More dplyr

Eric Hare, Andee Kaplan, Carson Sievert

June 10, 2015

Using the packages tidyr, dplyr

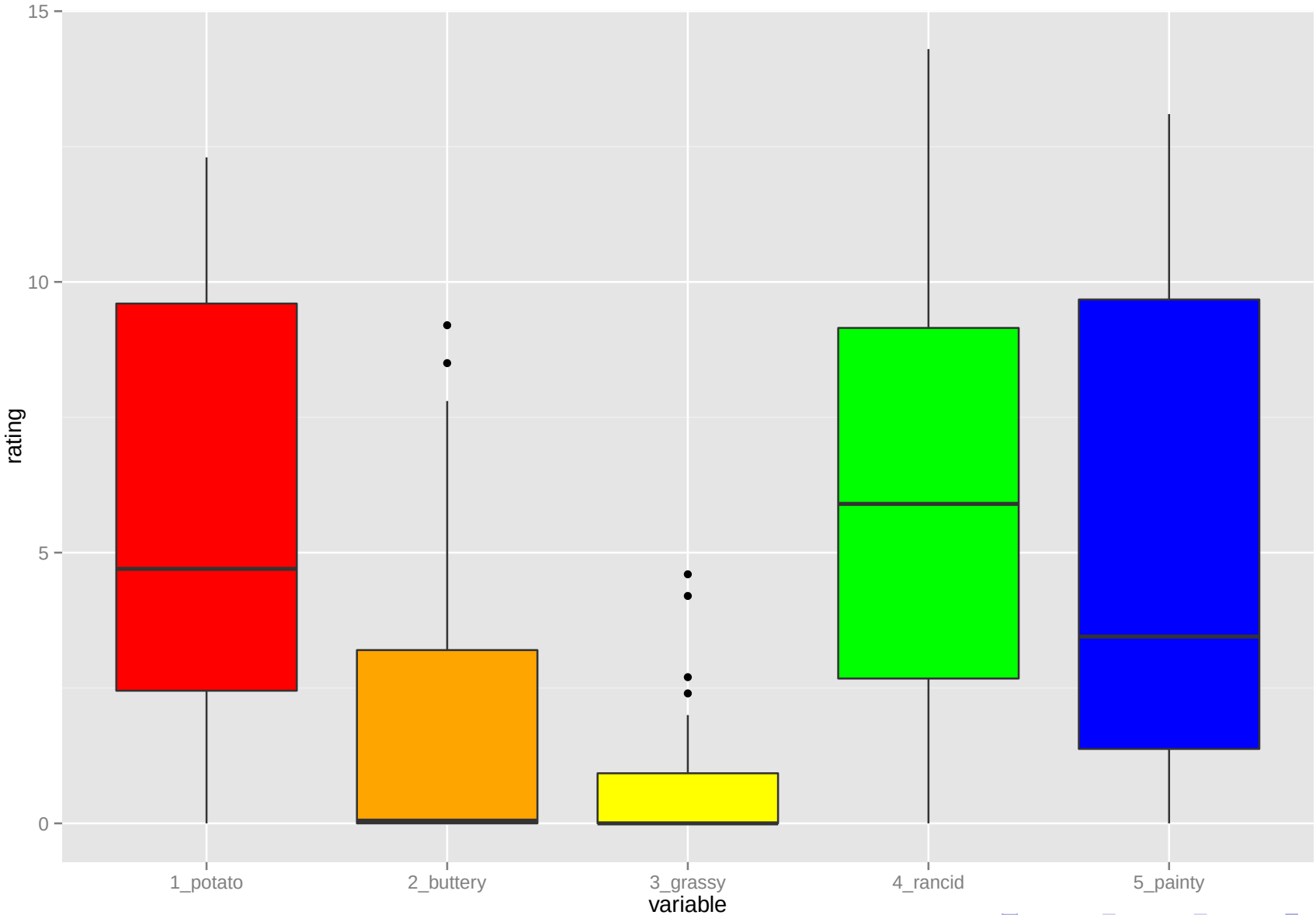
During a ten week sensory experiment, 12 individuals were asked to assess taste of french fries on several scales (how potato-y, buttery, grassy, rancid, paint-y do the fries taste?)

French fries were fried in one of three different oils, and each week individuals had to assess six batches of french fries (all three oils, replicated twice)

	time	treatment	subject	rep	potato	buttery	grassy	ran
61	1	1	3	1.00	2.90	0.00	0.00	0
25	1	1	3	2.00	14.00	0.00	0.00	1
62	1	1	10	1.00	11.00	6.40	0.00	0
26	1	1	10	2.00	9.90	5.90	2.90	2
63	1	1	15	1.00	1.20	0.10	0.00	1
27	1	1	15	2.00	8.80	3.00	3.60	1

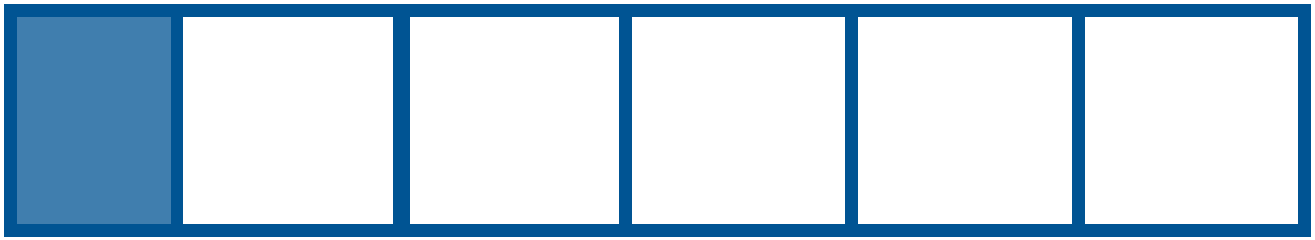
This format is not ideal for data analysis

What code would be needed to plot each of the ratings over time as a different color?

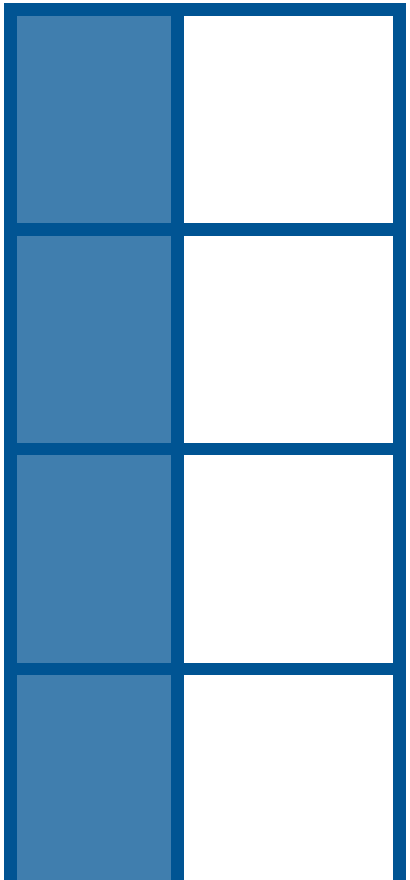


What we have and what we want

We want to change this **wide format**:



to this **long format**:



Gathering

- When gathering, you need to specify the **keys** (identifiers) and the **values** (measures).

Keys/Identifiers:

- Identify a record (must be unique)
- Example: Indices on an random variable
- Fixed by design of experiment (known in advance)
- May be single or composite (may have one or more variables)

Values/Measures:

- Collected during the experiment (not known in advance)
- Usually numeric quantities

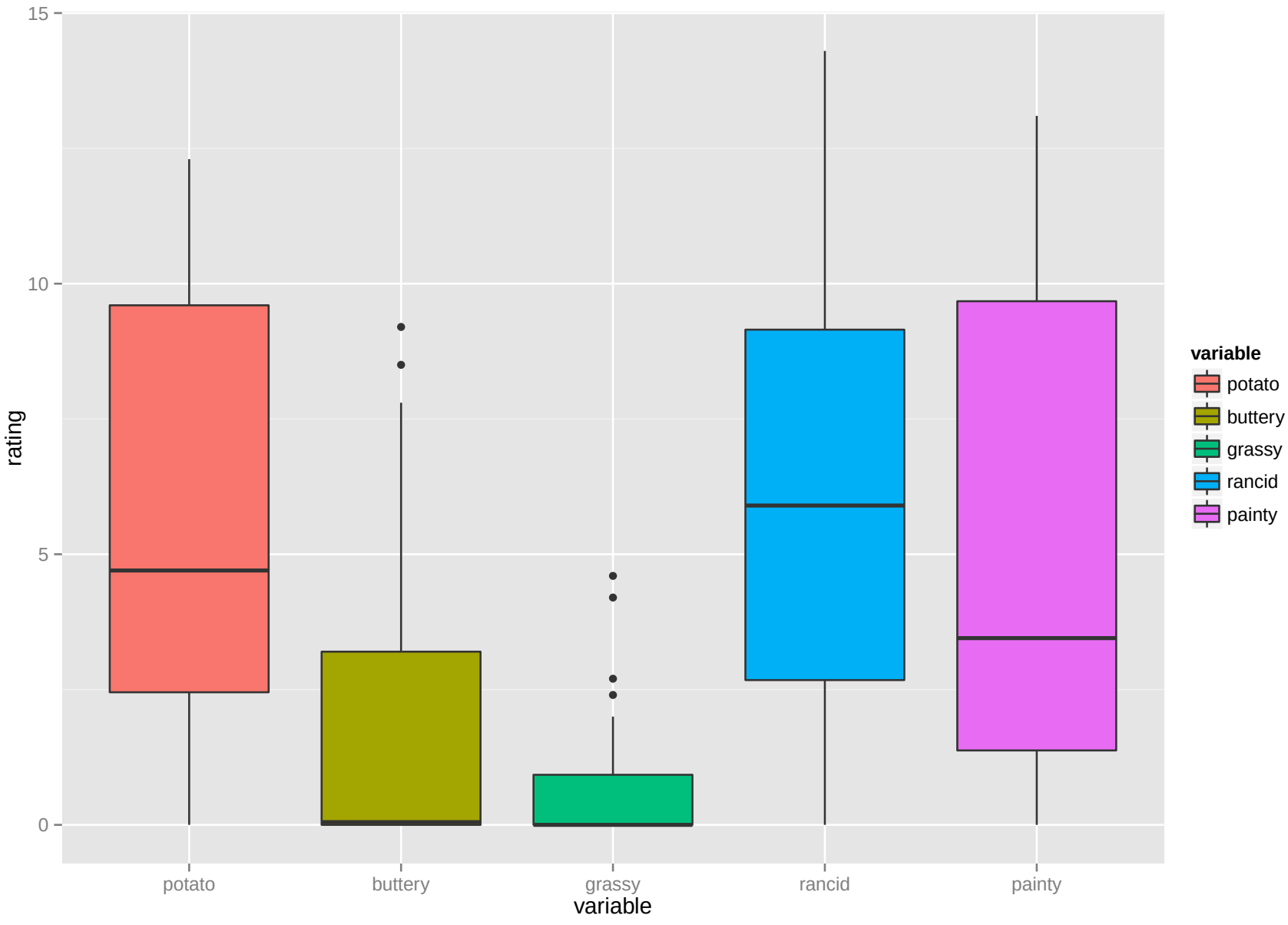
Gathering the French Fry Data

```
french_fries_long <- gather(french_fries, key = variable, v

head(french_fries_long)
```

```
##   time treatment subject rep variable rating
## 1    1         1       3   1   potato    2.9
## 2    1         1       3   2   potato   14.0
## 3    1         1      10   1   potato   11.0
## 4    1         1      10   2   potato    9.9
## 5    1         1      15   1   potato    1.2
## 6    1         1      15   2   potato    8.8
```

Let's Reconstruct our Plot

```
french_fries_long_sub <- french_fries_long[
  french_fries_long$time == 10,]

qplot(variable, rating, data = french_fries_long_sub, fill
```


15
10
5
0
rating
potato
buttery
grassy
rancid
painty
variable
variable
potato
buttery
grassy
rancid
painty

Long to Wide

In certain applications, we may wish to take a long dataset and convert it to a wide dataset (Perhaps displaying in a table).

```
head(french_fries_long)
```

```
##    time treatment subject rep variable rating
## 1     1         1       3   1   potato    2.9
## 2     1         1       3   2   potato   14.0
## 3     1         1      10   1   potato   11.0
## 4     1         1      10   2   potato    9.9
## 5     1         1      15   1   potato    1.2
## 6     1         1      15   2   potato    8.8
```

Spread

We use the **spread** function from tidyr to do this:

```
french_fries_wide <- spread(french_fries_long, key = variab

head(french_fries_wide)
```

```
##   time treatment subject rep potato buttery grassy ranci
## 1    1         1       3   1    2.9     0.0    0.0     0
## 2    1         1       3   2   14.0     0.0    0.0     1
## 3    1         1      10   1   11.0     6.4    0.0     0
## 4    1         1      10   2    9.9     5.9    2.9     2
## 5    1         1      15   1    1.2     0.1    0.0     1
## 6    1         1      15   2    8.8     3.0    3.6     1
```

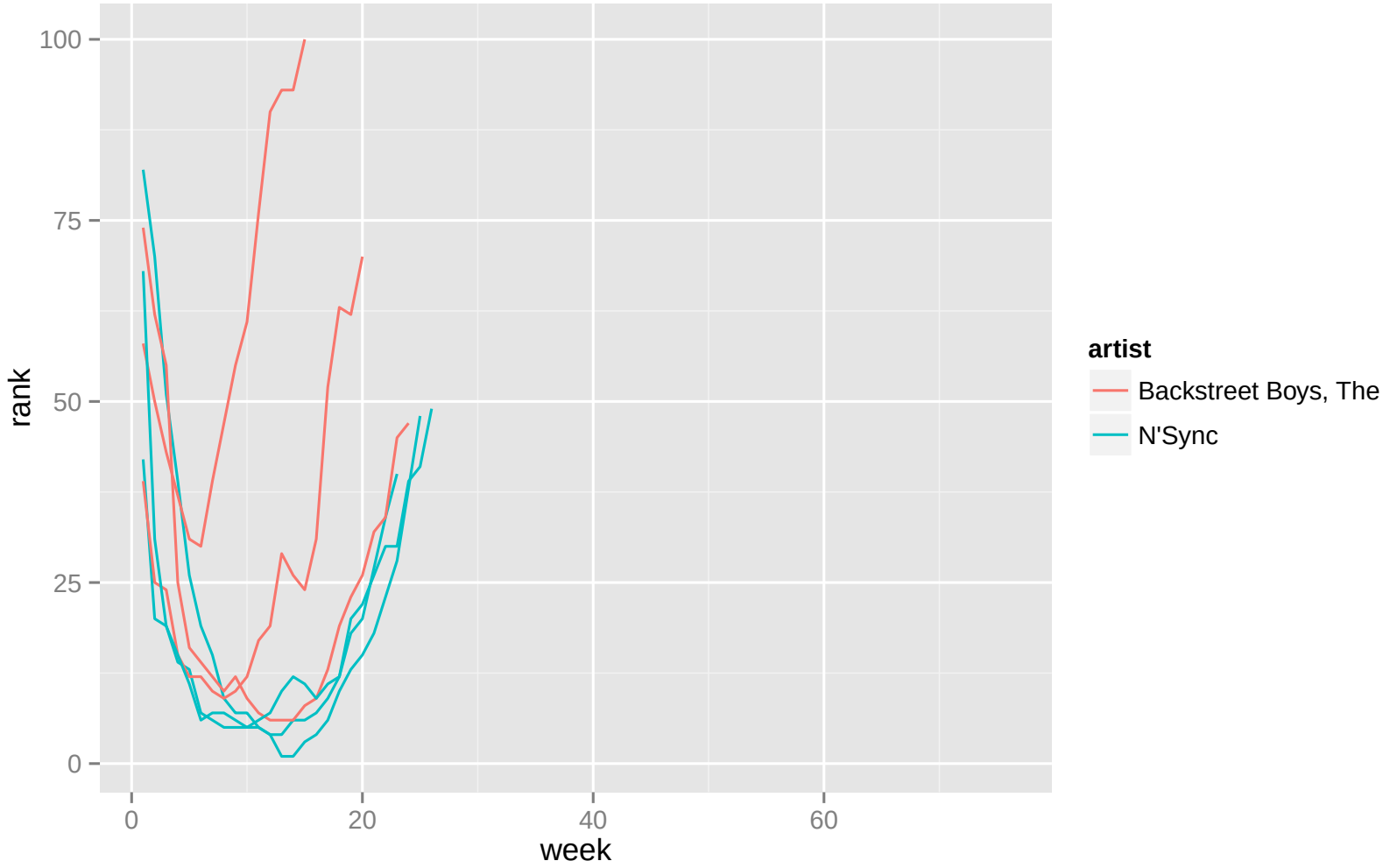
YOUR TURN

1. Read in the billboard top 100 music data, which contains N'Sync and Backstreet Boys songs that entered the billboard charts in the year 2000

```
billboard <- read.csv("http://heike.github.io/rwrks/03a
```

2. Use `tidyr` to convert this data into a long format appropriate for plotting a time series (date on the x axis, chart position on the y axis)

3. Use `ggplot2` to create this time series plot:



The Split-Apply-Combine Approach

- *Split* a dataset into many smaller sub-datasets
- *Apply* some function to each sub-dataset to compute a result
- *Combine* the results of the function calls into a one dataset

Split-Apply-Combine in dplyr

```
library(dplyr)

french_fries_split <- group_by(french_fries_long, variable)
french_fries_apply <- summarise(french_fries_split, rating
french_fries_apply
```

```
## Source: local data frame [5 x 2]
##
##   variable    rating
## 1   potato 6.9525180
## 2  buttery 1.8236994
## 3   grassy 0.6641727
## 4   rancid 3.8522302
## 5   painty 2.5217579
```

The pipe operator

- dplyr allows us to chain together these data analysis tasks using the %>% (pipe) operator
- `x %>% f(y)` is shorthand for `f(x, y)`
- Example:

```
french_fries %>%
    gather(key = variable, value = rating, potato:painty) %
    group_by(variable) %>%
    summarise(rating = mean(rating, na.rm = TRUE))
```

```
## Source: local data frame [5 x 2]
##
##   variable    rating
## 1   potato 6.9525180
## 2  buttery 1.8236994
## 3   grassy 0.6641727
## 4   rancid 3.8522302
## 5   painty 2.5217579
```

dplyr verbs

There are five primary dplyr **verbs**, representing distinct data analysis tasks:

- Filter: Remove the rows of a data frame, producing subsets
- Arrange: Reorder the rows of a data frame
- Select: Select particular columns of a data frame
- Mutate: Add new columns that are functions of existing columns
- Summarise: Create collapsed summaries of a data frame

Filter

```
french_fries %>%
    filter(subject == 3, time == 1)
```

```
##   time treatment subject rep potato buttery grassy ranci
## 1    1         1       3   1    2.9     0.0    0.0    0.
## 2    1         1       3   2   14.0     0.0    0.0    1.
## 3    1         2       3   1   13.9     0.0    0.0    3.
## 4    1         2       3   2   13.4     0.1    0.0    1.
## 5    1         3       3   1   14.1     0.0    0.0    1.
## 6    1         3       3   2    9.5     0.0    0.6    2.
```

Arrange

```
french_fries %>%
    arrange(desc(rancid)) %>%
    head
```

```
##    time treatment subject rep potato buttery grassy ranci
## 1     9         2      51   1    7.3     2.3      0    14.
## 2    10         1      86   2    0.7     0.0      0    14.
## 3     5         2      63   1    4.4     0.0      0    13.
## 4     9         2      63   1    1.8     0.0      0    13.
## 5     5         2      19   2    5.5     4.7      0    13.
## 6     4         3      63   1    5.6     0.0      0    13.
```

Select

```
french_fries %>%
    select(time, treatment, subject, rep, potato) %>%
    head
```

```
##    time treatment subject rep potato
## 61    1         1       3   1    2.9
## 25    1         1       3   2   14.0
## 62    1         1      10   1   11.0
## 26    1         1      10   2    9.9
## 63    1         1      15   1    1.2
## 27    1         1      15   2    8.8
```

Mutate

```
french_fries %>%
    mutate(rancid2 = rancid^2) %>%
    head
```

```
##   time treatment subject rep potato buttery grassy ranci
## 1    1         1       3   1    2.9     0.0    0.0     0
## 2    1         1       3   2   14.0     0.0    0.0     1
## 3    1         1      10   1   11.0     6.4    0.0     0
## 4    1         1      10   2    9.9     5.9    2.9     2
## 5    1         1      15   1    1.2     0.1    0.0     1
## 6    1         1      15   2    8.8     3.0    3.6     1
```

Summarise

```
french_fries %>%
    group_by(time, treatment) %>%
    summarise(mean_rancid = mean(rancid), sd_rancid = sd(ra
```

```
## Source: local data frame [30 x 4]
## Groups: time
##
##    time treatment mean_rancid sd_rancid
## 1     1         1    2.758333  3.212870
## 2     1         2    1.716667  2.714801
## 3     1         3    2.600000  3.202037
## 4     2         1    3.900000  4.374730
## 5     2         2    2.141667  3.117540
## 6     2         3    2.495833  3.378767
## 7     3         1    4.650000  3.933358
## 8     3         2    2.895833  3.773532
## 9     3         3    3.600000  3.592867
## 10    4         1    2.079167  2.394737
```

YOUR TURN

Read in the flights data:

```
flights <- read.csv("http://heike.github.io/rwrks/03a-r-fo
```

This dataset contains information on over 300,000 flights that departed from New York City in the year 2013.

1. Using dplyr and the pipe operator, create a data frame consisting of the average arrival delay (`arr_delay`) based on the destination airport (`dest`). Sort this data frame in descending order, so the destination airport with the largest delay is first.
2. Find out the most used airports for each airline carrier.

Dates and Times

Dates are deceptively hard to work with in R.

Example: 02/05/2012. Is it February 5th, or May 2nd?

Other things are difficult too:

- Time zones
- POSIXct format in base R is challenging

The **lubridate** package helps tackle some of these issues.

Basic Lubridate Use

```
library(lubridate)

now()
today()
now() + hours(4)
today() - days(2)
```

```
## [1] "2015-06-10 15:28:44 CDT"
## [1] "2015-06-10"
## [1] "2015-06-10 19:28:44 CDT"
## [1] "2015-06-08"
```

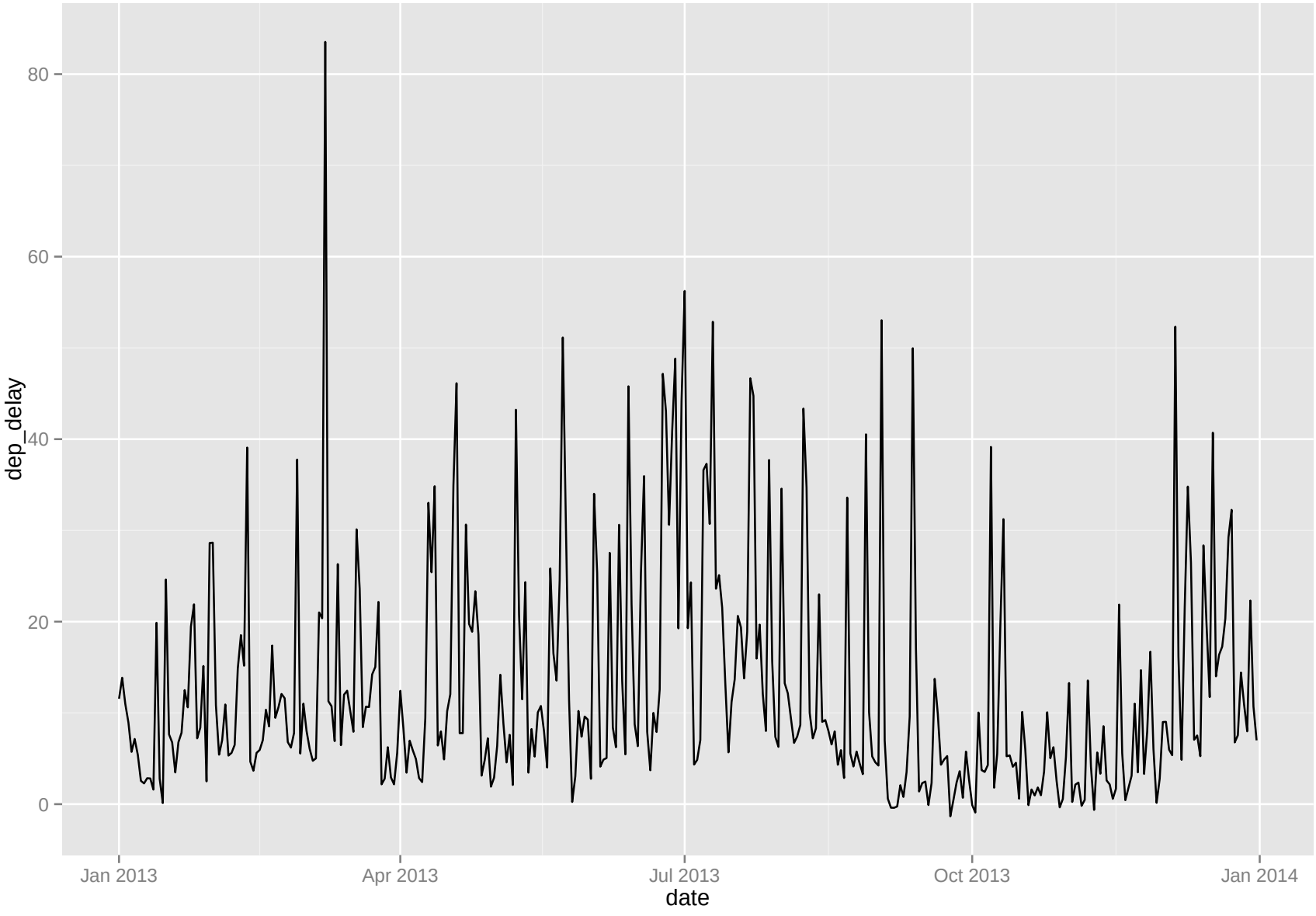
Parsing Dates

```
ymd("2013-05-14")
mdy("05/14/2013")
dmy("14052013")
ymd_hms("2013:05:14 14:50:30", tz = "America/Chicago")
```

```
## [1] "2013-05-14 UTC"
## [1] "2013-05-14 UTC"
## [1] "2013-05-14 UTC"
## [1] "2013-05-14 14:50:30 CDT"
```

YOUR TURN

1. Using the `flights` data, create a new column Date using lubridate. You will need to paste together the columns year, month, and day in order to do this. See the `paste` function.
2. Use dplyr to calculate the average departure delay for each date.
3. Plot a time series of the date versus the average departure delay


80
60
40
20
0
dep_delay
Jan 2013
Apr 2013
Jul 2013
Oct 2013
Jan 2014
date