타이디버스 치트시트

타이디버스 소개

데이터 과학은 데이터를 통한 질문 해결 과정이지만, 데이터 정리(wrangling)에 대부분의 시간이 소요됩니다. 이를 돕기 위해 R에서는 타이디버스(Tidyverse)라는 패키지 모음이 개발되었습니다. 타이디버스는 공통된 철학과 구조를 공유하며, 이 과정을 통해 깔끔한 데이터(tidy data) 개념과 타이디버스 활용법을 익히는 것이 목표입니다. 이 과정은 R 프로그래밍 기본 지식을 전제로 하며, R에 익숙하지 않다면 선행 학습을 권장합니다.

타이디버스를 사용하기 전에 먼저 설치해야 합니다.

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

깔끔한 데이터

깔끔한 데이터셋은 조작, 모델링, 시각화에 용이하도록 설계된 데이터입니다. 이는 일관된 원칙과 규칙을 따르기 때문인데, Hadley Wickham이 "깔끔한 데이터셋은 모두 비슷하지만, 모든 지저분한 데이터셋은 자신만의 방식으로 지저분하다"고 말했듯, 깔끔한 형식은 데이터 작업 과정을 크게 단순화합니다. 프로젝트 초기에 데이터를 깔끔한 형식으로 만들면 이후의 데이터 과학 과정이 훨씬 쉬워집니다.

깔끔한 데이터의 이점

깔끔한 데이터 프레임워크 내에서 작업하는 데는 아래 여러 가지 이점이 있습니다.

- 깔끔한 데이터는 일관된 데이터 구조를 가집니다. 이는 데이터가 저장될 수 있는 다양한 방식을 제거합니다. 통일된 데이터 구조를 부과함으로써, 분석가가 새로운 프로젝트를 시작할 때마다 발생하는 인지 부하가 최소화됩니다.
- 깔끔한 데이터는 도구 개발을 촉진합니다. 깔끔한 데이터 프레임워크 내에서 작동하는 소프트웨어는 서로 다른 개인이 개발하더라도 서로 잘 작동할 수 있으며, 궁극적으로 분석가가 새로운 도구를 배울 때마다 완전히 새로운 정신 모델을 배울 필요 없이 사용 가능한 도구의 다양성과 범위를 증가시킵니다.
- 깔끔한 데이터는 소수의 도구만 배우면 됩니다. 일관된 데이터 형식을 사용할 때, 소수의 도구만 필요하며 이 도구들은 한 프로젝트에서 다음 프로젝트로 재사용할 수 있습니다.
- 깔끔한 데이터는 데이터셋을 결합할 수 있도록 합니다. 데이터는 종종 여러 테이블이나 다른 위치에 저장됩니다. 각 테이블을 깔끔한 형식으로 만들면, 테이블이나 소스 간의 결합이 사소해집니다.

깔끔한 데이터를 저장하기 위한 규칙

네 가지 깔끔한 데이터 원칙 외에도, 데이터를 입력하거나 이미 주어진 지저분한 데이터를 깔끔한 형식으로 재구성할 때 따라야 할 여러 규칙이 있습니다. 이는 나중에 데이터 분석 및 시각화를 더 쉽게 만드는 데 도움이 되는 규칙들입니다. 이 규칙들은 두 명의 저명한 데이터 과학자인 Karl Broman과 Kara Woo가 작성한 "스프레드시트의 데이터 구성"이라는 논문에서 공식화되었습니다. 이 논문에서 그들은 데이터가 깔끔한지 확인하는 것 외에도 스프레드시트에 데이터를 입력할 때 다음 지침을 따를 것을 제안합니다.

- 일관성을 유지하십시오.
- 항목에 좋은 이름을 선택하십시오.
- 날짜는 `YYYY-MM-DD` 형식으로 작성하십시오.
- 빈 셀을 사용하지 마십시오.
- 한 셀에 한 가지만 넣으십시오.
- 글꼴 색상이나 강조 표시를 데이터로 사용하지 마십시오.
- 데이터를 일반 텍스트 파일로 저장하십시오.

이러한 각 규칙을 검토하여 훌륭한 깔끔한 스프레드시트가 어떤 모습인지 모두가 명확히 알 수 있도록 하겠습니다.

타이디버스에서 데이터 정리하기

데이터 과학자의 매우 중요한 기술은 지저분한 형식의 데이터를 가져와 깔끔한 형식으로 만드는 능력입니다. 이 과정은 종종 데이터 정리(data wrangling)라고 불립니다. 일반적으로 데이터 정리 기술은 현재 형식의 데이터를 원하는 깔끔한 형식으로 정리할 수 있도록 하는 기술입니다. 데이터 정리 외에도, 가지고 있는 데이터가 정확하고 관심 있는 질문에 답하는 데 필요한 것인지 확인하는 것도 중요합니다. 데이터를 깔끔한 형식으로 정리한 후에도 데이터를 정리하기 위해 추가 작업이 필요한 경우가 많습니다.

깔끔한 데이터 검토

더 나아가기 전에, 깔끔한 데이터셋의 요구 사항을 검토해 봅시다.

- 각 변수는 열에 저장됩니다.
- 각 관측치는 행에 저장됩니다.
- 각 셀은 단일 값을 저장합니다.

데이터 재구성

깔끔한 데이터는 일반적으로 와이드(wide) 데이터와 롱(long) 데이터의 두 가지 형태로 존재합니다. 두 가지 유형의 데이터는 데이터 분석에서 모두 사용되고 필요하며, 다행히 와이드 형식을 롱 형식으로, 롱 형식을 와이드 형식으로 변경할 수 있는 도구가 있습니다. 이를 통해 모든 깔끔한 데이터셋을 쉽게 작업할 수 있습니다. 와이드 및 롱 데이터가 무엇이며 R에서 두 형식 간에 전환하는 방법에 대한 기본 사항을 논의할 것입니다. 데이터를 올바른 형식으로 만드는 것은 나중에 데이터를 요약하고 시각화할 때 매우 중요할 것입니다.

와이드 데이터

와이드 데이터는 각 변수마다 열이 있고 각 관측치마다 행이 있습니다. 데이터는 종종 이런 방식으로 입력되고 저장됩니다. 이는 와이드 데이터가 한눈에 이해하기 쉽기 때문입니다.

롱 데이터

반면에 롱 데이터는 해당 행에 포함된 변수 유형을 나타내는 열이 하나 있고, 해당 변수에 대한 값을 위한 별도의 행이 있습니다. 각 행은 단일 변수에 대한 단일 관측치를 포함합니다.

팩터(Factors)로 작업하기

R에서 범주형 데이터는 팩터(factors)로 처리됩니다. 정의상 범주형 데이터는 가질 수 있는 가능한 값의 수가 제한적입니다. 예를 들어, 한 달은 12개월이 있습니다. 월 변수에서 각 관측치는 이 12개 값 중 하나만 가질 수 있습니다. 따라서 가능한 값의 수가 제한되어 있으므로 월은 범주형 변수입니다. 이 단원의 나머지 부분에서는 팩터라고 불릴 범주형 데이터는 데이터에서 정기적으로 발견됩니다. 이러한 유형의 변수로 효과적으로 작업하는 방법을 배우는 것은 매우 유용할 것입니다.

팩터로 작업하는 것을 더 간단하게 만들기 위해, 우리는 핵심 타이디버스 패키지인 `forcats` 패키지를 활용할 것입니다. `forcats` 내의 모든 함수는 `fct_`로 시작하여 찾아보기 쉽고 기억하기 쉽습니다. 이전과 마찬가지로 사용 가능한 함수를 보려면 RStudio 콘솔에 `?fct_`를 입력하면 됩니다. 가능한 모든 `forcats` 함수가 포함된 드롭다운 메뉴가 나타날 것입니다.

R에서 팩터는 두 가지 구성 요소로 이루어져 있습니다: 데이터의 실제 값과 팩터 내의 가능한 수준(levels)입니다. 따라서 팩터를 생성하려면 이 두 가지 정보를 모두 제공해야 합니다.

예를 들어, 12개월의 문자 벡터를 만들려면 다음과 같이 할 수 있습니다.

```
## all 12 months
all_months ← c("Jan", "Feb", "Mar",
"Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec")

## our data
some_months ← c("Mar", "Dec", "Jan",
"Apr", "Jul")
```

그러나 이 벡터를 정렬하면 R은 이 벡터를 알파벳순으로 정렬할 것입니다.

```
# alphabetical sort
sort(some_months)
## [1] "Apr" "Dec" "Jan" "Jul" "Mar"
```

우리와 여러분은 이것이 월이 정렬되어야 하는 방식이 아니라는 것을 알지만, 아직 R에게 알려주지 않았습니다. 그렇게 하려면, R에게 그것이 팩터 변수이고 해당 팩터 변수의 수준이 무엇이어야 하는지 알려주어야 합니다.

```
# create factor
mon ← factor(some_months, levels =
all_months)
# look at factor
mon
## [1] Mar Dec Jan Apr Jul
## Levels: Jan Feb Mar Apr May Jun Jul
Aug Sep Oct Nov Dec
# look at sorted factor
sort(mon)
## [1] Jan Mar Apr Jul Dec
## Levels: Jan Feb Mar Apr May Jun Jul
Aug Sep Oct Nov Dec
```

타이디버스를 통한 쉬운 분석

이제 R의 기본 사항을 알았으니, 방금 배운 모든 것을 훨씬 더 잘 할 수 있는 방법이 있다는 것을 배울 때입니다!

타이디버스 소개

타이디버스는 R을 더 쉽게 사용할 수 있게 해주는 패키지들의 묶음입니다. 이 패키지들은 모두 함께 작동하도록 설계되었기 때문입니다. 대부분의 "타이디" 함수는 다음과 같은 이유로 서로 잘 작동합니다.

- 데이터프레임을 입력으로 받습니다.
- 데이터프레임을 출력으로 반환합니다.

분석에서 타이디버스 패키지를 모두 사용하지 않을 수도 있지만, 대부분의 분석 시작 시 모두 로드하여 표준 도구 세트를 사용할 수 있습니다. 모두 로드하려면 다음을 사용하십시오.

예제에서는 `carData` 패키지의 `Cowles` 데이터를 사용합니다. 먼저 패키지를 설치하고 로드해야 합니다.

```
# install.packages("carData") # 필요한 경우 설치
library(carData)
```

```
library(tidyverse)
cow = Cowles
```

dplyr: 복잡한 분석을 간단한 단계로 전환

분석이 복잡해질수록 코드도 더 복잡해질 수 있습니다. 원하는 답을 얻으려면 다음을 수행해야 할 수도 있습니다.

- 데이터셋에서 유효하지 않거나 관련 없는 특정 행을 삭제합니다.
- 각 처리 그룹에 대해 별도로 통계를 계산합니다.
- 학교 평균과 같은 요약 변수를 사용하여 표준화된 점수를 계산합니다.

그리고 더 중요하게는, 수행하는 각 계산에 대해 이러한 여러 가지 작업을 결합해야 할 수도 있습니다.

타이디버스에 있는 `dplyr` 패키지는 복잡한 작업을 수행하기 위해 쉽게 결합할 수 있는 소수의 간단한 동사를 제공하여 이를 더 쉽게 만듭니다. 각 동사는 현재 데이터를 가져와 변경한 다음 반환합니다. 가장 일반적으로 사용될 동사는 다음과 같습니다.

- `filter()`: 논리적 테스트를 기반으로 유지할 행을 선택하고 나머지는 삭제합니다.
- `arrange()`: 데이터를 정렬합니다.
- `select()`: 유지할 열을 선택합니다.
- `mutate()`: 새 열을 추가합니다.
- `summarize()`: 데이터를 단일 행으로 요약하는 열을 만듭니다.
- `count()`: 데이터의 행 수를 계산합니다.
- `left_join()` (및 `right_join()`, `inner_join()` 등): 공통 식별자를 기반으로 데이터셋을 병합합니다.

그리고 가장 중요한 것은 다음과 같습니다.

- `group_by()`: 데이터를 그룹으로 분할하여 이후의 모든 단계가 각 그룹에 대해 별도로 수행되도록 합니다.

아래에서 이 모든 예시를 다룰 것입니다.

규칙 구부리기: 비표준 평가

`dplyr` 및 기타 타이디버스 패키지는 R 구문의 규칙을 구부려서 `survey1$group`, `survey1$time`처럼 매번 데이터프레임 이름을 모두 입력할 필요 없이 `group` 및 `time`과 같은 열 이름만 입력할 수 있도록 합니다. 타이디버스 함수 호출 내에서는 열 이름만 입력하면 됩니다. 예를 들면:

```
cow %>%
    # Within the brackets, no $ needed
    count(sex, volunteer)
## # A tibble: 4 x 3
##   sex    volunteer     n
##   <fct>  <fct>     <int>
## 1 female no          431
## 2 female yes         349
## 3 male   no          393
## 4 male   yes         248
```

dplyr을 사용한 일반적인 작업

select를 이용한 열 선택

`select()`를 사용하여 데이터프레임에서 특정 열을 선택할 수 있습니다. 열 이름 앞에 `-`를 사용하면 해당 열을 제외합니다.

```
# These both give the same result:
inclusion vs. exclusion
cow %>%
    select(neuroticism, extraversion)
%>%
    head(2)
##   neuroticism extraversion
## 1          16           13
## 2           8           14
cow %>%
    select(-sex, -volunteer) %>%
    head(2)
##   neuroticism extraversion
## 1          16           13
## 2           8           14
```

열 이름 대신 `starts_with()` 및 `num_range()`와 같은 다양한 도우미 함수를 사용하여 특정 패턴과 일치하는 여러 열을 선택할 수도 있습니다.

mutate를 이용한 열 생성/변경

데이터프레임에 `mutate`를 사용하여 하나 이상의 열을 추가하거나 변경할 수 있습니다.

```
cow %>%
    # high_extraversion and
high_neuroticism are the names
    #   of new columns that will be
created
    mutate(high_extraversion =
extraversion ⩾ 15,
           high_neuroticism =
neuroticism ⩾ 15) %>%
    head()
##   neuroticism extraversion    sex
volunteer high_extraversion
## 1          16           13 female
no             FALSE
## 2           8           14   male
no             FALSE
## 3           5           16   male
no              TRUE
## 4           8           20 female
no              TRUE
## 5           9           19   male
no              TRUE
## 6           6           15   male
no              TRUE
```

이러한 변경 사항을 저장하려면 결과를 원래 변수에 다시 저장해야 합니다. 기본적으로 `mutate()`는 원본 데이터의 변경된 복사본을 반환할 뿐 원본을 변경하지는 않습니다.

```
# If we want to actually save the
changes
cow = cow %>%
    mutate(high_extraversion =
extraversion ⩾ 15,
           high_neuroticism =
neuroticism ⩾ 15)
```

그 자체로 `mutate`가 제공하는 주요 장점은 데이터프레임 이름을 포함하지 않고도 계산을 명확하게 표현할 수 있다는 것입니다. 그러나 다른 동사와 결합하면 매우 유용할 수 있습니다.

summarize를 이용한 데이터 요약

`summarize`는 `mutate`와 유사합니다. 열을 추가하거나 변경합니다. 그러나 `summarize`는 데이터를 단일 행으로 축소하므로 계산하는 값은 평균 또는 개수와 같은 단일 값이어야 합니다.

```
cow %>%
    summarize(
        extraversion =
mean(extraversion),
        volunteers = sum(volunteer ==
"yes")
    )
##   extraversion volunteers
## 1     12.37298         597
```

`summarize`는 `group_by`와 함께 사용하면 유용하며, 데이터를 그룹당 한 행으로 축소합니다.

filter를 이용한 행 선택

`filter()`를 사용하여 논리적 테스트로 행을 선택하는 것은 논리 벡터로 데이터를 서브셋팅하는 것과 같습니다. 단지 일련의 단계의 일부로 수행하기가 더 쉽습니다.

```
cow %>%
  filter((sex == "male") & (volunteer ==
"yes")) %>%
  head()
##   neuroticism extraversion  sex
volunteer
## 1          17           19 male
yes
## 2           7           15 male
yes
## 3          17           12 male
yes
## 4           6           13 male
yes
## 5           8            9 male
yes
## 6           5           16 male
yes
```

arrange를 이용한 정렬

`arrange`를 사용하면 하나 이상의 열로 정렬할 수 있습니다. `desc(column)` (내림차순의 약어)을 사용하여 해당 열을 반대 방향으로 정렬하십시오.

```
cow %>%
  arrange(sex, volunteer,
desc(extraversion)) %>%
  head()
##   neuroticism extraversion    sex
volunteer
## 1          15           23 female
no
## 2          15           21 female
no
## 3           9           21 female
no
## 4          10           21 female
no
## 5           8           20 female
no
## 6           5           20 female
no
```

group_by를 이용한 그룹 내 계산

`group_by()`는 데이터의 하위 그룹에서 다른 통계를 계산하는 데 매우 유용합니다. 이는 데이터로 수행해야 할 수 있는 더 복잡한 작업 중 상당 부분을 포함하므로 많은 가능성을 열어줍니다.

다음과 같은 작업을 수행할 수 있습니다.

- 남성과 여성에 대해 별도로 점수 평균 중심화:

```
cow %>%
    group_by(sex) %>%
    mutate(
        extraversion_centered =
extraversion - mean(extraversion)
    )
## # A tibble: 1,421 x 5
## # Groups:   sex [2]
##    neuroticism extraversion sex
volunteer extraversion_centered
##          <int>        <int> <fct>
<fct>                     <dbl>
##  1          16           13 female no
0.578
##  2           8           14 male   no
1.69
##  3           5           16 male   no
3.69
##  4           8           20 female no
7.58
```

• 하위 그룹에 대한 평균 및 표준 편차 계산:

```
cow %>%
    group_by(sex, volunteer) %>%
    summarise(mean = mean(neuroticism),
              sd = sd(neuroticism))
## # A tibble: 4 x 4
## # Groups:   sex [2]
##   sex    volunteer  mean    sd
##   <fct>  <fct>     <dbl> <dbl>
## 1 female no         12.2  4.75
## 2 female yes        12.3  4.79
## 3 male   no         10.5  4.75
## 4 male   yes        10.4  5.11
```

count를 이용한 빈도표

count()는 빈도표를 만드는 간단한 방법을 제공합니다. 백분율을 계산하는 내장된 방법은 없지만, mutate()를 사용하여 쉽게 추가할 수 있습니다.

```
cow %>%
    count(sex) %>%
    # count creates a column called 'n'
    mutate(percent = n / sum(n) * 100)
## # A tibble: 2 x 3
##   sex        n percent
##   <fct>  <int>   <dbl>
## 1 female   780    54.9
## 2 male     641    45.1
```

여러 그룹화 수준이 있는 경우 백분율을 어떻게 계산할지 고려해야 합니다. 각 성별 내에서 자원 봉사한 비율을 얻으려면 다음을 수행하십시오.

```
cow %>%
    count(sex, volunteer) %>%
    group_by(sex) %>%
    mutate(percent = n / sum(n) * 100)
## # A tibble: 4 x 4
## # Groups:   sex [2]
##   sex    volunteer     n percent
##   <fct>  <fct>     <int>   <dbl>
## 1 female no          431    55.3
## 2 female yes         349    44.7
## 3 male   no          393    61.3
## 4 male   yes         248    38.7
```

left_join()을 이용한 데이터 병합

두 데이터프레임을 결합하려면 left_join() 및 inner_join()과 같은 함수를 사용할 수 있습니다. 제 경험상 left_join()이 대부분의 경우에 원하는 것입니다. 데이터를 성공적으로 병합하려면 일반적으로 참가자 ID 또는 이와 유사한 공통 식별자와 같이 두 데이터셋에 모두 존재하는 열만 있으면 됩니다.

조인은 복잡한 요약을 계산해야 할 때도 유용할 수 있습니다. 요약 정보가 포함된 별도의 테이블을 만들고 이를 기본 데이터셋으로 다시 병합할 수 있습니다. 간단한 예로, 성별 및 자원 봉사 여부에 따라 외향성을 요약하고 이를 기본 데이터셋으로 다시 병합해 보겠습니다.

```
extraversion_info = cow %>%
    group_by(sex, volunteer) %>%
    summarize(mean_extraversion =
mean(extraversion))

extraversion_info
## # A tibble: 4 x 3
## # Groups:   sex [2]
##   sex    volunteer mean_extraversion
##   <fct>  <fct>                 <dbl>
## 1 female no                     12.0
## 2 female yes                    12.9
## 3 male   no                     11.9
## 4 male   yes                    12.9
cow %>%
  left_join(extraversion_info, by =
c("sex", "volunteer")) %>%
  head()
##   neuroticism extraversion    sex
volunteer mean_extraversion
## 1          16           13 female
no          12.00696
## 2           8           14   male
no          11.91349
## 3           5           16   male
no          11.91349
## 4           8           20 female
no          12.00696
## 5           9           19   male
no          11.91349
## 6           6           15   male
no          11.91349
```