Tidy data and data wrangling

Prof. Maria Tackett

STA 199

Click for PDF of slides

Tidy data

Tidy data

> Happy families are all alike; every unhappy family is unhappy in its own way.
>
> Leo Tolstoy

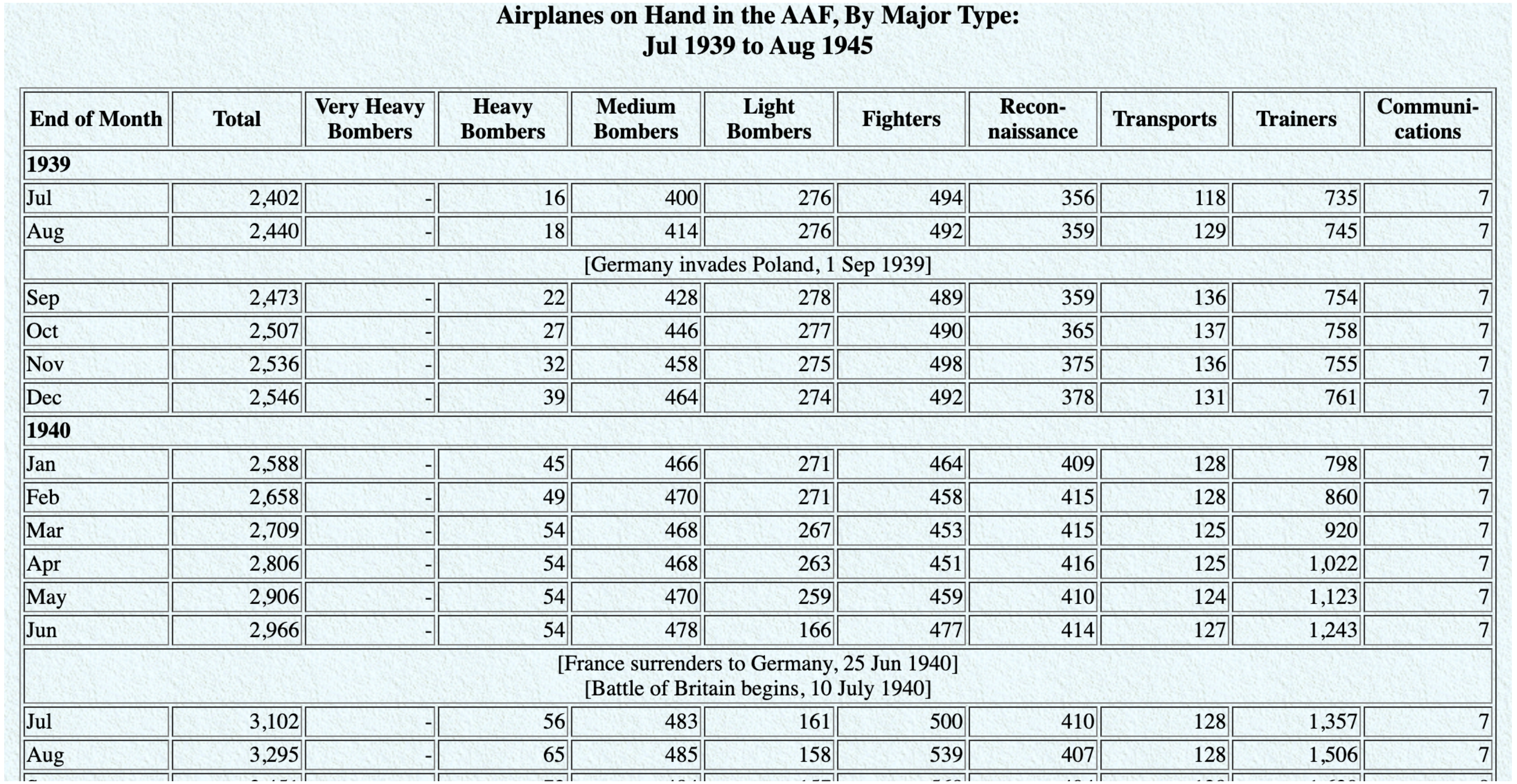
Characteristics of tidy data:

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.

Characteristics of untidy data:

!@#$%^&*()

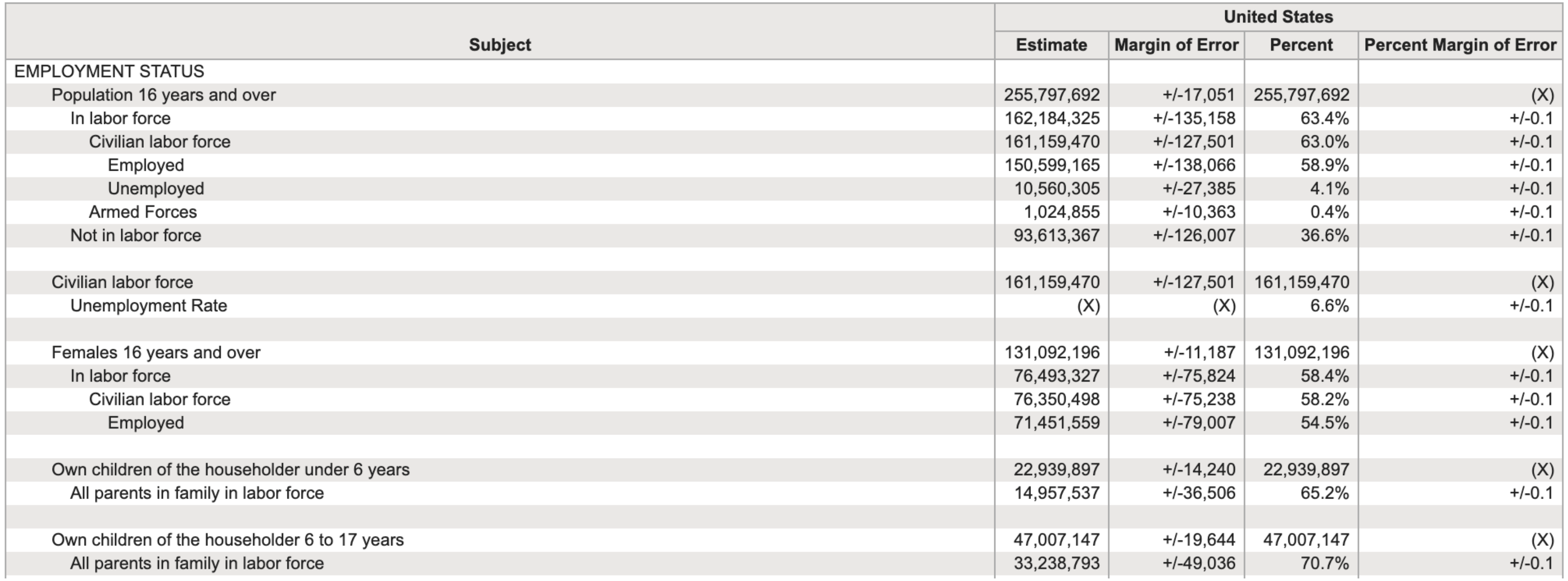
What makes this data not tidy?



Airplanes on Hand in the AAF, By Major Type:
Jul 1939 to Aug 1945

End of Month	Total	Very Heavy Bombers	Heavy Bombers	Medium Bombers	Light Bombers	Fighters	Recon-naissance	Transports	Trainers	Communi-cations
1939										
Jul	2,402	-	16	400	276	494	356	118	735	7
Aug	2,440	-	18	414	276	492	359	129	745	7
[Germany invades Poland, 1 Sep 1939]										
Sep	2,473	-	22	428	278	489	359	136	754	7
Oct	2,507	-	27	446	277	490	365	137	758	7
Nov	2,536	-	32	458	275	498	375	136	755	7
Dec	2,546	-	39	464	274	492	378	131	761	7
1940										
Jan	2,588	-	45	466	271	464	409	128	798	7
Feb	2,658	-	49	470	271	458	415	128	860	7
Mar	2,709	-	54	468	267	453	415	125	920	7
Apr	2,806	-	54	468	263	451	416	125	1,022	7
May	2,906	-	54	470	259	459	410	124	1,123	7
Jun	2,966	-	54	478	166	477	414	127	1,243	7
[France surrenders to Germany, 25 Jun 1940] [Battle of Britain begins, 10 July 1940]										
Jul	3,102	-	56	483	161	500	410	128	1,357	7
Aug	3,295	-	65	485	158	539	407	128	1,506	7

What makes this data not tidy?



Subject	United States			
	Estimate	Margin of Error	Percent	Percent Margin of Error
EMPLOYMENT STATUS				
Population 16 years and over	255,797,692	+/-17,051	255,797,692	(X)
In labor force	162,184,325	+/-135,158	63.4%	+/-0.1
Civilian labor force	161,159,470	+/-127,501	63.0%	+/-0.1
Employed	150,599,165	+/-138,066	58.9%	+/-0.1
Unemployed	10,560,305	+/-27,385	4.1%	+/-0.1
Armed Forces	1,024,855	+/-10,363	0.4%	+/-0.1
Not in labor force	93,613,367	+/-126,007	36.6%	+/-0.1
Civilian labor force	161,159,470	+/-127,501	161,159,470	(X)
Unemployment Rate	(X)	(X)	6.6%	+/-0.1
Females 16 years and over	131,092,196	+/-11,187	131,092,196	(X)
In labor force	76,493,327	+/-75,824	58.4%	+/-0.1
Civilian labor force	76,350,498	+/-75,238	58.2%	+/-0.1
Employed	71,451,559	+/-79,007	54.5%	+/-0.1
Own children of the householder under 6 years	22,939,897	+/-14,240	22,939,897	(X)
All parents in family in labor force	14,957,537	+/-36,506	65.2%	+/-0.1
Own children of the householder 6 to 17 years	47,007,147	+/-19,644	47,007,147	(X)
All parents in family in labor force	33,238,793	+/-49,036	70.7%	+/-0.1

[US Census Fact Finder, General Economic Characteristics, ACS 2017]

Summary tables

Is each of the following a dataset or a summary table?

```
## # A tibble: 87 x 3
##    name               height  mass
##    <chr>               <int> <dbl>
##  1 Luke Skywalker        172    77
##  2 C-3PO                 167    75
##  3 R2-D2                  96    32
##  4 Darth Vader           202   136
##  5 Leia Organa           150    49
##  6 Owen Lars             178   120
##  7 Beru Whitesun lars    165    75
##  8 R5-D4                  97    32
##  9 Biggs Darklighter     183    84
## 10 Obi-Wan Kenobi        182    77
## # … with 77 more rows
```

```
## # A tibble: 3 x 2
##   gender    avg_height
##   <chr>          <dbl>
## 1 feminine        165.
## 2 masculine       177.
## 3 <NA>            181.
```

Displaying data

```
starwars %>%
  select(name, height, mass)
```

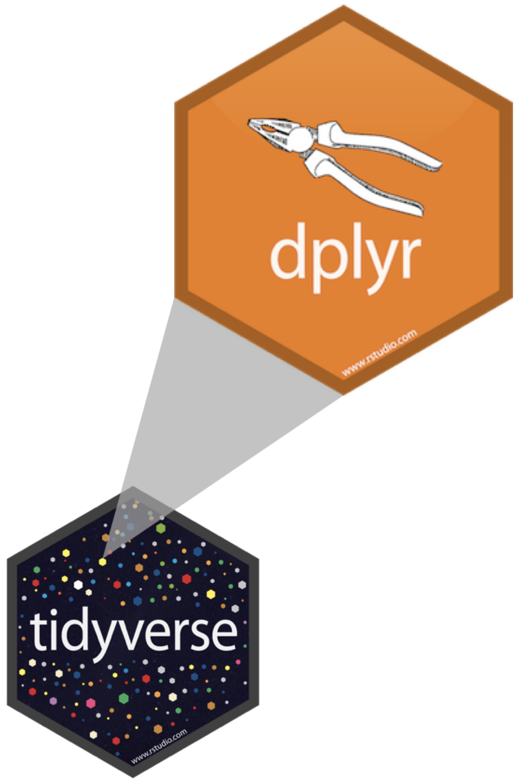
Summarizing data

```
starwars %>%
  group_by(gender) %>%
  summarize(
    avg_height = mean(height, na.rm = TRUE) %>% round(2)
  )
```

Grammar of data wrangling

A grammar of data wrangling...

... based on the concepts of functions as verbs that manipulate data frames



- **`select`**: pick columns by name
- **`arrange`**: reorder rows
- **`slice`**: pick rows using index(es)
- **`filter`**: pick rows matching criteria
- **`distinct`**: filter for unique rows
- **`mutate`**: add new variables
- **`summarise`**: reduce variables to values
- **`group_by`**: for grouped operations

Rules of dplyr functions

- First argument is *always* a data frame
- Subsequent arguments say what to do with that data frame
- Always return a data frame

Data: Hotel bookings

- Data from two hotels: one resort and one city hotel
- Observations: Each row represents a hotel booking
- Goal for original data collection: Development of prediction models to classify a hotel booking's likelihood to be cancelled (Antonia et al., 2019)
- Featured in TidyTuesday!

```r
hotels <- read_csv("data/hotels.csv")
```

First look: Variables

```r
names(hotels)
```

```
##  [1] "hotel"                          "is_canceled"
##  [3] "lead_time"                      "arrival_date_year"
##  [5] "arrival_date_month"             "arrival_date_week_number"
##  [7] "arrival_date_day_of_month"      "stays_in_weekend_nights"
##  [9] "stays_in_week_nights"           "adults"
## [11] "children"                       "babies"
## [13] "meal"                           "country"
## [15] "market_segment"                 "distribution_channel"
## [17] "is_repeated_guest"              "previous_cancellations"
## [19] "previous_bookings_not_canceled" "reserved_room_type"
## [21] "assigned_room_type"             "booking_changes"
## [23] "deposit_type"                   "agent"
## [25] "company"                        "days_in_waiting_list"
## [27] "customer_type"                  "adr"
```

Second look: Overview

```r
glimpse(hotels)
```

```
## Rows: 119,390
## Columns: 32
## $ hotel                          <chr> "Resort Hotel", "Resort Hotel", "Res
## $ is_canceled                    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
## $ lead_time                      <dbl> 342, 737, 7, 13, 14, 14, 0, 9, 85,
## $ arrival_date_year              <dbl> 2015, 2015, 2015, 2015, 2015, 2015,
## $ arrival_date_month             <chr> "July", "July", "July", "July", "Ju
## $ arrival_date_week_number       <dbl> 27, 27, 27, 27, 27, 27, 27, 27, 27,
## $ arrival_date_day_of_month      <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## $ stays_in_weekend_nights        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## $ stays_in_week_nights           <dbl> 0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 4, 4,
## $ adults                         <dbl> 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,
## $ children                       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## $ babies                         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Select a single column

View only the **`lead_time`** (number of days between booking and arrival date):

```
hotels %>%
  select(lead_time)
```

```
## # A tibble: 119,390 x 1
##    lead_time
##        <dbl>
##  1       342
##  2       737
##  3         7
##  4        13
##  5        14
##  6        14
##  7         0
```

- Start with a data frame
- Pass it to the **`select()`** function.
- Second argument is variable we want to select: **`lead_time`**
- The result is a data frame with 119,300 and 1 column: --dplyr functions always expect a data frame and always yield a data frame.

Select multiple columns

View only the **`hotel`** type and **`lead_time`**:

```
hotels %>%
  select(hotel, lead_time)
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       342
##  2 Resort Hotel       737
##  3 Resort Hotel         7
##  4 Resort Hotel        13
##  5 Resort Hotel        14
##  6 Resort Hotel        14
##  7 Resort Hotel         0
##  8 Resort Hotel         0
```

What if we wanted to select these columns, and then arrange the data in descending order of lead time?

Data wrangling, step-by-step

Select:

```r
hotels %>%
  select(hotel, lead_time)
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       342
##  2 Resort Hotel       737
##  3 Resort Hotel         7
##  4 Resort Hotel        13
##  5 Resort Hotel        14
##  6 Resort Hotel        14
##  7 Resort Hotel         0
##  8 Resort Hotel         9
```

Select, then arrange:

```r
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       737
##  2 Resort Hotel       709
##  3 City Hotel         629
##  4 City Hotel         629
##  5 City Hotel         629
##  6 City Hotel         629
##  7 City Hotel         629
```

Pipes

What is a pipe?

In programming, a pipe is a technique for passing information from one process to another.

- Start with the data frame **`hotels`**, and pass it to the **`select()`** function,

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       737
##  2 Resort Hotel       709
##  3 City Hotel         629
##  4 City Hotel         629
##  5 City Hotel         629
##  6 City Hotel         629
##  7 City Hotel         629
##  8 City Hotel         629
##  9 City Hotel         629
## 10 City Hotel         629
## # … with 119,380 more rows
```

What is a pipe?

In programming, a pipe is a technique for passing information from one process to another.

- Start with the data frame **`hotels`**, and pass it to the **`select()`** function,
- then we select the variables **`hotel`** and **`lead_time`**,

```r
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       737
##  2 Resort Hotel       709
##  3 City Hotel         629
##  4 City Hotel         629
##  5 City Hotel         629
##  6 City Hotel         629
##  7 City Hotel         629
##  8 City Hotel         629
##  9 City Hotel         629
## 10 City Hotel         629
## # … with 119,380 more rows
```

What is a pipe?

In programming, a pipe is a technique for passing information from one process to another.

- Start with the data frame **hotels**, and pass it to the **select()** function,
- then we select the variables **hotel** and **lead_time**,
- and then we arrange the data frame by **lead_time** in descending order.

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       737
##  2 Resort Hotel       709
##  3 City Hotel         629
##  4 City Hotel         629
##  5 City Hotel         629
##  6 City Hotel         629
##  7 City Hotel         629
##  8 City Hotel         629
##  9 City Hotel         629
## 10 City Hotel         629
## # … with 119,380 more rows
```

Aside

The pipe operator is implemented in the package **magrittr**, though we don't need to load this package explicitly since **tidyverse** does this for us.

> Any guesses as to why the package is called magrittr?

Ceci n'est pas une pipe.

%>%
magrittr
Ceci n'est pas un pipe.

How does a pipe work?

- You can think about the following sequence of actions - find keys, start car, drive to work, park.
- Expressed as a set of nested functions in R pseudocode this would look like:

```
park(drive(start_car(find("keys")), to = "work"))
```

- Writing it out using pipes give it a more natural (and easier to read) structure:
    - Read the pipe as "and then"

```
find("keys") %>%
  start_car() %>%
  drive(to = "work") %>%
  park()
```

What about other arguments?

Use the dot to

- send results to a function argument other than first one or
- use the previous result for multiple arguments

```
hotels %>%
  filter(hotel == "Resort Hotel") %>%
  lm(adr ~ lead_time, data = .)
```

```
##
## Call:
## lm(formula = adr ~ lead_time, data = .)
##
## Coefficients:
## (Intercept)    lead_time
##    93.16876      0.01925
```

Working with a single data frame

You have a single data frame, and you want to process it and prepare it for anlaysis!

select to keep variables

```r
hotels %>%
  select(hotel, lead_time)
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       342
##  2 Resort Hotel       737
##  3 Resort Hotel         7
##  4 Resort Hotel        13
##  5 Resort Hotel        14
##  6 Resort Hotel        14
##  7 Resort Hotel         0
##  8 Resort Hotel         9
##  9 Resort Hotel        85
## 10 Resort Hotel        75
## # … with 119,380
```

`select` to exclude variables

```r
hotels %>%
  select(-agent)
```

```
## # A tibble: 119,390 x 31
##    hotel is_canceled lead_time arrival_date_ye… arrival_date_mo…
##    <chr>       <dbl>     <dbl>            <dbl> <chr>
##  1 Reso…           0       342             2015 July
##  2 Reso…           0       737             2015 July
##  3 Reso…           0         7             2015 July
##  4 Reso…           0        13             2015 July
##  5 Reso…           0        14             2015 July
##  6 Reso…           0        14             2015 July
##  7 Reso…           0         0             2015 July
##  8 Reso…           0         9             2015 July
##  9 Reso…           1        85             2015 July
## 10 Reso…           1        75             2015 July
## # … with 119,380 more rows, and 26 more variables:
## #   arrival_date_week_number <dbl>, arrival_date_day_of_month <dbl>,
## #   stays_in_weekend_nights <dbl>, stays_in_week_nights <dbl>, adults <dbl>,
## #   children <dbl>, babies <dbl>, meal <chr>, country <chr>,
## #   market_segment <chr>, distribution_channel <chr>, is_repeated_guest <dbl>,
## #   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>,
## #   reserved_room_type <chr>, assigned_room_type <chr>, booking_changes <dbl>,
```

select a range of variables

```r
hotels %>%
  select(hotel:arrival_date_month)
```

```
## # A tibble: 119,390 x 5
##    hotel        is_canceled lead_time arrival_date_year arrival_date_month
##    <chr>              <dbl>     <dbl>             <dbl> <chr>
##  1 Resort Hotel           0       342              2015 July
##  2 Resort Hotel           0       737              2015 July
##  3 Resort Hotel           0         7              2015 July
##  4 Resort Hotel           0        13              2015 July
##  5 Resort Hotel           0        14              2015 July
##  6 Resort Hotel           0        14              2015 July
##  7 Resort Hotel           0         0              2015 July
##  8 Resort Hotel           0         9              2015 July
##  9 Resort Hotel           1        85              2015 July
## 10 Resort Hotel           1        75              2015 July
```

arrange in ascending / descending order

```r
hotels %>%
  select(adults, children, babies
  arrange(babies)
```

```
## # A tibble: 119,390 x 3
##    adults children babies
##     <dbl>    <dbl>  <dbl>
##  1      2        0      0
##  2      2        0      0
##  3      1        0      0
##  4      1        0      0
##  5      2        0      0
##  6      2        0      0
##  7      2        0      0
##  8      2        0      0
##  9      2        0      0
```

```r
hotels %>%
  select(adults, children, babies
  arrange(desc(babies))
```

```
## # A tibble: 119,390 x 3
##    adults children babies
##     <dbl>    <dbl>  <dbl>
##  1      2        0     10
##  2      1        0      9
##  3      2        0      2
##  4      2        0      2
##  5      2        0      2
##  6      2        0      2
##  7      2        0      2
##  8      2        0      2
##  9      2        0      2
```

`slice` for certain row numbers

```r
# first five
hotels %>%
  slice(1:5)
```

```
## # A tibble: 5 x 32
##   hotel is_canceled lead_time arrival_date_ye… arrival_date_mo… arrival_date_we…
##   <chr>       <dbl>     <dbl>            <dbl> <chr>                       <dbl>
## 1 Reso…           0       342             2015 July                           27
## 2 Reso…           0       737             2015 July                           27
## 3 Reso…           0         7             2015 July                           27
## 4 Reso…           0        13             2015 July                           27
## 5 Reso…           0        14             2015 July                           27
## # … with 26 more variables: arrival_date_day_of_month <dbl>,
## #   stays_in_weekend_nights <dbl>, stays_in_week_nights <dbl>, adults <dbl>,
## #   children <dbl>, babies <dbl>, meal <chr>, country <chr>,
## #   market_segment <chr>, distribution_channel <chr>, is_repeated_guest <dbl>,
## #   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>,
## #   reserved_room_type <chr>, assigned_room_type <chr>, booking_changes <dbl>,
## #   deposit_type <chr>, agent <chr>, company <chr>, days_in_waiting_list <dbl>,
## #   customer_type <chr>, adr <dbl>, required_car_parking_spaces <dbl>,
```

Tip:

In R, you can use the **#** (hashtag or pound sign, depending on your age 😜) for adding comments to your code. Any text following **#** will be printed as is, and won't be run as R code. This is useful for leaving comments in your code and for temporarily disabling certain lines of code while debugging.

```
hotels %>%
  # slice the first five rows  # this line is a comment
  #select(hotel) %>%           # this one doesn't run
  slice(1:5)                   # this line runs
```

```
## # A tibble: 5 x 32
##   hotel is_canceled lead_time arrival_date_ye… arrival_date_mo… arrival_date_we…
##   <chr>       <dbl>     <dbl>            <dbl> <chr>                       <dbl>
## 1 Reso…           0       342             2015 July                           27
## 2 Reso…           0       737             2015 July                           27
## 3 Reso…           0         7             2015 July                           27
## 4 Reso…           0        13             2015 July                           27
## 5 Reso…           0        14             2015 July                           27
## # … with 26 more variables: arrival_date_day_of_month <dbl>,
## #   stays_in_weekend_nights <dbl>, stays_in_week_nights <dbl>, adults <dbl>,
```

`slice` for certain row numbers

```r
# last five
last_row <- nrow(hotels) # nrow() gives the number of rows in a data frame
hotels %>%
  slice((last_row - 4):last_row)
```

```
## # A tibble: 5 x 32
##   hotel is_canceled lead_time arrival_date_ye… arrival_date_mo… arrival_date_we…
##   <chr>       <dbl>     <dbl>            <dbl> <chr>                       <dbl>
## 1 City…           0        23             2017 August                         35
## 2 City…           0       102             2017 August                         35
## 3 City…           0        34             2017 August                         35
## 4 City…           0       109             2017 August                         35
## 5 City…           0       205             2017 August                         35
## # … with 26 more variables: arrival_date_day_of_month <dbl>,
## #   stays_in_weekend_nights <dbl>, stays_in_week_nights <dbl>, adults <dbl>,
## #   children <dbl>, babies <dbl>, meal <chr>, country <chr>,
## #   market_segment <chr>, distribution_channel <chr>, is_repeated_guest <dbl>,
## #   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>,
## #   reserved_room_type <chr>, assigned_room_type <chr>, booking_changes <dbl>,
## #   deposit_type <chr>, agent <chr>, company <chr>, days_in_waiting_list <dbl>,
```

`filter` to select a subset of rows

```r
# bookings in City Hotels
hotels %>%
  filter(hotel == "City Hotel")
```

```
## # A tibble: 79,330 x 32
##    hotel is_canceled lead_time arrival_date_ye… arrival_date_mo…
##    <chr>       <dbl>     <dbl>            <dbl> <chr>
##  1 City…           0         6             2015 July
##  2 City…           1        88             2015 July
##  3 City…           1        65             2015 July
##  4 City…           1        92             2015 July
##  5 City…           1       100             2015 July
##  6 City…           1        79             2015 July
##  7 City…           0         3             2015 July
##  8 City…           1        63             2015 July
##  9 City…           1        62             2015 July
## 10 City…           1        62             2015 July
## # … with 79,320 more rows, and 27 more variables:
## #   arrival_date_week_number <dbl>, arrival_date_day_of_month <dbl>,
## #   stays_in_weekend_nights <dbl>, stays_in_week_nights <dbl>, adults <dbl>,
```

`filter` for many conditions at once

```r
hotels %>%
  filter(
    adults == 0,
    children >= 1
    ) %>%
  select(adults, babies, children)
```

```
## # A tibble: 223 x 3
##    adults babies children
##     <dbl>  <dbl>    <dbl>
##  1      0      0        3
##  2      0      0        2
##  3      0      0        2
##  4      0      0        2
##  5      0      0        2
##  6      0      0        3
```

filter for more complex conditions

```r
# bookings with no adults and some children or babies in the room
hotels %>%
  filter(
    adults == 0,
    children >= 1 | babies >= 1     # | means or
    ) %>%
  select(adults, babies, children)
```

```
## # A tibble: 223 x 3
##    adults babies children
##     <dbl>  <dbl>    <dbl>
##  1      0      0        3
##  2      0      0        2
##  3      0      0        2
##  4      0      0        2
##  5      0      0        2
```

Logical operators in R

operator	definition	operator	definition
`<`	less than	`x | y`	`x` OR `y`
`<=`	less than or equal to	`is.na(x)`	test if `x` is `NA`
`>`	greater than	`!is.na(x)`	test if `x` is not `NA`
`>=`	greater than or equal to	`x %in% y`	test if `x` is in `y`
`==`	exactly equal to	`!(x %in% y)`	test if `x` is not in `y`
`!=`	not equal to	`!x`	not `x`
`x & y`	`x` AND `y`		

Demo

distinct to filter for unique rows

... and **arrange** to order alphabetically

```
hotels %>%
  distinct(market_segment) %>%
  arrange(market_segment)
```

```
## # A tibble: 8 x 1
##   market_segment
##   <chr>
## 1 Aviation
## 2 Complementary
## 3 Corporate
## 4 Direct
## 5 Groups
## 6 Offline TA/TO
## 7 Online TA
## 8 Undefined
```

```
hotels %>%
  distinct(hotel, market_segment) %>%
  arrange(hotel, market_segment)
```

```
## # A tibble: 14 x 2
##    hotel        market_segment
##    <chr>        <chr>
##  1 City Hotel   Aviation
##  2 City Hotel   Complementary
##  3 City Hotel   Corporate
##  4 City Hotel   Direct
##  5 City Hotel   Groups
##  6 City Hotel   Offline TA/TO
##  7 City Hotel   Online TA
##  8 City Hotel   Undefined
##  9 Resort Hotel Complementary
## 10 Resort Hotel Corporate
## 11 Resort Hotel Direct
## 12 Resort Hotel Groups
## 13 Resort Hotel Offline TA/TO
```

count to create frequency tables

```r
# alphabetical order by default
hotels %>%
  count(market_segment)
```

```
## # A tibble: 8 x 2
##   market_segment     n
##   <chr>          <int>
## 1 Aviation         237
## 2 Complementary    743
## 3 Corporate       5295
## 4 Direct         12606
## 5 Groups         19811
## 6 Offline TA/TO  24219
## 7 Online TA      56477
## 8 Undefined          2
```

```r
# descending frequency order
hotels %>%
  count(market_segment,
        sort = TRUE)
```

```
## # A tibble: 8 x 2
##   market_segment     n
##   <chr>          <int>
## 1 Online TA      56477
## 2 Offline TA/TO  24219
## 3 Groups         19811
## 4 Direct         12606
## 5 Corporate       5295
## 6 Complementary    743
## 7 Aviation         237
## 8 Undefined          2
```

count and arrange

```r
# ascending frequency order
hotels %>%
  count(market_segment) %>%
  arrange(n)
```

```
## # A tibble: 8 x 2
##   market_segment     n
##   <chr>          <int>
## 1 Undefined          2
## 2 Aviation         237
## 3 Complementary    743
## 4 Corporate       5295
## 5 Direct         12606
## 6 Groups         19811
## 7 Offline TA/TO  24219
## 8 Online TA      56477
```

```r
# descending frequency order
# just like adding sort = TRUE
hotels %>%
  count(market_segment) %>%
  arrange(desc(n))
```

```
## # A tibble: 8 x 2
##   market_segment     n
##   <chr>          <int>
## 1 Online TA      56477
## 2 Offline TA/TO  24219
## 3 Groups         19811
## 4 Direct         12606
## 5 Corporate       5295
## 6 Complementary    743
## 7 Aviation         237
```

count for multiple variables

```r
hotels %>%
  count(hotel, market_segment)
```

```
## # A tibble: 14 x 3
##    hotel        market_segment     n
##    <chr>        <chr>          <int>
##  1 City Hotel   Aviation         237
##  2 City Hotel   Complementary    542
##  3 City Hotel   Corporate       2986
##  4 City Hotel   Direct          6093
##  5 City Hotel   Groups         13975
##  6 City Hotel   Offline TA/TO  16747
##  7 City Hotel   Online TA      38748
##  8 City Hotel   Undefined          2
##  9 Resort Hotel Complementary    201
## 10 Resort Hotel Corporate       2309
```

order matters when you **count**

```r
# hotel type first
hotels %>%
  count(hotel, market_segment)
```

```
## # A tibble: 14 x 3
##    hotel        market_segment     n
##    <chr>        <chr>          <int>
##  1 City Hotel   Aviation         237
##  2 City Hotel   Complementary    542
##  3 City Hotel   Corporate       2986
##  4 City Hotel   Direct          6093
##  5 City Hotel   Groups         13975
##  6 City Hotel   Offline TA/TO  16747
##  7 City Hotel   Online TA      38748
##  8 City Hotel   Undefined          2
##  9 Resort Hotel Complementary    201
## 10 Resort Hotel Corporate       2309
## 11 Resort Hotel Direct          6513
## 12 Resort Hotel Groups          5836
```

```r
# market segment first
hotels %>%
  count(market_segment, hotel)
```

```
## # A tibble: 14 x 3
##    market_segment hotel            n
##    <chr>          <chr>        <int>
##  1 Aviation       City Hotel     237
##  2 Complementary  City Hotel     542
##  3 Complementary  Resort Hotel   201
##  4 Corporate      City Hotel    2986
##  5 Corporate      Resort Hotel  2309
##  6 Direct         City Hotel    6093
##  7 Direct         Resort Hotel  6513
##  8 Groups         City Hotel   13975
##  9 Groups         Resort Hotel  5836
## 10 Offline TA/TO  City Hotel   16747
## 11 Offline TA/TO  Resort Hotel  7472
## 12 Online TA      City Hotel   38748
```

Demo

`mutate` to add a new variable

```r
hotels %>%
  mutate(little_ones = children + babies) %>%
  select(children, babies, little_ones) %>%
  arrange(desc(little_ones))
```

```
## # A tibble: 119,390 x 3
##    children babies little_ones
##       <dbl>  <dbl>       <dbl>
##  1       10      0          10
##  2        0     10          10
##  3        0      9           9
##  4        2      1           3
##  5        2      1           3
##  6        2      1           3
##  7        3      0           3
##  8        2      1           3
```

Little ones in resort and city hotels

```r
# Resort Hotel
hotels %>%
  mutate(little_ones = children + babies) %>%
  filter(
    little_ones >= 1,
    hotel == "Resort Hotel"
    ) %>%
  select(hotel, little_ones)
```

```
## # A tibble: 3,929 x 2
##    hotel        little_ones
##    <chr>              <dbl>
##  1 Resort Hotel           1
##  2 Resort Hotel           2
##  3 Resort Hotel           2
##  4 Resort Hotel           2
##  5 Resort Hotel           1
##  6 Resort Hotel           1
##  7 Resort Hotel           2
##  8 Resort Hotel           2
##  9 Resort Hotel           1
## 10 Resort Hotel           1
## #   with 3,919 more rows
```

```r
# City Hotel
hotels %>%
  mutate(little_ones = children + babies) %>%
  filter(
    little_ones > 1,
    hotel == "City Hotel"
    )  %>%
  select(hotel, little_ones)
```

```
## # A tibble: 2,140 x 2
##    hotel      little_ones
##    <chr>            <dbl>
##  1 City Hotel           2
##  2 City Hotel           2
##  3 City Hotel           2
##  4 City Hotel           2
##  5 City Hotel           2
##  6 City Hotel           2
##  7 City Hotel           2
##  8 City Hotel           2
##  9 City Hotel           2
## 10 City Hotel           3
## #   with 2,130 more rows
```

What is happening in the following chunk?

```
hotels %>%
  mutate(little_ones = children + babies) %>%
  count(hotel, little_ones) %>%
  mutate(prop = n / sum(n))
```

```
## # A tibble: 12 x 4
##    hotel        little_ones     n       prop
##    <chr>              <dbl> <int>      <dbl>
##  1 City Hotel             0 73923 0.619
##  2 City Hotel             1  3263 0.0273
##  3 City Hotel             2  2056 0.0172
##  4 City Hotel             3    82 0.000687
##  5 City Hotel             9     1 0.00000838
##  6 City Hotel            10     1 0.00000838
##  7 City Hotel            NA     4 0.0000335
##  8 Resort Hotel           0 36131 0.303
##  9 Resort Hotel           1  2183 0.0183
## 10 Resort Hotel           2  1716 0.0144
```

`summarise` for summary stats

```r
# mean average daily rate for all bookings
hotels %>%
  summarise(mean_adr = mean(adr))
```

```
## # A tibble: 1 x 1
##   mean_adr
##      <dbl>
## 1     102.
```

Tip:

`summarise()` changes the data frame entirely, it collapses rows down to a single summary statistics, and removes all columns that are irrelevant to the calculation.

Tip:

`summarise()` also lets you get away with being sloppy and not naming your new column, but that's not recommended!

❌

```
hotels %>%
  summarise(mean(adr))
```

```
## # A tibble: 1 x 1
##   `mean(adr)`
##         <dbl>
## 1        102.
```

✅

```
hotels %>%
  summarise(mean_adr = mean(adr))
```

```
## # A tibble: 1 x 1
##   mean_adr
##      <dbl>
## 1     102.
```

group_by for grouped operations

```r
# mean average daily rate for all booking at city and resort hotels
hotels %>%
  group_by(hotel) %>%
  summarise(mean_adr =  mean(adr))
```

```
## # A tibble: 2 x 2
##   hotel        mean_adr
##   <chr>           <dbl>
## 1 City Hotel      105. 
## 2 Resort Hotel     95.0
```

Calculating frequencies

The following two give the same result, so **`count`** is simply short for **`group_by`** then determine frequencies

```r
hotels %>%
  group_by(hotel) %>%
  summarise(n = n())
```

```
## # A tibble: 2 x 2
##   hotel            n
##   <chr>        <int>
## 1 City Hotel   79330
## 2 Resort Hotel 40060
```

```r
hotels %>%
  count(hotel)
```

```
## # A tibble: 2 x 2
##   hotel            n
##   <chr>        <int>
## 1 City Hotel   79330
## 2 Resort Hotel 40060
```

Multiple summary statistics

`summarise` can be used for multiple summary statistics as well

```
hotels %>%
  summarise(
    min_adr = min(adr),
    mean_adr = mean(adr),
    median_adr = median(adr),
    max_adr = max(adr)
    )
```

```
## # A tibble: 1 x 4
##   min_adr mean_adr median_adr max_adr
##     <dbl>    <dbl>      <dbl>   <dbl>
## 1   -6.38     102.       94.6    5400
```

Demo