7. 데이터 랭글링

Data Wrangling이란?

데이터 랭글링은 원자료(raw data)를 또다른 형태로 수작업으로 전환하거나 매핑하는 과정이며,

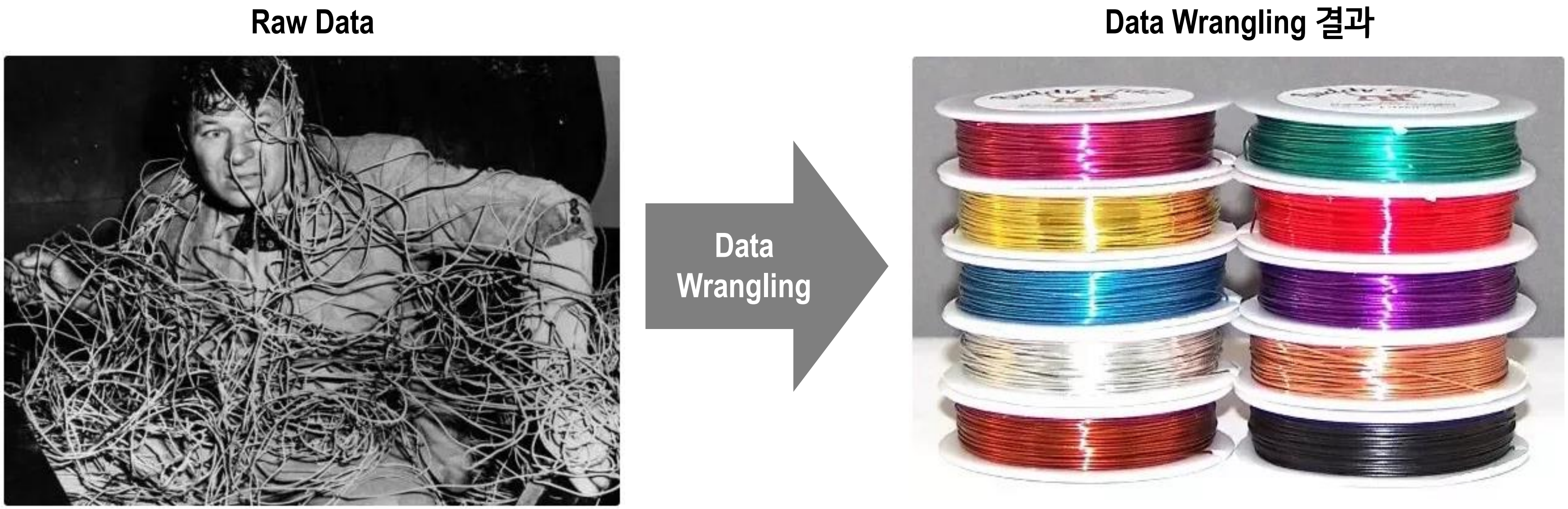
데이터 먼징(Data Munging)이라고도 한다.

□ Data Wrangling의 용도

- 데이터 시각화, 데이터 집합, 통계모형 학습 뿐만 아니라 많은 다른 잠재적 용도도 포함된다.

□ Data Wrangling 과정

- 데이터 원천(Data Source)으로부터 원래 최초 형태로 자료를 추출한다.
- 알고리듬(예 : 정렬)을 사용해서 원자료를 사전 정의된 자료구조로 데이터를 파싱(parsing)한다.
- 마지막으로 저장이나 미래 사용을 위해서 작업완료한 콘텐츠를 데이터 싱크(data sink)에 저장한다.



Raw Data

Data Wrangling

Data Wrangling 결과

dplyr 소개

데이터 랭글링을 위한 패키지 dplyr을 소개하면 하기와 같다.

❑ **dplyr 소개**

- Data Frame 즉 Excel 데이터를 쉽게 조작하는 데 사용함
- 앞서 Excel, DB 에서 불러들이 데이터는 모두 data.frame 으로 저장됨

구분	설명
select()	열(Column)을 선택하는 데 사용
filter()	행(Row)을 선택하는데 사용
subset()	열과 행을 조작하는데 사용
mutate(), transform()	데이터를 조작하거나 새로 생성함
arrange()	데이터 정렬
summarize()	데이터 summary
group_by()	데이터 Grouping
%>%	연결처리

사전 준비

문자열, 비정형 데이터 처리를 위하여 사전에 준비해야 할 사항은 하기와 같다.

❑ package 설치

- dplyr 설치 : install.packages(dplyr)

- hflights 설치 : install.packages(hflights)

❑ package 불러오기

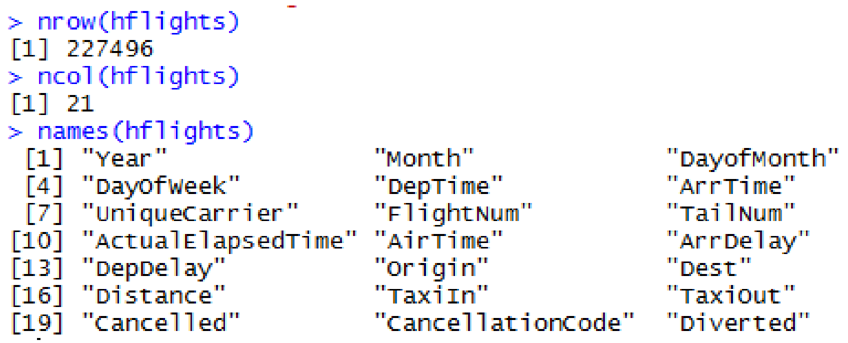
library(dplyr)

library(hflights)

❑ 처리 대상 데이터 살펴보기

- 총 행개수 : 22,7496개

- 총 열개수 : 21개



```
> nrow(hflights)
[1] 227496
> ncol(hflights)
[1] 21
> names(hflights)
 [1] "Year"              "Month"             "DayofMonth"
 [4] "DayOfWeek"         "DepTime"           "ArrTime"
 [7] "UniqueCarrier"     "FlightNum"         "TailNum"
[10] "ActualElapsedTime" "AirTime"           "ArrDelay"
[13] "DepDelay"          "Origin"            "Dest"
[16] "Distance"          "TaxiIn"            "TaxiOut"
[19] "Cancelled"         "CancellationCode"  "Diverted"
```

열처리 – select()

분석 데이터의 열이 많을 때, 필요한 열만 선택해야 하는 경우에 사용한다.

❑ 사용법 : select(데이터프레임, 선택하고 싶은 열조건 표현)

- 열조건 표현 : 열의 이름을 사용하거나, 순서로 표현함(벡터로 표현)

❑ 예제

```
> a<-select(hflights, ActualElapsedTime, AirTime, ArrDelay, DepDelay)
> names(a)
[1] "ActualElapsedTime" "AirTime"          "ArrDelay"
[4] "DepDelay
```

```
> a<-select(hflights, c(10, 11, 12, 13))
> names(a)
[1] "ActualElapsedTime" "AirTime"          "ArrDelay"
[4] "DepDelay“
```

❑ 예제

```
> b<-select(hflights, Origin:Cancelled)
> names(b)
[1] "Origin"    "Dest"      "Distance"  "TaxiIn"    "TaxiOut"
[6] "Cancelled"
```

열처리 – select()

분석 데이터의 열이 많을 때, 필요한 열만 선택해야 하는 경우에 사용한다.

❑ 사용법 : select(데이터프레임, 선택하고 싶은 열조건 표현)

- 열조건 표현 : 열의 이름을 사용하거나, 순서로 표현함(벡터로 표현)

❑ 예제

```
> c<-select(hflights, -(DepTime:AirTime))
> names(c)
 [1] "Year"           "Month"           "DayofMonth"
 [4] "DayOfWeek"       "ArrDelay"        "DepDelay"
 [7] "Origin"          "Dest"            "Distance"
[10] "TaxiIn"          "TaxiOut"         "Cancelled"
[13] "CancellationCode" "Diverted"

> d<-select(hflights, UniqueCarrier, ends_with("Num"),
    starts_with("Cancel"))
> names(d)
[1] "UniqueCarrier"   "FlightNum"       "TailNum"
[4] "Cancelled"       "CancellationCode"
```

❑ 예제

```
> e<-select(hflights, contains("Tim"), contains("Del"))
> names(e)
[1] "DepTime"         "ArrTime"         "ActualElapsedTime"
[4] "AirTime"         "ArrDelay"        "DepDelay“

> f<-select(hflights, Year:ArrTime)
> names(f)
[1] "Year"       "Month"      "DayofMonth" "DayOfWeek"  "DepTime"
[6] "ArrTime"

> f<-select(hflights, Year:ArrTime, -DayofMonth)
> names(f)
[1] "Year"      "Month"     "DayOfWeek" "DepTime"   "ArrTime"
```

행 처리 – filter()

분석 데이터의 행이 많을 때, 필요한 행만 선택해야 하는 경우에 사용한다.

❑ 사용법 : filter(데이터프레임, 선택하고 싶은 행 조건 표현)

- 열 표현 : 열의 이름을 사용하거나, 순서로 표현함(수식으로 표현)

❑ 예제

```
> a<-filter(hflights, Distance >= 3000)
> min(a$Distance)
[1] 3266

> b<-filter(hflights, Month>3 & Month<6)
> range(b$Month)
[1] 4 5

> c<-filter(hflights, DepDelay > 0, ArrDelay < 0)
> range(c$DepDelay)
[1]  1 54
> range(c$ArrDelay)
[1] -51  -1
```

열 & 행처리 – subset()

분석 데이터의 행과 열에 대한 선별이 필요한 경우에 사용한다.

❑ 사용법 : subset(데이터프레임, 행 선택 조건 표현, select = c(열))

❑ 예제

```
> a<-subset( hflights, DepDelay > 0 & ArrDelay < 0, select = c(ActualElapsedTime, AirTime, ArrDelay, DepDelay) )

> names(a)
[1] "ActualElapsedTime" "AirTime"           "ArrDelay"
[4] "DepDelay“

> range(a$DepDelay)
[1]  1 54

> range(a$ArrDelay)
[1] -51  -1
```

열 변환처리 – mutate()

분석 데이터의 값들을 조작하여 새로운 데이터나 기존 데이터를 변경해야 하는 경우에 사용한다.

❑ 사용법 : mutate(데이터프레임, 신규 열명 = 변경수식)

```
> a<-mutate( hflights, ActualGroundTime = ActualElapsedTime - AirTime)
> names(a)
 [1] "Year"              "Month"             "DayofMonth"
 [4] "DayOfWeek"         "DepTime"           "ArrTime"
 [7] "UniqueCarrier"     "FlightNum"         "TailNum"
[10] "ActualElapsedTime" "AirTime"           "ArrDelay"
[13] "DepDelay"          "Origin"            "Dest"
[16] "Distance"          "TaxiIn"            "TaxiOut"
[19] "Cancelled"         "CancellationCode"  "Diverted"
[22] "ActualGroundTime“
```

```
> b<-mutate( hflights, loss = ArrDelay - DepDelay, loss_percent = (ArrDelay - DepDelay) / DepDelay * 100)
> names(b)
 [1] "Year"              "Month"             "DayofMonth"
 [4] "DayOfWeek"         "DepTime"           "ArrTime"
 [7] "UniqueCarrier"     "FlightNum"         "TailNum"
[10] "ActualElapsedTime" "AirTime"           "ArrDelay"
[13] "DepDelay"          "Origin"            "Dest"
[16] "Distance"          "TaxiIn"            "TaxiOut"
[19] "Cancelled"         "CancellationCode"  "Diverted"
[22] "loss"              "loss_percent"
```

열 변환처리 – transform()

분석 데이터의 값들을 조작하여 새로운 데이터나 기존 데이터를 변경해야 하는 경우에 사용한다.

❑ 사용법 : transform(데이터프레임, 신규 열명 = 변경수식)

```
> a<-transform( hflights, ActualGroundTime = ActualElapsedTime - AirTime)
> names(a)
 [1] "Year"              "Month"             "DayofMonth"
 [4] "DayOfWeek"         "DepTime"           "ArrTime"
 [7] "UniqueCarrier"     "FlightNum"         "TailNum"
[10] "ActualElapsedTime" "AirTime"           "ArrDelay"
[13] "DepDelay"          "Origin"            "Dest"
[16] "Distance"          "TaxiIn"            "TaxiOut"
[19] "Cancelled"         "CancellationCode"  "Diverted"
[22] "ActualGroundTime"
```

```
> b<-transform( hflights, AverageSpeed = Distance / AirTime * 60)
> names(b)
 [1] "Year"              "Month"             "DayofMonth"
 [4] "DayOfWeek"         "DepTime"           "ArrTime"
 [7] "UniqueCarrier"     "FlightNum"         "TailNum"
[10] "ActualElapsedTime" "AirTime"           "ArrDelay"
[13] "DepDelay"          "Origin"            "Dest"
[16] "Distance"          "TaxiIn"            "TaxiOut"
[19] "Cancelled"         "CancellationCode"  "Diverted"
[22] "AverageSpeed"
```

정렬 처리 – arrange()

분석 데이터의 행을 특정 열을 기준으로 오름차순, 내림차순으로 정리하는 경우에 사용한다.

❑ 사용법 : arrange(데이터프레임, 오름차선/내림차순 설정 + 기준 열명)

```
arrange( hflights, DepDelay)                    # 오름차순
arrange( hflights, desc(DepDelay) )             # 내림차순
arrange( hflights, ActualGroundTime)            # 오름차순
arrange( hflights, desc(ActualGroundTime))   # 오름차순
arrange( hflights, CancellationCode)
arrange( hflights, UniqueCarrier, DepDelay)     # 두개 항의 조합
arrange( hflights, UniqueCarrier, desc(DepDelay))
```

❑ 응용 : selection, filter, subset 명령과 함께 사용하기

```
arrange(filter(hflights, Dest == "DFW", DepTime < 800), desc(AirTime)
arrange(select(hflights, Year:ArrTime), Year)
```

요약처리 – summarize()

분석 데이터 객체에 대한 기술통계값을 계산하여 보여주어야 하는 경우에 사용한다.

❑ 사용법 : summarize(데이터프레임, 열명 = 수식)

```
> summarise(hflights, min_dist = min(Distance), max_dist = max(Distance))
  min_dist max_dist
1       79     3904
```

```
> summarise(hflights, n_obs = n(),
+           n_carrier = n_distinct(UniqueCarrier),
+           n_dest = n_distinct(Dest),
+           dest100 = nth(Dest, 100))

   n_obs     n_carrier     n_dest     dest100
1  227496    15            116        DFW
```

연결 처리 – %>%

데이터 manipulation 명령어를 순서대로 나열하듯 실행하여 해당 코드의 애해와 독해를 빠르게 하기 위해 사용한다.

❑ 사용법 : %>% (순차적 연결 명령)

```
> hflights %>%
+ mutate(diff=TaxiOut-TaxiIn) %>%
+ filter(!is.na(diff)) %>%
+ summarise(avg=mean(diff))

    avg
1   8.992064
```

```
> hflights %>%
+ select(Dest, Cancelled,Distance, ActualElapsedTime, Diverted) %>%
+ mutate(RealTime=ActualElapsedTime+100, mph=Distance/RealTime*60) %>%
+ filter(mph<105|Cancelled==1|Diverted==1) %>%
+ summarise(n_non=n(),
+ p_non=n_non/nrow(hflights)*100,
+ n_dest=n_distinct(Dest),
+ min_dist=min(Distance),
+ max_dist=max(Distance))

    n_non     p_non        n_dest    min_dist     max_dist
1   42400    18.63769    113        79            3904
```

그룹연산 처리 – group_by

데이터 특정 기준으로 Group화하여 연산처리를 하는데 사용한다.

❑ 사용법 : group_by(그룹화 기준)

```
> hflights %>%
+ group_by(UniqueCarrier) %>%
+ summarise(n_flights=n(),
+ n_canc=sum(Cancelled==1),
+ p_canc=mean(Cancelled==1)*199,
+ avg_delay=mean(ArrDelay, na.rm=TRUE)) %>%
+ arrange(avg_delay, p_canc)


# A tibble: 15 x 5
  UniqueCarrier n_flights n_canc   p_canc  avg_delay
          <chr>     <int>  <int>    <dbl>      <dbl>
1            US      4082     46 2.242528 -0.6307692
2            AA      3244     60 3.680641  0.8917558
3            FL      2139     21 1.953717  1.8536239
4            AS       365      0 0.000000  3.1923077
5            YV        79      1 2.518987  4.0128205
6            DL      2641     42 3.164710  6.0841374
7            CO     70032    475 1.349740  6.0986983
8            MQ      4648    135 5.779905  7.1529751
9            EV      2204     76 6.862069  7.2569543
10           WN     45343    703 3.085305  7.5871430
11           F9       838      6 1.424821  7.6682692
12           XE     73053   1132 3.083624  8.1865242
13           OO     16061    224 2.775419  8.6934922
14           B6       695     18 5.153957  9.8588410
15           UA      2072     34 3.265444 10.4628628
```

```
> hflights %>%
+ group_by(DayOfWeek) %>%
+ summarise(avg_taxi=mean(TaxiIn+TaxiOut, na.rm=TRUE)) %>%
+ arrange(desc(avg_taxi))


# A tibble: 7 x 2
   DayOfWeek    avg_taxi
   <int>           <dbl>
1     1         21.77027
2     2         21.43505
3     4         21.26076
4     3         21.19055
5     5         21.15805
6     7         20.93726
7     6         20.43061
```

열조인 방법

두 개의 테이블을 조인하는 방법은 여러가지가 있으며 해당 방법과 결과는 하기와 같다.

❑ Join 방식

- Inner Join : 두 집합에서 일치하지 않는 Data는 모두 결과에서 생략함
- Left Outer Join : 왼쪽 집합(Left Table)의 Data는 오른쪽 집합(Right Table)과 일치하지 않는 항목이 있더라도 결과로 출력함
- Right Outer Join : 오른쪽 집합(Right Table)의 Data는 왼쪽 집합(Left Table)과 일치하지 않는 항목이 있더라도 결과로 출력함
- Full Outer Join : 두 집합에서 일치하지 않는 Data라도 모두 결과로 출력함

❑ 실행 예제

Data A, B

Dataset – A
ID 이름
1, 김대중
2, 김영삼
3, 노태우
4, 전두환

Dataset – B
ID 차량
2, SM5
3, Sonata
5, Spark

① Inner Join(A, B)

Join연산 결과

2, 김영삼, SM5
3, 노태우, Sonata

② Left Outer Join

Join연산 결과

1, 김대중,
2, 김영삼, SM5
3, 노태우, Sonata
4, 전두환,

③ Right Outer Join

Join연산 결과

2, SM5, 김영삼
3, Sonata, 노태우
5, Spark,

④ Full Outer Join

Join연산 결과

1, 김대중,
2, 김영삼, SM5
3, 노태우, Sonata
4, 전두환,
5, , Spark

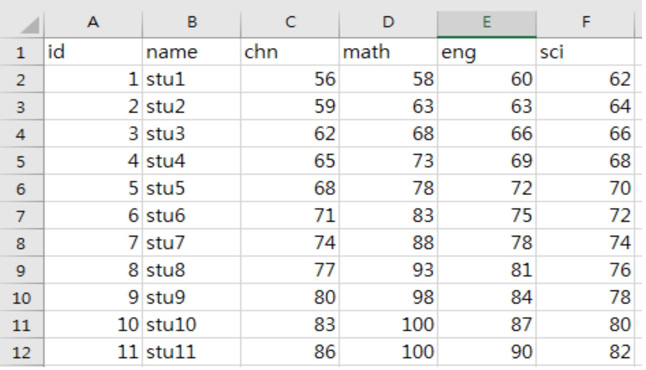
열조인 처리 – merge()

두 개의 파일이나 두개의 데이터 프레임을 특정 값을 기준으로 하여 하나로 합쳐야 하는 경우에 사용한다.

❑ 사용법 : merge(데이터프레임1, 데이터프레임2, by=c(기준항목1, 기준항목2, …))

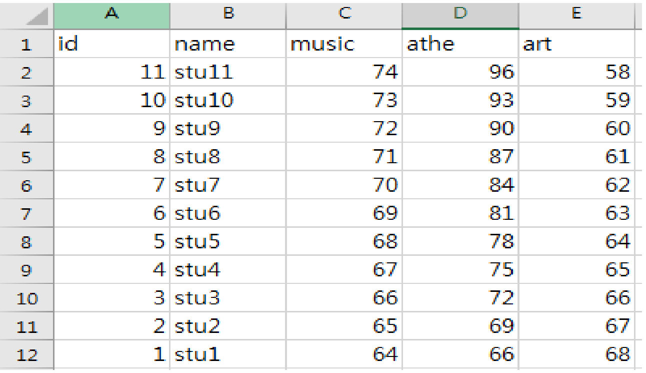
→ 기준항목의 값은 Unique하고 독립적인 key이어야 함

1.csv



	A	B	C	D	E	F
1	id	name	chn	math	eng	sci
2	1	stu1	56	58	60	62
3	2	stu2	59	63	63	64
4	3	stu3	62	68	66	66
5	4	stu4	65	73	69	68
6	5	stu5	68	78	72	70
7	6	stu6	71	83	75	72
8	7	stu7	74	88	78	74
9	8	stu8	77	93	81	76
10	9	stu9	80	98	84	78
11	10	stu10	83	100	87	80
12	11	stu11	86	100	90	82

2.csv



	A	B	C	D	E
1	id	name	music	athe	art
2	11	stu11	74	96	58
3	10	stu10	73	93	59
4	9	stu9	72	90	60
5	8	stu8	71	87	61
6	7	stu7	70	84	62
7	6	stu6	69	81	63
8	5	stu5	68	78	64
9	4	stu4	67	75	65
10	3	stu3	66	72	66
11	2	stu2	65	69	67
12	1	stu1	64	66	68

```
> df1<-read.csv('d:/1.csv',header=TRUE)
> df2<-read.csv('d:/2.csv',header=TRUE)
> out<-merge(df1, df2, by=c('id','name'))
> out
```

```
    id  name  chn  math  eng  sci  music  athe  art
1   1   stu1  56   58    60   62   64     66    68
2   10  stu10 83   100   87   80   73     93    59
3   11  stu11 86   100   90   82   74     96    58
4   2   stu2  59   63    63   64   65     69    67
5   3   stu3  62   68    66   66   66     72    66
6   4   stu4  65   73    69   68   67     75    65
7   5   stu5  68   78    72   70   68     78    64
8   6   stu6  71   83    75   72   69     81    63
9   7   stu7  74   88    78   74   70     84    62
10  8   stu8  77   93    81   76   71     87    61
11  9   stu9  80   98    84   78   72     90    60
```

Join옵션

# Inner Join
merge(x=t1, y=t2, by="필드명")
# Full Outer Join
merge(x=t1, y=t2, by="필드명", all=TRUE)
# Left Outer Join
merge(x=t1, y=t2, by="필드명", all.x=TRUE)
# Right Outer Join
merge(x=t1, y=t2, by="필드명", all.y=TRUE)
# Cross Join Join
merge(x=t1, y=t2, by=NULL)

행조인 처리 – rbind()

두 개의 파일이나 두개의 데이터 프레임을 행기준으로 하나로 합쳐야 하는 경우에 사용한다.

❑ **사용법 : rbind(데이터프레임1, 데이터프레임2)**

```
> df11
   id  name chn math eng sci
1 1   stu1  56  58   60  62
2 2   stu2  59  63   63  64
3 3   stu3  62  68   66  66
4 4   stu4  65  73   69  68
5 5   stu5  68  78   72  70


> df12
     id  name chn math eng sci
6    6  stu6  71  83   75  72
7    7  stu7  74  88   78  74
8    8  stu8  77  93   81  76
9    9  stu9  80  98   84  78
10 10  stu10  83  100  87  80
11 11  stu11  86  100  90  82
```

```
> rbind(df11,df12)
    id  name  chn math  eng  sci
1    1  stu1  56  58   60   62
2    2 stu2   59  63   63   64
3    3 stu3   62  68   66   66
4    4 stu4   65  73   69   68
5    5 stu5   68  78   72   70
6    6 stu6   71  83   75   72
7    7 stu7   74  88   78   74
8    8 stu8   77  93   81   76
9    9 stu9   80  98   84   78
10 10 stu10   83  100  87   80
11 11 stu11  86  100  90   82
```

reshape2, tidyr기타 패키지 소개

데이터 랭글링을 위한 패키지 reshape2, tidyr을 소개하면 하기와 같다.

❑ **reshape2**

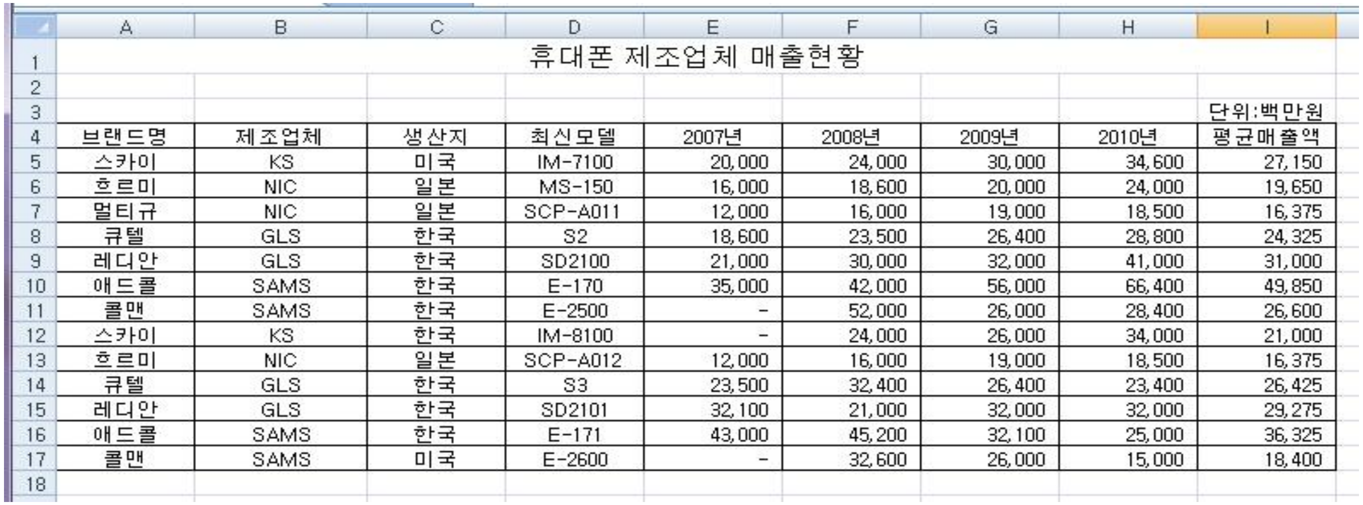
- melt, dcast를 이용하여 원시 데이터 프레임을 조작하는데 사용함
- 설치 : install.packages('reshape2')

❑ **tidyr**

- gather, spread, separate, unite를 이용하여 원시 데이터를 조작하는데 사용함
- 설치 : install.packages('tidyr')

피벗테이블

피벗테이블이란 데이터 값을 여러가지 항목을 기준으로 분류하여 분석하고 싶을 때 사용하는 테이블이다.

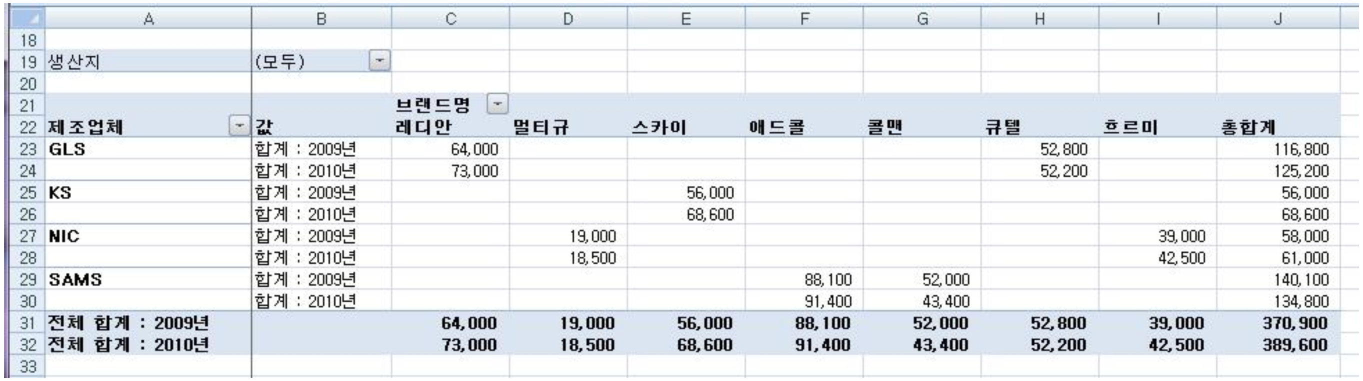


휴대폰 제조업체 매출현황

단위:백만원

브랜드명	제조업체	생산지	최신모델	2007년	2008년	2009년	2010년	평균매출액
스카이	KS	미국	IM-7100	20,000	24,000	30,000	34,600	27,150
흐르미	NIC	일본	MS-150	16,000	18,600	20,000	24,000	19,650
멀티규	NIC	일본	SCP-A011	12,000	16,000	19,000	18,500	16,375
큐텔	GLS	한국	S2	18,600	23,500	26,400	28,800	24,325
레디안	GLS	한국	SD2100	21,000	30,000	32,000	41,000	31,000
애드콜	SAMS	한국	E-170	35,000	42,000	56,000	66,400	49,850
콜맨	SAMS	한국	E-2500	-	52,000	26,000	28,400	26,600
스카이	KS	한국	IM-8100	-	24,000	26,000	34,000	21,000
흐르미	NIC	일본	SCP-A012	12,000	16,000	19,000	18,500	16,375
큐텔	GLS	한국	S3	23,500	32,400	26,400	23,400	26,425
레디안	GLS	한국	SD2101	32,100	21,000	32,000	32,000	29,275
애드콜	SAMS	한국	E-171	43,000	45,200	32,100	25,000	36,325
콜맨	SAMS	미국	E-2600	-	32,600	26,000	15,000	18,400

매출액 정보를 브랜드명 / 제조업체 / 연도별로 요약하여 분석하기 위해서는 피벗기능을 사용하여 정리한다.



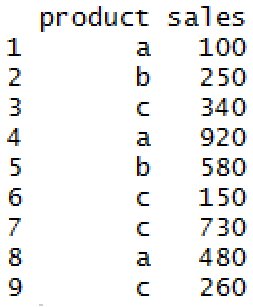
생산지 (모두)

		브랜드명							
제조업체	**값**	**레디안**	**멀티규**	**스카이**	**애드콜**	**콜맨**	**큐텔**	**흐르미**	**총합계**
GLS	합계 : 2009년	64,000					52,800		116,800
	합계 : 2010년	73,000					52,200		125,200
KS	합계 : 2009년			56,000					56,000
	합계 : 2010년			68,600					68,600
NIC	합계 : 2009년		19,000					39,000	58,000
	합계 : 2010년		18,500					42,500	61,000
SAMS	합계 : 2009년				88,100	52,000			140,100
	합계 : 2010년				91,400	43,400			134,800
전체 합계 : 2009년		**64,000**	**19,000**	**56,000**	**88,100**	**52,000**	**52,800**	**39,000**	**370,900**
전체 합계 : 2010년		**73,000**	**18,500**	**68,600**	**91,400**	**43,400**	**52,200**	**42,500**	**389,600**

피벗테이블

피벗테이블의 용도는 여러가지이지만 주로 2가지 방법으로 데이터를 정리하는 경우에 많이 사용한다고 볼 수 있다.

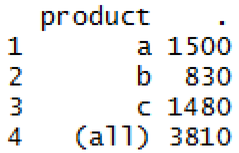
항목별 데이터 정리



```
  product sales
1       a   100
2       b   250
3       c   340
4       a   920
5       b   580
6       c   150
7       c   730
8       a   480
9       c   260
```

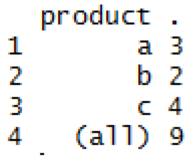
- 행이될 항목 : product
- 열이될 항목 : 없음

계산값 : sum



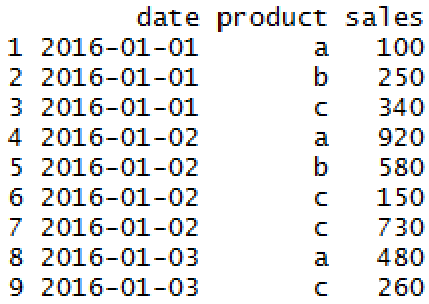
```
  product    .
1       a 1500
2       b  830
3       c 1480
4   (all) 3810
```

계산값 : sum



```
  product .
1       a 3
2       b 2
3       c 4
4   (all) 9
```

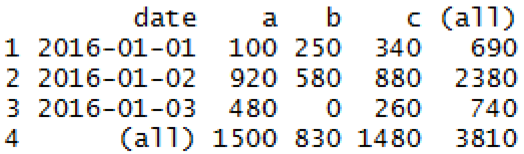
여러 항목을 행/열 기준으로 데이터 정리



```
        date product sales
1 2016-01-01       a   100
2 2016-01-01       b   250
3 2016-01-01       c   340
4 2016-01-02       a   920
5 2016-01-02       b   580
6 2016-01-02       c   150
7 2016-01-02       c   730
8 2016-01-03       a   480
9 2016-01-03       c   260
```

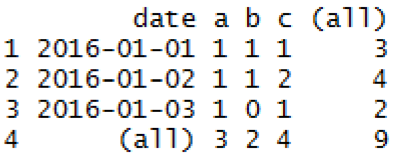
- 행이될 항목 : date
- 열이될 항목 : product

계산값 : sum



```
        date    a   b    c (all)
1 2016-01-01  100 250  340   690
2 2016-01-02  920 580  880  2380
3 2016-01-03  480   0  260   740
4      (all) 1500 830 1480  3810
```

계산값 : length



```
        date a b c (all)
1 2016-01-01 1 1 1     3
2 2016-01-02 1 1 2     4
3 2016-01-03 1 0 1     2
4      (all) 3 2 4     9
```

여러 개의 열 -> 한 개 열로 통합 – melt()

여러 열로 나열된 측정데이터를 식별항목(id), 측정항목(variable), 측정치(value)형태로 데이터를 재구성하는 경우에 사용한다.

❑ 사용법 : melt(데이터, 식별항목(id), na.rm=TRUE)

```
> library(reshape2)
> library(dplyr)
> names(airquality) <- tolower(names(airquality))
> names(airquality)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"

> head(airquality)

 Ozone Solar.R Wind Temp Month Day
1   41     190  7.4   67     5   1
2   36     118  8.0   72     5   2
3   12     149 12.6   74     5   3
4   18     313 11.5   62     5   4
5   NA      NA 14.3   56     5   5
6   28      NA 14.9   66     5   6
```

```
> # 식별항목 : month, day
> # 측정치 항목 : ozone, solar.r, wind, temp
> aqm<-melt(airquality, id=c(‘Month',‘Day'),na.rm=TRUE)
> head(aqm)
 Month Day variable value
1    5   1    Ozone    41
2    5   2    Ozone    36
3    5   3    Ozone    12
4    5   4    Ozone    18
6    5   6    Ozone    28
7    5   7    Ozone    23
```

한 개의 열 → 여러 개의 열로 분리(피벗테이블) – dcast()

하나의 변수를 여러 개로 분리할 때, 즉 key value pair를 여러 개의 열로 구분할 때 사용한다.

❑ **사용법 : dcast(원데이터, 행이될 항목~열이될 항목, value.var=값으로 사용할 항목, fill=NA, convert=FALSE, drop=TRUE)**

→ 주의 : value.var에 사용하는 값으로 사용할 항목명은 인용부호안에 넣어주어야 함

```
> levels(aqm$variable)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"
> range(aqm$month)
[1] 5 9
> names(aqm)
[1] "Month"   "Day"     "variable" "value"
```

```
> dcast(aqm, Month ~ variable, mean)
    Month   Ozone     Solar.R      Wind         Temp
1   5       23.61538 181.2963  11.622581  65.54839
2   6       29.44444 190.1667  10.266667  79.10000
3   7       59.11538 216.4839   8.941935   83.90323
4   8       59.96154 171.8571   8.793548   83.96774
5   9       31.44828 167.4333  10.180000   76.90000
```

```
> dcast(aqm, Month ~ variable, value.var="value", mean, margins=TRUE)
    Month   Ozone       Solar.R      Wind         Temp         (all)
1   5       23.61538   181.2963  11.622581  65.54839   68.70696
2   6       29.44444   190.1667  10.266667  79.10000   87.38384
3   7       59.11538   216.4839   8.941935  83.90323   93.49748
4   8       59.96154   171.8571   8.793548  83.96774   79.71207
5   9       31.44828   167.4333  10.180000  76.90000   71.82689
6  (all)    42.12931   185.9315   9.957516  77.88235   80.05722
```

```
library(plyr) # needed to access . function

> dcast(aqm, variable~month, mean, subset=.(variable=="ozone"))
   variable      5              6           7           8           9
1  ozone     23.61538 29.44444 59.11538 59.96154 31.44828
```

```
> dcast(aqm, variable~month, mean, subset=.(month==5))
   variable       5
1  ozone   23.61538
2  solar.r 181.29630
3   wind   11.62258
4   temp   65.54839
```

한 개의 열 → 여러 개의 열로 분리(피벗테이블) – dcast()

```
> names(ChickWeight)
[1] "weight" "time"   "chick"  "diet“


> head(ChickWeight)
  weight time chick diet
1     42    0     1    1
2     51    2     1    1
3     59    4     1    1
4     64    6     1    1
5     76    8     1    1
6     93   10     1    1

> names(ChickWeight)<-tolower(names(ChickWeight))
> chick_m<-melt(ChickWeight, id=2:4, na.rm=TRUE)

# time별 평균몸무게 효과
> dcast(chick_m, time~variable, mean)

# diet별 평균몸무게 효과
> dcast(chick_m, diet~variable, mean)

# diet별 데이터 수를 알려고 할 경우
> dcast(chick_m, diet~variable, length)

# diet & time별 평균몸무게 효과
# 행기준항목(diet), 열기준항목(time)으로 설정하여 분석하는 경우
> dcast(chick_m, diet~time, mean)
```

❑ time의 평균효과

```
   time    weight
1     0  41.06000
2     2  49.22000
3     4  59.95918
4     6  74.30612
5     8  91.24490
6    10 107.83673
7    12 129.24490
8    14 143.81250
9    16 168.08511
10   18 190.19149
11   20 209.71739
12   21 218.68889
```

❑ diet의 평균효과와 데이터 개수

```
  diet   weight
1    1 102.6455
2    2 122.6167
3    3 142.9500
4    4 135.2627
```

```
  diet weight
1    1    220
2    2    120
3    3    120
4    4    118
```

❑ diet와 time의 평균효과

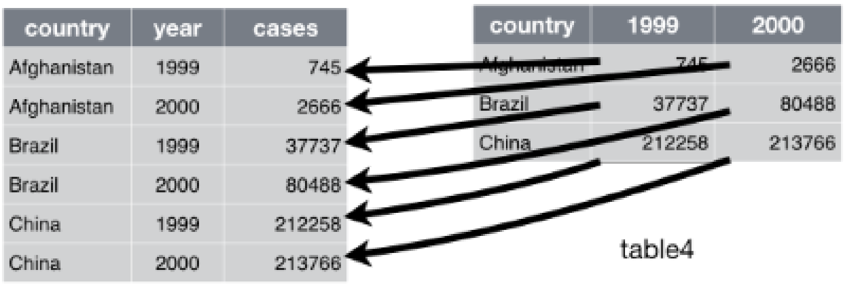
```
  diet    0     2        4        6         8        10
1    1 41.4 47.25 56.47368 66.78947  79.68421  93.05263
2    2 40.7 49.40 59.80000 75.40000  91.70000 108.50000
3    3 40.8 50.40 62.20000 77.90000  98.40000 117.10000
4    4 41.0 51.80 64.50000 83.90000 105.60000 126.00000
        12       14       16       18       20       21
1 108.5263 123.3889 144.6471 158.9412 170.4118 177.7500
2 131.3000 141.9000 164.7000 187.7000 205.6000 214.7000
3 144.4000 164.5000 197.4000 233.1000 258.9000 270.3000
4 151.4000 161.8000 182.0000 202.9000 233.8889 238.5556
```

여러 개의 열 → 한 개 열로 통합 – gather()

여러 열로 나열된 측정데이터를 식별항목(id), 측정항목(variable), 측정값(value)형태로 데이터를 재구성하는 경우에 사용한다(melt함수와 동일함).

❑ 사용법 : gather(데이터, 측정항목명칭, 측정값명칭, -식별항목(id), na.rm=TRUE)

```
> table4a
# A tibble: 3 x 3
      country   `1999`     `2000`
*       <chr>    <int>       <int>
1 Afghanistan    745         2666
2      Brazil     37737      80488
3       China     212258   213766
```



country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

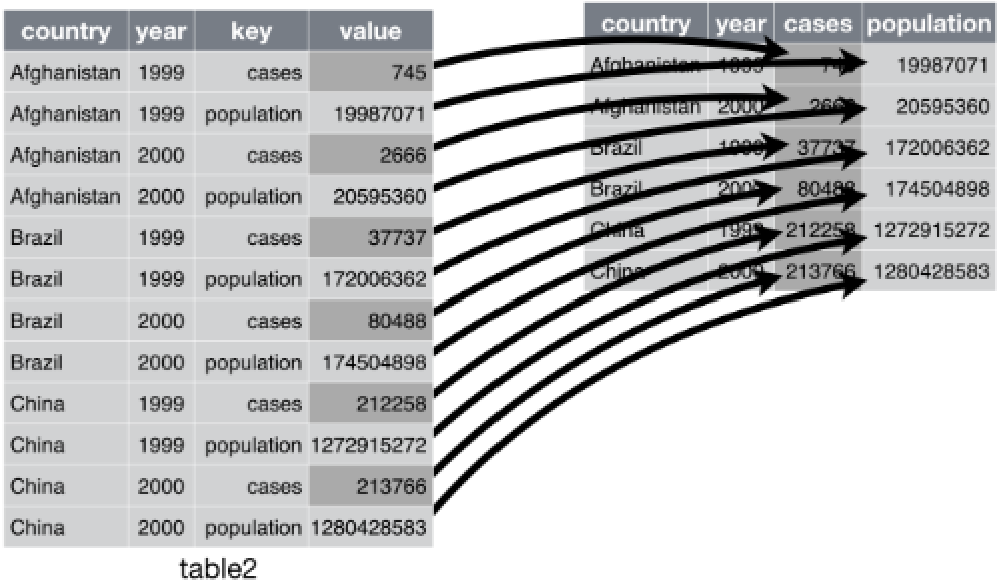
```
# 데이터 : table4a
# 측정항목명칭 : year
# 측정값명칭 : cases
# 식별항목 : 1
# melt(table4a, id=c(‘country’))와 동일한 결과
# gather(table4a, “year”, “case”, -1)과 동일한 결과
> gather(table4a, "year", "cases", 2:3)
# A tibble: 6 x 3
     country          year      cases
       <chr>          <chr>     <int>
1  Afghanistan     1999          745
2      Brazil      1999        37737
3       China      1999       212258
4  Afghanistan     2000         2666
5      Brazil      2000        80488
6       China      2000       213766
```

한 개의 열 → 여러 개의 열로 분리(피벗테이블) – spread()

하나의 변수를 여러 개로 분리할 때, 즉 key와 value의 쌍을 여러 개의 열로 구분할 때 사용한다(dcast함수와 동일함).

❑ 사용법 : spread(원데이터, 열이될 항목, 열의 값으로 사용할 항목)

```
> names(table2)<-c('country', 'year', 'key', 'value')
> table2
```



country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

```
# 원데이터 : table2
# 열이될 항목 : key열 데이터인 cases, population
# 열의 값으로 사용할 항목 : value

> spread(table2, key, value)
# A tibble: 6 x 4
     country       year    cases     population
  *      <chr>      <int>  <int>      <int>
1  Afghanistan  1999    745       19987071
2  Afghanistan  2000   2666       20595360
3      Brazil       1999  37737      172006362
4      Brazil       2000  80488      174504898
5       China       1999 212258     1272915272
6       China       2000 213766     1280428583
```

하나의 변수를 두 개로 분리/합치기 처리 – separate(), unite()

하나의 변수를 두 개로 분리하거나, 두개의 변수를 한 개로 합치려고 할 때 사용한다.

❑ **사용법 : separate(원데이터, 분리대상 열의 명칭, 분리되는 열의 명칭, 분리시작점-1)**

unite(원데이터, 합친열의 명칭, 대상열1의 명칭, 대상열2의 명칭, 분리문자)

```
> table3
# A tibble: 6 x 3
     country        year          rate
*      <chr>        <int>         <chr>
1 Afghanistan   1999         745/19987071
2 Afghanistan   2000         2666/20595360
3      Brazil       1999         37737/172006362
4      Brazil       2000         80488/174504898
5       China       1999         212258/1272915272
6       China       2000         213766/1280428583
```

```
> table6<-separate(table3, year, into = c("century", "year"), sep = 2)
> table6
# A tibble: 6 x 4
     country        century  year           rate
*      <chr>        <chr> <chr>          <chr>
1 Afghanistan    19    99            745/19987071
2 Afghanistan    20    00            2666/20595360
3      Brazil        19    99            37737/172006362
4      Brazil        20    00            80488/174504898
5       China        19    99            212258/1272915272
6       China        20    00            213766/1280428583
```

```
> separate(table3, year, into = c("century", "year"), sep = 2)
# A tibble: 6 x 4
     country         century  year            rate
*      <chr>          <chr> <chr>           <chr>
1 Afghanistan       19    99         745/19987071
2 Afghanistan       20    00         2666/20595360
3      Brazil           19    99         37737/172006362
4      Brazil           20    00         80488/174504898
5       China           19    99         212258/1272915272
6       China           20    00         213766/1280428583
```

```
> unite(table6, "new", century, year, sep = "")
# A tibble: 6 x 3
     country        new          rate
*      <chr>        <chr>        <chr>
1 Afghanistan   1999        745/19987071
2 Afghanistan   2000        2666/20595360
3      Brazil       1999        37737/172006362
4      Brazil       2000        80488/174504898
5       China       1999        212258/1272915272
6       China       2000        213766/1280428583
```

lubridate 소개

데이터 랭글링을 위한 패키지 lubridate를 소개하면 하기와 같다.

❑ lubridate란?

- 날짜시각 데이터를 원활하게 가공하는 데 도움을 주기 위한 목적으로 개발한 패키지
- 설치 : install.packages("lubridate")

❑ 제작자

- Garrett Grolemund와 Hadley Wickham외 8명이 협업하여 만든 날짜처리 패키지이다.

❑ 왜 사용해야 하나?

- 분석시 기준 Timezone이 변경되는 경우 알아서 해결해 준다.
- 날짜와 시각이 조합된 데이터의 산술연산이 필요한 경우 알아서 해결해 준다.
  (예 : 시간별 평균, 요일별 평균, 월별 평균 등)
- 공휴일, 주말을 제외한 날짜의 데이터만을 남기고 싶을때 알아서 해결해 준다
- 주기성 데이터의 주기를 맞추기 위한 시간 스케일링을 만들어 준다.
  (예 : 10분단위의 주기인 01:00, 01:10, 01:20, 01:30, …형태로 데이터를 설정하고 싶은 경우)

날짜 데이터 전처리 – ymd(), year(), with_tz(), …

날짜 데이터를 파싱하여 원하는 형태의 데이터로 변환하거나 해당 값을 가져온다.

- ❑ **사용법 : ymd(), ymd_hms, dmy(), dmy_hms, mdy(), …**
- ❑ **사용법 : year(), month(), mday(), hour(), minute() and second():**
- ❑ **사용법 : with_tz(), force_tz(), as.Date()**

```
ymd(20101215)
#> [1] "2010-12-15"
mdy("4/1/17")
#> [1] "2017-04-01"

bday <- dmy("14/10/1979")
month(bday)
#> [1] 10
wday(bday, label = TRUE)
#> [1] Sun
#> Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat

year(bday) <- 2016
wday(bday, label = TRUE)
#> [1] Fri
#> Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

```
time <- ymd_hms("2010-12-13 15:30:30")
time
#> [1] "2010-12-13 15:30:30 UTC"

# Changes printing
with_tz(time, "America/Chicago")
#> [1] "2010-12-13 09:30:30 CST"

# Changes time
force_tz(time, "America/Chicago")
#> [1] "2010-12-13 15:30:30 CST“

# Extract only date info
> as.Date("2011-06-04 13:30:50")
[1] "2011-06-04"
```

날짜 데이터 전처리 – 부분 정보 추출, 날짜 데이터 연산

날짜 데이터를 파싱하여 원하는 형태의 데이터로 변환하거나 해당 값을 가져온다.

❑ **사용법 : second(), minute(), hour(), day(), wday(), yday(), week(), month(), year()**

```
> ld1 <- ymd_hms("2011-11-04 13:30:50")
> year(ld1)
[1] 2011
> month(ld1)
[1] 11
> day(ld1)
[1] 4
> wday(ld1)
[1] 7
> yday(ld1)
[1] 155
> hour(ld1)
[1] 13
> minute(ld1)
[1] 30
> second(ld1)
[1] 50
> ld1+years(1);ld1+months(1);ld1+days(1)
[1] "2012-11-04 13:30:50 UTC"
[1] "2011-12-04 13:30:50 UTC"
[1] "2011-11-05 13:30:50 UTC“
> ld1+hours(1);ld1+minutes(1);ld1+seconds(1)
[1] "2011-11-04 14:30:50 UTC"
[1] "2011-11-04 13:31:50 UTC"
[1] "2011-11-04 13:30:51 UTC“
> ld1<-hm("22:30");ld1
[1] "22H 30M 0S"
> ld1<-hms("22:30:15");ld1
[1] "22H 30M 15S"
```

```
> ymd("2016-01-30") - days(1:30)
 [1] "2016-01-29" "2016-01-28" "2016-01-27" "2016-01-26" "2016-01-25"
 [6] "2016-01-24" "2016-01-23" "2016-01-22" "2016-01-21" "2016-01-20"
[11] "2016-01-19" "2016-01-18" "2016-01-17" "2016-01-16" "2016-01-15"
[16] "2016-01-14" "2016-01-13" "2016-01-12" "2016-01-11" "2016-01-10"
[21] "2016-01-09" "2016-01-08" "2016-01-07" "2016-01-06" "2016-01-05"
[26] "2016-01-04" "2016-01-03" "2016-01-02" "2016-01-01" "2015-12-31“
```

```
> am(ld1)
[1] FALSE
> pm(ld1)
[1] TRUE
```

```
> round_date(ld1, "hour")
[1] "2011-06-04 14:00:00 UTC"
> floor_date(ld1, "hour")
[1] "2011-06-04 13:00:00 UTC"
> ceiling_date(ld1, "day")
[1] "2011-06-05 UTC"
```

R Cheat sheet - dplyr

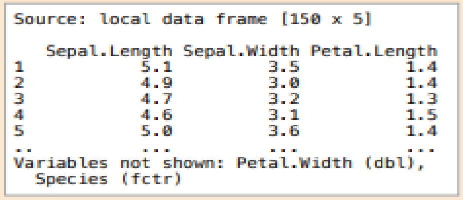
Data Wrangling
with dplyr and tidyr
Cheat Sheet

RStudio

Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:



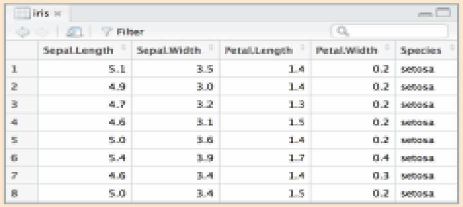
```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

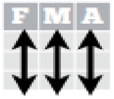
```
      x %>% f(y)  is the same as  f(x, y)
  y %>% f(x, ., z)  is the same as  f(x, y, z )
```

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

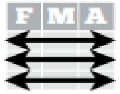
Tidy Data - A foundation for wrangling in R

In a tidy data set:



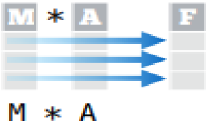
Each **variable** is saved in its own **column**

&

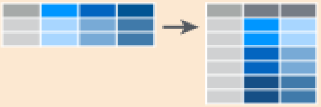


Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

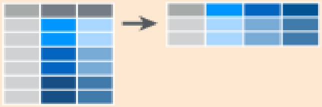


Reshaping Data - Change the layout of a data set



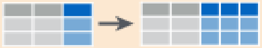
tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.



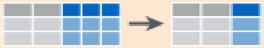
tidyr::**spread(pollution, size, amount)**

Spread rows into columns.



tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.



tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

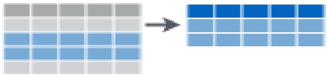
dplyr::**arrange(mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)



dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

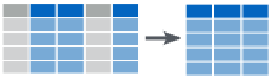
Select rows by position.

dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic			
<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&,|,!,xor,any,all	Boolean operators

Subset Variables (Columns)



dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, **contains**("."))

Select columns whose name contains a character string.

select(iris, **ends_with**("Length"))

Select columns whose name ends with a character string.

select(iris, **everything**())

Select every column.

select(iris, **matches**(".t."))

Select columns whose name matches a regular expression.

select(iris, **num_range**("x", 1:5))

Select columns named x1, x2, x3, x4, x5.

select(iris, **one_of**(c("Species", "Genus")))

Select columns whose names are in a group of names.

select(iris, **starts_with**("Sepal"))

Select columns whose name starts with a character string.

select(iris, Sepal.Length**:**Petal.Width)

Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, **-**Species)

Select all columns except Species.

R Cheat sheet - dplyr

Summarise Data

dplyr::**summarise(iris, avg = mean(Sepal.Length))**
Summarise data into single row of values.

dplyr::**summarise_each(iris, funs(mean))**
Apply summary function to each column.

dplyr::**count(iris, Species, wt = Sepal.Length)**
Count number of rows with each unique value of variable (with or without weights).

summary function

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::**first**
First value of a vector.

dplyr::**last**
Last value of a vector.

dplyr::**nth**
Nth value of a vector.

dplyr::**n**
# of values in a vector.

dplyr::**n_distinct**
# of distinct values in a vector.

IQR
IQR of a vector.

min
Minimum value in a vector.

max
Maximum value in a vector.

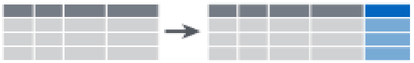
mean
Mean value of a vector.

median
Median value of a vector.

var
Variance of a vector.

sd
Standard deviation of a vector.

Make New Variables



dplyr::**mutate(iris, sepal = Sepal.Length + Sepal. Width)**
Compute and append one or more new columns.

dplyr::**mutate_each(iris, funs(min_rank))**
Apply window function to each column.

dplyr::**transmute(iris, sepal = Sepal.Length + Sepal. Width)**
Compute one or more new columns. Drop original columns.

window function

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

dplyr::**lead**
Copy with values shifted by 1.

dplyr::**lag**
Copy with values lagged by 1.

dplyr::**dense_rank**
Ranks with no gaps.

dplyr::**min_rank**
Ranks. Ties get min rank.

dplyr::**percent_rank**
Ranks rescaled to [0, 1].

dplyr::**row_number**
Ranks. Ties got to first value.

dplyr::**ntile**
Bin vector into n buckets.

dplyr::**between**
Are values between a and b?

dplyr::**cume_dist**
Cumulative distribution.

dplyr::**cumall**
Cumulative `all`

dplyr::**cumany**
Cumulative `any`

dplyr::**cummean**
Cumulative `mean`

cumsum
Cumulative `sum`

cummax
Cumulative `max`

cummin
Cumulative `min`

cumprod
Cumulative `prod`

pmax
Element-wise `max`

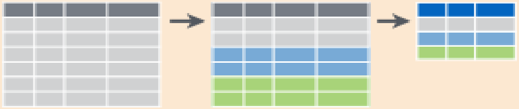
pmin
Element-wise `min`

Group Data

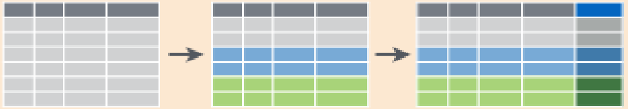
dplyr::**group_by(iris, Species)**
Group data into rows with the same value of Species.

dplyr::**ungroup(iris)**
Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(...)
Compute separate summary row for each group.



iris %>% group_by(Species) %>% mutate(...)
Compute new variables by group.



Combine Data Sets

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::**left_join(a, b, by = "x1")**
Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::**right_join(a, b, by = "x1")**
Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::**inner_join(a, b, by = "x1")**
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::**full_join(a, b, by = "x1")**
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::**semi_join(a, b, by = "x1")**
All rows in a that have a match in b.

x1	x2
C	3

dplyr::**anti_join(a, b, by = "x1")**
All rows in a that do not have a match in b.

y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

Set Operations

x1	x2
B	2
C	3

dplyr::**intersect(y, z)**
Rows that appear in both y and z.

x1	x2
A	1
B	2
C	3
D	4

dplyr::**union(y, z)**
Rows that appear in either or both y and z.

x1	x2
A	1

dplyr::**setdiff(y, z)**
Rows that appear in y but not z.

Binding

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

dplyr::**bind_rows(y, z)**
Append z to y as new rows.

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

dplyr::**bind_cols(y, z)**
Append z to y as new columns.
Caution: matches rows by position.

R Cheat sheet - dplyr

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries. Also **summarise_**().
summarise(mtcars, avg = mean(mpg))

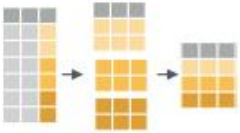
count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to created a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.



```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE**)**
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …**)**
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

filter(.data, …**)** Extract rows that meet logical criteria. Also **filter_()**. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values. Also **distinct_()**. *distinct(iris, Species)*

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows. *sample_frac(iris, 0.5, replace = TRUE)*

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …**)** Select rows by position. Also **slice_()**. *slice(iris, 10:15)*

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …**)**
Order rows by values of a column (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL**)**
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

select(.data, …**)**
Extract columns by name. Also **select_if()**
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)**	**num_range(**prefix, range**)**	:, e.g. mpg:cyl
ends_with(match**)**	**one_of(**…**)**	-, e.g, -Species
matches(match**)**	**starts_with(**match**)**	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …**)**
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …**)**
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …**)** Apply funs to every column. Use with **funs()**.
mutate_all(faithful, funs(log(.), log2(.)))

mutate_at(.tbl, .cols, .funs, …**)** Apply funs to specific columns. Use with **funs(), vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

mutate_if(.tbl, .predicate, .funs, …**)**
Apply funs to all columns of one type.
Use with **funs()**.
mutate_if(iris, is.numeric, funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL**)** Add new column(s).
add_column(mtcars, new = 1:32)

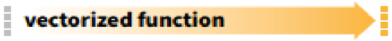
rename(.data, …**)** Rename columns.
rename(iris, Length = Sepal.Length)

R Cheat sheet - dplyr

Vectorized Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.



OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

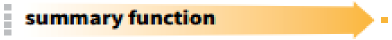
MISC

dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.



COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

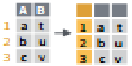
quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

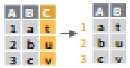
IQR() - Inter-Quartile Range
mad() - mean absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.



rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

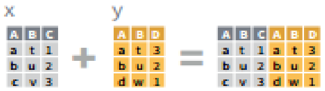


column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also has_**rownames()**, **remove_rownames()**

Summary Functions

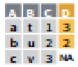
COMBINE VARIABLES



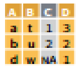
Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

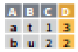
Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.



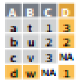
left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join matching values from y to x.



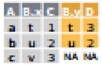
right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join matching values from x to y.



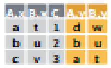
inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join data. Retain only rows with matches.



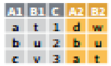
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join data. Retain all values, all rows.



Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

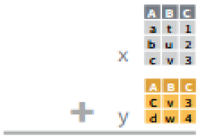


Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

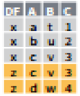


Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

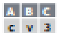
COMBINE CASES



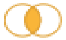
Use **bind_rows()** to paste tables below each other as they are.

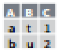


bind_rows(…, .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

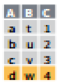


intersect(x, y, …)
Rows that appear in both x and z.





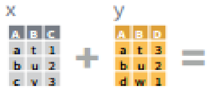
setdiff(x, y, …)
Rows that appear in x but not z.



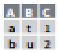
union(x, y, …)
Rows that appear in x or z. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

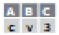
EXTRACT ROWS



Use a "**Filtering Join**" to filter one table against the rows of another.



semi_join(x, y, by = NULL, …)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.



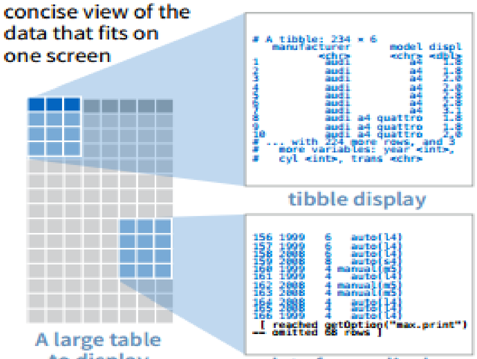
anti_join(x, y, by = NULL, …)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

R Cheat sheet – tidyr

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen



```
# A tibble: 234 × 6
   manufacturer        model displ
          <chr>        <chr> <dbl>
1          audi           a4   1.8
2          audi           a4   1.8
3          audi           a4   2.0
4          audi           a4   2.0
5          audi           a4   2.8
6          audi           a4   2.8
7          audi           a4   3.1
8          audi   a4 quattro   1.8
9          audi   a4 quattro   1.8
10         audi   a4 quattro   2.0
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

tibble display

```
156 1999    6    auto(l4)
157 1999    6    auto(l4)
158 2008    6    auto(l4)
159 2008    8    auto(s4)
160 1999    4  manual(m5)
161 1999    4    auto(l4)
162 2008    4  manual(m5)
163 2008    4  manual(m5)
164 2008    4    auto(l4)
165 2008    4    auto(l4)
166 1999    4    auto(l4)
 [ reached getOption("max.print")
-- omitted 68 rows ]
```

data frame display

A large table to display

- Control the default appearance with options:

  options(tibble.print_max = n,
  tibble.print_min = m, tibble.width = Inf)

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(…)
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…)
Construct by rows.

```
tribble( ~x,  ~y,
          1,  "a",
          2,  "b",
          3,  "c")
```

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <dbl>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …) Convert data frame to tibble.

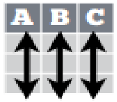
enframe(x, name = "name", value = "value")
Convert named vector to a tibble

is_tibble(x) Test whether x is a tibble.

Tidy Data with Tidyr

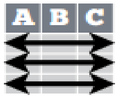
Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:



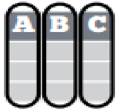
Each **variable** is in its own **column**

&



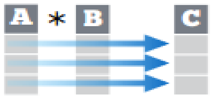
Each **observation**, or **case**, is in its own **row**

Tidy data:



Makes variables easy to access as vectors

A * B -> C



Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

Gather moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

Spread moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

key value

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...)
Drop rows containing NA's in … columns.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(x, x2)

fill(data, ..., .direction = c("down", "up"))
Fill in NA's in … columns with most recent non-NA values.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(x, x2)

replace_na(data, replace = list(), ...)
Replace NA's by column.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(x,list(x2 = 2), x2)

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list())
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, ...)
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...)

Separate each cell in a column to make several columns.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate(table3, rate, into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE)

Separate each cell in a column to make several rows. Also **separate_rows_()**.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

separate_rows(table3, rate)

unite(data, col, ..., sep = "_", remove = TRUE)

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

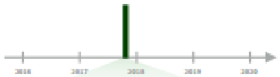
→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

unite(table5, century, year, col = "year", sep = "")

R Cheat sheet – lubridate

Date-times



2017-11-28 12:00:00

2017-11-28 12:00:00

A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

*dt <- **as_datetime**(1511870400)*
## "2017-11-28 12:00:00 UTC"

2017-11-28

A **date** is a day stored as the number of days since 1970-01-01

*d <- **as_date**(17498)*
## "2017-11-28"

12:00:00

An hms is a **time** stored as the number of seconds since 00:00:00

*t <- hms::**as.hms**(85)*
## 00:01:25

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

2017-11-28T14:02:00 — **ymd_hms**(), **ymd_hm**(), **ymd_h**(). *ymd_hms("2017-11-28T14:02:00")*

2017-22-12 10:00:00 — **ydm_hms**(), **ydm_hm**(), **ydm_h**(). *ydm_hms("2017-22-12 10:00:00")*

11/28/2017 1:02:03 — **mdy_hms**(), **mdy_hm**(), **mdy_h**(). *mdy_hms("11/28/2017 1:02:03")*

1 Jan 2017 23:59:59 — **dmy_hms**(), **dmy_hm**(), **dmy_h**(). *dmy_hms("1 Jan 2017 23:59:59")*

20170131 — **ymd**(), **ydm**(). *ymd(20170131)*

July 4th, 2000 — **mdy**(), **myd**(). *mdy("July 4th, 2000")*

4th of July '99 — **dmy**(), **dym**(). *dmy("4th of July '99")*

2001: Q3 — **yq**() Q for quarter. *yq("2001: Q3")*

2:01 — hms::**hms**() Also lubridate::**hms**(), **hm**() and **ms**(), which return periods.* *hms::hms(sec = 0, min= 1, hours = 2)*

2017.5 — **date_decimal**(decimal, tz = "UTC") *date_decimal(2017.5)*

now(tzone = "") Current time in tz (defaults to system tz). *now()*

today(tzone = "") Current date in a tz (defaults to system tz). *today()*

fast_strptime() Faster strptime. *fast_strptime('9/1/01', '%y/%m/%d')*

parse_date_time() Easier strptime. *parse_date_time("9/1/01", "ymd")*

GET AND SET COMPONENTS

Use an accessor function to get a component.

d ## "2017-11-28"
day(d) ## 28

Assign into an accessor function to change a component in place.

day(d) <- 1
d ## "2017-11-01"

2018-01-31 11:59:59 — **date**(x) Date component. *date(dt)*

2018-01-31 11:59:59 — **year**(x) Year. *year(dt)*
isoyear(x) The ISO 8601 year.
epiyear(x) Epidemiological year.

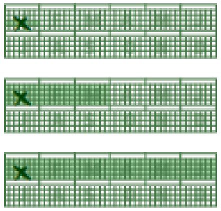
2018-01-31 11:59:59 — **month**(x, label, abbr) Month. *month(dt)*

2018-01-31 11:59:59 — **day**(x) Day of month. *day(dt)*
wday(x,label,abbr) Day of week.
qday(x) Day of quarter.

2018-01-31 11:59:59 — **hour**(x) Hour. *hour(dt)*

2018-01-31 11:59:59 — **minute**(x) Minutes. *minute(dt)*

2018-01-31 11:59:59 — **second**(x) Seconds. *second(dt)*



week(x) Week of the year. *week(dt)*
isoweek() ISO 8601 week.
epiweek() Epidemiological week.

quarter(x, with_year = FALSE) Quarter. *quarter(dt)*

semester(x, with_year = FALSE) Semester. *semester(dt)*

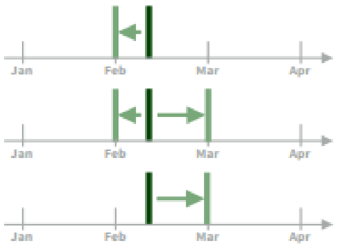
am(x) Is it in the am? *am(dt)*
pm(x) Is it in the pm? *pm(dt)*

dst(x) Is it daylight savings? *dst(d)*

leap_year(x) Is it a leap year? *leap_year(d)*

update(object, ..., simple = FALSE) *update(dt, mday = 2, hour = 1)*

Round Date-times



floor_date(x, unit = "second") Round down to nearest unit. *floor_date(dt, unit = "month")*

round_date(x, unit = "second") Round to nearest unit. *round_date(dt, unit = "month")*

ceiling_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit. *ceiling_date(dt, unit = "month")*

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. *rollback(dt)*

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date**() and **stamp_time**().

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

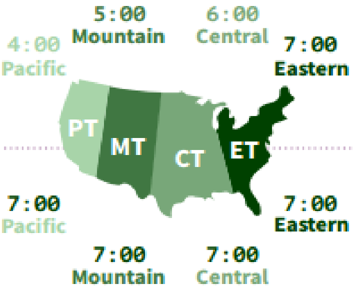
Tip: use a date with day > 12

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. *OlsonNames()*



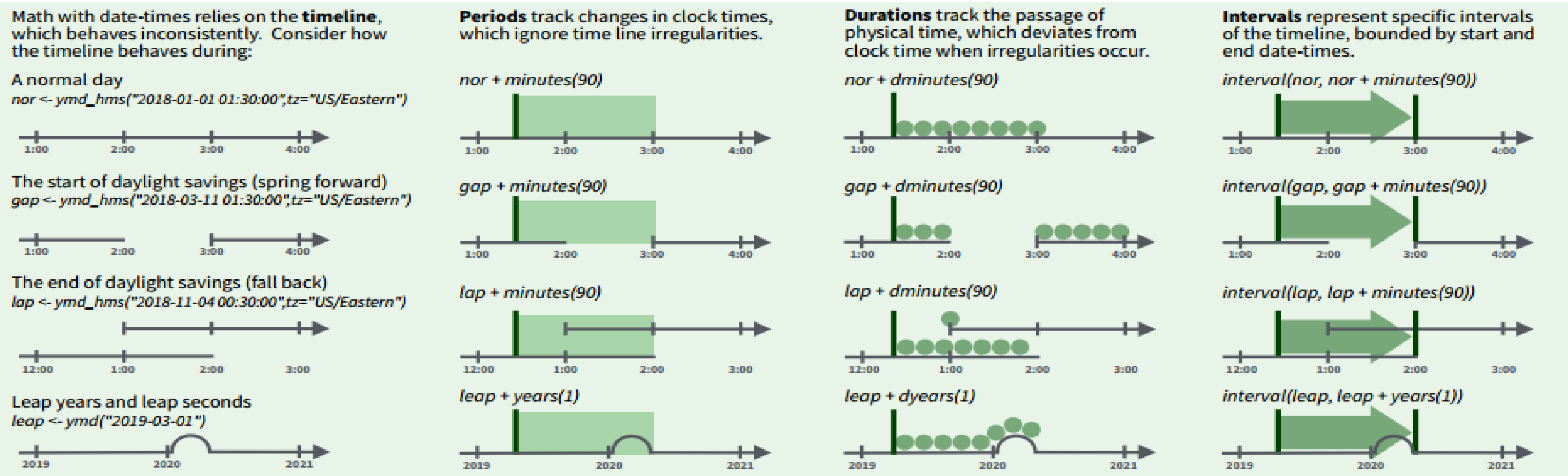
with_tz(time, tzone = "") Get the **same date-time** in a new time zone (a new clock time). *with_tz(dt, "US/Pacific")*

force_tz(time, tzone = "") Get the **same clock time** in a new time zone (a new date-time). *force_tz(dt, "US/Pacific")*

Studio

R Cheat sheet – lubridate

Math with Date-times – Lubridate provides three classes of timespans to facilitate math with dates and date-times



Math with date-times relies on the **timeline**, which behaves inconsistently. Consider how the timeline behaves during:

A normal day
nor <- ymd_hms("2018-01-01 01:30:00",tz="US/Eastern")

The start of daylight savings (spring forward)
gap <- ymd_hms("2018-03-11 01:30:00",tz="US/Eastern")

The end of daylight savings (fall back)
lap <- ymd_hms("2018-11-04 00:30:00",tz="US/Eastern")

Leap years and leap seconds
leap <- ymd("2019-03-01")

Periods track changes in clock times, which ignore time line irregularities.

nor + minutes(90)

gap + minutes(90)

lap + minutes(90)

leap + years(1)

Durations track the passage of physical time, which deviates from clock time when irregularities occur.

nor + dminutes(90)

gap + dminutes(90)

lap + dminutes(90)

leap + dyears(1)

Intervals represent specific intervals of the timeline, bounded by start and end date-times.

interval(nor, nor + minutes(90))

interval(gap, gap + minutes(90))

interval(lap, lap + minutes(90))

interval(leap, leap + years(1))

Not all years are 365 days due to **leap days**.

Not all minutes are 60 seconds due to **leap seconds**.

It is possible to create an imaginary date by adding **months**, e.g. February 31st

```
jan31 <- ymd(20180131)
jan31 + months(1)
## NA
```

%m+% and **%m-%** will roll imaginary dates to the last day of the previous month.

```
jan31 %m+% months(1)
## "2018-02-28"
```

add_with_rollback(e1, e2, roll_to_first = TRUE) will roll imaginary dates to the first day of the new month.

```
add_with_rollback(jan31, months(1),
roll_to_first = TRUE)
## "2018-03-01"
```

PERIODS

Add or subtract periods to model events that happen at specific clock times, like the NYSE opening bell.

Make a period with the name of a time unit ***pluralized***, e.g.

```
p <- months(3) + days(12)
p
"3m 12d 0H 0M 0S"
```

Number of months | Number of days | etc.

years(x = 1) x years.
months(x) x months.
weeks(x = 1) x weeks.
days(x = 1) x days.
hours(x = 1) x hours.
minutes(x = 1) x minutes.
seconds(x = 1) x seconds.
milliseconds(x = 1) x milliseconds.
microseconds(x = 1) x microseconds
nanoseconds(x = 1) x nanoseconds.
picoseconds(x = 1) x picoseconds.

period(num = NULL, units = "second", ...) An automation friendly period constructor. *period(5, unit = "years")*

as.period(x, unit) Coerce a timespan to a period, optionally in the specified units. Also **is.period**(). *as.period(i)*

period_to_seconds(x) Convert a period to the "standard" number of seconds implied by the period. Also **seconds_to_period**(). *period_to_seconds(p)*

DURATIONS

Add or subtract durations to model physical processes, like battery life. Durations are stored as seconds, the only time unit with a consistent length. **Difftimes** are a class of durations found in base R.

Make a duration with the name of a period prefixed with a ***d***, e.g.

```
dd <- ddays(14)
dd
"1209600s (~2 weeks)"
```

Exact length in seconds | Equivalent in common units

dyears(x = 1) 31536000x seconds.
dweeks(x = 1) 604800x seconds.
ddays(x = 1) 86400x seconds.
dhours(x = 1) 3600x seconds.
dminutes(x = 1) 60x seconds.
dseconds(x = 1) x seconds.
dmilliseconds(x = 1) x × 10^{-3} seconds.
dmicroseconds(x = 1) x × 10^{-6} seconds.
dnanoseconds(x = 1) x × 10^{-9} seconds.
dpicoseconds(x = 1) x × 10^{-12} seconds.

duration(num = NULL, units = "second", ...) An automation friendly duration constructor. *duration(5, unit = "years")*

as.duration(x, …) Coerce a timespan to a duration. Also **is.duration**(), **is.difftime**(). *as.duration(i)*

make_difftime(x) Make difftime with the specified number of units. *make_difftime(99999)*

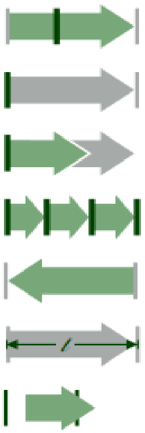
INTERVALS

Divide an interval by a duration to determine its physical length, divide and interval by a period to determine its implied length in clock time.

Make an interval with **interval**() or **%--%**, e.g.

```
i <- interval(ymd("2017-01-01"), d)      ## 2017-01-01 UTC--2017-11-28 UTC
j <- d %--% ymd("2017-12-31")            ## 2017-11-28 UTC--2017-12-31 UTC
```

Start Date | End Date



a **%within%** b Does interval or date-time *a* fall within interval *b*? *now() %within% i*

int_start(int) Access/set the start date-time of an interval. Also **int_end**(). *int_start(i) <- now(); int_start(i)*

int_aligns(int1, int2) Do two intervals share a boundary? Also **int_overlaps**(). *int_aligns(i, j)*

int_diff(times) Make the intervals that occur between the date-times in a vector. *v <-c(dt, dt + 100, dt + 1000); int_diff(v)*

int_flip(int) Reverse the direction of an interval. Also **int_standardize**(). *int_flip(i)*

int_length(int) Length in seconds. *int_length(i)*

int_shift(int, by) Shifts an interval up or down the timeline by a timespan. *int_shift(i, days(-1))*

as.interval(x, start, …) Coerce a timespans to an interval with the start date-time. Also **is.interval**(). *as.interval(days(1), start = now())*