# DATA 106 - Notes 3

*Jillian Morrison*

*September 8, 2019*

## Difference between dataframes in Wide and Long formats

- This matters for:
  - Plotting
  - Analysis

## Example of Wide format

Dataset: Annual Percentage Rate of Interest Charged on New-Car Loans at Nine of the Largest Banks in 3 American Cities
(Source: Consumer Reports, October 1985, pp 582-586)

- ID: id of Bank
- City.1: Atlanta
- City.2: Chicago
- City.3: Houston

```
  ID City.1 City.2 City.3
1  1  13.75  14.25  14.00
2  2  13.75  13.00  14.00
3  3  13.50  12.75  13.51
4  4  13.50  12.50  13.50
5  5  13.00  12.50  13.50
6  6  13.00  12.40  13.25
7  7  13.00  12.30  13.00
8  8  12.75  11.90  12.50
9  9  12.50  11.90  12.50
```

## Long format: an example

- ID: participant ID
- City: a factor with three levels City.1, City.2, City.3
- Rate: Rate Charged by the banks

Below are 4 ways to create the same long dataset

```
>
> #Using reshape2 Package
> library(reshape2)
> long_ <- melt(data=wide, id=c("ID"), value.name = "Rate", variable.name='City')
>
> head(long_, n=10)
   ID   City  Rate
1   1 City.1 13.75
```

```
2   2 City.1 13.75
3   3 City.1 13.50
4   4 City.1 13.50
5   5 City.1 13.00
6   6 City.1 13.00
7   7 City.1 13.00
8   8 City.1 12.75
9   9 City.1 12.50
10  1 City.2 14.25
```

## Long format: an example

Below are 4 ways to create the same long dataset

```
> library(dplyr)
>
> #Using tidyr Package
> library(dplyr)
> library(tidyr)
> long <- gather(data=wide, key='City', value='Rate', -ID)              #METHOD 1
> long <- wide %>% gather( key = City, value = Rate, City.1:City.3)     #METHOD 2
```

```
> #Using reshape2 Package
> library(dplyr)
> library(reshape2)
> long <- melt(data=wide, id=c("ID"), value.name = "Rate", variable.name='City')     #METHOD 3
> long <- wide %>% melt( id=c("ID"), value.name = "Rate", variable.name='City')      #METHOD 4
>
> head(long, n=10)
   ID   City  Rate
1   1 City.1 13.75
2   2 City.1 13.75
3   3 City.1 13.50
4   4 City.1 13.50
5   5 City.1 13.00
6   6 City.1 13.00
7   7 City.1 13.00
8   8 City.1 12.75
9   9 City.1 12.50
10  1 City.2 14.25
```

use `?melt` or `?gather` for more information

## Show the different levels of the variable City in the dataset

```
> # show levels of City
> library(dplyr)
> long %>%group_by(City) %>% slice(1)   ##METHOD 1  - produces a dataset with the first entry of each l
# A tibble: 3 x 3
# Groups:   City [3]
```

```
     ID City    Rate
  <int> <fct>  <dbl>
1     1 City.1  13.8
2     1 City.2  14.2
3     1 City.3  14
```

```
> unique(long$City)     ##METHOD 2
[1] City.1 City.2 City.3
Levels: City.1 City.2 City.3
```

```
>
> levels(long$City)     ##METHOD 3
[1] "City.1" "City.2" "City.3"
```

## Converting from Long to Wide

Convert the previous Long format back to Wide format (4 ways):

```
> #Using tidyr Package
> library(tidyr)
> library(dplyr)
> WIDE_from_long <- spread(data= long, key = City, value = Rate)     ##METHOD 1
> WIDE_from_long <- long %>% spread(key = City, value = Rate)        ##METHOD 2
```

```
>
> #Using reshape2 Package
> library(dplyr)
> library(reshape2)
> WIDE_from_long <- dcast(data=long, formula = ID ~ City, value.var = "Rate")     ##METHOD 3
> WIDE_from_long <-  long %>% dcast( formula = ID ~ City, value.var = "Rate")     ##METHOD 4
>
>
> WIDE_from_long
  ID City.1 City.2 City.3
1  1  13.75  14.25  14.00
2  2  13.75  13.00  14.00
3  3  13.50  12.75  13.51
4  4  13.50  12.50  13.50
5  5  13.00  12.50  13.50
6  6  13.00  12.40  13.25
7  7  13.00  12.30  13.00
8  8  12.75  11.90  12.50
9  9  12.50  11.90  12.50
```

use `?spread` or `?dcast` for more information

## R implementation: summary

Wide to Long

- `gather` (gather {tidyr}): convert wide format to long format

- `melt` (melt {reshape2}): convert wide format to long format

Long to Wide

- `spread` (spread {tidyr}) : convert long format to wide format
- `dcast` (dcast {reshape2}) : convert long format to wide format

## Separating and Uniting

Materials used in this section

- Chapter 9 of book "R for data science" by Wickham and Grolemun - See syllabus for how to access this text
- R libraries used: `tidyverse` and `tidyr`

## The "table3"

```
> library(tidyverse)
> 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
```

Important info: - "rate" - ratio of "cases" to "population"

INSTRUCTIONS:

Extract "cases" and "population" from "rate"

How would you suggest we do this?

## Separating columns using criteria

```
> # separate{tidyr}
> table3 %>% separate(rate, into = c("cases", "population"),
+                     sep = "/",convert = TRUE)
# 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
```

SO, separating `rate` into 2 columns by the `/` symbol. The first part that appears before `/` becomes `cases` column and the second part after `/` becomes `population` column

# Subsetting datasets

## Accessing entries of dataframe

- access via numeric index
  - You can access entry using [row index, column index]
  - You can access multiple entries by using the vector function c(), with a vector of indices to access

```
> da = as.data.frame(matrix(1:6,nrow=2,ncol=3))    #Creating dataframe called da to use as example
> da
  V1 V2 V3
1  1  3  5
2  2  4  6
```

```
> da[c(1,2),]  #Accesses 1st and 2nd rows of da
  V1 V2 V3
1  1  3  5
2  2  4  6
```

```
> da[,c(1,2)]  #Accesses 1st and 2nd columns of da
  V1 V2
1  1  3
2  2  4
```

```
> da[1,2]      #Accesses entry at row 1, column 2 position of da
[1] 3
```

## Accessing entries of dataframe

- access via logic index
  - A logical object in R is either `TRUE` or `FALSE`
  - You can create a vector of logical entries to access columns or rows listed as `TRUE`

```
> da.2 = as.data.frame(matrix(5:13,ncol=3,nrow=3))  #creating dataframe
> da.2
  V1 V2 V3
1  5  8 11
2  6  9 12
3  7 10 13
```

```
> da.2[c(F,T),]   #Accesses second row of da.2
  V1 V2 V3
2  6  9 12
```

```
> da.2[,c(F,T)]   #Accesses second column of da.2.
[1]  8  9 10
```

Notice that R everything after the last `T` is assumed to be `False`

## Sampling

`sample` - (sample {base}) samples a specified size from the elements of x using either with or without replacement:

```
sample(x, size, replace = FALSE, prob = NULL)
```

Example:

```
> set.seed(556)    ##What is a seed?
> sample(x=1:10,size=5,replace = F)
[1] 2 5 8 4 7
```

```
> sample(x=1:10,size=5,replace = T)
[1] 9 8 2 8 2
```

Why are the samples the same everytime and for everyone?

## Operating on rows

Materials used in this section

- Chapters 3 and 10 of book "R for data science" by Wickham and Grolemun
- R libraries used: `tidyverse`, `dplyr`, `nycflights13`

## NYC flight data: `nycflights13`

nycflights13 contains 5 tibbles ... what is a tibble?

- airlines: carrier name
- airports: info about the airports
- planes: info about planes - by tail number
- weather: hourly weather at each NYC airport
- flights: airplane scheduled departure and arrival times, tailnumber, etc

for more info on Tibbles: https://r4ds.had.co.nz/tibbles.html

## NYC flight data: flights

Looking at weather dataset

```
> #install.packages("nycflights13")
> library(nycflights13)   #Need to use library nycflights13 to get dataset
> head(flights)
# A tibble: 6 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      517            515         2      830
2  2013     1     1      533            529         4      850
3  2013     1     1      542            540         2      923
4  2013     1     1      544            545        -1     1004
5  2013     1     1      554            600        -6      812
6  2013     1     1      554            558        -4      740
# ... with 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

BTW, tibbles are data frames, but they tweak some older behaviours to make life a little easier.

## Looking at the `flights` tibble

```
> names(flights)   ### column names of the weather dataset
 [1] "year"           "month"          "day"
 [4] "dep_time"       "sched_dep_time" "dep_delay"
 [7] "arr_time"       "sched_arr_time" "arr_delay"
[10] "carrier"        "flight"         "tailnum"
[13] "origin"         "dest"           "air_time"
[16] "distance"       "hour"           "minute"
[19] "time_hour"
```

```
> head(flights)   ##First 6 entries of weather dataset
# A tibble: 6 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      517            515         2      830
2  2013     1     1      533            529         4      850
3  2013     1     1      542            540         2      923
4  2013     1     1      544            545        -1     1004
5  2013     1     1      554            600        -6      812
6  2013     1     1      554            558        -4      740
# ... with 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
> unique(flights$year)  ## Unique entries of Year variable in weather dataset
[1] 2013
```

## Subseting using `filter`

`filter` - (filter{dplyr}) subsets observations using conditional statements based on the values of ther entries

Want to subset the flights tibble to get a dataset with data from 2013, months 1 and 2 and tailnumber N14228

```
> fl_2013<- filter(flights, year == 2013, month %in% c(1, 2), tailnum == 'N14228')
> fl_2013
# A tibble: 22 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     8     1435           1440        -5     1717
 3  2013     1     9      717            700        17      812
 4  2013     1     9     1143           1144        -1     1425
 5  2013     1    13      835            824        11     1030
 6  2013     1    16     1829           1730        59     2117
 7  2013     1    22     1902           1808        54     2214
 8  2013     1    23     1050           1056        -6     1143
 9  2013     1    23     1533           1529         4     1641
10  2013     1    25      724            720         4     1000
# ... with 12 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

Note: You can add as much or as little conditons as you would like in the filter function.

## Subsetting using `filter`

- What if you wanted to do an `OR` conditional

```
> filter(flights, !(arr_delay > 120 | dep_delay > 120))
# A tibble: 316,050 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ... with 316,040 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

- We filtered the dataset by arrivals or delays more than 120.
- What is the difference between `OR` and `AND` in this case?

### Rearranging using arrange

arrange (arrange{dplyr}) - allows you to change the order of the rows

```
> arrange(.data=flights, year, month, day, dep_time)  #Ordering dataset by year, month, day and departu
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ... with 336,766 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

## Operating on columns

### Selecting columns with select

select allows to select column gives by their column names

Suppose you want to select all columns between year and dep_time, including year and dep_time?

```
> sel<-select(.data=flights, year:dep_time)  #Method 1
> head(sel, n=3)                             #viewing first 3 rows of sel... without n=3, recall head f
# A tibble: 3 x 4
   year month   day dep_time
  <int> <int> <int>    <int>
1  2013     1     1      517
2  2013     1     1      533
3  2013     1     1      542
```

```
> sel<-select(.data=flights, year, month, day, dep_time)   #Method2
> head(sel, n=3)                                           #viewing first 3 rows of sel... without n=3,
# A tibble: 3 x 4
   year month   day dep_time
  <int> <int> <int>    <int>
1  2013     1     1      517
2  2013     1     1      533
3  2013     1     1      542
```

### Selecting columns with select

Suppose you wanted to select all columns except those from year to day (inclusive)?

```
>
> sel.2<-select(.data = flights, -(year:day))
> head(sel.2)
# A tibble: 6 x 16
  dep_time sched_dep_time dep_delay arr_time sched_arr_time
     <int>          <int>     <dbl>    <int>          <int>
1      517            515         2      830            819
2      533            529         4      850            830
3      542            540         2      923            850
4      544            545        -1     1004           1022
5      554            600        -6      812            837
6      554            558        -4      740            728
# ... with 11 more variables: arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

## Adding a new variable in column

- `mutate` (mutate{dplyr}) adds new columns at the end of a dataset,
- You can set these new columns to be often functions of existing columns

Let's create a smaller dataset with columns from year to day, all columns that ends with delay and distance and air_time

```
>
> flights_sml <- select(flights, year:day, ends_with("delay"), distance, air_time)
> head(flights_sml)
# A tibble: 6 x 7
   year month   day dep_delay arr_delay distance air_time
  <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
1  2013     1     1         2        11     1400      227
2  2013     1     1         4        20     1416      227
3  2013     1     1         2        33     1089      160
4  2013     1     1        -1       -18     1576      183
5  2013     1     1        -6       -25      762      116
6  2013     1     1        -4        12      719      150
```

## Adding column(s) via `mutate`

Now, lets: - add a new column called gain which is the arrival delay minus the departure delay - add a new column called speed which is the ratio of distance to air time multiplied by 60

```
> ##Method 1
> new_Fl = mutate(flights_sml, gain = arr_delay - dep_delay, speed = distance / air_time * 60)
```

```
> #Method 2 - uses piping (i.e. %>%) from dplyr packages
> new_Fl = flights_sml%>% mutate( gain = arr_delay - dep_delay, speed = distance / air_time * 60)
> head(new_Fl)
# A tibble: 6 x 9
   year month   day dep_delay arr_delay distance air_time  gain
```

```
  <int> <int> <int>      <dbl>      <dbl>     <dbl>     <dbl> <dbl>
1  2013     1     1          2         11      1400       227     9
2  2013     1     1          4         20      1416       227    16
3  2013     1     1          2         33      1089       160    31
4  2013     1     1         -1        -18      1576       183   -17
5  2013     1     1         -6        -25       762       116   -19
6  2013     1     1         -4         12       719       150    16
# ... with 1 more variable: speed <dbl>
```