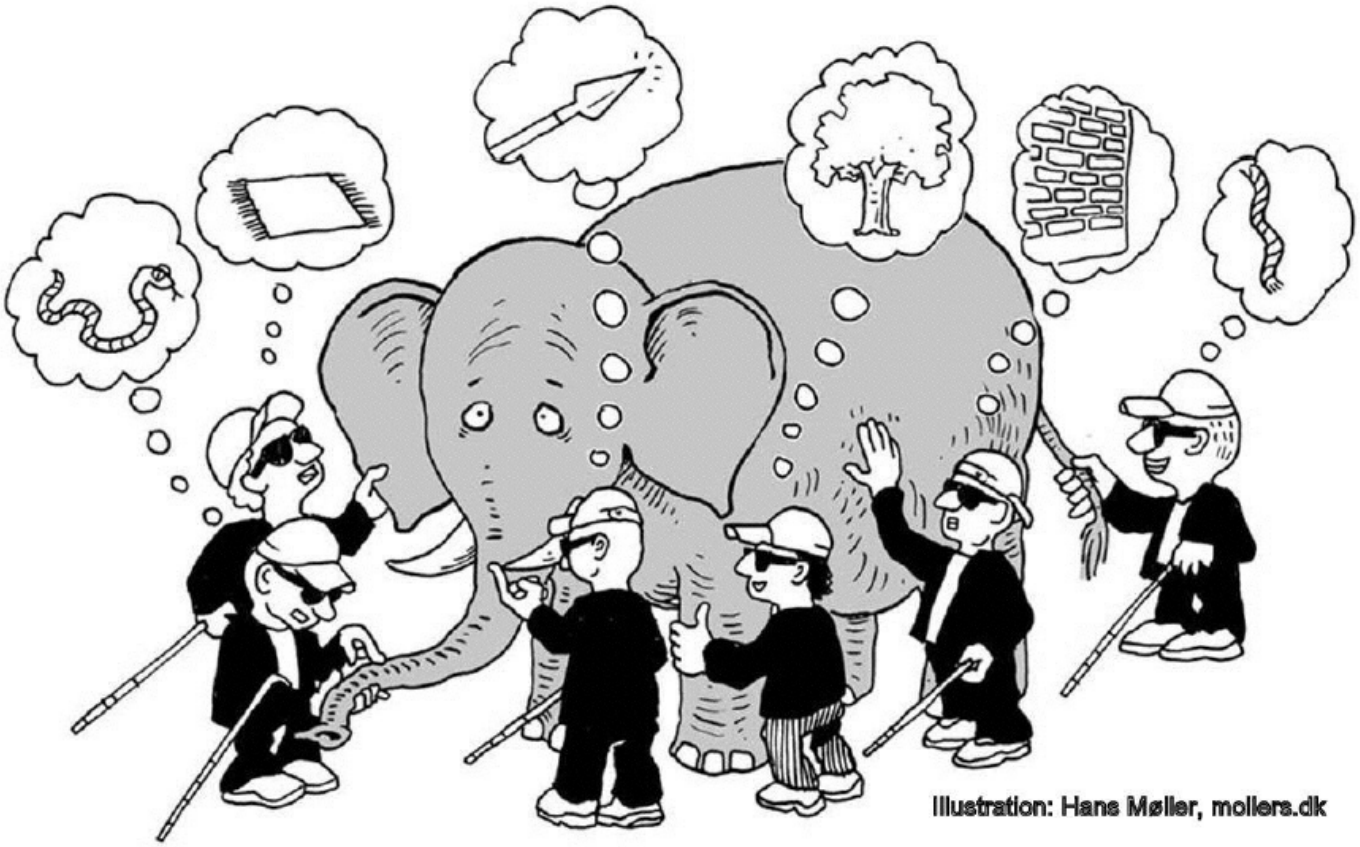
title: “Introduction to R”
subtitle: “Data wrangling with tidyverse”
author: “Jan-Philipp Kolb”
date: “03 März, 2020”
location: “GESIS, Mannheim, Germany”
output:
xaringan::moon_reader:
lib_dir: libs
css: [“default”, “default-fonts”, “../workshop.css”]
nature:
highlightStyle: “github”
highlightLines: true
countIncrementalSlides: false



“Unless the data is something I’ve analyzed a lot before, I usually feel like the blind men and the elephant.”

Jeff Leek (http://uc-r.github.io/data_wrangling/week-3)

FIRST THINGS TO DO (http://uc-r.github.io/data_wrangling/week-3)

Don’t try to kiss your data on the first date; rather, you just want to get to know the data:

1. Import the data
2. Review the codebook
3. **Learn about the data**
4. Quick visual understanding of the data

WHAT IS TIDY DATA? (http://uc-r.github.io/data_wrangling/week-3)

- One variable per column
- One observation per row

```
##        userid age dob_day dob_year dob_month gender tenure friend_count
## 1     2094382  14      19     1999        11   male    266            0
## 2     1192601  14       2     1999        11 female      6            0
## 3     2083884  14      16     1999        11   male     13            0
## 4     1203168  14      25     1999        12 female     93            0
## 5     1733186  14       4     1999        12   male     82            0
## 6     1524765  14       1     1999        12   male     15            0
## 7     1136133  13      14     2000         1   male     12            0
## 8     1680361  13       4     2000         1 female      0            0
## 9     1365174  13       1     2000         1   male     81            0
## 10    1712567  13       2     2000         2   male    171            0
## 11    1612453  13      22     2000         2   male     98            0
```

What is `tidyverse`

```
library(tidyverse)
```

`tidyr`

Four key `tidyr` functions

You'll learn four key `tidyr` functions that allow you to solve the vast majority of your data tidying challenges:

- `gather` : transforms data from wide to long
- `spread` : transforms data from long to wide
- `separate` : splits a single column into multiple columns
- `unite` : combines multiple columns into a single column

Exercise: random numbers (https://www.datacamp.com/community/tutorials/pipe-r-tutorial)

1. Draw 8 random numbers from the uniform distribution (`rnorm`) and save them in a vector `x`
2. Compute the natural logarithm of `x` ,
3. and round the result

The package `magrittr`

```
library(magrittr)
```

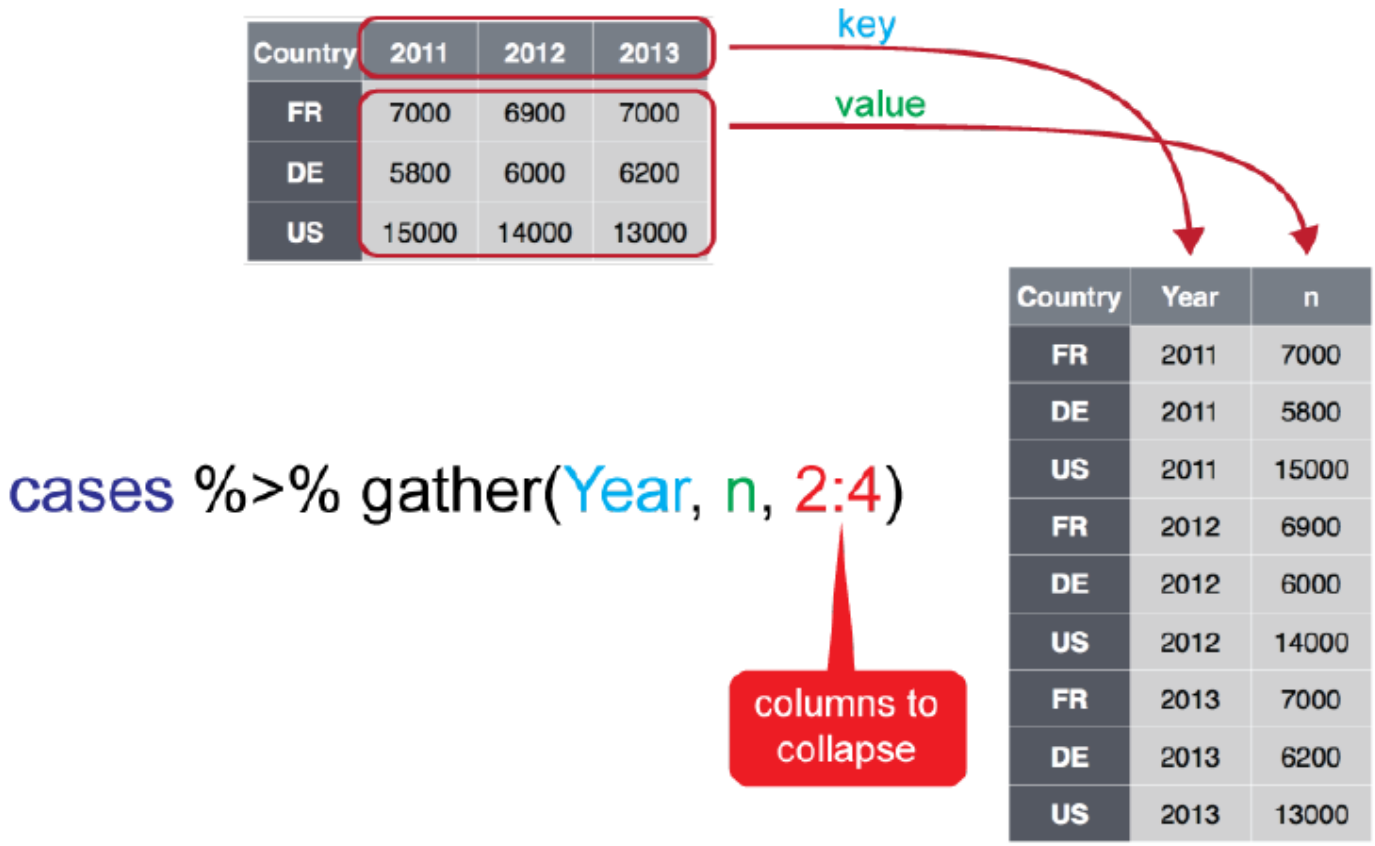
The pipe operator (https://www.datacamp.com/community/tutorials/pipe-r-tutorial)

```
library(magrittr)

# Perform the same computations on `x` as above
x %>% log() %>%
    round(1)
```

```
## [1]  0.0 -0.8 -0.1 -0.2 -0.1 -1.5 -0.7  0.0
```

How `gather` function works



The `gather` function

```
load("../data/tidy_data.RData")
cases %>% gather(Year,n,2:4)
```

```
##    country Year     n
## 1       FR 2011  7000
## 2       DE 2011  5800
## 3       US 2011 15000
## 4       FR 2012  6900
## 5       DE 2012  6000
## 6       US 2012 14000
## 7       FR 2013  7000
## 8       DE 2013  6200
## 9       US 2013 13000
```

The gather function (http://uc-r.github.io/data_wrangling/week-3)

Code alternatives:

```
# These all produce the same results:
cases %>% gather(Year, n, "2011":"2013")
cases %>% gather(Year, n, "2011", "2012", "2013")
cases %>% gather(Year, n, 2:4)
cases %>% gather(Year, n, -country)
```

- Also note that if you do not supply arguments for `na.rm` or convert values then the defaults are used

The seperate function

```
storms <- storms %>%
  separate(date, c("year", "month", "day"))
```

- By default, if no separator is specified, will separate by any regular expression that matches any sequence of non-alphanumeric values

```
storms %>%
  separate(date, c("year", "month", "day"), sep = "-")
```

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

The unite function

```
# same results:
storms %>% unite(date, year, month, day, sep = "_")
storms %>% unite(date, year, month, day)
# If no separator is identified,
# "_" will automatically be used
```

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

The R-package `data.table`

Get an overview

```
data(airquality)
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
```

Overview with `data.table`

```
library(data.table)
(airq <- data.table(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
##  ---
## 149:    30     193  6.9   70     9  26
## 150:    NA     145 13.2   77     9  27
## 151:    14     191 14.3   75     9  28
## 152:    18     131  8.0   76     9  29
## 153:    20     223 11.5   68     9  30
```

The `dplyr` package

```
library(dplyr)
```

Introduction to dplyr

When working with data you must:

- Figure out what you want to do.
- Describe those tasks in the form of a computer program.
- Execute the program.

The dplyr package makes these steps fast and easy:

- By constraining your options, it helps you think about your data manipulation challenges.
- It provides simple "verbs", functions that correspond to the most common data manipulation tasks, to help you translate your thoughts into code.
- It uses efficient backends, so you spend less time waiting for the computer.

Important functions of `dplyr`

- `filter` : pick observations based on values
- `arrange` : reorder data
- `select` : pick variables
- `mutate` : create new variables
- `summarize` : summarize data by functions of choice
- `group_by` : group data by categorical levels

PACKAGE and data

```
library(nycflights13)
```

```
flights
```

```
> flights
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1  2013     1     1      517            515         2      830            819        11
 2  2013     1     1      533            529         4      850            830        20
 3  2013     1     1      542            540         2      923            850        33
 4  2013     1     1      544            545        -1     1004           1022       -18
 5  2013     1     1      554            600        -6      812            837       -25
 6  2013     1     1      554            558        -4      740            728        12
 7  2013     1     1      555            600        -5      913            854        19
 8  2013     1     1      557            600        -3      709            723       -14
 9  2013     1     1      557            600        -3      838            846        -8
10  2013     1     1      558            600        -2      753            745         8
# ... with 336,766 more rows, and 10 more variables: carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

Exercise: Vignette and additional documentation

- Are there vignettes for the `dplyr` package?
- Can you find additional documentation explaining the `flights` data set?

the `filter` command

Filter values based on defined conditions

Filter based on one or more variables

```
filter(flights,month==1)
```

```
filter(flights, month == 1)
# A tibble: 27,004 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515         2      830            819        11
2   2013     1     1      533            529         4      850            830        20
3   2013     1     1      542            540         2      923            850        33
4   2013     1     1      544            545        -1     1004           1022       -18
5   2013     1     1      554            600        -6      812            837       -25
6   2013     1     1      554            558        -4      740            728        12
7   2013     1     1      555            600        -5      913            854        19
8   2013     1     1      557            600        -3      709            723       -14
```

BASIC FILTERING

Filter based on one or more variables

```
filter(flights,month==1,day==1)
```

```
filter(flights, month == 1, day == 1)
# A tibble: 842 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515         2      830            819        11
2   2013     1     1      533            529         4      850            830        20
3   2013     1     1      542            540         2      923            850        33
4   2013     1     1      544            545        -1     1004           1022       -18
5   2013     1     1      554            600        -6      812            837       -25
6   2013     1     1      554            558        -4      740            728        12
7   2013     1     1      555            600        -5      913            854        19
8   2013     1     1      557            600        -3      709            723       -14
```

Filter based on one or more variables

```
filter(flights,month==1,day==1,dep_delay > 0)
```

```
filter(flights, month == 1, day == 1, dep_delay > 0)
# A tibble: 352 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515         2      830            819        11
2   2013     1     1      533            529         4      850            830        20
3   2013     1     1      542            540         2      923            850        33
4   2013     1     1      601            600         1      844            850        -6
5   2013     1     1      608            600         8      807            735        32
6   2013     1     1      611            600        11      945            931        14
7   2013     1     1      613            610         3      925            921         4
8   2013     1     1      623            610        13      920            915         5
```

Comparison

What will these operations produce?

```
filter(flights,month==12)
filter(flights,month!=12)
filter(flights,month %in% c(11,12))
filter(flights,arr_delay <= 120)
filter(flights,!(arr_delay <= 120))
filter(flights,is.na(tailnum))
```

MULTIPLE COMPARISONS (http://uc-r.github.io/data_wrangling/week-4)

Using comma is same as using &

```
filter(flights,month==12, day==25)
filter(flights,month==12 & day==25)
```

Use `%in%` as a shortcut for `|`

```
filter(flights,month==11 | month==12)
filter(flights,month %in% c(11,12))
```

Are these the same?

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
filter(flights, arr_delay <= 120,dep_delay <= 120)
```

Exercise: `dplyr` and `flights` dataset

Find the number of flights that:

a. Had an arrival delay of two or more hours

b. Flew to Houston (IAH or HOU)

c. Arrived more than two hours late, but didn't leave late

`arrange` - reorder data

Order data based on one or more variables

```
arrange(flights, dep_delay)
```

```
arrange(flights, dep_delay)
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1   2013    12     7     2040           2123       -43       40           2352        48      B6     97
2   2013     2     3     2022           2055       -33     2240           2338       -58      DL   1715
3   2013    11    10     1408           1440       -32     1549           1559       -10      EV   5713
4   2013     1    11     1900           1930       -30     2233           2243       -10      DL   1435
5   2013     1    29     1703           1730       -27     1947           1957       -10      F9    837
6   2013     8     9      729            755       -26     1002            955         7      MQ   3478
7   2013    10    23     1907           1932       -25     2143           2143         0      EV   4361
8   2013     3    30     2030           2055       -25     2213           2250       -37      MQ   4573
9   2013     3     2     1431           1455       -24     1601           1631       -30      9E   3318
10  2013     5     5      934            958       -24     1225           1309       -44      B6    375
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
```

ORDERING YOUR DATA

Order data based on one or more variables

```
arrange(flights, dep_delay, arr_delay)
```

```
arrange(flights, dep_delay, arr_delay)
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1   2013    12     7     2040           2123       -43       40           2352        48      B6     97
2   2013     2     3     2022           2055       -33     2240           2338       -58      DL   1715
3   2013    11    10     1408           1440       -32     1549           1559       -10      EV   5713
4   2013     1    11     1900           1930       -30     2233           2243       -10      DL   1435
5   2013     1    29     1703           1730       -27     1947           1957       -10      F9    837
6   2013     8     9      729            755       -26     1002            955         7      MQ   3478
7   2013     3    30     2030           2055       -25     2213           2250       -37      MQ   4573
8   2013    10    23     1907           1932       -25     2143           2143         0      EV   4361
9   2013     5     5      934            958       -24     1225           1309       -44      B6    375
10  2013     9    18     1631           1655       -24     1812           1845       -33      AA   2223
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
```

Reverse ordering

Reverse the order by using `desc()`

```
arrange(flights, desc(dep_delay))
```

```
arrange(flights, desc(dep_delay))
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1   2013     1     9      641            900      1301     1242           1530      1272      HA     51
2   2013     6    15     1432           1935      1137     1607           2120      1127      MQ   3535
3   2013     1    10     1121           1635      1126     1239           1810      1109      MQ   3695
4   2013     9    20     1139           1845      1014     1457           2210      1007      AA    177
5   2013     7    22      845           1600      1005     1044           1815       989      MQ   3075
6   2013     4    10     1100           1900       960     1342           2211       931      DL   2391
7   2013     3    17     2321            810       911      135           1020       915      DL   2119
8   2013     6    27      959           1900       899     1236           2226       850      DL   2007
9   2013     7    22     2257            759       898      121           1026       895      DL   2047
10  2013    12     5      756           1700       896     1058           2020       878      AA    172
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
```

Ordering and missing values

- Note that missing values are always sorted at the end:

```
df <- tibble(x = c(5, 2, 5, NA))
# A tibble: 4 × 1
      x
  <dbl>
1     5
2     2
3     5
4    NA
```

```
arrange(df, x)
# A tibble: 4 × 1
      x
  <dbl>
1     2
2     5
3     5
4    NA
```

```
arrange(df, desc(x))
# A tibble: 4 × 1
      x
  <dbl>
1     5
2     5
3     2
4    NA
```

Exercise: Sorting `flights` data

Sort and find flights

a. Sort flights to find those with largest departure delays.
b. Find the flights that left earliest based on departure time.
c. Which flights traveled the longest distance?
d. Which traveled the shortest?

`select` - select variables of concern

SELECTING VARIABLES

Select one or more variables

```
select(flights, year, month, day)
```

```
select(flights, year, month, day)
# A tibble: 336,776 × 3
    year month   day
   <int> <int> <int>
1   2013     1     1
2   2013     1     1
3   2013     1     1
4   2013     1     1
5   2013     1     1
6   2013     1     1
7   2013     1     1
8   2013     1     1
9   2013     1     1
```

Same

Results

```
select(flights, year:day)
# A tibble: 336,776 × 3
    year month   day
   <int> <int> <int>
1   2013     1     1
2   2013     1     1
3   2013     1     1
4   2013     1     1
5   2013     1     1
6   2013     1     1
7   2013     1     1
8   2013     1     1
9   2013     1     1
```

Deselect one or more variables

```
select(flights, -(year:day))
```

```
> select(flights, -(year:day))
# A tibble: 336,776 x 16
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
      <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>    <int>
 1      517            515         2      830            819        11 UA        1545
 2      533            529         4      850            830        20 UA        1714
 3      542            540         2      923            850        33 AA        1141
 4      544            545        -1     1004           1022       -18 B6         725
 5      554            600        -6      812            837       -25 DL         461
 6      554            558        -4      740            728        12 UA        1696
 7      555            600        -5      913            854        19 B6         507
 8      557            600        -3      709            723       -14 EV        5708
 9      557            600        -3      838            846        -8 B6          79
10      558            600        -2      753            745         8 AA         301
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>,
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

USEFUL `select` FUNCTIONS

Blue functions come in `dplyr`

-	Select everything but
:	Select range
contains()	Select columns whose name contains a character string
ends_with()	Select columns whose name ends with a string
everything()	Select every column
matches()	Select columns whose name matches a regular expression
num_range()	Select columns named `x1, x2, x3, x4, x5`
one_of()	Select columns whose names are in a group of names
starts_with()	Select columns whose name starts with a character string

SELECTING VARIABLES

Select variables based on name patterns

```
select(flights, ends_with("time"))
# A tibble: 336,776 × 5
   dep_time sched_dep_time arr_time sched_arr_time air_time
      <int>          <int>    <int>          <int>    <dbl>
1       517            515      830            819      227
2       533            529      850            830      227
3       542            540      923            850      160
4       544            545     1004           1022      183
5       554            600      812            837      116
6       554            558      740            728      150
7       555            600      913            854      158
8       557            600      709            723       53
9       557            600      838            846      140
10      558            600      753            745      138
```

RENAMING VARIABLES ()

Other times we just want to rename our variables:

```
rename(flights, ANNOYING = dep_delay)
```

```
rename(flights, ANNOYING = dep_delay)
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time ANNOYING arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>    <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515        2      830            819        11
2   2013     1     1      533            529        4      850            830        20
3   2013     1     1      542            540        2      923            850        33
4   2013     1     1      544            545       -1     1004           1022       -18
5   2013     1     1      554            600       -6      812            837       -25
6   2013     1     1      554            558       -4      740            728        12
7   2013     1     1      555            600       -5      913            854        19
8   2013     1     1      557            600       -3      709            723       -14
9   2013     1     1      557            600       -3      838            846        -8
10  2013     1     1      558            600       -2      753            745         8
```

Exercises: the commands `select` and `one_of()`

a. What happens if you include the name of a variable multiple times in a `select` call?
b. What does the `one_of()` function do? Why might it be helpful in conjunction with this vector?

```
vars <- c("MONTH","month","day","dep_delay","arr_delay")
```

c. Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

```
select(flights,contains("TIME"))
```

```
> select(flights,contains("TIME"))
# A tibble: 336,776 x 6
   dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
      <int>          <int>    <int>          <int>    <dbl> <dttm>
 1      517            515      830            819      227 2013-01-01 05:00:00
 2      533            529      850            830      227 2013-01-01 05:00:00
 3      542            540      923            850      160 2013-01-01 05:00:00
 4      544            545     1004           1022      183 2013-01-01 05:00:00
 5      554            600      812            837      116 2013-01-01 06:00:00
 6      554            558      740            728      150 2013-01-01 05:00:00
 7      555            600      913            854      158 2013-01-01 06:00:00
 8      557            600      709            723       53 2013-01-01 06:00:00
 9      557            600      838            846      140 2013-01-01 06:00:00
10      558            600      753            745      138 2013-01-01 06:00:00
# ... with 336,766 more rows
```

REDUCE OUR DATA (http://uc-r.github.io/data_wrangling/week-4)

Lets work with a smaller data set

```
(flights_sml <- select(flights,year:day,
                       ends_with("delay"),
                       distance,air_time))
```

```
## # A tibble: 336,776 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
##  7  2013     1     1        -5        19     1065      158
##  8  2013     1     1        -3       -14      229       53
##  9  2013     1     1        -3        -8      944      140
## 10  2013     1     1        -2         8      733      138
## # ... with 336,766 more rows
```

CREATE NEW VARIABLES (http://uc-r.github.io/data_wrangling/week-4)

`mutate` creates new variables with functions of existing variables:

```
mutate(flights_sml,gain=arr_delay - dep_delay,
       speed = distance/air_time * 60)
```

```
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  speed = distance / air_time * 60
)
# A tibble: 336,776 × 9
    year month   day dep_delay arr_delay distance air_time  gain    speed
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl>    <dbl>
1   2013     1     1         2        11     1400      227     9 370.0441
2   2013     1     1         4        20     1416      227    16 374.2731
3   2013     1     1         2        33     1089      160    31 408.3750
4   2013     1     1        -1       -18     1576      183   -17 516.7213
5   2013     1     1        -6       -25      762      116   -19 394.1379
6   2013     1     1        -4        12      719      150    16 287.6000
7   2013     1     1        -5        19     1065      158    24 404.4304
8   2013     1     1        -3       -14      229       53   -11 259.2453
9   2013     1     1        -3        -8      944      140    -5 404.5714
10  2013     1     1        -2         8      733      138    10 318.6957
```

CREATE NEW VARIABLES

Note: you can create variables based on columns that you've just created:

```
mutate(flights_sml,gain=arr_delay - dep_delay,
       hours = air_time / 60, gain_per_hour = gain / hours)
```

```
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
# A tibble: 336,776 × 10
    year month   day dep_delay arr_delay distance air_time  gain     hours gain_per_hour
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl>     <dbl>         <dbl>
1   2013     1     1         2        11     1400      227     9 3.7833333      2.378855
2   2013     1     1         4        20     1416      227    16 3.7833333      4.229075
3   2013     1     1         2        33     1089      160    31 2.6666667     11.625000
4   2013     1     1        -1       -18     1576      183   -17 3.0500000     -5.573770
5   2013     1     1        -6       -25      762      116   -19 1.9333333     -9.827586
6   2013     1     1        -4        12      719      150    16 2.5000000      6.400000
7   2013     1     1        -5        19     1065      158    24 2.6333333      9.113924
8   2013     1     1        -3       -14      229       53   -11 0.8833333    -12.452830
9   2013     1     1        -3        -8      944      140    -5 2.3333333     -2.142857
```

MANY USEFUL CREATION FUNCTIONS

There are a wide variety of functions you can use with `mutate()`

Functions	Description
`+, -, *, /, ^`	arithmetic
`x / sum(x)`	arithmetic w/aggregate functions
`%/%, %%`	modular arithmetic
`log, exp, sqrt`	transformations
`lag, lead`	offsets
`cumsum, cumprod, cum…`	cum/rolling aggregates
`>, >=, <, <=, !=, ==`	logical comparisons
`min_rank, dense_rank, etc`	ranking
`between`	are values between a and b?

Exercise: `tidyverse` convert

a. Create a new variable `distance_km` that converts distance in miles to kilometers.

b. Create a `time_per_km` variable based on `air_time` and `distance_km`.

Convert all columns of a vector to character (https://stackoverflow.com/questions/43789278/convert-all-columns-to-characters-in-a-data-frame)

```
library(dplyr)
mtcars %>%  mutate_all(as.character)
```

```
##      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1    21   6   160 110  3.9  2.62 16.46  0  1    4    4
## 2    21   6   160 110  3.9 2.875 17.02  0  1    4    4
## 3  22.8   4   108  93 3.85  2.32 18.61  1  1    4    1
## 4  21.4   6   258 110 3.08 3.215 19.44  1  0    3    1
## 5  18.7   8   360 175 3.15  3.44 17.02  0  0    3    2
## 6  18.1   6   225 105 2.76  3.46 20.22  1  0    3    1
## 7  14.3   8   360 245 3.21  3.57 15.84  0  0    3    4
## 8  24.4   4 146.7  62 3.69  3.19    20  1  0    4    2
## 9  22.8   4 140.8  95 3.92  3.15  22.9  1  0    4    2
## 10 19.2   6 167.6 123 3.92  3.44  18.3  1  0    4    4
## 11 17.8   6 167.6 123 3.92  3.44  18.9  1  0    4    4
## 12 16.4   8 275.8 180 3.07  4.07  17.4  0  0    3    3
## 13 17.3   8 275.8 180 3.07  3.73  17.6  0  0    3    3
## 14 15.2   8 275.8 180 3.07  3.78    18  0  0    3    3
## 15 10.4   8   472 205 2.93  5.25 17.98  0  0    3    4
## 16 10.4   8   460 215    3 5.424 17.82  0  0    3    4
## 17 14.7   8   440 230 3.23 5.345 17.42  0  0    3    4
## 18 32.4   4  78.7  66 4.08   2.2 19.47  1  1    4    1
## 19 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## 20 33.9   4  71.1  65 4.22 1.835  19.9  1  1    4    1
## 21 21.5   4 120.1  97  3.7 2.465 20.01  1  0    3    1
## 22 15.5   8   318 150 2.76  3.52 16.87  0  0    3    2
## 23 15.2   8   304 150 3.15 3.435  17.3  0  0    3    2
## 24 13.3   8   350 245 3.73  3.84 15.41  0  0    3    4
## 25 19.2   8   400 175 3.08 3.845 17.05  0  0    3    2
## 26 27.3   4    79  66 4.08 1.935  18.9  1  1    4    1
## 27   26   4 120.3  91 4.43  2.14  16.7  0  1    5    2
## 28 30.4   4  95.1 113 3.77 1.513  16.9  1  1    5    2
## 29 15.8   8   351 264 4.22  3.17  14.5  0  1    5    4
## 30 19.7   6   145 175 3.62  2.77  15.5  0  1    5    6
## 31   15   8   301 335 3.54  3.57  14.6  0  1    5    8
## 32 21.4   4   121 109 4.11  2.78  18.6  1  1    4    2
```

`summarise` - Collapse many values down to a single summary statistic

We can create summary statistics of one or more variables:

```
summarise(flights, dep_delay_mean = mean(dep_delay, na.rm = TRUE))
# A tibble: 1 × 1
  dep_delay_mean
           <dbl>
1       12.63907
```

Important: try this without `na.rm = TRUE` and see what happens. Why does this happen?

Get summary statistics by group (http://www.sthda.com/english/wiki/one-way-anova-test-in-r)

```
library(dplyr)
group_by(iris, Species) %>%
  summarise(
    count = n(),
    mean = mean(Sepal.Length, na.rm = TRUE),
    sd = sd(Sepal.Length, na.rm = TRUE)
  )
```

```
## # A tibble: 3 x 4
##   Species    count  mean    sd
##   <fct>      <int> <dbl> <dbl>
## 1 setosa        50  5.01 0.352
## 2 versicolor    50  5.94 0.516
## 3 virginica     50  6.59 0.636
```

Links and resources

- UC Business Analytics R Programming Guide - **Course: Data Wrangling with R** (http://uc-r.github.io/data_wrangling/week-3)
- **Manipulating, analyzing and exporting data with tidyverse** (https://datacarpentry.org/R-ecology-lesson/03-dplyr.html) **Pipes in R Tutorial For Beginners** (https://www.datacamp.com/community/tutorials/pipe-r-tutorial)
- **Convert all columns** to characters in a data.frame (https://stackoverflow.com/questions/43789278/convert-all-columns-to-characters-in-a-data-frame)