서 만들어낸 그룹별 요약 + 서브세트와 정확하게 동일합니다. 즉, 앞서 만들어낸 그룹별 요약는데 이어서, ggplot2의 계층을 규칙대로 쌓아주기만 하면, 시각화 자료가 만들어집니다. 그러면 ggplot2 문법 부분을 차례대로 설명해 볼까요? 1. ggplot(): 이제부터 ggplot2를 이용해 시각화를 시작한다는 의미입니다. 이 앞의 코드를 통해 만들어 낸 그룹별 요약의 결과 역시 데이터 프레임이라고 했고, 이를 파이프를 통해 ggplot() 함수와 연결 시켰기 때문에, 앞서 만든 요약 결과가 ggplot2를 통한 시각화의 'Data' 계층에 해당한다는 것을 알 수 있습니다. 2. aes(x=school_state, y=avg_size, fill=school_state): aes()는 Aesthetics 계층을 의미합니다. 즉, 여기서는 '내부변수'인 x, y, fill을 '외부변수'인 school_state, avg_size, school_state와 연결시키고 있는 것입니다. 물론 이들 외부 변수 이름은 Data 계층에 존재하지 않는다면 에러가 발생할 것입니다. x와 fill 모두에 school_state이 연결되어 있다는 것에도 주목해보세요. x와 school_state이 연결되었기 때문에, x축의 한 자리마다 하나의 주에 해당하는 바 그래프가 그려진 것이지만, 또, fill과 school_state도 연결되었기

때문에, 각 주는 다른 색깔의 바 그래프를 가지게 된 것입니다. 내부변수 fill을 사용하지 않으면 어떻게 되는지 확인하고 싶다면, fill=school_state 부분을 지우고 다시 실행시켜보세요. 3. geom_bar(stat="identity"): 먼저 geom_bar()는 Geometries 계층으로 바 그래프를 선택했다는 것을 의미합니다. 그런데 그 안에 stat="identity"라는 인수가 있지요? 이는 Statistics 계층에 대한 조작입니다. geom_bar() 함수는 우리가 아무런 지시도 하지 않으면 기본 동작으로 Data 계층에서 숫자를 센다음 이를 내부 변수 y로 삼으려는 계산 즉, Statistics 계층에서의 행동을 기본 값으로 가지고 있습니다. 다른 geometries 함수들은 대개 그렇지 않은데, 이는 geom_bar() 함수의 특성입니다. 그런데, 우리는 우리가 원하는 y 값으로 avg_size를 미리 계산해 두었지요? 따라서, stat="identity"라는 옵션을 줌으로써 이용자가 Data 계층을 통해 공급한 외부 변수와, Aestheics 계층에서 만든 내부변수 y와의 연결을 "시키는 그대로(=identity)" 받아들이라고 명령하는 것입니다.

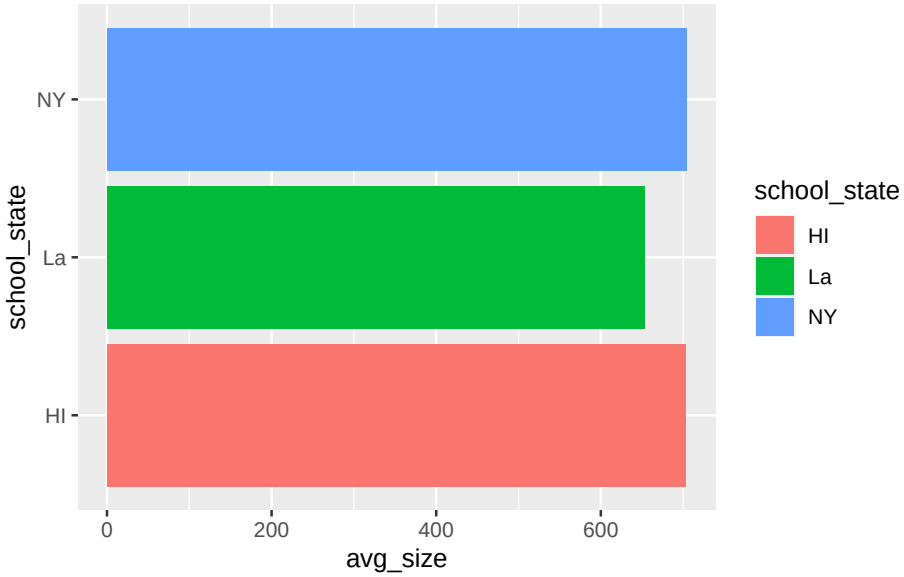
여기까지 해서 우리는 위의 코드를 통해 Statistics 계층까지를 지정해준 것입니다. 그 외의 계층은 별도로 건드리지 않으면 기본값에 따라 시각화 표현을 만들어 줍니다.

또 하나 유의해야 할 것은 ggplot()이라고 시각화 코드가 시작하면, 거기서부터는 |>라는 지금까지 써왔던 파이프 대신 + 를 파이프로 사용한다는 것입니다. 이를 잊어버리는 것도 에러를 발생시키는 흔한 원인이 되니 유의하기 바랍니다.

경우에 따라서는 막대 그래프를 세로가 아니라 가로로 그리고 싶을 때도 있습니다. 그럴 때는 어떻게 하면 좋을까요? 내부변수 x에 대응하는 외부변수와 y에 대응하는 외부변수를 바꿔지기만 하면 됩니다. 즉, Aesthetics 계층만 살짝 수정해부면 되는 것이지요. 이제 왜 ggplot2의 계층 구조가 편리한 것인지 조금 감이 오실 것입니다. 다음 코드가 앞의 코드와 다른 점을 잘 찾아보세요.

```
projects |>
    group_by(school_state) |>
    summarise(avg_size = mean(total_price_excluding_optional_support),
```

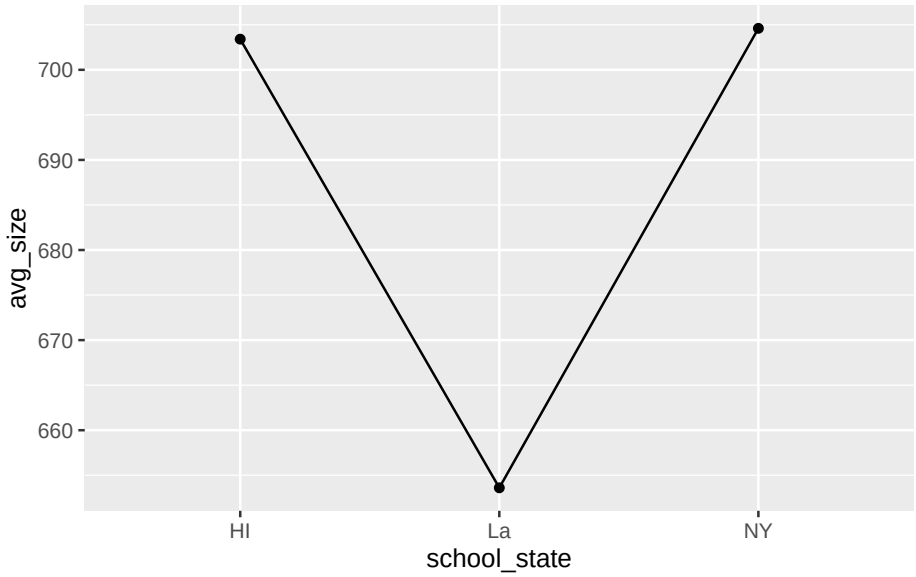
```
            count = n()) |>
  filter(avg_size > 600) |>
  ggplot() +
  aes(x=avg_size, y=school_state, fill=school_state) +
  geom_bar(stat="identity")
```



10.3 선그래프와 시간 데이터의 표현

이번에는 선 그래프를 그려보도록 하겠습니다. 선 그래프를 그리기 위해서는 Geometries 계층만 선그래프에 해당하는 것으로 바꿔주면 됩니다. geom_bar()를 geom_line()으로 대체해주는 것이지요.

```
projects |>
    group_by(school_state) |>
    summarise(avg_size = mean(total_price_excluding_optional_support),
              count = n()) |>
    filter(avg_size > 600) |>
    ggplot() +
    aes(x=school_state, y=avg_size) +
    geom_line(aes(group=1)) +
    geom_point()
```



이 그래프 역시 맨 처음의 막대 그래프와 거의 아무런 차이가 없습니다. 단, Geometries를 변경하기 위해 geom_bar() 대신 geom_line()을 사용해 주었지요. 그리고, geom_line() 안에도 Aesthetics, 즉, aes() 함수가 있습니다. 이 역시 내부 변수와 외부 변수를 연결하는 역할을 합니다. group 역시 ggplot2가 사용하는 내부변수라고 위에서 설명했지요? 그런

데, 왜 geom_line() 안에 있을까요? 해당 연결은 그래프 전체를 그릴 때가 아니라, 선을 그릴 때만 유효했으면 하기 때문입니다. '선을 그릴 때만'이라니, 다른 것도 그린다는 이야기인가요? 그렇습니다. 위의 코드에서 geom_line() 뒤에 바로 geom_point()를 덧붙였지요? 이렇게 여러게의 Geometries 계층을 덧붙일 수 있다는 것을 잘 알아두세요! 선 뿐만 아니라 점도 함께 그린 것입니다(만약 이해가 잘 안 된다면, + geom_point() 부분만 지우고 실행시켜보세요). 그런데, 위의 코드는 (선을 그릴 때가 아니라) 점을 그릴 때에는 group이라는 내부 변수에 무언가를 연결한 것이 유효하지 않다는 것을 의미합니다.

그런데 group=1은 무엇을 의미하는 것일까요? 사실 여기에서 설명하기엔 조금 복잡한 세부사항이 숨어있지만, '그래프에 존재하는 모든 점이 하나의 그룹이라고 생각해라'라는 것을 의미한다고 일단 이해하시면 좋겠습니다. geom_line()은 '선을 그린다'는 행위만을 알고 있을 뿐이지, (많은 경우) 어떤 점들을 이어서 선을 그어야 하는지는 모르거든요. 그래서 여기서는 모든 점을 다 이어서 선을 그리라는 의미로 group=1이라고 한 것입니다.

또 하나 첫번째 코드와의 차이점은 stat="identity"라는 표현이 이번에는 없다는 것입니다. geom_bar()를 사용할 때 stat="identity"라는 옵션은 데이터의 숫자를 세는 geom_bar()가 기본값으로 가지는 행동을 억누르고, 주는 데이터를 있는 그대로 받아들이라는 뜻이라고 했지요? 그런데, geom_line()은 원래부터, 주는 데이터를 그대로 받아들입니다. 즉, geom_line()에게 stat="identity"는 기본 옵션인 것이지요. 따라서, stat="identity"라는 표현은 이 경우 써도 그만, 안 써도 그만 입니다.

그런데, 여러분은 새로 그린 선 그래프가 막대 그래프보다 마음에 드나요? 저라면 이 경우에는 선 그래프를 추천하지 않을 것 같습니다. 나중에 데이터 스토리텔링을 이야기 하면서 좀 더 자세히 배우겠지만, 이렇게 선으로 연결된 점들을 보면, 인간의 뇌는 이 세 점 사이에 어떠한 연속성, 순서가 있는 것으로 자연스럽게 여기는 경향이 있습니다. 하지만, 위의 선 그래프에 표현된 세 개의 주 사이에는 어떠한 연속성도, 순서도 없지요. 이를, '범주형(categorical) 자료' 또는 '명목(nominal) 변수'라고 합니다. 즉, x축에 범주형 자료가 와야 할 때는 선 그래프가 잘 어울리지 않는 것이지요.

그렇다면, 언제 선 그래프가 잘 어울릴까요? 여러분도 익숙하다시피, x 축에 시간이 올 때 입니다. 그러면 시간 데이터를 이용한 선 그래프를 그려볼까요? 시간 데이터는 장에서 한 번 다룬 바가 있습니다. 연도별 프로젝트 크기를 구하는 코드를 그대로 가져와 보도록 하지요.

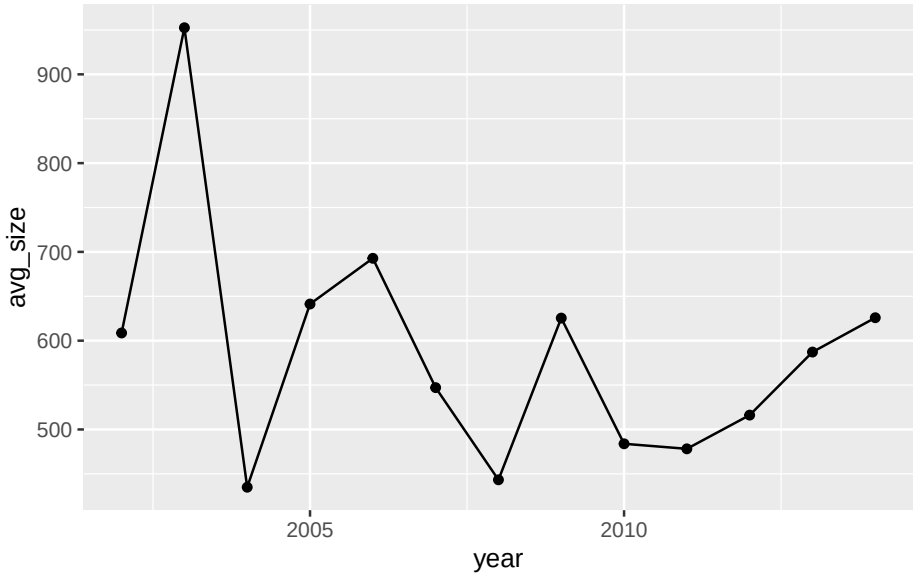
```
library(lubridate)
```

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(year) |>
    summarise(avg_size = mean(total_price_excluding_optional_support))
```

```
# A tibble: 13 x 2
    year avg_size
   <dbl>    <dbl>
 1  2002     609.
 2  2003     952.
 3  2004     435.
 4  2005     641.
 5  2006     693.
 6  2007     547.
 7  2008     443.
 8  2009     625.
 9  2010     484.
10  2011     478.
11  2012     516.
12  2013     587.
13  2014     626.
```

이제 자연스러운 '순서를 갖는' year 변수가 있으니, 선그래프가 적합해 보입니다.

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(year) |>
    summarise(avg_size = mean(total_price_excluding_optional_support)) |>
    ggplot() +
    aes(x=year, y=avg_size) +
    geom_line() +
    geom_point()
```



x축에 시간이 있으니, 조금 더 익숙한 모양의 그래프가 되지요? 우리는 자연스럽게 시간에 따른 '경향(trend)'를 파악할 수 있습니다.

물론 지금까지 배운 것들을 결합하면 여러개의 경향을 한 번에 표현할 수도 있습니다. 장에서 다룬 코드를 다시 한 번 가져와 보도록 하지요.

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(school_state, year) |>
    summarise(avg_size = mean(total_price_excluding_optional_support))
```

`summarise()` has grouped output by 'school_state'. You can override using the `.groups` argument.

```
# A tibble: 455 x 3
# Groups:   school_state [52]
   school_state  year avg_size
   <chr>        <dbl>    <dbl>
 1 AK            2007     380.
 2 AK            2008     414.
 3 AK            2009     412.
 4 AK            2010     419.
 5 AK            2011     442.
 6 AK            2012     462.
 7 AK            2013     577.
 8 AK            2014     522.
 9 AL            2003     339.
10 AL            2004     546.
# i 445 more rows
```

보다시피, 연도 뿐만이 아니라, 각 주에 대해서도 프로젝트 목표액 평균이 구해졌습니다. 모든 주에 대해 그래프를 그리려면 너무 많을 테니, 앞서 분석한 세 개의 주 중, 하와이(HI)와 뉴욕(NY)에 대한 데이터만 filter()를 이용해 서브세팅 하기로 하지요.

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(school_state, year) |>
```

```
    summarise(avg_size = mean(total_price_excluding_optional_support)) |>
    filter(school_state %in% c("HI", "NY"))
```

`summarise()` has grouped output by 'school_state'. You can override using the `.groups` argument.

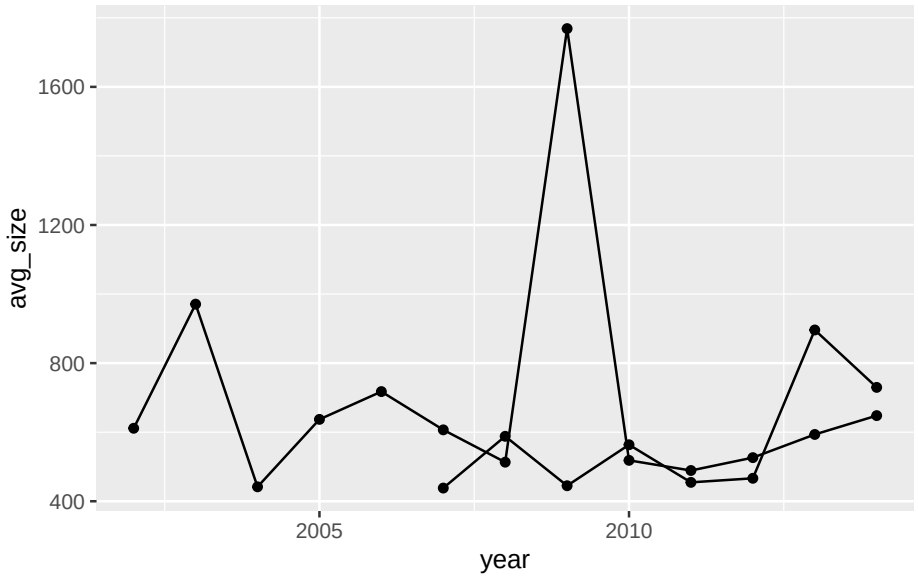
```
# A tibble: 21 x 3
# Groups:   school_state [2]
   school_state  year avg_size
   <chr>        <dbl>    <dbl>
 1 HI           2007     438.
 2 HI           2008     588.
 3 HI           2009     445.
 4 HI           2010     564.
 5 HI           2011     455.
 6 HI           2012     466.
 7 HI           2013     896.
 8 HI           2014     730.
 9 NY           2002     611.
10 NY           2003     971.
# i 11 more rows
```

%in%을 이용한 서브세팅에 주목해 보세요. 이제 Aesthetics 계층에서 x축과 연결할 데이터와 y축에 연결할 변수 뿐만 아니라, 다른 선에 연결할 변수, school_state도 구해졌습니다. 앞서 보았던 group 변수를 이용해서 다음과 같이 표현해 봅시다.

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(school_state, year) |>
    summarise(avg_size = mean(total_price_excluding_optional_support)) |>
    filter(school_state %in% c("HI", "NY")) |>
```

```
ggplot() +
aes(x=year, y=avg_size) +
geom_line(aes(group=school_state)) +
geom_point()
```

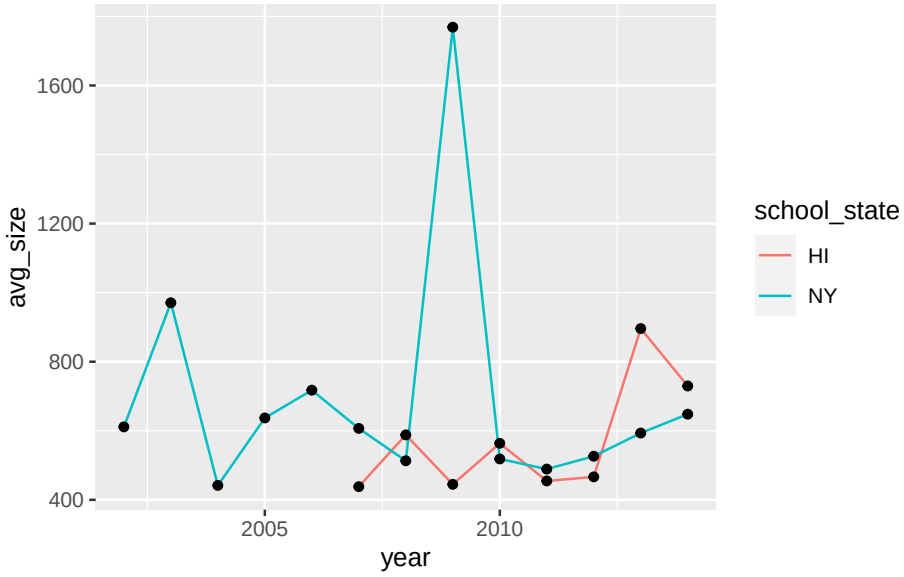
`summarise()` has grouped output by 'school_state'. You can override using the `.groups` argument.



group=school_state라고 써 줌으로써, geom_line()에게 뉴욕주는 뉴욕주끼리, 하와이주는 하와주끼리 연결하라고 한 것입니다. 하지만, 이렇게 하면 문제가 하나 있습니다. 어느 선이 어느 주인지 알 수가 없다는 것이죠. 그래서 이번에는 group 변수 대신에 color 변수를 사용해 보겠습니다.

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(school_state, year) |>
    summarise(avg_size = mean(total_price_excluding_optional_support)) |>
    filter(school_state %in% c("HI", "NY")) |>
    ggplot() +
    aes(x=year, y=avg_size) +
    geom_line(aes(color=school_state)) +
    geom_point()
```

`summarise()` has grouped output by 'school_state'. You can override using the `.groups` argument.

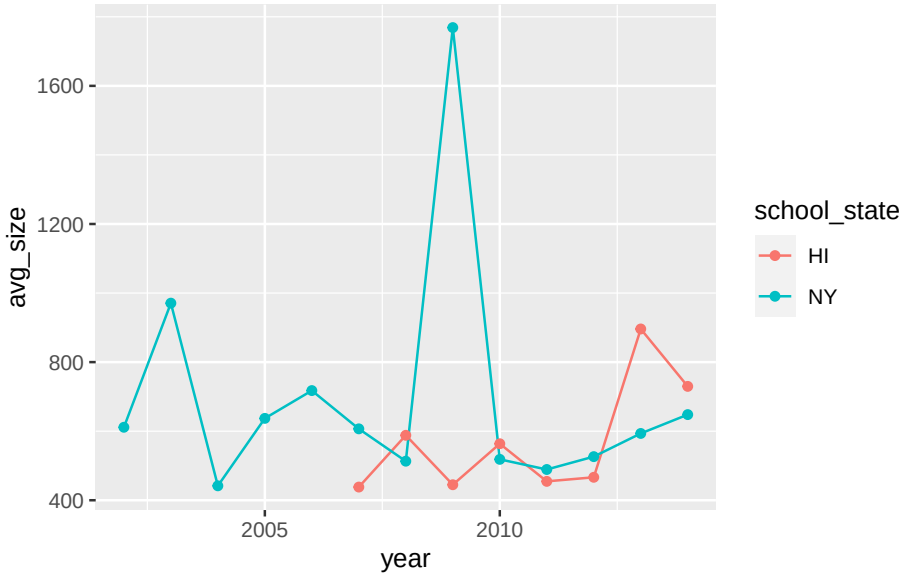


좀 더 명확하지요? 단, 점은 검은 색이라서 어색하긴 합니다. 왜냐하면 앞서 이야기 한 것처럼 color변수에 대한 설정을 geom_line()에만 적

용하고, geom_point()에는 적용되지 않았기 때문입니다. 이럴 때에는 color=school_state를 ggplot 전체에 적용되는 Aesthetics 계층으로 옮겨주면 됩니다.

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(school_state, year) |>
    summarise(avg_size = mean(total_price_excluding_optional_support)) |>
    filter(school_state %in% c("HI", "NY")) |>
    ggplot() +
    aes(x=year, y=avg_size, color=school_state) +
    geom_line() +
    geom_point()
```

`summarise()` has grouped output by 'school_state'. You can override using the `.groups` argument.

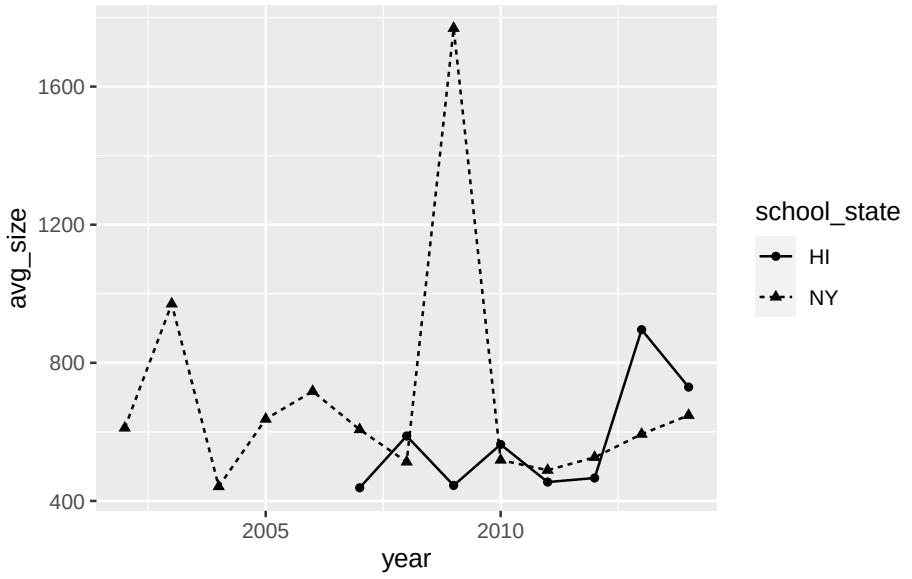


요즘에는 그럴 일이 별로 없지만, 만약 흑백 매체라면, 이렇게 그룹별로 색깔만 달리한 그래프는 차이를 표현해 주지 못하겠지요? 그럴 때는 linetype, shape과 같은 변수를 이용하면 됩니다.

```
projects |>
    mutate(year = year(date_posted)) |>
    group_by(school_state, year) |>
    summarise(avg_size = mean(total_price_excluding_optional_support)) |>
    filter(school_state %in% c("HI", "NY")) |>
    ggplot() +
    aes(x=year, y=avg_size, linetype=school_state, shape=school_state) +
    geom_line() +
    geom_point()
```

`summarise()` has grouped output by 'school_state'. You can override using the

`.groups` argument.

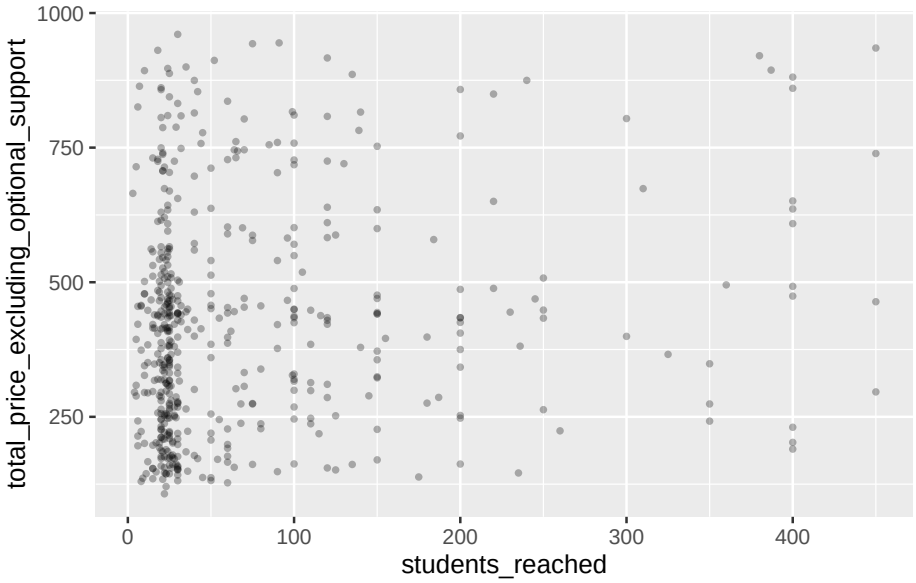


10.4 산포도와 상관관계 표현.

막대 그래프는 하나의 축이 범주형 자료를 표현할 때, 선 그래프는 하나의 축이 시간을 표현할 때 사용하는 것이 좋다고 하였습니다. 두 경우 모두 나머지 하나의 축은 연속적인 숫자, 즉, 금액을 나타내었지요. 그런데, 만약 두 변수 모두 연속적인 변수라면 어떤 그래프를 사용할까요? 여러 방법이 있지만, 산포도(scatters plot)을 이용해서 두 연속형 변수 간의 상관관계(correlation)을 찾아내는 것이 일반적입니다. 산포도란, 데이터에 포함된 두 개의 변수를 x축 y축 상의 점으로 모두 표현해준 것을 의미합니다. 사실 ggplot2의 관점에서는 이미 사용해본 geom_point()를 활용한다는 것 이상의 의미를 가지지 않습니다.

이번에는 크라우드펀딩 프로젝트를 통해 수혜를 얻는 학생의 수(students_reached)와 프로젝트 목표액 간에 상관관계가 있는지 알아보도록 하지요.

```
projects |>
    filter(total_price_excluding_optional_support < 1000 & students_reached < 500
    sample_n(500) |>
    ggplot() +
    aes(x=students_reached, y=total_price_excluding_optional_support) +
    geom_point(alpha=0.3, stroke=NA)
```



간단하지요? 편의상 포함한 filter() 부분과 sample_n() 부분을 제외하면, x와 y만 정해준 후, geom_point()를 이용하면 됩니다. filter()를 이용해 서브세팅을 한 이유는 크라우드펀딩 프로젝트 치고는 너무 큰 (목표액이 너무 크거나, 수혜를 입는 학생들이 너무 많은) 예외적인 경우를 제외해

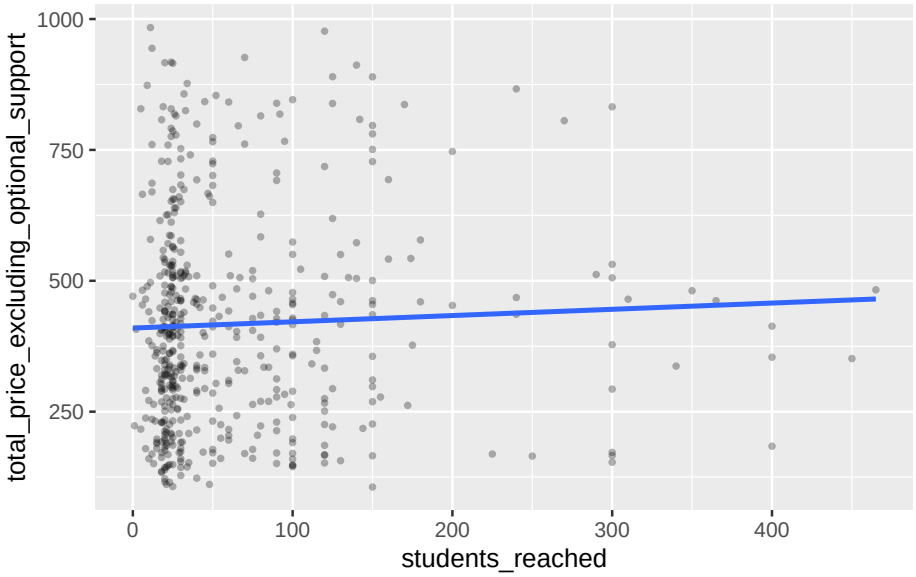
주기 위해서입니다. sample_n(500)은 전체 데이터에서 500개 행만 무작위로 뽑아달라는 의미입니다. 둘 다 꼭 해야 한다기 보다는 보기 좋은 그래프를 그리기 위한 편의상의 선택입니다.

마찬가지 이유로 geom_point() 안에 옵션도 추가했습니다. alpha=0.3이라는 옵션은 산포도에 찍히는 점들을 투명하게 만들어주기 위한 것입니다. 숫자가 작을수록 투명해지는 것이고, alpha=1이 되면 완전히 불투명해 지는 것이지요. stroke=NA는 각 점의 테두리를 없애주기 위한 옵션입니다.

자 두 변수 간에 상관 관계가 있는 것처럼 보이나요? 제 눈에는 별로 그렇게 보이지 않는데요, 이 상관관계 역시 시각적으로 표현할 수 있습니다. 위의 그래프에 하나의 계층만 추가하면 됩니다.

```
projects |>
    filter(total_price_excluding_optional_support < 1000 & students_reached < 500) |>
    sample_n(500) |>
    ggplot() +
    aes(x=students_reached, y=total_price_excluding_optional_support) +
    geom_point(alpha=0.3, stroke=NA) +
    geom_smooth(method = lm, se = FALSE)
```

```
`geom_smooth()` using formula = 'y ~ x'
```



여기서는 geom_smooth()라는 하나의 Geometries 계층만을 추가했습니다. 간단하게 상관관계를 직선으로 표현하고 있는데요, 우리 눈에 명확하지는 않지만, 양의 기울기를 갖는 것을 수혜를 입는 학생 수가 늘 수록, 프로젝트 목표액이 조금 느는 경향이 있는 것 같네요.

geom_smooth()에도 옵션을 사용했는데요, method=lm은 상관관계를 곡선이 아닌 직선으로(Linear Model) 표현해달라는 것이고, se=FALSE는 상관관계를 나타내는 해당 직선의 불확실성을 시각화하지 말라는 옵션입니다. 불확실성을 표현한다는 것이 무엇인지 확인하고 싶으시다면, 이 옵션을 TRUE로 바꿔서 다시 한 번 시각화 해 보세요.

10.5 시각화를 위한 데이터 처리 워크플로우

그래프에서부터 뒤로 만들어나가는 것이라는 것 설명하기.