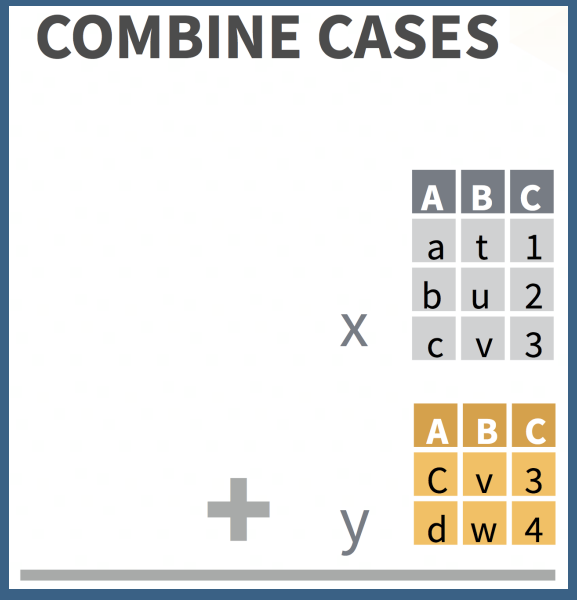
Tidy Data

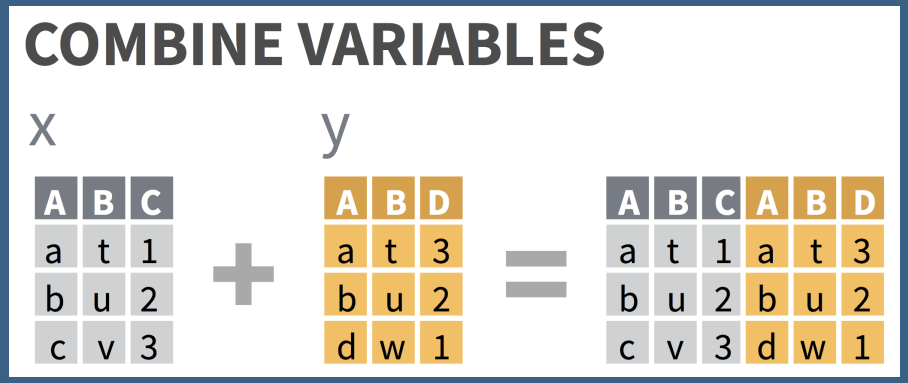
STAT 220

Bastola

January 24 2022

Last time: Combining data sets

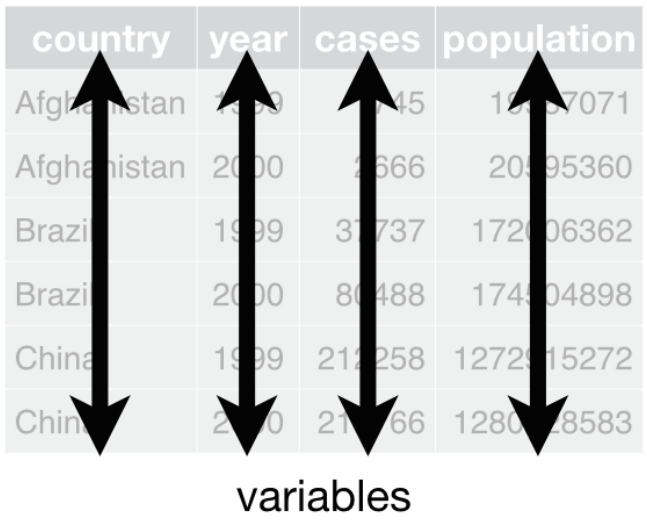


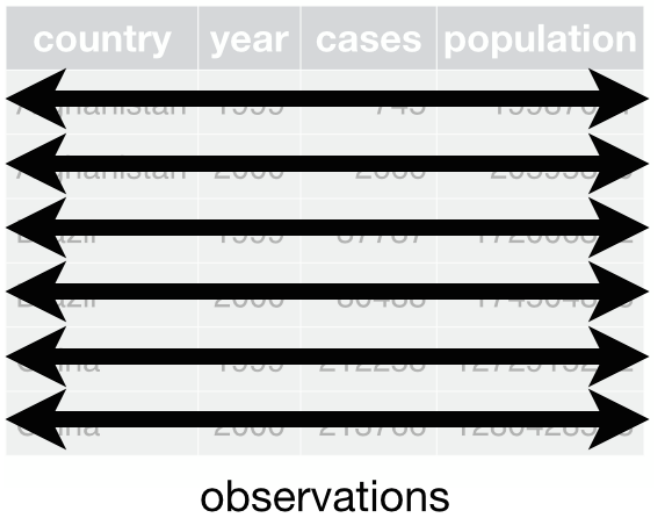


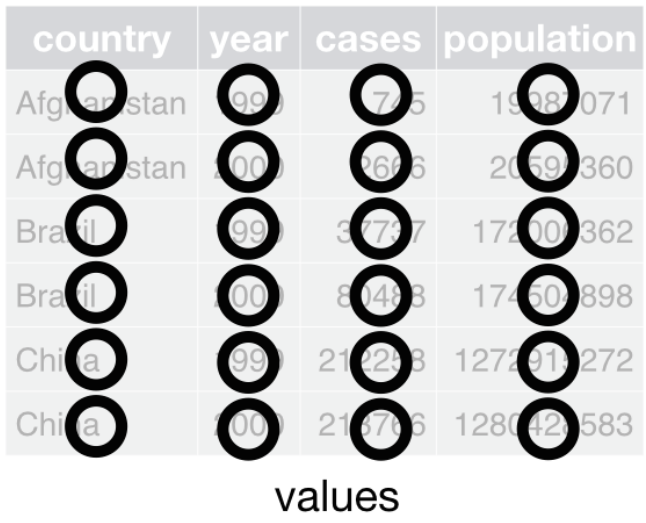
dplyr data transformation cheatsheet

What are tidy data?

1. Each variable forms a column
2. Each observation forms a row
3. Each value has its own cell







G. Grolemond & H. Wickham's R for Data Science

Untidy data: example 1

```
untidy_data <- tibble(
  name = c("Ana","Bob","Cara"),
  meds = c("advil 600mg 2xday","tylenol 650mg 4xday", "advil 200mg 3xday")
)
```

```
untidy_data
# A tibble: 3 × 2
  name  meds
  <chr> <chr>
1 Ana   advil 600mg 2xday
2 Bob   tylenol 650mg 4xday
3 Cara  advil 200mg 3xday
```

Tidy data: example 1

We will learn how to do this!

```
untidy_data %>%
  separate(col = meds,
           into = c("med_name","dose_mg","times_per_day"),
           sep=" ") %>%
  mutate(times_per_day = as.numeric(str_remove(times_per_day, "xday")),
         dose_mg = as.numeric(str_remove(dose_mg, "mg")))
# A tibble: 3 × 4
  name  med_name dose_mg times_per_day
  <chr> <chr>      <dbl>         <dbl>
1 Ana   advil        600             2
2 Bob   tylenol      650             4
3 Cara  advil        200             3
```

Untidy data: example 2

```
untidy_data2 <- tibble(
  name = c("Ana","Bob","Cara"),
  wt_07_01_2021 = c(100, 150, 140),
  wt_08_01_2021 = c(104, 155, 138),
  wt_09_01_2021 = c(NA, 160, 142)
)
```

```
untidy_data2
# A tibble: 3 × 4
  name  wt_07_01_2021 wt_08_01_2021 wt_09_01_2021
  <chr>         <dbl>         <dbl>         <dbl>
1 Ana             100           104            NA
2 Bob             150           155           160
3 Cara            140           138           142
```

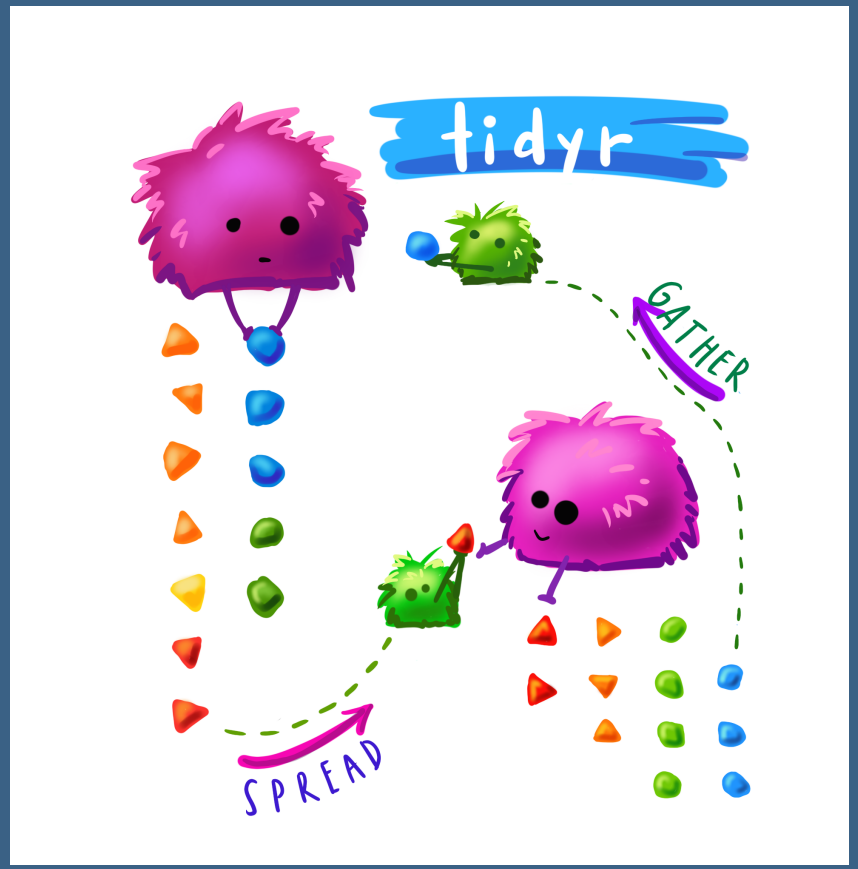
Tidy data: example 2

We will learn how to do this!

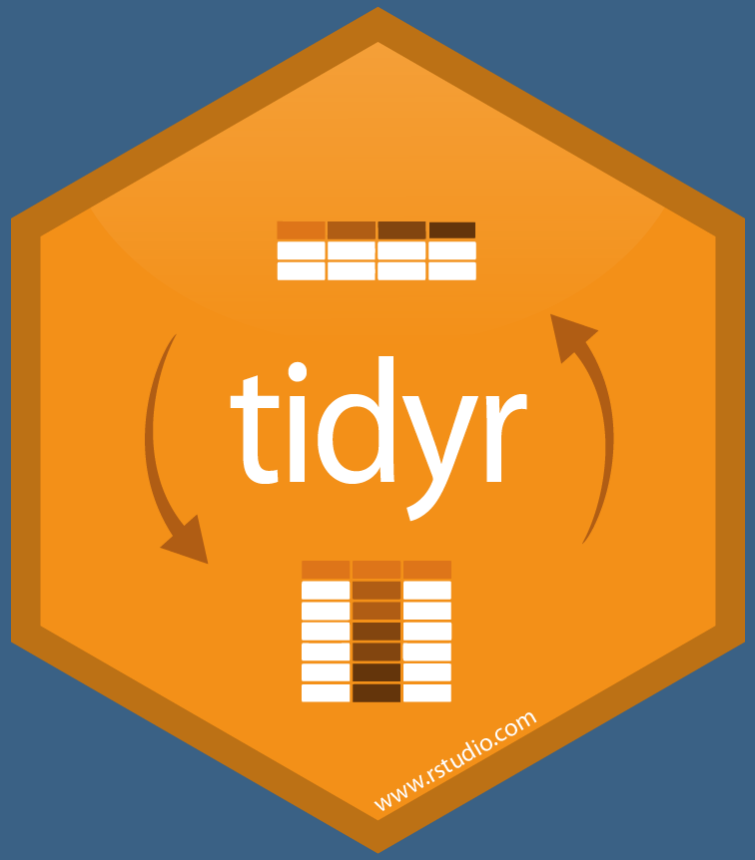
```
untidy_data2 %>%
  gather(key = "date", value = "weight", -name) %>%
  mutate(date = str_remove(date,"wt_"),
         date = dmy(date))     # dmy() is a function in the lubridate package
# A tibble: 9 × 3
  name  date       weight
  <chr> <date>      <dbl>
1 Ana   2021-01-07    100
2 Bob   2021-01-07    150
3 Cara  2021-01-07    140
4 Ana   2021-01-08    104
5 Bob   2021-01-08    155
6 Cara  2021-01-08    138
7 Ana   2021-01-09     NA
8 Bob   2021-01-09    160
9 Cara  2021-01-09    142
```

Reshaping data

wide vs. long



Allison Horst



tidyr

Wide vs. long data

- **Wide** data has one row per subject, with multiple columns for their repeated measurements
- **Long** data has multiple rows per subject, with one column for the measurement variable and another indicating from when/where the repeated measures are from

wide

id	SBP_visit1	SBP_visit2	SBP_visit3
a	130	110	112
b	120	116	122
c	130	136	138
d	119	106	118

long

id	visit	SBP
a	1	130
b	1	120
c	1	130
d	1	119
a	2	110
b	2	116
c	2	136
d	2	106
a	3	112
b	3	122
c	3	138
d	3	118

Your Turn 1

Please git clone the repository on tidy R to your local folder.

03:00

```
BP_wide
# A tibble: 4 × 5
  id    sex   SBP_v1 SBP_v2 SBP_v3
  <chr> <chr>  <dbl>  <dbl>  <dbl>
1 a     F        130    110    112
2 b     M        120    116    122
3 c     M        130    136    138
4 d     F        119    106    118
```

- What do you think the data in the table are measures of? How can we tell the data are wide?
- What would a long data look like? How many columns will it have?

Wide to long (Old Version) : gather()

```
BP_wide
# A tibble: 4 × 5
  id    sex   SBP_v1 SBP_v2 SBP_v3
  <chr> <chr>  <dbl>  <dbl>  <dbl>
1 a     F        130    110    112
2 b     M        120    116    122
3 c     M        130    136    138
4 d     F        119    106    118
```

gather columns into rows to make the data long. Need to **specify**:

- **new column names**
  - **key**: stores row names of wide data's gathered columns
  - **value**: stores data values
- **which columns to gather**

```
BP_long1 <- BP_wide %>%
  gather(key = "visit", value = "SBP",
         SBP_v1:SBP_v3)
BP_long1
# A tibble: 12 × 4
   id    sex   visit    SBP
   <chr> <chr> <chr>  <dbl>
 1 a     F     SBP_v1   130
 2 b     M     SBP_v1   120
 3 c     M     SBP_v1   130
 4 d     F     SBP_v1   119
 5 a     F     SBP_v2   110
 6 b     M     SBP_v2   116
 7 c     M     SBP_v2   136
 8 d     F     SBP_v2   106
 9 a     F     SBP_v3   112
10 b     M     SBP_v3   122
11 c     M     SBP_v3   138
12 d     F     SBP_v3   118
```

Wide to long: `pivot_longer()`

```
BP_wide
# A tibble: 4 × 5
  id    sex   SBP_v1 SBP_v2 SBP_v3
  <chr> <chr>  <dbl>  <dbl>  <dbl>
1 a     F        130    110    112
2 b     M        120    116    122
3 c     M        130    136    138
4 d     F        119    106    118
```

`pivot_longer` lengthens data, increasing the number of rows and decreasing the number of columns. Need to **specify**:

- **new column names**
  - **names_to**: stores row names of wide data's columns
  - **values_to**: stores data values
- **which columns to pivot**

```
BP_long2 <- BP_wide %>%
  pivot_longer(names_to = "visit",
               values_to = "SBP",
               cols = SBP_v1:SBP_v3)
BP_long2
# A tibble: 12 × 4
   id    sex   visit    SBP
   <chr> <chr> <chr>  <dbl>
 1 a     F     SBP_v1   130
 2 a     F     SBP_v2   110
 3 a     F     SBP_v3   112
 4 b     M     SBP_v1   120
 5 b     M     SBP_v2   116
 6 b     M     SBP_v3   122
 7 c     M     SBP_v1   130
 8 c     M     SBP_v2   136
 9 c     M     SBP_v3   138
10 d     F     SBP_v1   119
11 d     F     SBP_v2   106
12 d     F     SBP_v3   118
```

Long to wide (Old Version) : spread()

```
BP_long1
# A tibble: 12 × 4
   id    sex   visit    SBP
   <chr> <chr> <chr>  <dbl>
 1 a     F     SBP_v1   130
 2 b     M     SBP_v1   120
 3 c     M     SBP_v1   130
 4 d     F     SBP_v1   119
 5 a     F     SBP_v2   110
 6 b     M     SBP_v2   116
 7 c     M     SBP_v2   136
 8 d     F     SBP_v2   106
 9 a     F     SBP_v3   112
10 b     M     SBP_v3   122
11 c     M     SBP_v3   138
12 d     F     SBP_v3   118
```

spread rows into columns to make the data wide. Need to **specify** which columns in the long data to use:

- **key** column: has the variable names
- **value** column: has the data values

```
BP_wide1 <- BP_long1 %>%
  spread(key = "visit", value = "SBP")
BP_wide1
# A tibble: 4 × 5
  id    sex   SBP_v1 SBP_v2 SBP_v3
  <chr> <chr>  <dbl>  <dbl>  <dbl>
1 a     F        130    110    112
2 b     M        120    116    122
3 c     M        130    136    138
4 d     F        119    106    118
```

Long to wide: `pivot_wider()`

```
BP_long1
# A tibble: 12 × 4
   id    sex   visit    SBP
   <chr> <chr> <chr>  <dbl>
 1 a     F     SBP_v1   130
 2 b     M     SBP_v1   120
 3 c     M     SBP_v1   130
 4 d     F     SBP_v1   119
 5 a     F     SBP_v2   110
 6 b     M     SBP_v2   116
 7 c     M     SBP_v2   136
 8 d     F     SBP_v2   106
 9 a     F     SBP_v3   112
10 b     M     SBP_v3   122
11 c     M     SBP_v3   138
12 d     F     SBP_v3   118
```

`pivot_wider` increases number of columns and decreases the number of rows. Need to **specify**:

- **new column names**
  - **names_from**: get the name of the column
  - **values_from**: get the cell values from

```
BP_wide2 <- BP_long1 %>%
  pivot_wider(names_from = "visit",
              values_from = "SBP")
BP_wide2
# A tibble: 4 × 5
  id    sex   SBP_v1 SBP_v2 SBP_v3
  <chr> <chr>  <dbl>  <dbl>  <dbl>
1 a     F        130    110    112
2 b     M        120    116    122
3 c     M        130    136    138
4 d     F        119    106    118
```

Separate Info

```
BP_long2
# A tibble: 12 × 4
   id    sex   visit    SBP
   <chr> <chr> <chr>  <dbl>
 1 a     F     SBP_v1   130
 2 a     F     SBP_v2   110
 3 a     F     SBP_v3   112
 4 b     M     SBP_v1   120
 5 b     M     SBP_v2   116
 6 b     M     SBP_v3   122
 7 c     M     SBP_v1   130
 8 c     M     SBP_v2   136
 9 c     M     SBP_v3   138
10 d     F     SBP_v1   119
11 d     F     SBP_v2   106
12 d     F     SBP_v3   118
```

```
BP_long3 <- BP_long2 %>%
  separate(visit, c("acrnym", "visit"))
BP_long3
# A tibble: 12 × 5
   id    sex   acrnym visit   SBP
   <chr> <chr> <chr>  <chr> <dbl>
 1 a     F     SBP    v1      130
 2 a     F     SBP    v2      110
 3 a     F     SBP    v3      112
 4 b     M     SBP    v1      120
 5 b     M     SBP    v2      116
 6 b     M     SBP    v3      122
 7 c     M     SBP    v1      130
 8 c     M     SBP    v2      136
 9 c     M     SBP    v3      138
10 d     F     SBP    v1      119
11 d     F     SBP    v2      106
12 d     F     SBP    v3      118
```

Remove and Clean up Columns

```
BP_long3
# A tibble: 12 × 5
   id    sex   acrnym visit   SBP
   <chr> <chr> <chr>  <chr> <dbl>
 1 a     F     SBP    v1      130
 2 a     F     SBP    v2      110
 3 a     F     SBP    v3      112
 4 b     M     SBP    v1      120
 5 b     M     SBP    v2      116
 6 b     M     SBP    v3      122
 7 c     M     SBP    v1      130
 8 c     M     SBP    v2      136
 9 c     M     SBP    v3      138
10 d     F     SBP    v1      119
11 d     F     SBP    v2      106
12 d     F     SBP    v3      118
```

Goal: Get rid of `acrnym` column and remove the string `v` from the `visit` variable's values.

```
BP_long3 <- BP_long3 %>%
  select(-acrnym) %>%
  mutate(visit = str_replace(visit,"v",""))
BP_long3
# A tibble: 12 × 4
   id    sex   visit   SBP
   <chr> <chr> <chr> <dbl>
 1 a     F     1       130
 2 a     F     2       110
 3 a     F     3       112
 4 b     M     1       120
 5 b     M     2       116
 6 b     M     3       122
 7 c     M     1       130
 8 c     M     2       136
 9 c     M     3       138
10 d     F     1       119
11 d     F     2       106
12 d     F     3       118
```

Make cleaned-up long data wide

```
head(BP_long3, 2)
# A tibble: 2 × 4
  id    sex   visit   SBP
  <chr> <chr> <chr> <dbl>
1 a     F     1       130
2 a     F     2       110
BP_wide4 <- BP_long3 %>%
  pivot_wider(names_from = "visit",
              values_from = "SBP")
BP_wide4
# A tibble: 4 × 5
  id    sex     `1`   `2`   `3`
  <chr> <chr> <dbl> <dbl> <dbl>
1 a     F       130   110   112
2 b     M       120   116   122
3 c     M       130   136   138
4 d     F       119   106   118
```

Problem: have numbers as column names

Solution: have row names start with the key column's name separated by a character

```
BP_wide5 <- BP_long3 %>%
  pivot_wider(names_from = "visit",
              values_from = "SBP",
              names_prefix = "value_")
BP_wide5
# A tibble: 4 × 5
  id    sex   value_1 value_2 value_3
  <chr> <chr>   <dbl>   <dbl>   <dbl>
1 a     F         130     110     112
2 b     M         120     116     122
3 c     M         130     136     138
4 d     F         119     106     118
```

Your Turn 2

```
DBP_wide
# A tibble: 4 × 6
  id    sex   v1.DBP v2.DBP v3.DBP   age
  <chr> <chr>  <dbl>  <dbl>  <dbl> <dbl>
1 a     F         88     78     94    23
2 b     M         84     78     82    56
3 c     M        102     96     94    41
4 d     F         70     76     74    38
```

- Transform
- Clean-up
- Rename
- Join

```
BP_long2
# A tibble: 12 × 4
   id    sex   visit   SBP
   <chr> <chr> <chr> <dbl>
 1 a     F     1       130
 2 b     M     1       120
 3 c     M     1       130
 4 d     F     1       119
 5 a     F     2       110
 6 b     M     2       116
 7 c     M     2       136
 8 d     F     2       106
 9 a     F     3       112
10 b     M     3       122
11 c     M     3       138
12 d     F     3       118
```

06:00

Data cleaning

(messy NAs, dates, factors)

Removing missing data: drop_na()

A simple example:

```
foo <- tibble(id = 3:5,
                  name = c("Al",
                           "Wu",
                           "Flo"),
                  height = c(3, NA, 2.8),
                  years = c(50,33,NA))
foo
# A tibble: 3 × 4
     id name  height years
  <int> <chr>  <dbl> <dbl>
1     3 Al       3      50
2     4 Wu      NA      33
3     5 Flo      2.8    NA
```

Remove *all* rows with **any missing data**

```
foo %>% drop_na()
# A tibble: 1 × 4
     id name  height years
  <int> <chr>  <dbl> <dbl>
1     3 Al       3      50
```

Remove rows with NA in **selected columns**

```
foo %>% drop_na(height)
# A tibble: 2 × 4
     id name  height years
  <int> <chr>  <dbl> <dbl>
1     3 Al       3      50
2     5 Flo      2.8    NA
```

Replace NAs with another value: `replace_na()`

Use with `mutate()`

```
foo
# A tibble: 3 × 4
     id name  height years
  <int> <chr>  <dbl> <dbl>
1     3 Al         3    50
2     4 Wu        NA    33
3     5 Flo      2.8    NA
```

```
foo %>%
  mutate(height = replace_na(height, "Unknown"),
         years = replace_na(years, 0) )
# A tibble: 3 × 4
     id name  height  years
  <int> <chr> <chr>   <dbl>
1     3 Al    3          50
2     4 Wu    Unknown    33
3     5 Flo   2.8         0
```

--

Dates with `lubridate`

- Convert characters to special "Date" type
- Easy date magic examples:
  - add and subtract dates
  - convert to minutes/years/etc
  - change timezones
  - add 1 month to a date...
- `lubridate` cheat sheet

Allison Horst

What kind of date do you have?

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)*

`lubridate` cheat sheet

```
timedata <- tibble(name = c("Yi","Mo","Dee"),
                   dob=c("10/31/1952",
                         "1/12/1984",
                         "2/02/2002"),
                   age= c("69", "38", "19"))

timedata %>%
  mutate(dob_date = mdy(dob),
         age_num = as.numeric(age))
# A tibble: 3 × 5
  name  dob        age   dob_date   age_num
  <chr> <chr>      <chr> <date>       <dbl>
1 Yi    10/31/1952 69    1952-10-31      69
2 Mo    1/12/1984  38    1984-01-12      38
3 Dee   2/02/2002  19    2002-02-02      19
```

Math with dates

```
timedata %>%
  mutate(dob = mdy(dob),                          # convert to a date
         dob_year = year(dob),                    # extract the year
         time_since_birth = dob %--% today(),     # create an "interval"
         age = time_since_birth %/% years(1),     # modulus on "years"
         dobplus = dob + days(30)                 # add 30 days
  )
# A tibble: 3 × 6
  name  dob          age dob_year time_since_birth                dobplus
  <chr> <date>     <dbl>    <dbl> <Interval>                       <date>
1 Yi    1952-10-31    69     1952 1952-10-31 UTC--2022-01-24 UTC 1952-11-30
2 Mo    1984-01-12    38     1984 1984-01-12 UTC--2022-01-24 UTC 1984-02-11
3 Dee   2002-02-02    19     2002 2002-02-02 UTC--2022-01-24 UTC 2002-03-04
```

Allison Horst

Clean messy column names with clean_names()

```
bar <- tibble("First Name"= c("Yi","DJ"), "last init" = c("C","R"), "% in" = c(0.1, 0.5),
              "ñ$$$"= 1:2, " "=3:2," hi"=c("a","b"),  "null"=c(NA,NA))
bar
# A tibble: 2 × 7
  `First Name` `last init` `% in` `ñ$$$`   ` ` ` hi` null
  <chr>        <chr>         <dbl>  <int> <int> <chr> <lgl>
1 Yi           C               0.1      1     3 a     NA
2 DJ           R               0.5      2     2 b     NA
```

```
bar %>% clean_names() %>%         # in the janitor package
  remove_empty(c("rows","cols"))    # also useful
# A tibble: 2 × 6
  first_name last_init percent_in     n     x hi
  <chr>      <chr>          <dbl> <int> <int> <chr>
1 Yi         C                0.1     1     3 a
2 DJ         R                0.5     2     2 b
```

Your Turn 3

10:00

```
messy_data
# A tibble: 3 × 3
  NAME  `months follow up` `Date of visit`
  <chr> <chr>              <chr>
1 J N   ""                 July 31, 2003
2 A C   "10"               Nov 12, 2005
3 D E   "11"               Aug 3, 2007
```

Perform the following data cleaning tasks

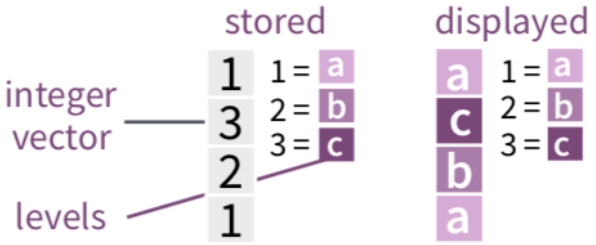
- Clean column names
- Replace missing data
- Convert column types
- Math with date
- Missing data
- String manipulation

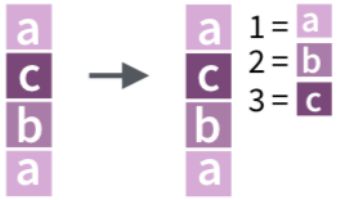
Factors - categorical data

- Clean and order factors with `forcats` package
- Important for visualization, statistical modeling (i.e. for `lm()`), and creating tables
- See `forcats` cheatsheet and `forcats` vignette

Factors

R represents categorical data with factors. A **factor** is an integer vector with a **levels** attribute that stores a set of mappings between integers and categorical values. When you view a factor, R displays not the integers, but the values associated with them.

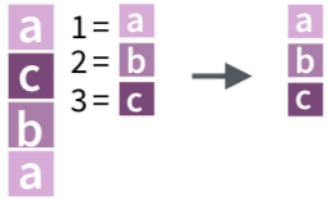




Create a factor with factor()

factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA) Convert a vector to a factor. Also **as_factor**.

f <- factor(c("a", "c", "b", "a"), levels = c("a", "b", "c"))



Return its levels with levels()

levels(x) Return/set the levels of a factor. *levels(f); levels(f) <- c("x","y","z")*

Use unclass() to see its structure

forcats examples - specify levels fct_relevel()

```
mydata <- tibble(
  id = 1:4,
  grade=c("9th","10th","11th","9th")) %>%
  mutate(grade_fac = factor(grade))
levels(mydata$grade_fac)
[1] "10th" "11th" "9th"
```

```
mydata %>%
  arrange(grade_fac)
# A tibble: 4 × 3
     id grade grade_fac
  <int> <chr> <fct>
1     2 10th  10th
2     3 11th  11th
3     1 9th   9th
4     4 9th   9th
```

```
mydata <- mydata %>%
  mutate(
    grade_fac =
      fct_relevel(grade_fac,
                  c("9th","10th","11th")))
levels(mydata$grade_fac)
[1] "9th"  "10th" "11th"
mydata %>% arrange(grade_fac)
# A tibble: 4 × 3
     id grade grade_fac
  <int> <chr> <fct>
1     1 9th   9th
2     4 9th   9th
3     2 10th  10th
4     3 11th  11th
```

forcats examples - collapse levels

```
mydata <- tibble(loc = c("SW","NW","NW","NE","SE","SE"))
mydata %>% mutate(
  loc_fac = factor(loc),
  loc2 = fct_collapse(loc_fac,                          # collapse levels
                      south = c("SW","SE"),
                      north = c("NE","NW")),
  loc3 = fct_lump(loc_fac, n=2, other_level = "other") # most common 2 levels + other
  )
# A tibble: 6 × 4
  loc   loc_fac loc2  loc3
  <chr> <fct>   <fct> <fct>
1 SW    SW      south other
2 NW    NW      north NW
3 NW    NW      north NW
4 NE    NE      north other
5 SE    SE      south SE
6 SE    SE      south SE
```

Acknowledgements

- Parts of the slides are adapted from Jessica Minnier's OCTRI BERD R Courses instruction materials.