Bootcamp outline

Day #1: Reproducible research, Command line introduction, Intro to R and RStudio, and data input

Day #2: Data cleaning, manipulation, processing

Day #3: R and data plotting, five-minute presentations

R

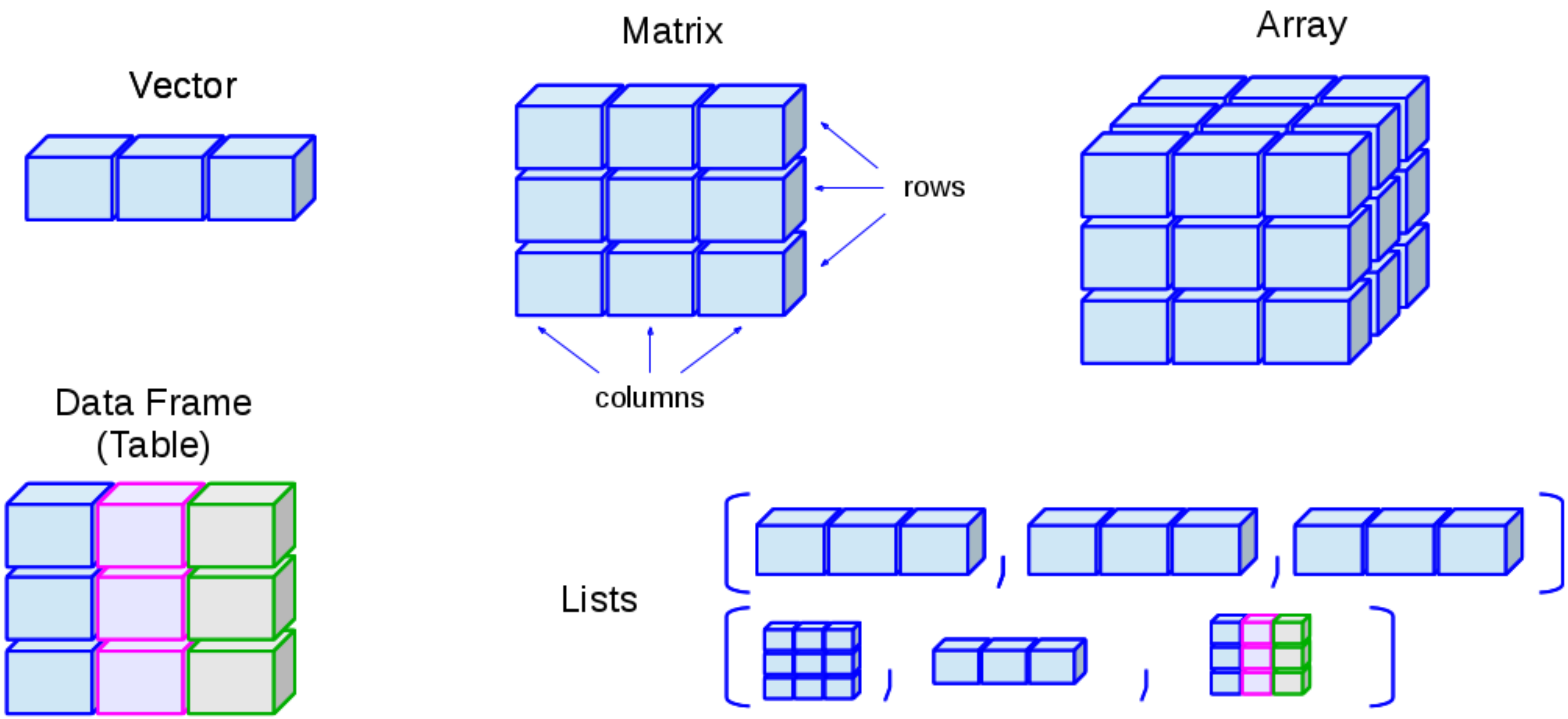
is a software environment
for statistical computing

Data structures in R

Vectors are one-dimensional data sets of any one data type

Data frames are two-dimensional data sets of any data type

Lists are groups of vectors, data frames, or other lists.



How do we explore data objects?

`names(df)` = gives the names of columns of df

`rownames(df)` = gives the names of rows of df

`colnames(df)` = gives the names of columns of df

`dim(df)` = outputs the number of rows then columns of df

`length(df)` = outputs the length of df

`summary(df)` = outputs summary statistics of df

Other useful functions in R

`match, %in%`	= match elements in one object to another
`grep`	= look for pattern in strings
`seq`	= generate a sequence of numbers
`is.na`	= gives TRUE or FALSE for NA data values
`unique`	= look for unique data values
`merge, join`	= merge two vectors or data frames
`c`	= combine data
`rbind, cbind`	= combine rows or columns, respectively
`rep`	= replicate data

Flow of data analysis

✓1. Read — Yesterday

2. Tidy
3. Process

} Today

4. Plot
5. Present

} Thursday

Take some time to move through your data

1. Are there missing values? `is.na()`

2. What are the column names? `colnames()`

3. What are the dimensions? `dim()`

4. What are the row values? `rownames()`

5. Are the data organized long or wide?

Some thoughts about organizing raw data

- Be consistent.
- Write dates as `YYYY-MM-DD` or YYYYMMDD.
- Fill in all of the cells.
- Put just one thing in a cell.
- Make it a rectangle.
- Create a data dictionary.
- No calculations in the raw data files.
- Don’t use font color or highlighting as data.
- Choose good names for things.
- Make backups.
- Use data validation to avoid data entry mistakes. Unit tests!
- Save the data in plain text files.

from Karl Broman (UW Madison), check out his fantastic blog
Other great rules here: https://github.com/jtleek/datasharing

The most important part of data analysis is to *think* about your data

What do you want to test?

How will you show that conclusion?

Put the goal at the top of your markdown report

Introducing tidy data

Tidy data are easy to manipulate (dplyr), visualize (ggplot2) and model (many packages).

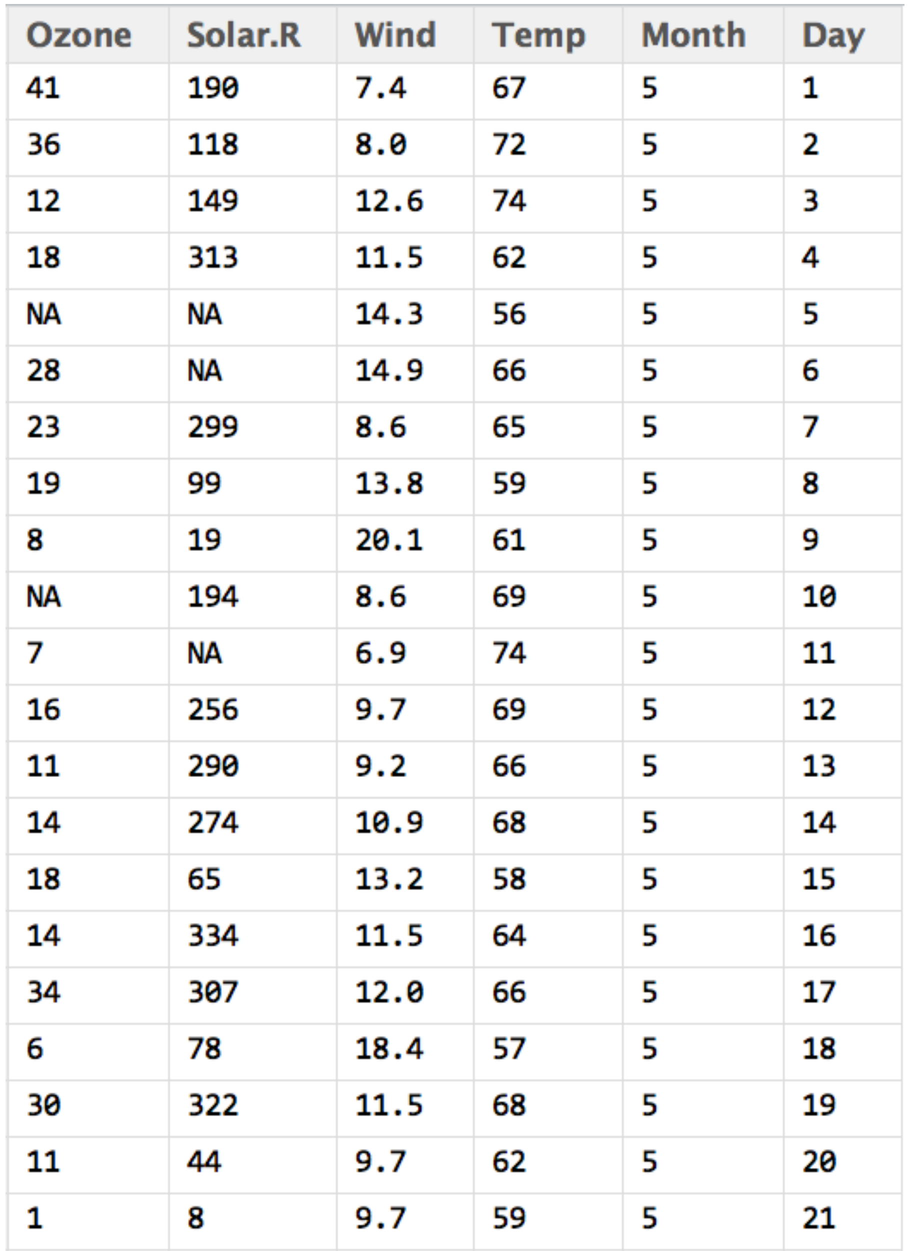
Each column is a variable (sample, replicate, phenotype, etc.).

Each row is an observation.

To you, it might seem like the data in some columns are repetitive.

Data Formats: Long *vs* Wide

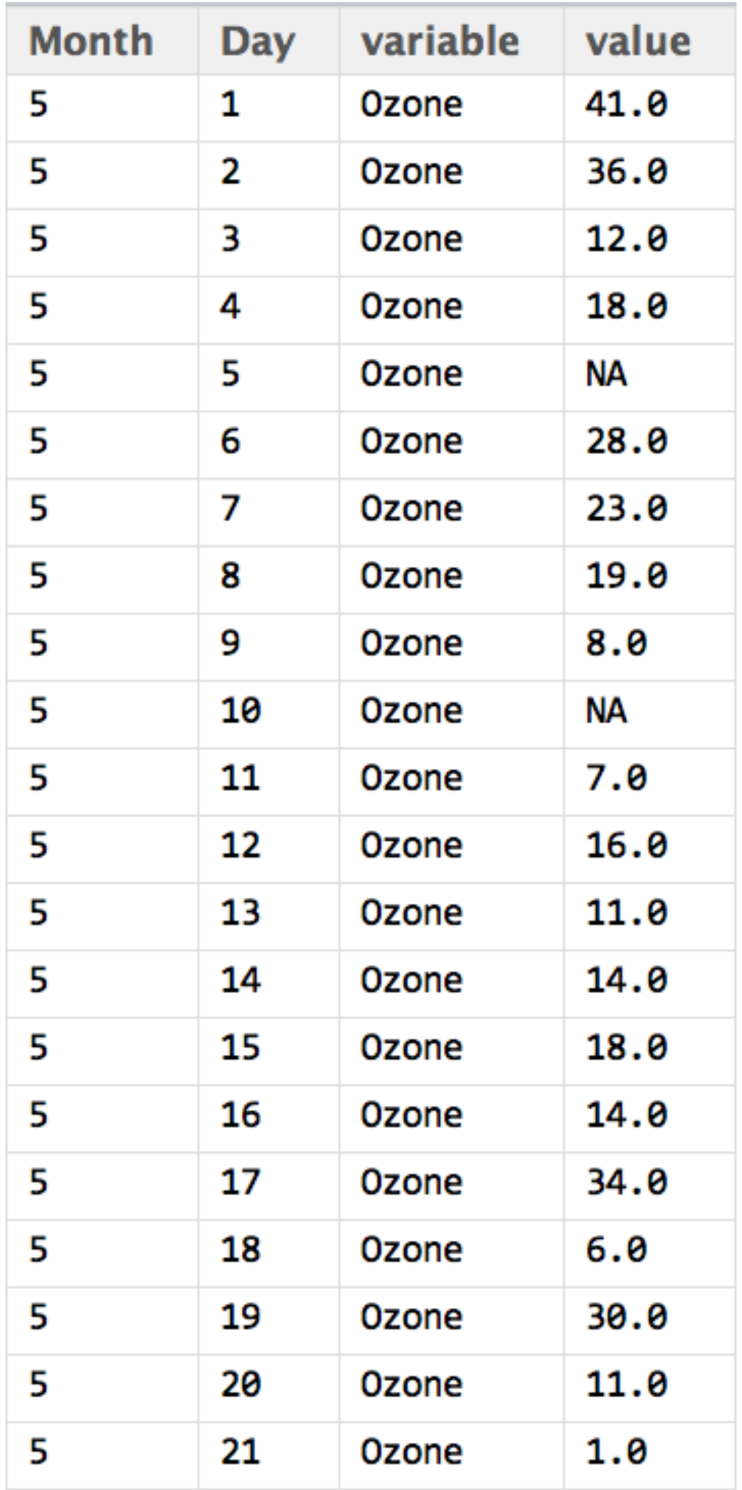
Variables Identifiers



Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6
23	299	8.6	65	5	7
19	99	13.8	59	5	8
8	19	20.1	61	5	9
NA	194	8.6	69	5	10
7	NA	6.9	74	5	11
16	256	9.7	69	5	12
11	290	9.2	66	5	13
14	274	10.9	68	5	14
18	65	13.2	58	5	15
14	334	11.5	64	5	16
34	307	12.0	66	5	17
6	78	18.4	57	5	18
30	322	11.5	68	5	19
11	44	9.7	62	5	20
1	8	9.7	59	5	21

Wide

Identifiers



Month	Day	variable	value
5	1	Ozone	41.0
5	2	Ozone	36.0
5	3	Ozone	12.0
5	4	Ozone	18.0
5	5	Ozone	NA
5	6	Ozone	28.0
5	7	Ozone	23.0
5	8	Ozone	19.0
5	9	Ozone	8.0
5	10	Ozone	NA
5	11	Ozone	7.0
5	12	Ozone	16.0
5	13	Ozone	11.0
5	14	Ozone	14.0
5	15	Ozone	18.0
5	16	Ozone	14.0
5	17	Ozone	34.0
5	18	Ozone	6.0
5	19	Ozone	30.0
5	20	Ozone	11.0
5	21	Ozone	1.0

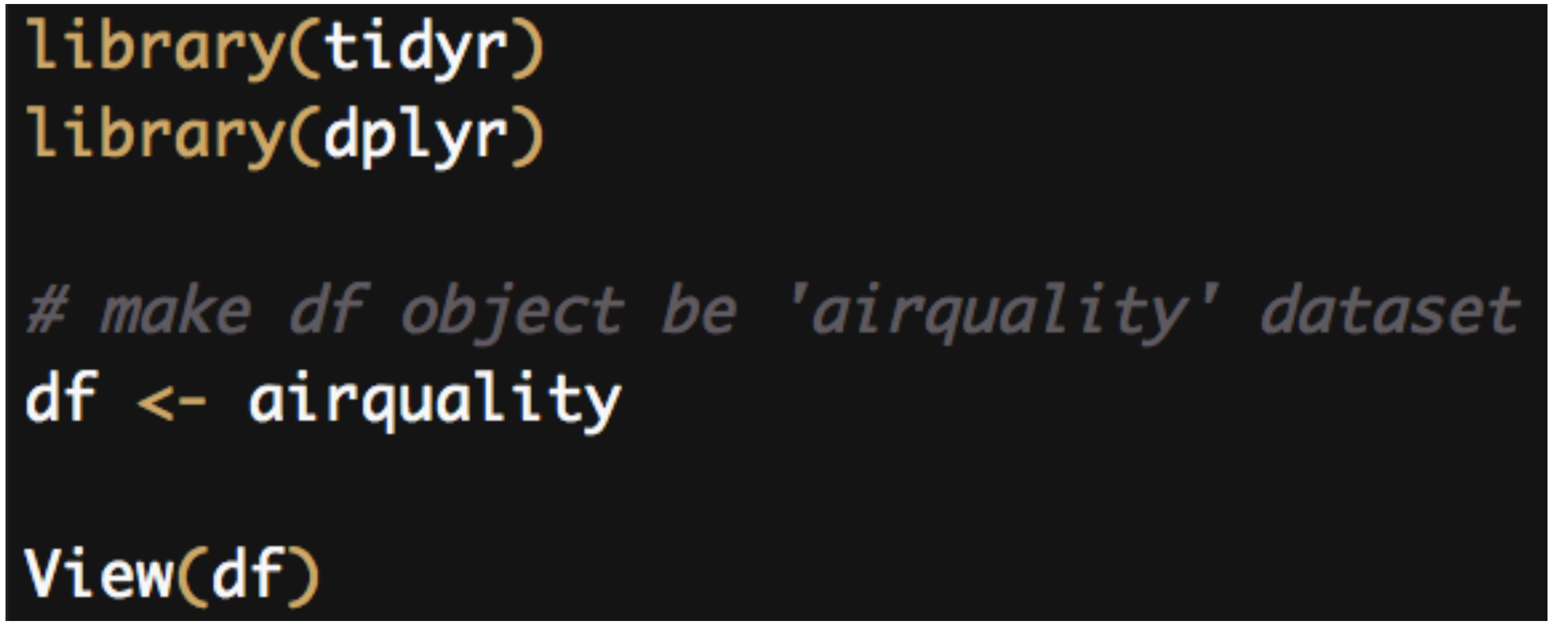
Long

Data Formats: Long *vs* Wide

How to convert between long and wide data?

tidyr - “Easily tidy data with spread and gather functions.”

An example:



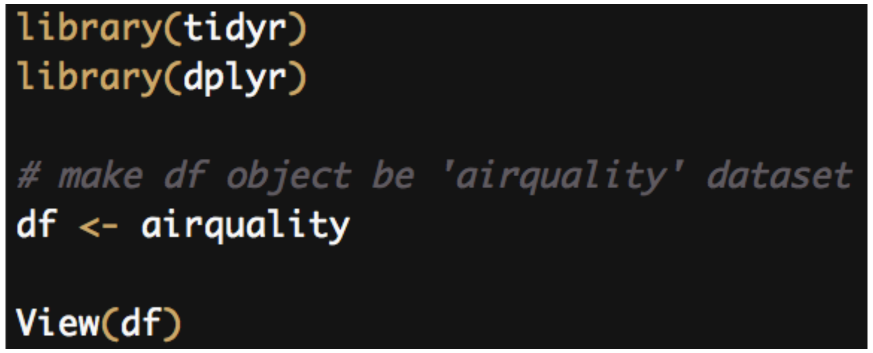
```
library(tidyr)
library(dplyr)

# make df object be 'airquality' dataset
df <- airquality

View(df)
```

Data Formats: Long *vs* Wide

An example:

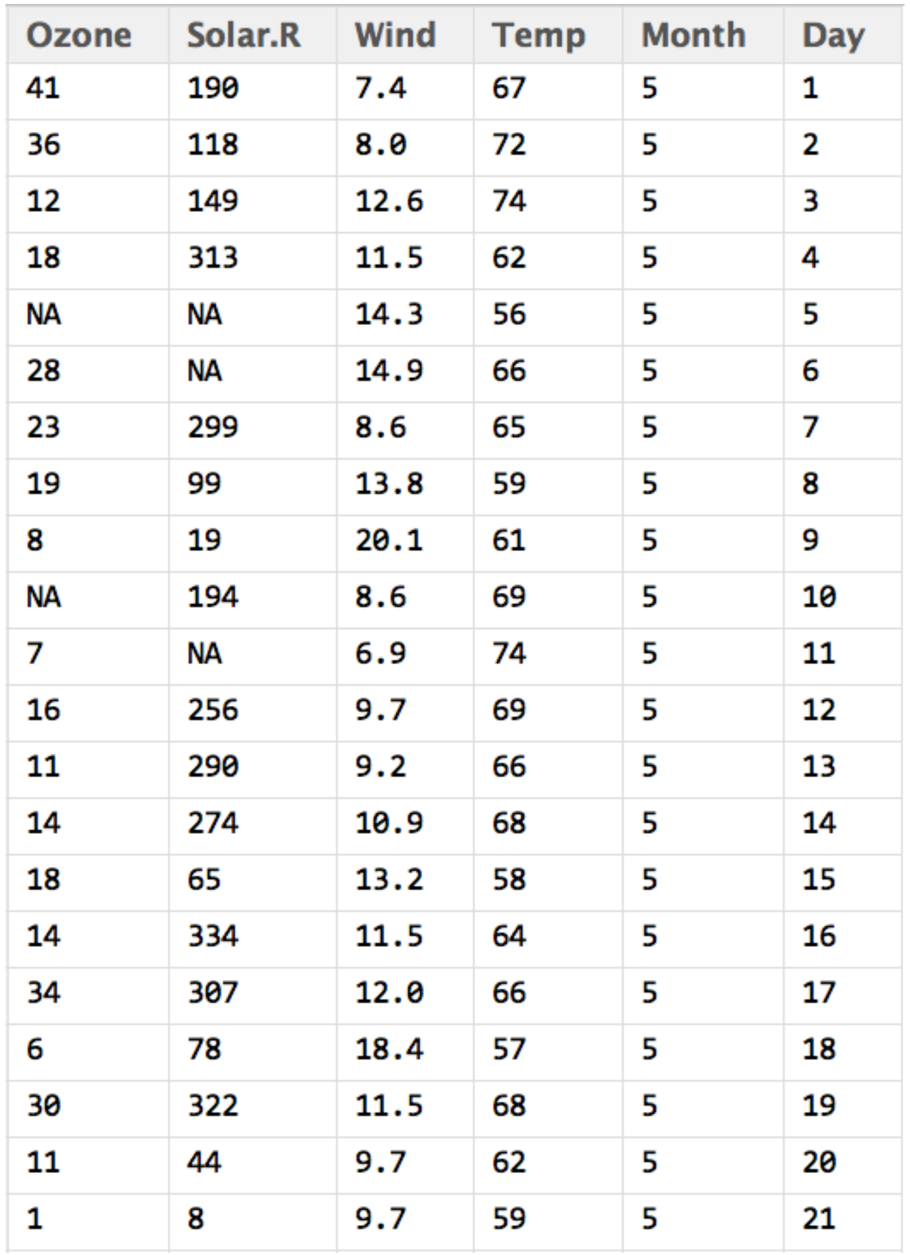


```
library(tidyr)
library(dplyr)

# make df object be 'airquality' dataset
df <- airquality

View(df)
```

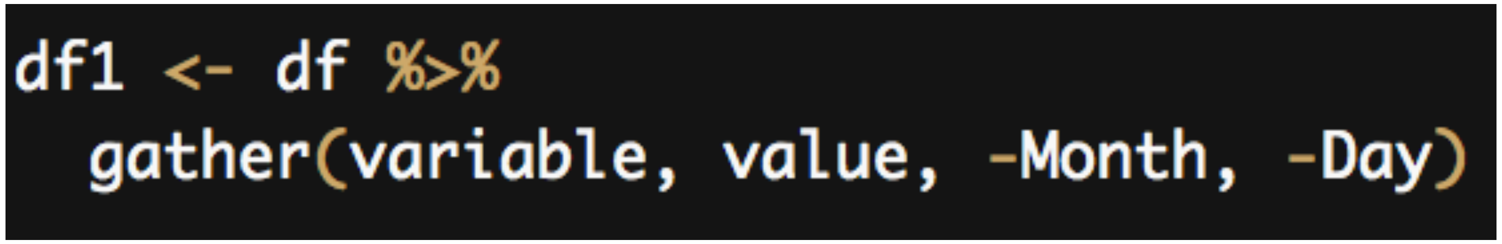
Variables Identifiers



Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6
23	299	8.6	65	5	7
19	99	13.8	59	5	8
8	19	20.1	61	5	9
NA	194	8.6	69	5	10
7	NA	6.9	74	5	11
16	256	9.7	69	5	12
11	290	9.2	66	5	13
14	274	10.9	68	5	14
18	65	13.2	58	5	15
14	334	11.5	64	5	16
34	307	12.0	66	5	17
6	78	18.4	57	5	18
30	322	11.5	68	5	19
11	44	9.7	62	5	20
1	8	9.7	59	5	21

Data Formats: Long *vs* Wide

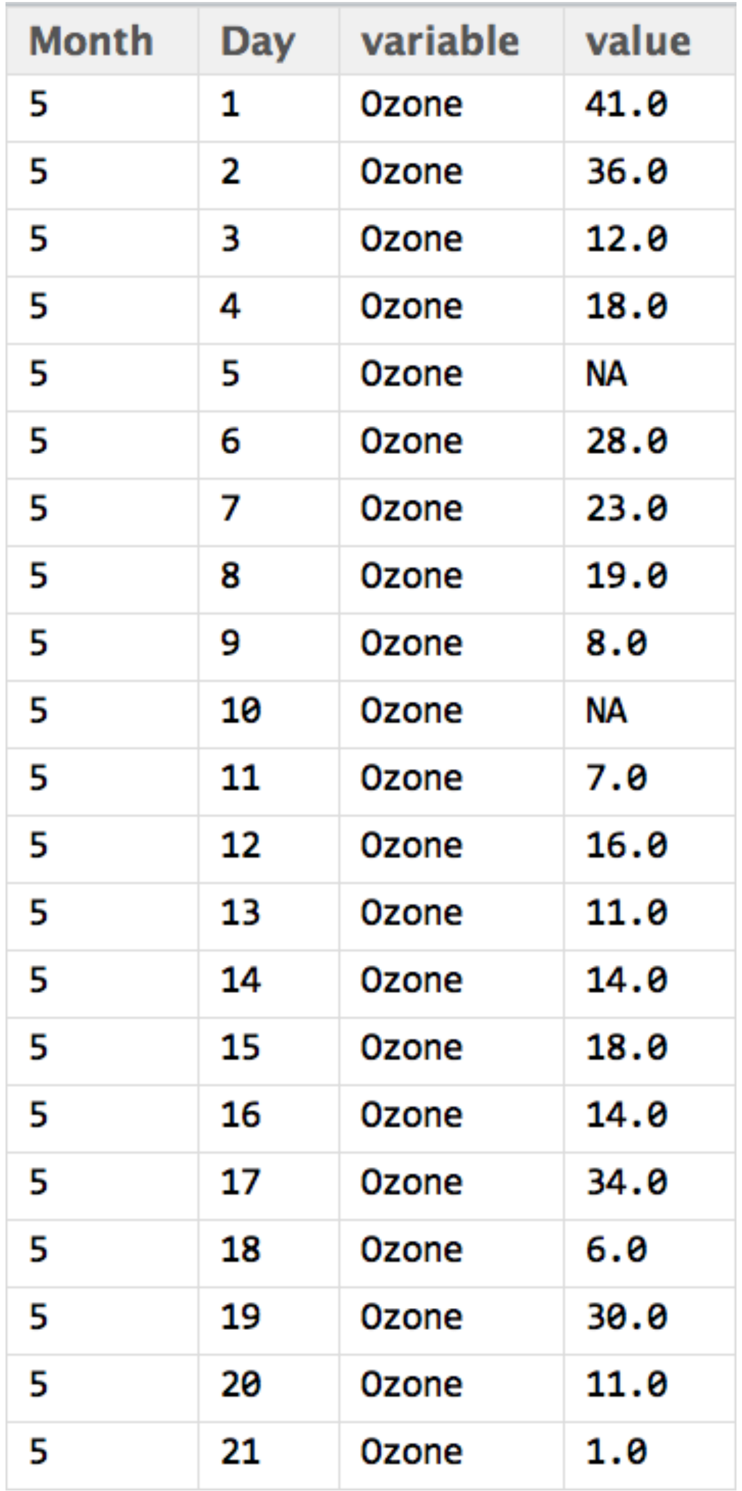
Convert to long format



```
df1 <- df %>%
  gather(variable, value, -Month, -Day)
```

tidyr integrates with dplyr

Identifiers

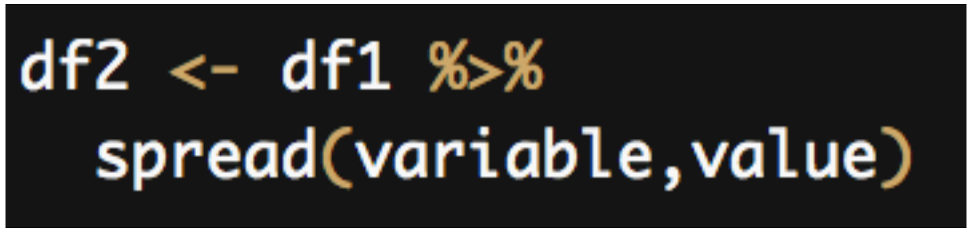


Month	Day	variable	value
5	1	Ozone	41.0
5	2	Ozone	36.0
5	3	Ozone	12.0
5	4	Ozone	18.0
5	5	Ozone	NA
5	6	Ozone	28.0
5	7	Ozone	23.0
5	8	Ozone	19.0
5	9	Ozone	8.0
5	10	Ozone	NA
5	11	Ozone	7.0
5	12	Ozone	16.0
5	13	Ozone	11.0
5	14	Ozone	14.0
5	15	Ozone	18.0
5	16	Ozone	14.0
5	17	Ozone	34.0
5	18	Ozone	6.0
5	19	Ozone	30.0
5	20	Ozone	11.0
5	21	Ozone	1.0

Long

Data Formats: Long *vs* Wide

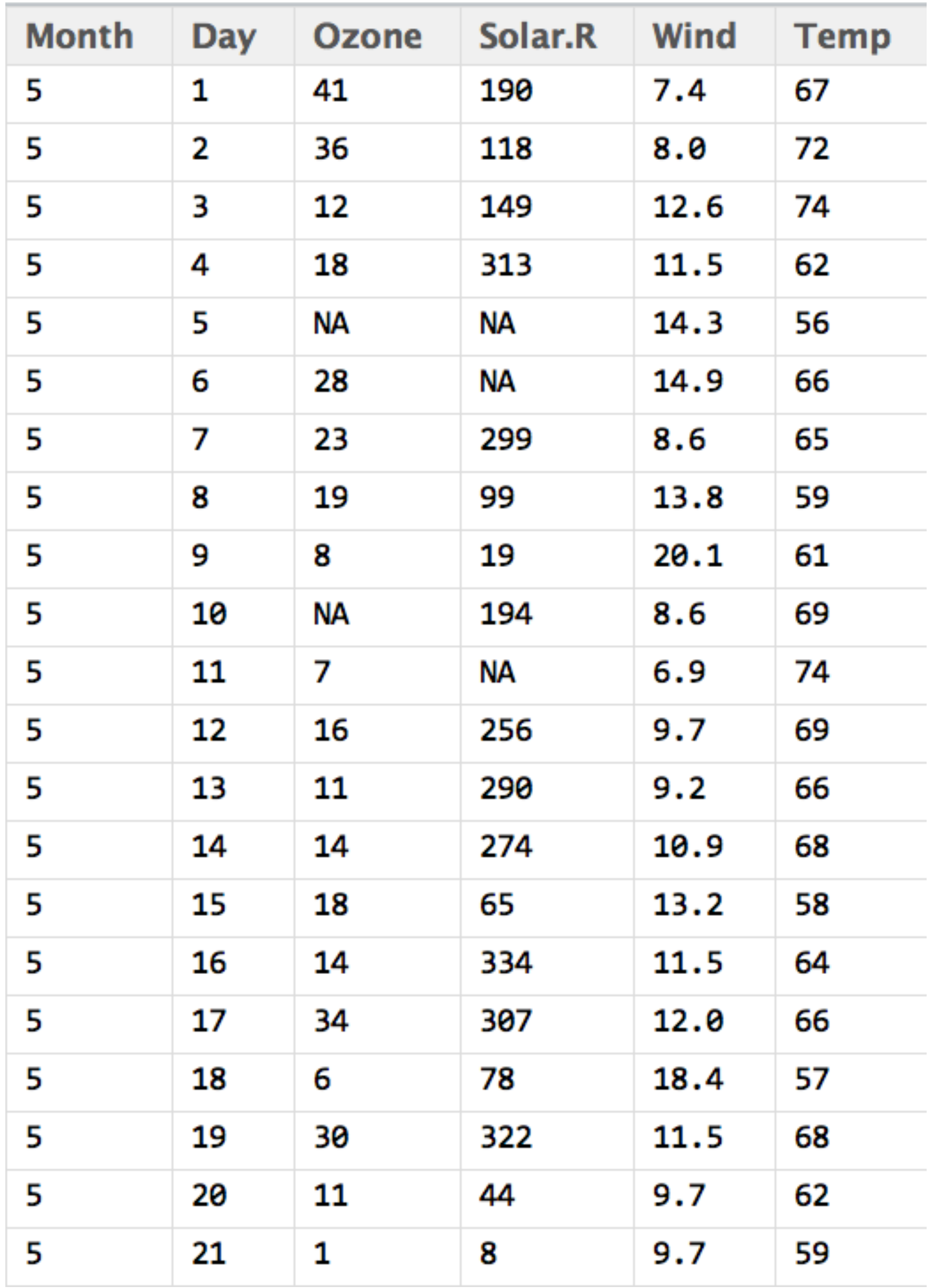
Convert back to wide format



```
df2 <- df1 %>%
  spread(variable,value)
```

Identifiers

Variables



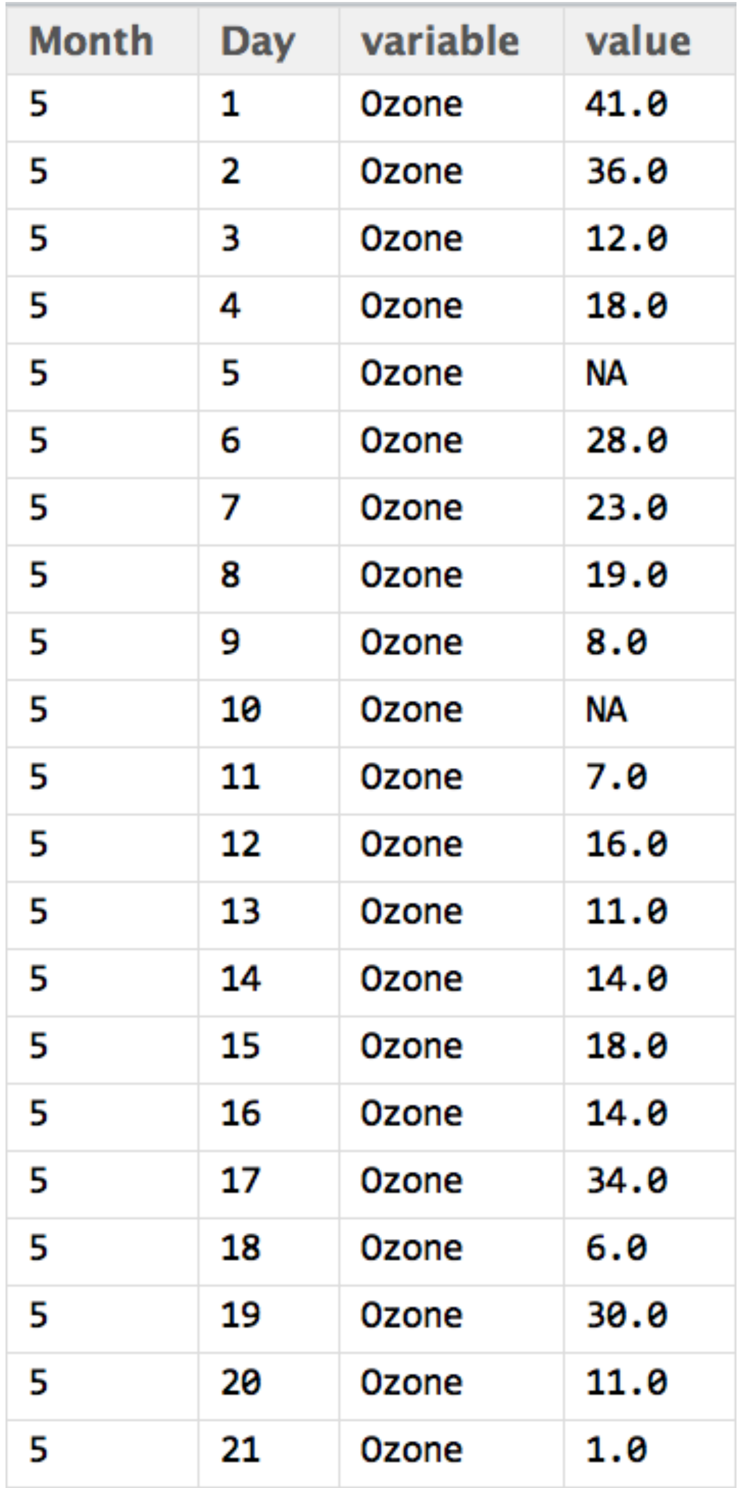
Month	Day	Ozone	Solar.R	Wind	Temp
5	1	41	190	7.4	67
5	2	36	118	8.0	72
5	3	12	149	12.6	74
5	4	18	313	11.5	62
5	5	NA	NA	14.3	56
5	6	28	NA	14.9	66
5	7	23	299	8.6	65
5	8	19	99	13.8	59
5	9	8	19	20.1	61
5	10	NA	194	8.6	69
5	11	7	NA	6.9	74
5	12	16	256	9.7	69
5	13	11	290	9.2	66
5	14	14	274	10.9	68
5	15	18	65	13.2	58
5	16	14	334	11.5	64
5	17	34	307	12.0	66
5	18	6	78	18.4	57
5	19	30	322	11.5	68
5	20	11	44	9.7	62
5	21	1	8	9.7	59

Why do we care about long vs. wide data?

Easy grouping leads to easy analysis

Easy summarization and plotting

Don’t forget `unique()`



Month	Day	variable	value
5	1	Ozone	41.0
5	2	Ozone	36.0
5	3	Ozone	12.0
5	4	Ozone	18.0
5	5	Ozone	NA
5	6	Ozone	28.0
5	7	Ozone	23.0
5	8	Ozone	19.0
5	9	Ozone	8.0
5	10	Ozone	NA
5	11	Ozone	7.0
5	12	Ozone	16.0
5	13	Ozone	11.0
5	14	Ozone	14.0
5	15	Ozone	18.0
5	16	Ozone	14.0
5	17	Ozone	34.0
5	18	Ozone	6.0
5	19	Ozone	30.0
5	20	Ozone	11.0
5	21	Ozone	1.0

Let’s go from messy to tidy

Use `gather()` to bring together a and b columns into key-value pairs (or variable-observation pairs)

Use `spread()` to do the opposite to make long data wide again

Use `separate()` to split column data into different columns

Use `unite()` to combine column data into one column

Flow of data analysis

✓ 1. Read

✓ 2. Tidy

3. Process

4. Plot

5. Present

Why do we care about processing data?

Measurement data can be overwhelming.

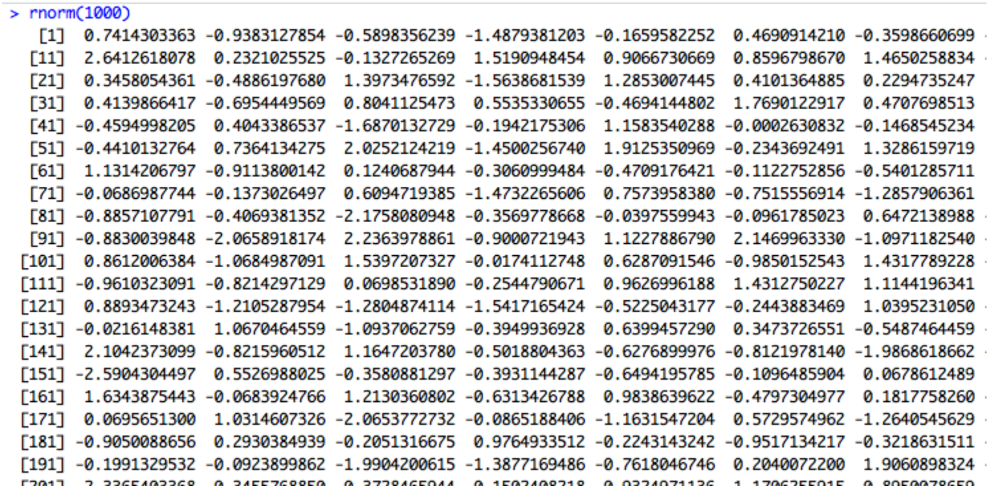
Summarize with means, medians, dispersion of data

Summaries help us think about replication and error

Sometimes we don’t care about processing data

dplyr: the grammar of data analysis

Hadley Wickham: R guru



```
> rnorm(1000)
   [1]  0.7414303363 -0.9383127854 -0.5898356239 -1.4879381203 -0.1659582252  0.4690914210 -0.3598660699
  [11]  2.6412618078  0.2321025525 -0.1327265269  1.5190948454  0.9066730669  0.8596798670  1.4650258834
  [21]  0.3458054361 -0.4886197680  1.3973476592 -1.5638681539  1.2853007445  0.4101364885  0.2294735247
  [31]  0.4139866417 -0.6954449569  0.8041125473  0.5535330655 -0.4694144802  1.7690122917  0.4707698513
  [41] -0.4594998205  0.4043386537 -1.6870132729 -0.1942175306  1.1583540288 -0.0002630832 -0.1468545234
  [51] -0.4410132764  0.7364134275  2.0252124219 -1.4500256740  1.9125350969 -0.2343692491  1.3286159719
  [61]  1.1314206797 -0.9113800142  0.1240687944 -0.3060999484 -0.4709176421 -0.1122752856 -0.5401285711
  [71] -0.0686987744 -0.1373026497  0.6094719385 -1.4732265606  0.7573958380 -0.7515556914 -1.2857906361
  [81] -0.8857107791 -0.4069381352 -2.1758080948 -0.3569778668 -0.0397559943 -0.0961785023  0.6472138988
  [91] -0.8830039848 -2.0658918174  2.2363978861 -0.9000721943  1.1227886790  2.1469963330 -1.0971182540
 [101]  0.8612006384 -1.0684987091  1.5397207327 -0.0174112748  0.6287091546 -0.9850152543  1.4317789228
 [111] -0.9610323091 -0.8214297129  0.0698531890 -0.2544790671  0.9626996188  1.4312750227  1.1144196341
 [121]  0.8893473243 -1.2105287954 -1.2804874114 -1.5417165424 -0.5225043177 -0.2443883469  1.0395231050
 [131] -0.0216148381  1.0670464559 -1.0937062759 -0.3949936928  0.6399457290  0.3473726551 -0.5487464459
 [141]  2.1042373099 -0.8215960512  1.1647203780 -0.5018804363 -0.6276899976 -0.8121978140 -1.9868618662
 [151] -2.5904304497  0.5526988025 -0.3580881297 -0.3931144287 -0.6494195785 -0.1096485904  0.0678612489
 [161]  1.6343875443 -0.0683924766  1.2130360802 -0.6313426788  0.9838639622 -0.4797304977  0.1817758260
 [171]  0.0695651300  1.0314607326 -2.0653772732 -0.0865188406 -1.1631547204  0.5729574962 -1.2640545629
 [181] -0.9050088656  0.2930384939 -0.2051316675  0.9764933512 -0.2243143242 -0.9517134217 -0.3218631511
 [191] -0.1991329532 -0.0923899862 -1.9904200615 -1.3877169486 -0.7618046746  0.2040072200  1.9060898324
 [201]  2.3365403368  0.3455768850  0.3728465944  0.1503408218  0.0334071136  1.1706355015  0.8050078659
```

Data

arrange **select**

mutate **filter**

summarise

Simple operations

Simple, efficient storage, fast algorithms, database integration

dplyr: simple verbs

select	select columns to keep
arrange	arrange the order of rows
filter	filter rows to keep
mutate	mutate columns into new columns
group_by	group data by specific column(s)
summarise	summarize data by group

select select columns to keep

```
> head(nut)

> r.nut <- dplyr::select(nut, date, Calories)

> head(r.nut)
```

arrange arrange the order of rows

```
> head(r.nut)

> r.nut <- dplyr::arrange(r.nut, protein)

> head(r.nut)
```

mutate mutate columns into new columns

```
> head(r.nut)

> r.nut <- dplyr::mutate(r.nut,
         total_fat = `Fat (g)`+`Saturated Fat`+
         `Polyunsaturated Fat`+`Monounsaturated Fat`+
         `Trans Fat`)

> head(r.nut)
```

filter filter rows to keep

```
> head(r.nut)

> r.nut <- dplyr::filter(r.nut, Meal == “Breakfast”)

> head(r.nut)
```

group_by group data by specific column(s)

summarise summarize data by group

```
> head(r.nut)

> r.nut <- dplyr::group_by(r.nut, date)

> r.nut <- dplyr::summarise(r.nut, totCal = sum(Calories))

> head(r.nut)
```

Divide and conquer

Let’s put it all together

%>%

Data tidying and processing time

https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Flow of data analysis

✓ 1. Read

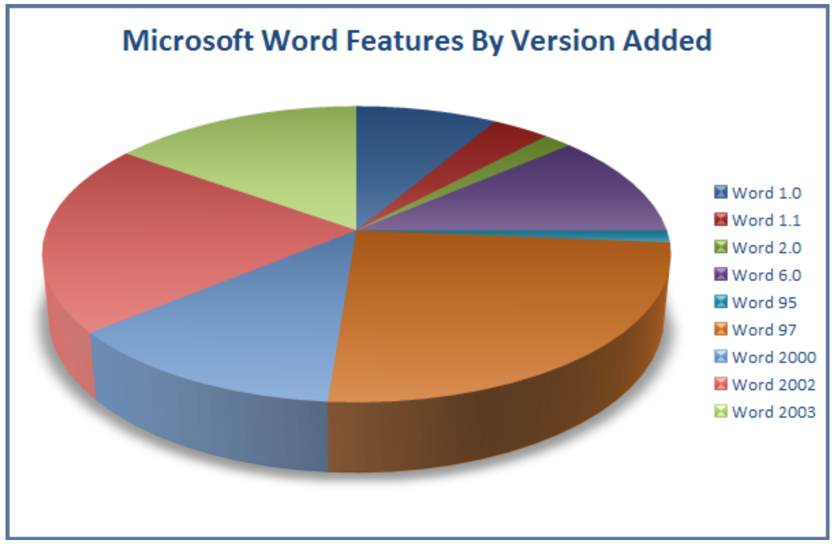
✓ 2. Tidy

✓ 3. Process

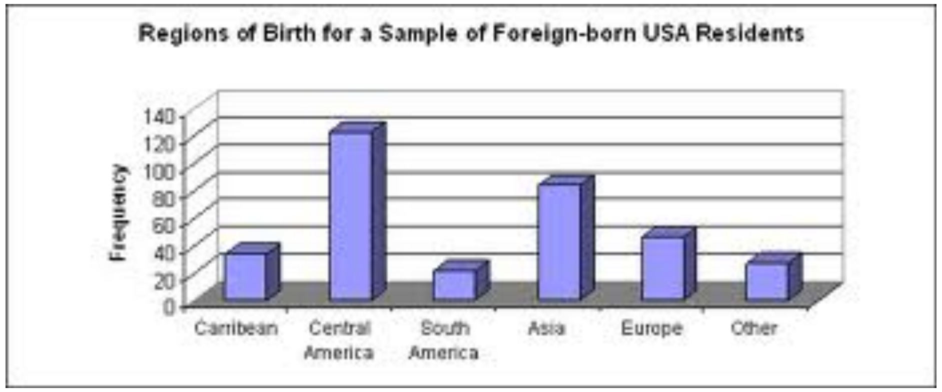
4. Plot

5. Present

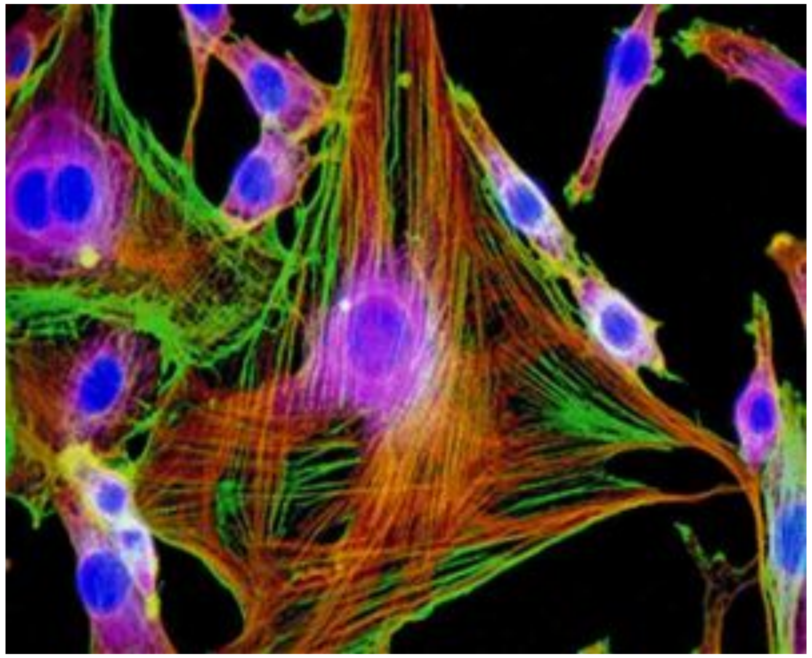
Plots should make data obvious and beautiful



Pie charts are impossible to interpret



Bar charts hide data



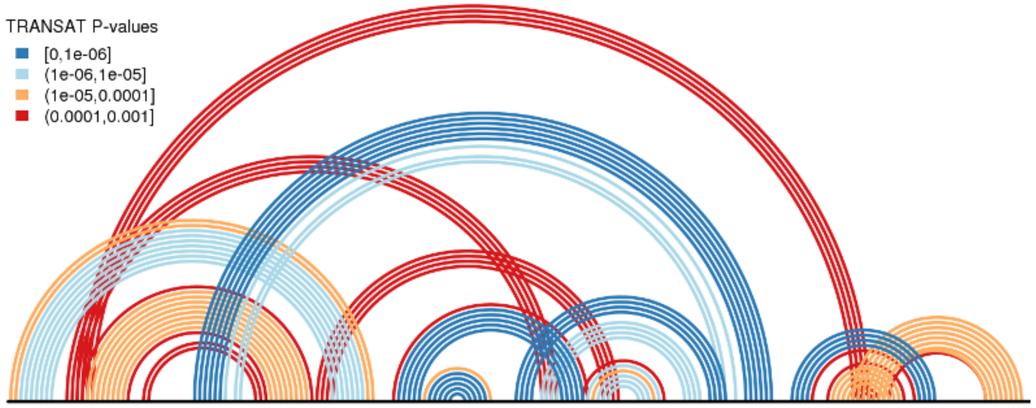
Beautiful, but…have to believe investigator on numbers, not quantitative, and hard to project

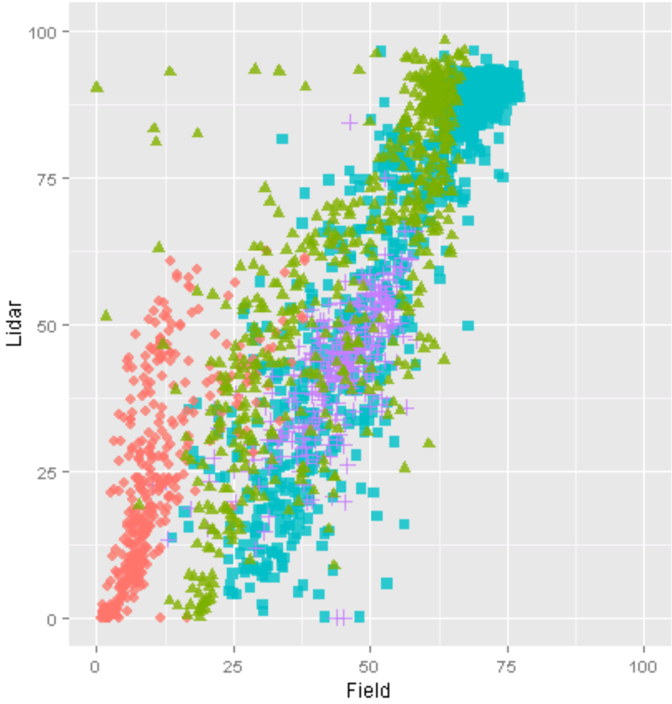
http://www.badfigure.org/

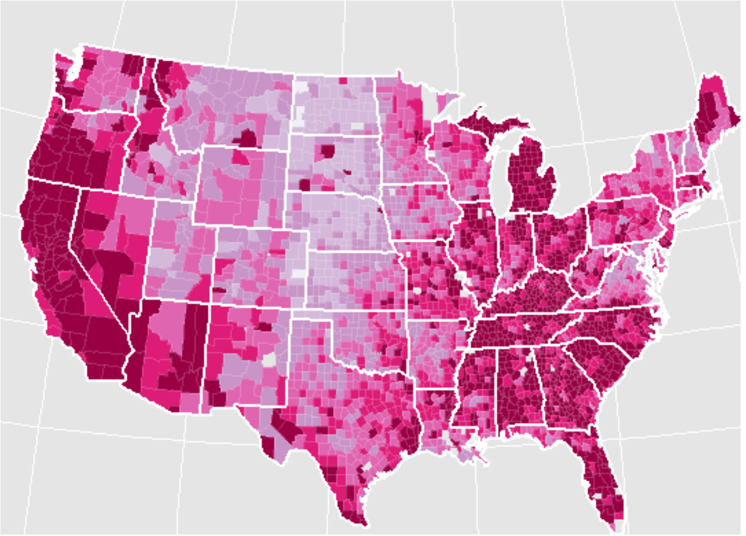
ggplot2: the grammar of graphics

Hadley Wickham: R guru

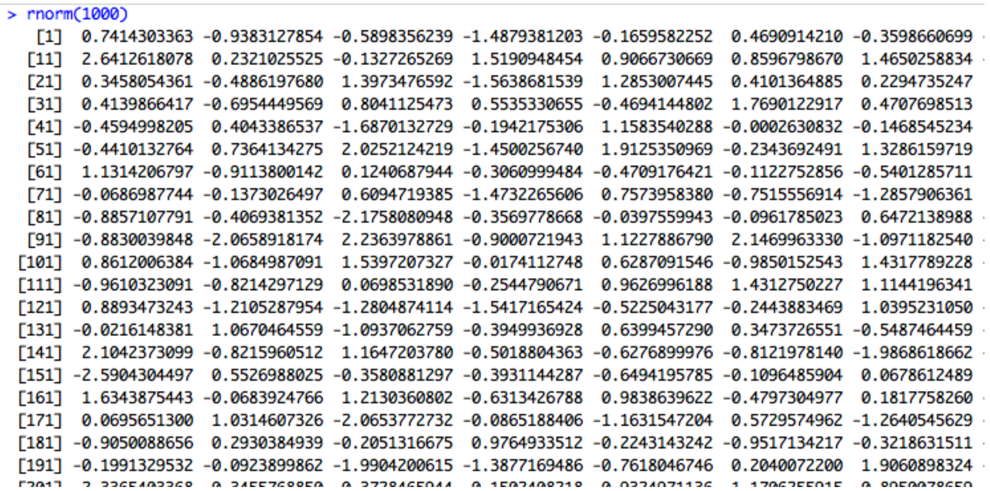
Edward Tufte: graphics guru







What is a grammar?



```
> rnorm(1000)
   [1]  0.7414303363 -0.9383127854 -0.5898356239 -1.4879381203 -0.1659582252  0.4690914210 -0.3598660699
  [11]  2.6412618078  0.2321025525 -0.1327265269  1.5190948454  0.9066730669  0.8596798670  1.4650258834
  [21]  0.3458054361 -0.4886197680  1.3973476592 -1.5638681539  1.2853007445  0.4101364885  0.2294735247
  [31]  0.4139866417 -0.6954449569  0.8041125473  0.5535330655 -0.4694144802  1.7690122917  0.4707698513
  [41] -0.4594998205  0.4043386537 -1.6870132729 -0.1942175306  1.1583540288 -0.0002630832 -0.1468545234
  [51] -0.4410132764  0.7364134275  2.0252124219 -1.4500256740  1.9125350969 -0.2343692491  1.3286159719
  [61]  1.1314206797 -0.9113800142  0.1240687944 -0.3060999484 -0.4709176421 -0.1122752856 -0.5401285711
  [71] -0.0686987744 -0.1373026497  0.6094719385 -1.4732265606  0.7573958380 -0.7515556914 -1.2857906361
  [81] -0.8857107791 -0.4069381352 -2.1758080948 -0.3569778668 -0.0397559943 -0.0961785023  0.6472138988
  [91] -0.8830039848 -2.0658918174  2.2363978861 -0.9000721943  1.1227886790  2.1469963330 -1.0971182540
 [101]  0.8612006384 -1.0684987091  1.5397207327 -0.0174112748  0.6287091546 -0.9850152543  1.4317789228
 [111] -0.9610323091 -0.8214297129  0.0698531890 -0.2544790671  0.9626996188  1.4312750227  1.1144196341
 [121]  0.8893473243 -1.2105287954 -1.2804874114 -1.5417165424 -0.5225043177 -0.2443883469  1.0395231050
 [131] -0.0216148381  1.0670464559 -1.0937062759 -0.3949936928  0.6399457290  0.3473726551 -0.5487464459
 [141]  2.1042373099 -0.8215960512  1.1647203780 -0.5018804363 -0.6276899976 -0.8121978140 -1.9868618662
 [151] -2.5904304497  0.5526988025 -0.3580881297 -0.3931144287 -0.6494195785 -0.1096485904  0.0678612489
 [161]  1.6343875443 -0.0683924766  1.2130360802 -0.6313426788  0.9838639622 -0.4797304977  0.1817758260
 [171]  0.0695651300  1.0314607326 -2.0653772732 -0.0865188406 -1.1631547204  0.5729574962 -1.2640545629
 [181] -0.9050088656  0.2930384939 -0.2051316675  0.9764933512 -0.2243143242 -0.9517134217 -0.3218631511
 [191] -0.1991329532 -0.0923899862 -1.9904200615 -1.3877169486 -0.7618046746  0.2040072200  1.9060898324
```

Data

x y z

color

Aesthetics

points boxplot

line

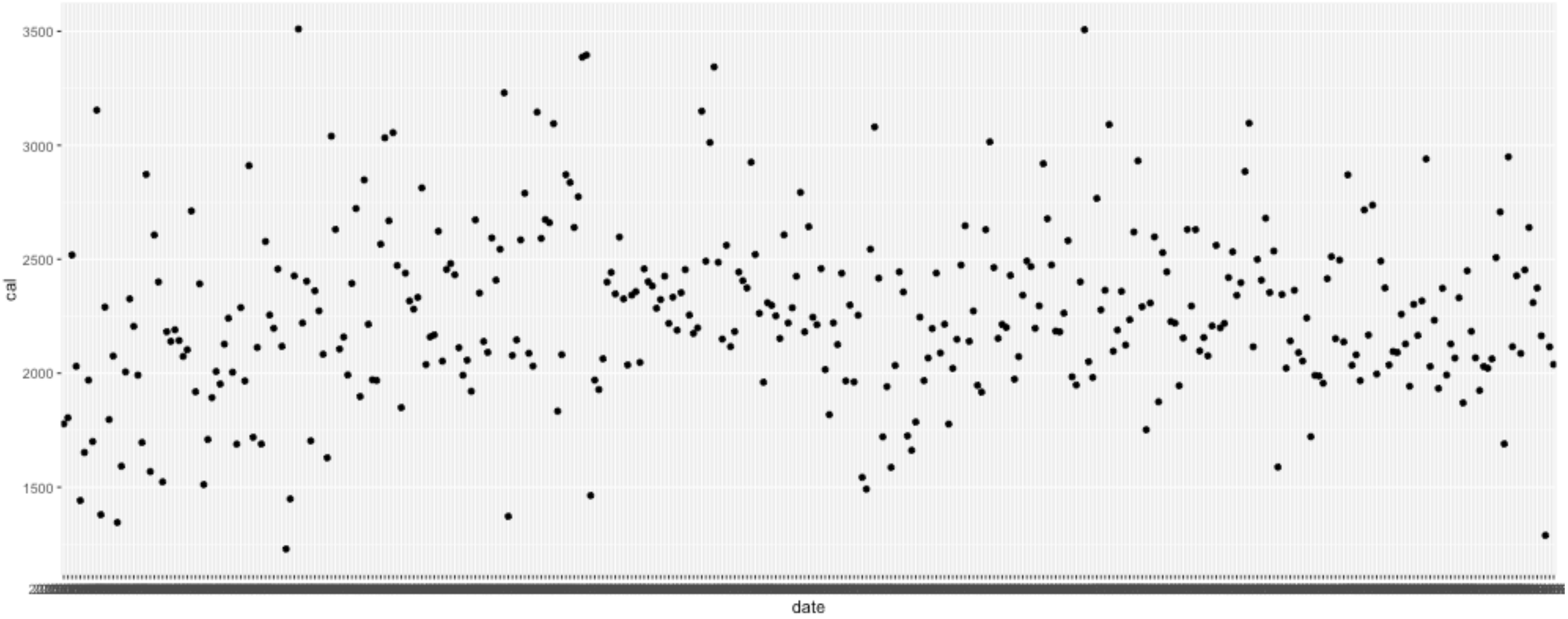
Geometric objects

http://ggplot2.org

Simple ggplot2 example

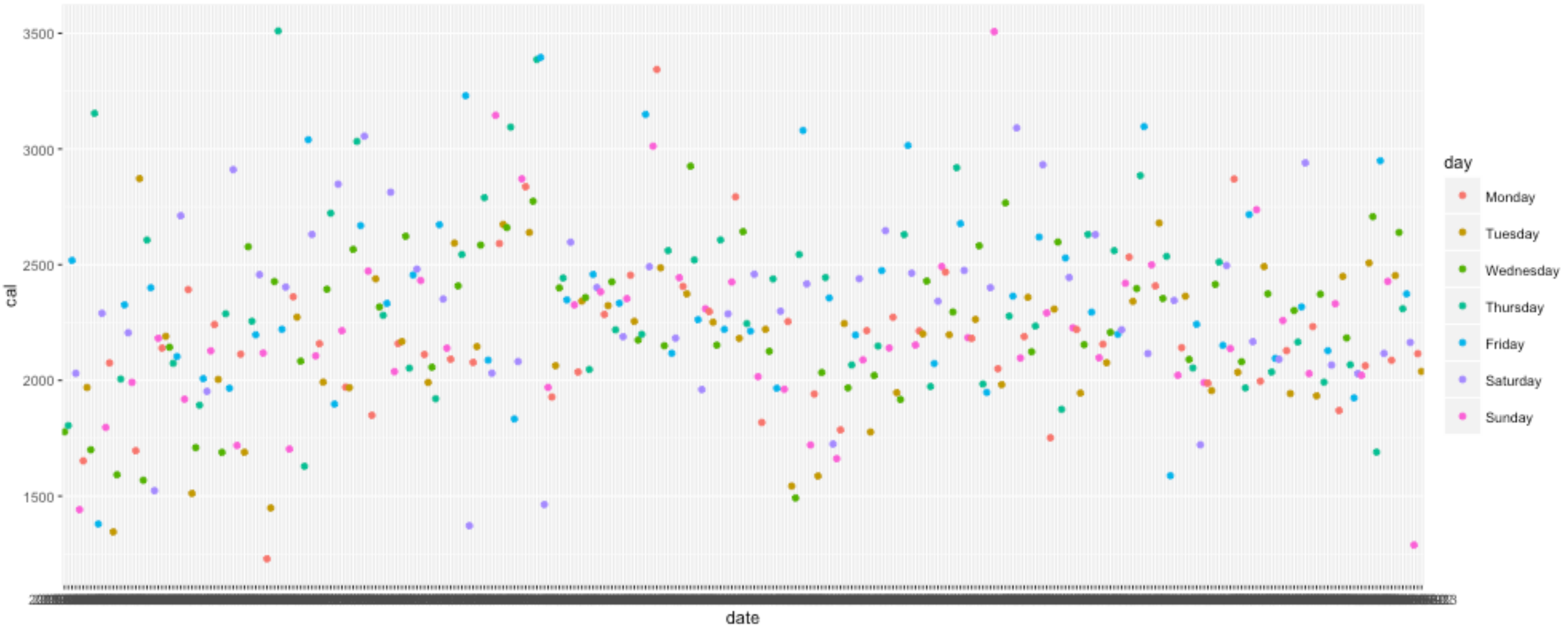
```
ggplot(data=proc.nut) + aes(x=date, y=cal) + geom_point()
```

Data **Aesthetics** **Geometric objects**



Let’s add some aesthetics

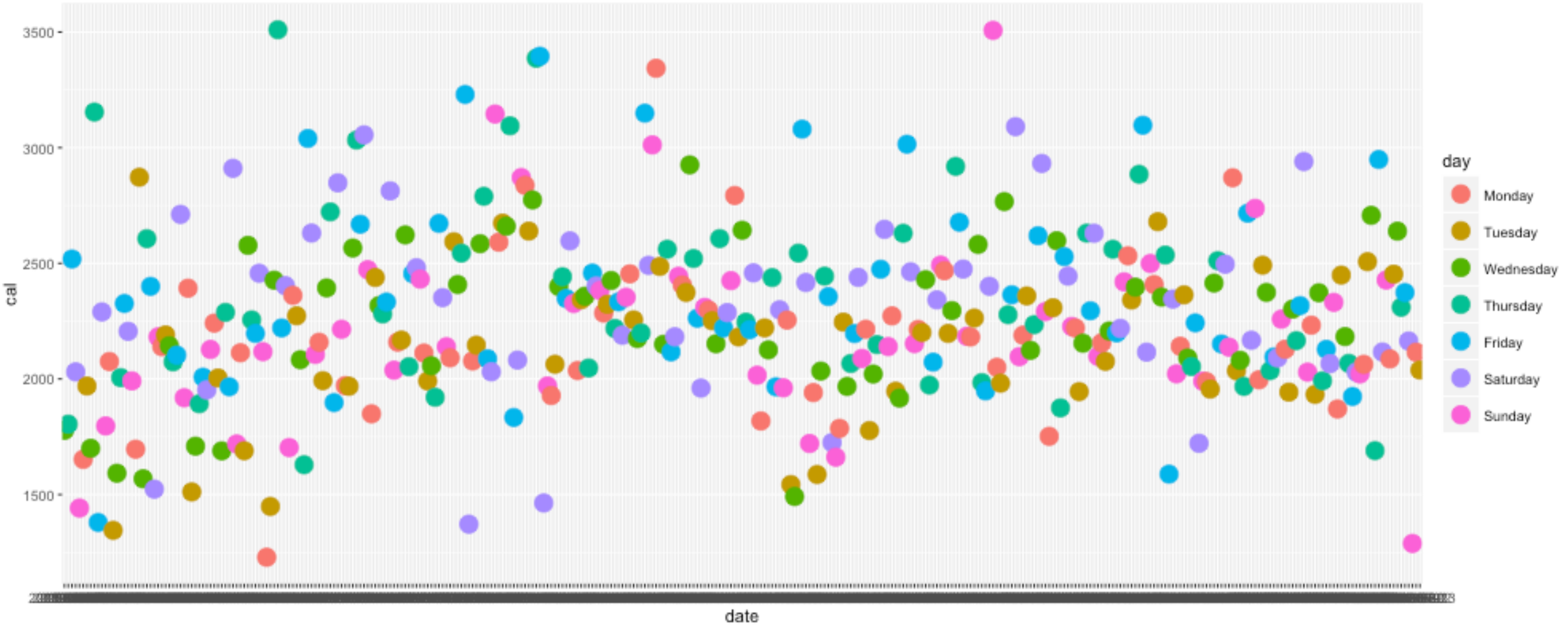
```
ggplot(data=proc.nut) + aes(x=date, y=cal, color=day) + geom_point()
```



Let’s add some aesthetics

```
ggplot(data=proc.nut) + aes(x=date, y=cal, color=day) + geom_point(size = 5)
```

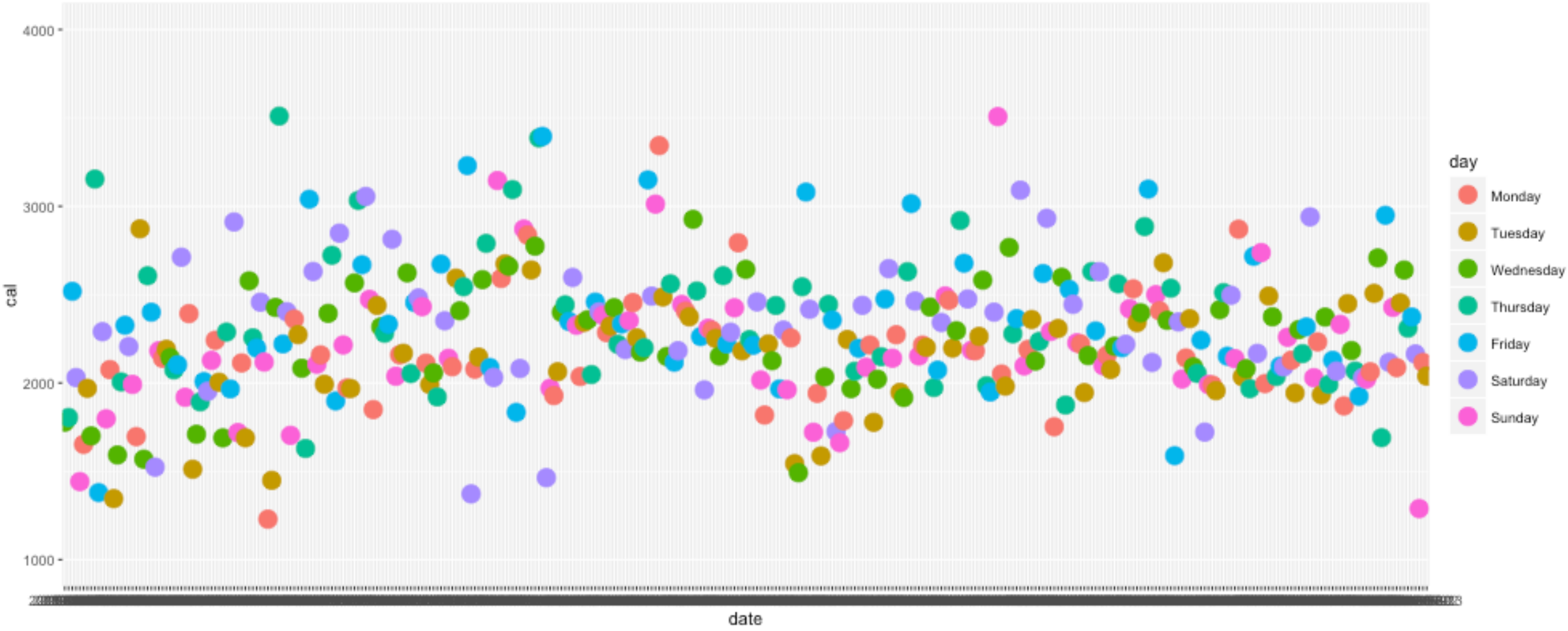
variable aesthetic **constant aesthetic**



Another aspect of grammar: coordinates

```
ggplot(data=proc.nut) + aes(x=date, y=cal, color=day) + geom_point(size=5) + ylim(1000,4000)
```

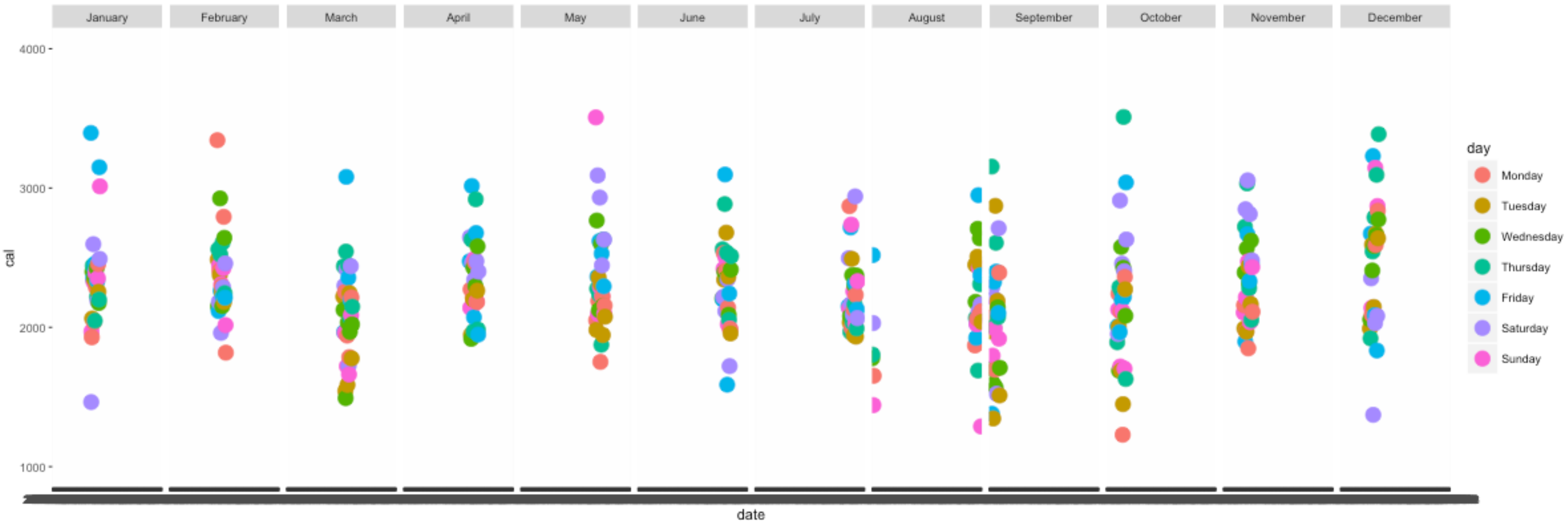
Data **Aesthetics** **Geometric objects** **Coordinates**



Another aspect of grammar: facets

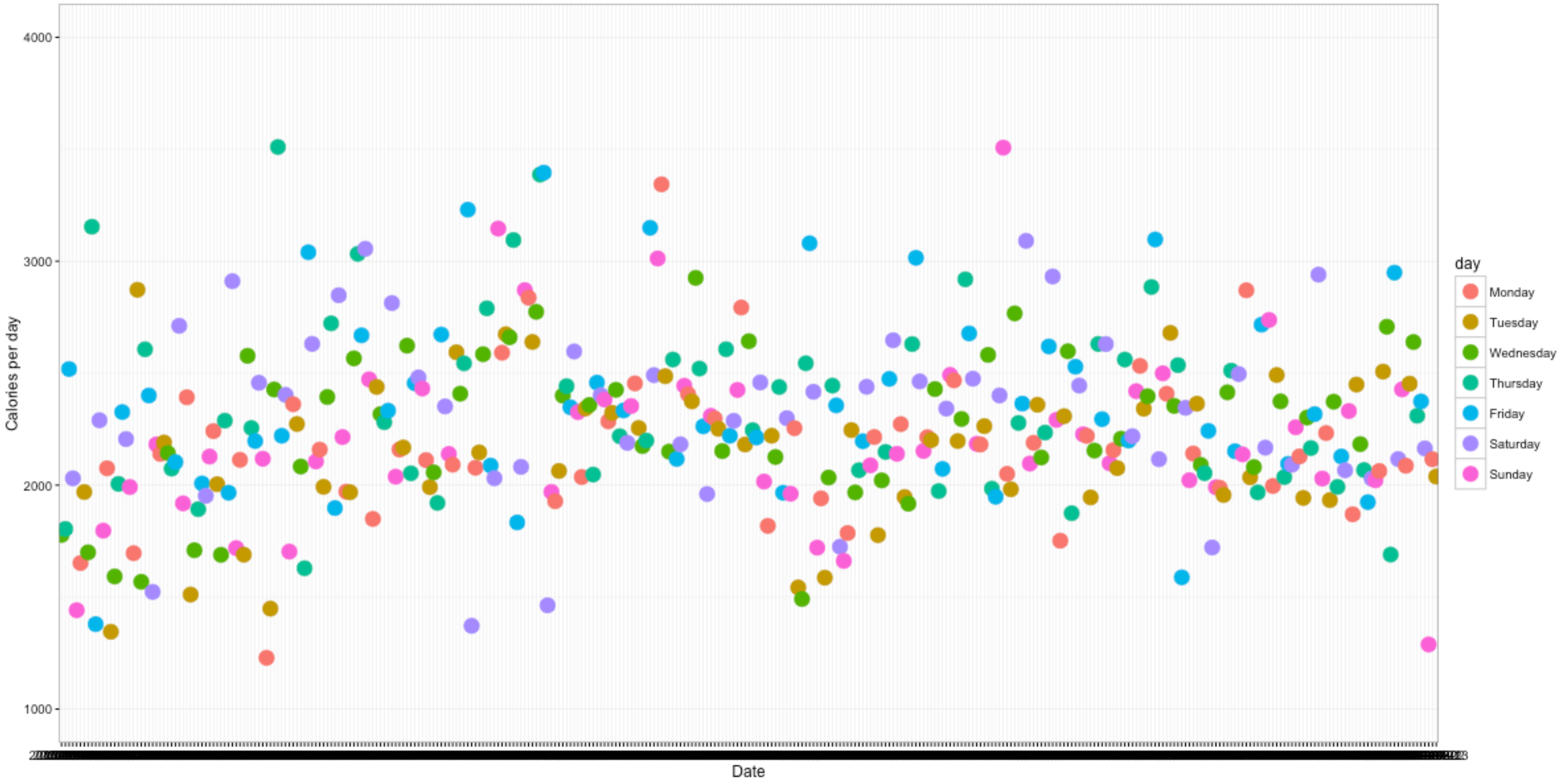
```
ggplot(data=proc.nut) + aes(x=date, y=cal, color=day) + geom_point(size=5) + ylim(1000,4000) + facet_grid(.~month)
```

Data **Aesthetics** **Geometric objects** **Coordinates** **Facets**



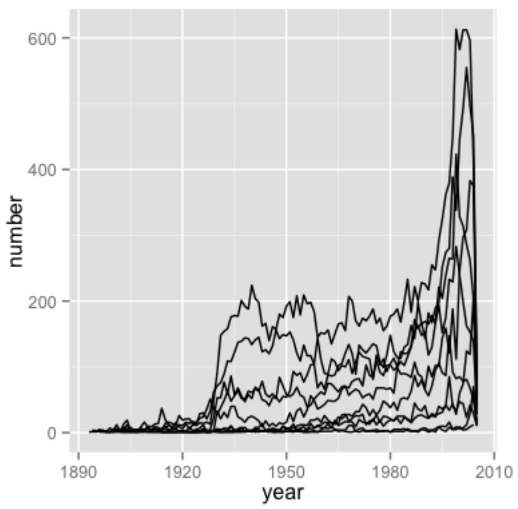
How to make the plot look prettier

```
ggplot(proc.nut) +
  aes(x = date, y=cal, color=day) +
  geom_point(size = 5) +
  ylim(1000,4000) +
  labs(x="Date", y="Calories per day") +
  theme_bw()
```

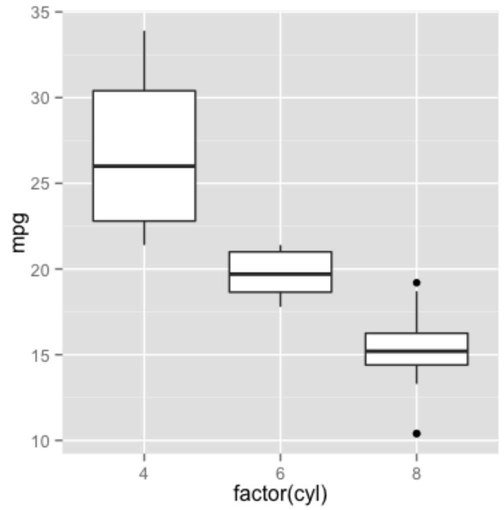


http://ggplot2.org

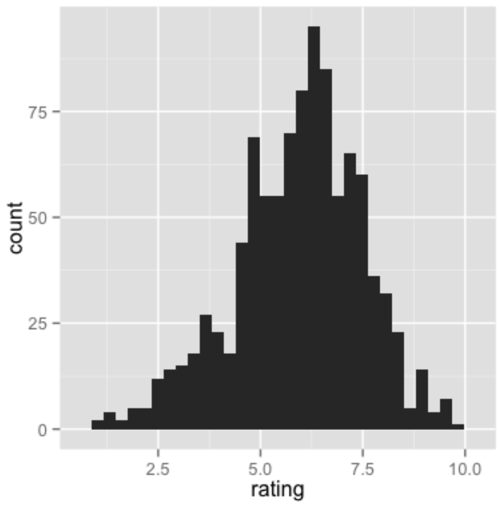
Other geometric objects



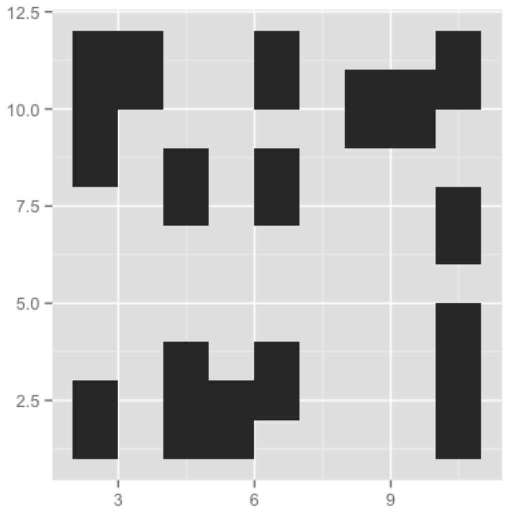
`geom_line`



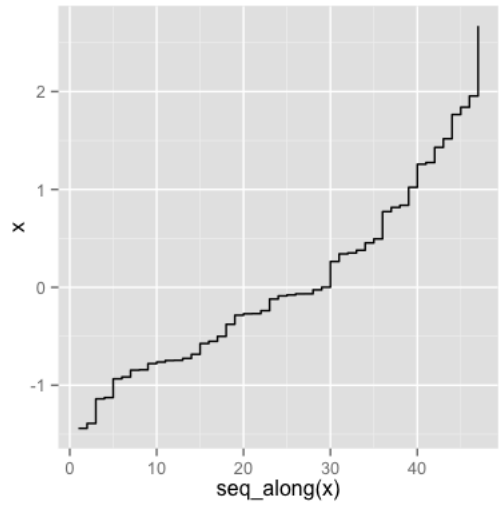
`geom_boxplot`



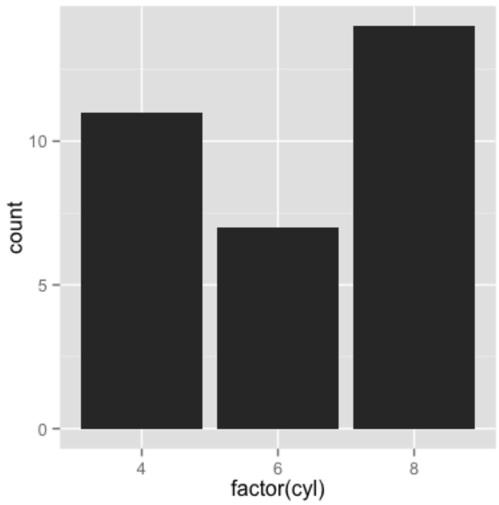
`geom_histogram`



`geom_rect`



`geom_step`



`geom_bar`

http://docs.ggplot2.org/current/

As always, ggplot2 is not the only way to plot data in R. Keep up with current trends.

http://flowingdata.com/

Data plotting time

https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Flow of data analysis

✓ 1. Read

✓ 2. Tidy

✓ 3. Process

✓ 4. Plot

5. Present

What to do when you are lost?

Google

stackoverflow

http://www.r-bloggers.com

http://gettinggeneticsdone.blogspot.com

http://rseek.org

Day #2 Homework

1. Think about your data analysis
2. Work on tidying, processing, and plotting your data