Course outline

Day #1: Basic command line interface and reproducible research

Day #2: Command line tools and sequence alignment, Intro to R and RStudio

Day #3: R, data manipulation, and plotting

Day #2 R/RStudio wrap-up

Read in data = `read.delim()`

Structure of data = `View(), str(), summary()`

Data types = data frames, vectors, lists

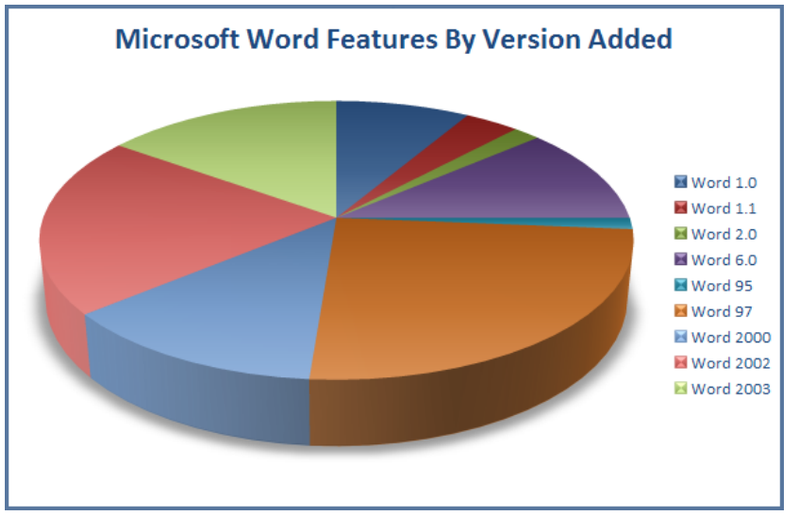
Markdown = .Rmd files in RStudio, knitr

Start an .Rmd file for today's work

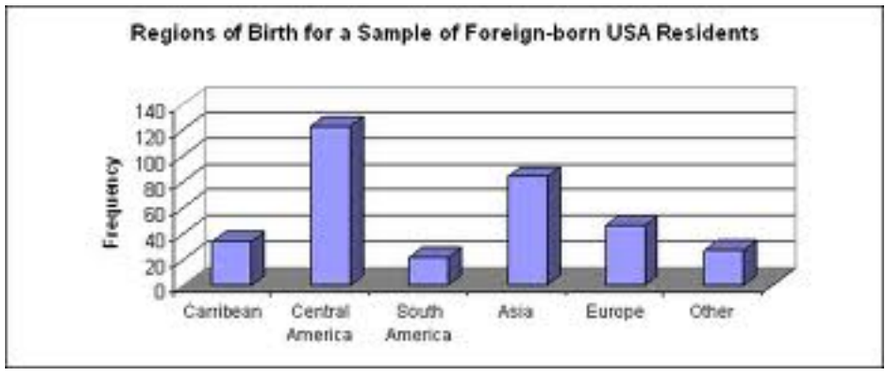
1. Enter a level three header: Analysis of sorter data
2. Write whatever description you like beneath it
3. Above the first code chunk, make a level four header: Read in data
4. Remove existing code. Add in code for reading in data.

Layout of the Rmd file

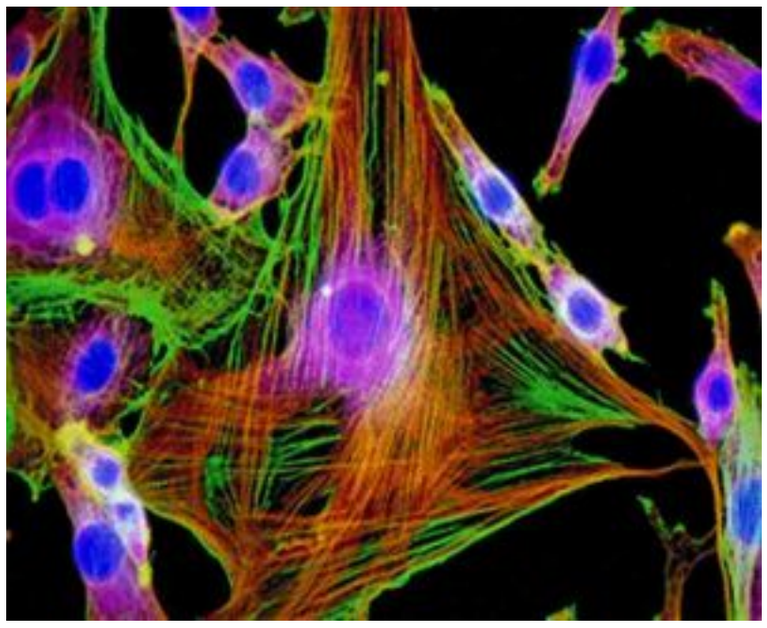
Plots should make data obvious and beautiful



Pie charts are impossible to interpret



Bar charts hide data



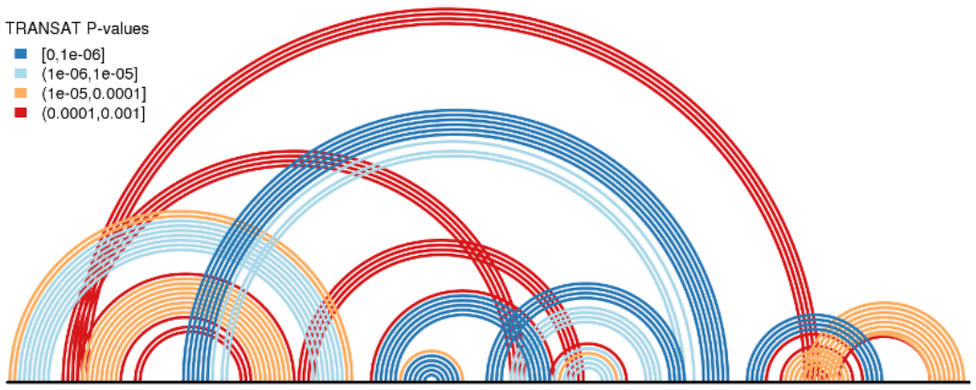
Beautiful, but…have to believe investigator on numbers, not quantitative, and hard to project

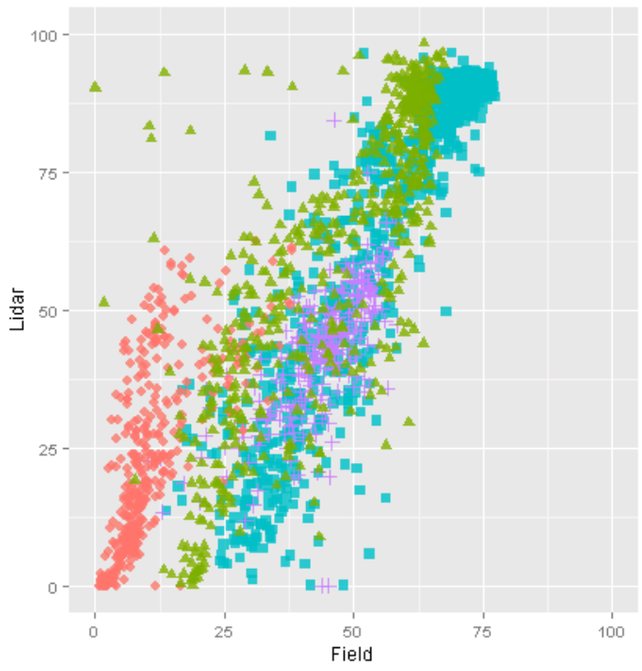
http://www.badfigure.org/

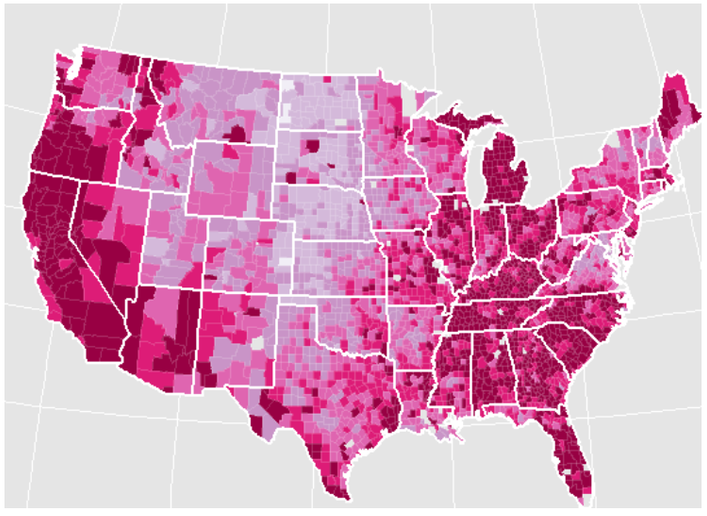
ggplot2: the grammar of graphics

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

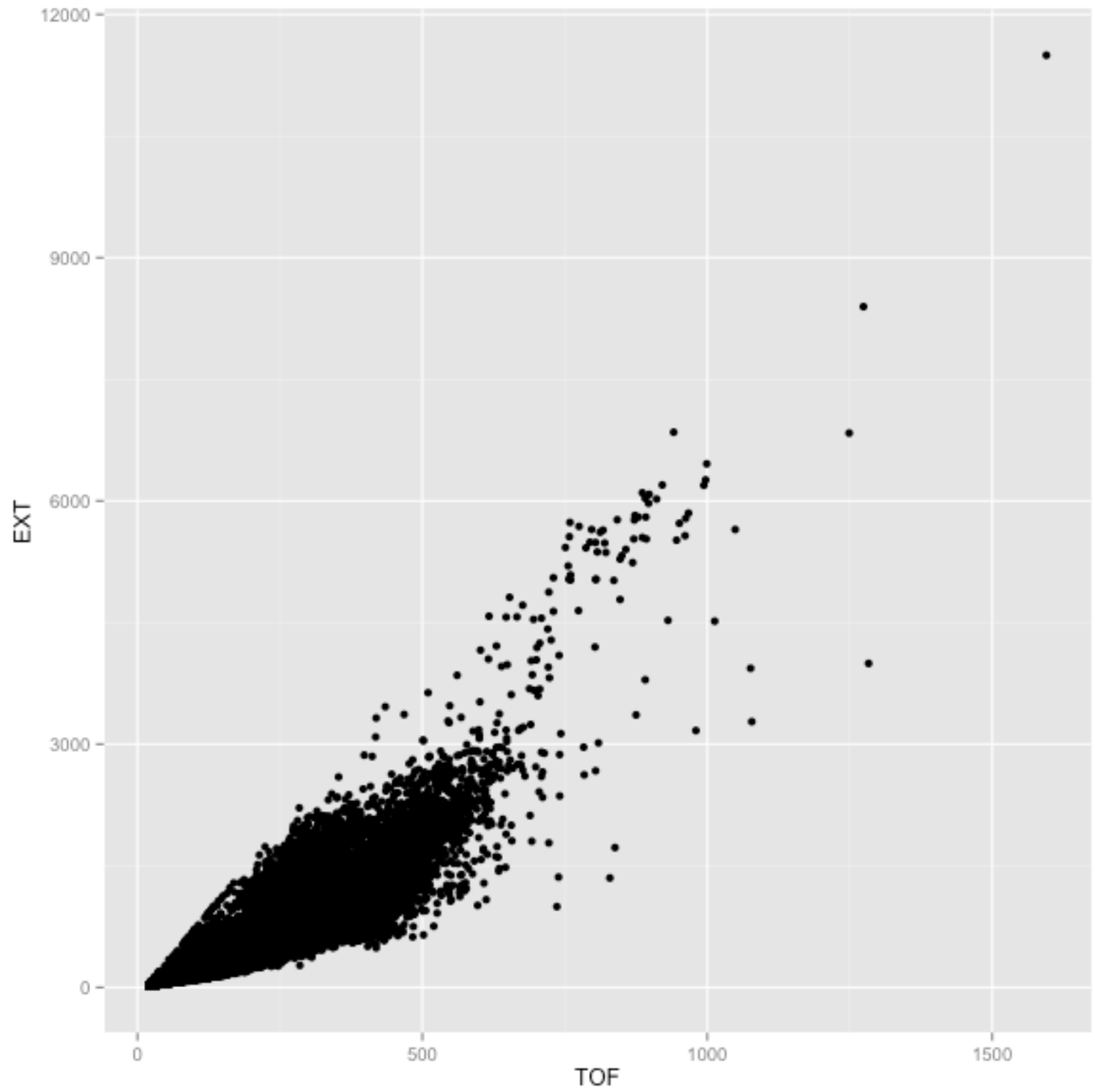
Geometric objects

http://ggplot2.org

Simple ggplot2 example

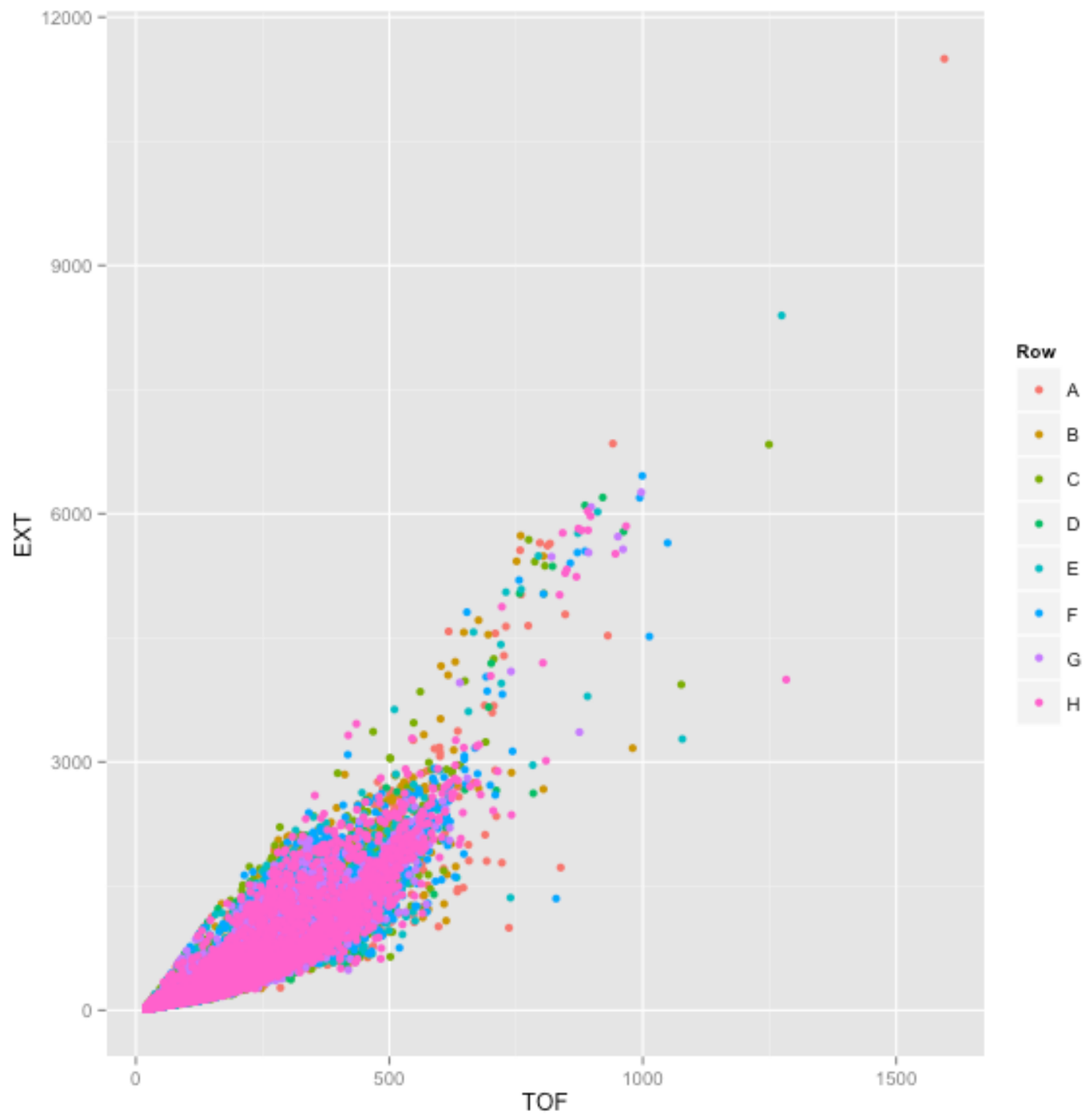
```
ggplot(data=df) + aes(x=TOF, y=EXT) + geom_point()
```

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



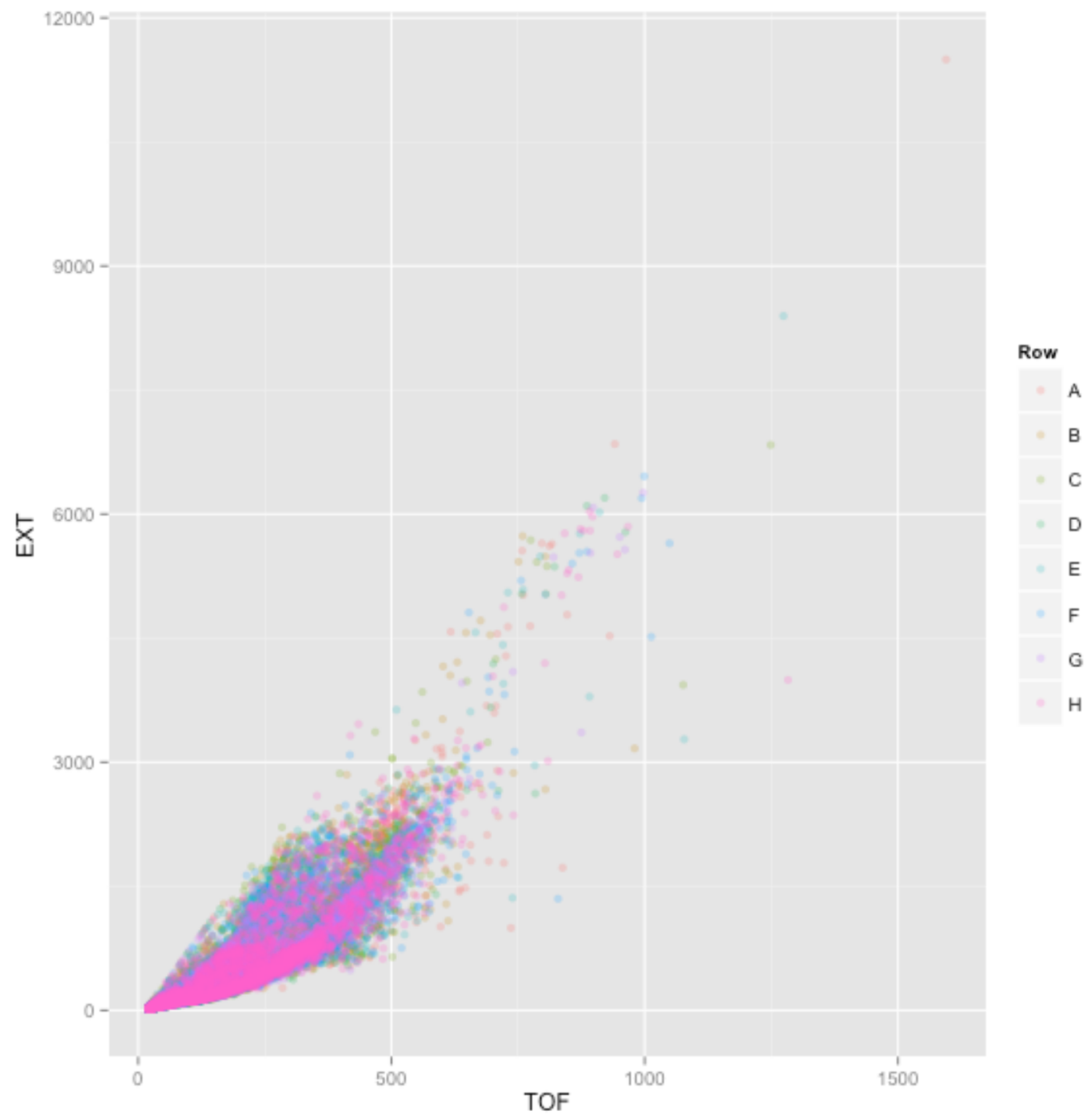
Let’s add some aesthetics

```
ggplot(data=df) + aes(x=TOF, y=EXT, color=Row) + geom_point()
```



Let’s add some aesthetics

```
ggplot(data=df) + aes(x=TOF, y=EXT, color=Row) + geom_point(alpha=0.25)
```



Aesthetics apply to data and geoms

```
ggplot(data=df) + aes(x=TOF, y=EXT, color=Row) + geom_point(alpha=0.25)
```

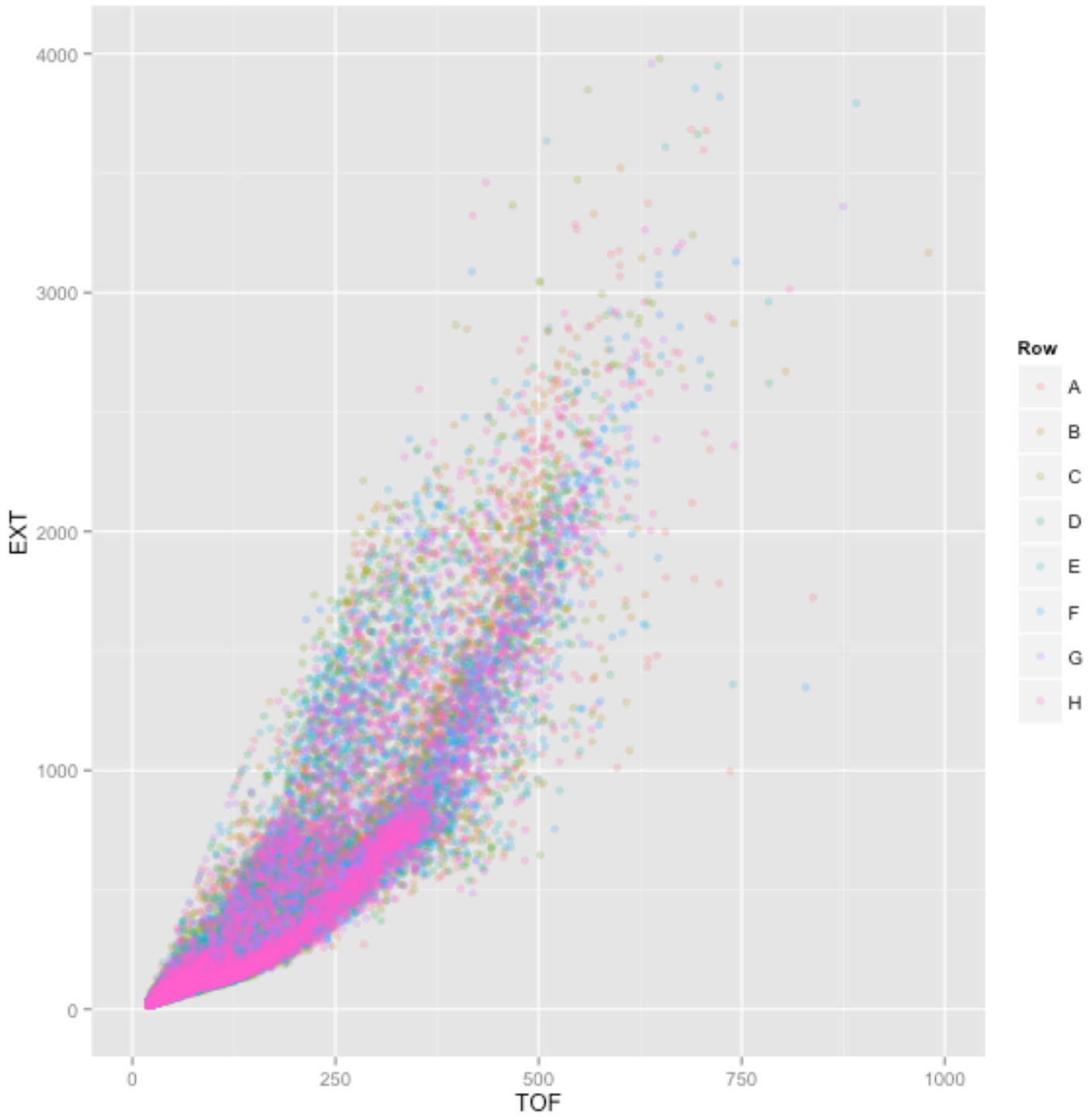
aesthetic from data **aesthetic constant**

Challenge:
Make a scatter plot with size from Column and shape = 2

Another aspect of grammar: coordinates

```
ggplot(data=df) + aes(x=TOF, y=EXT, color=Row) + geom_point(alpha=0.25) + xlim(0, 1000) + ylim(0,4000)
```

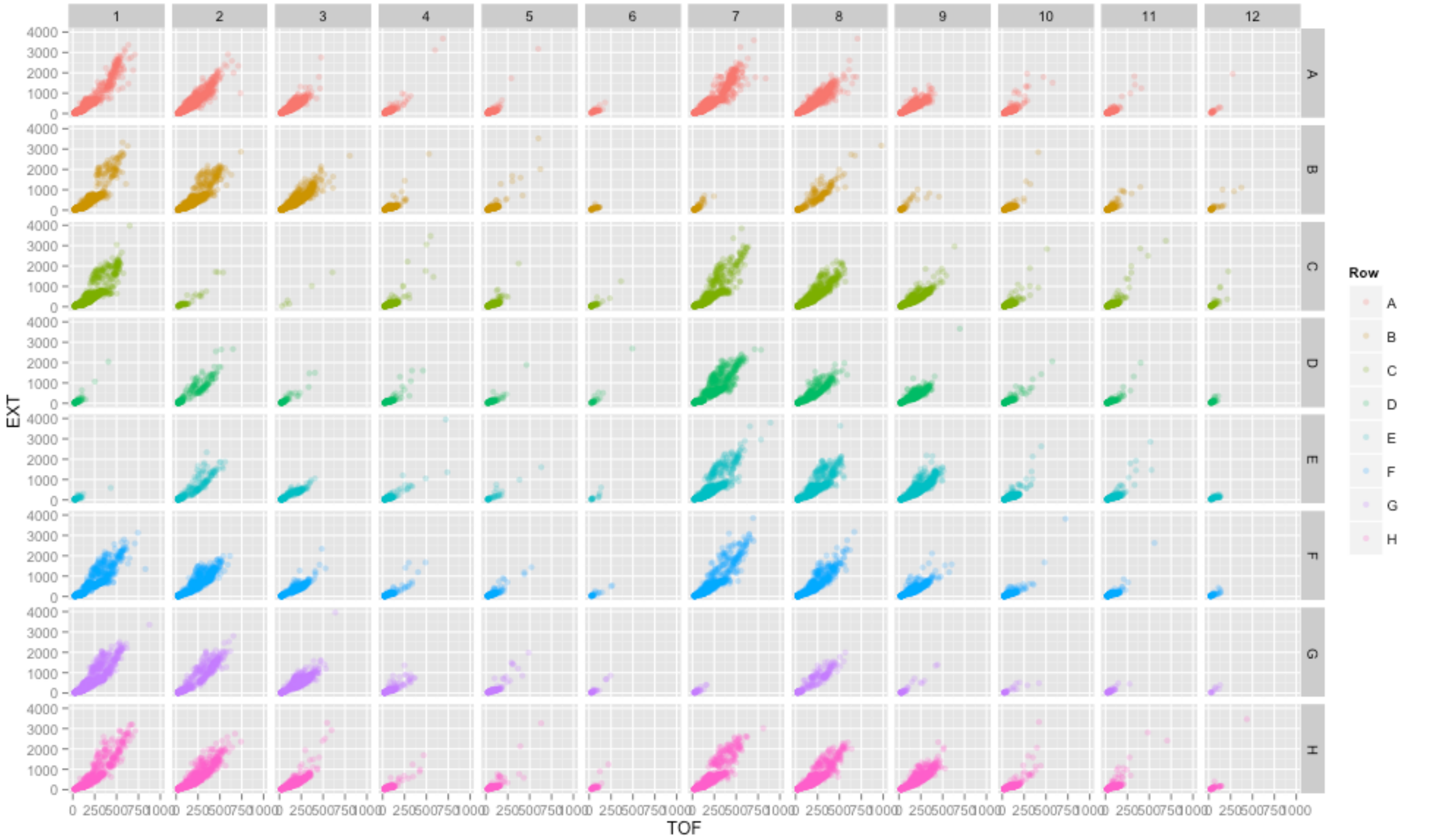
Data **Aesthetics** **Geometric objects** **Coordinates**



Another aspect of grammar: facets

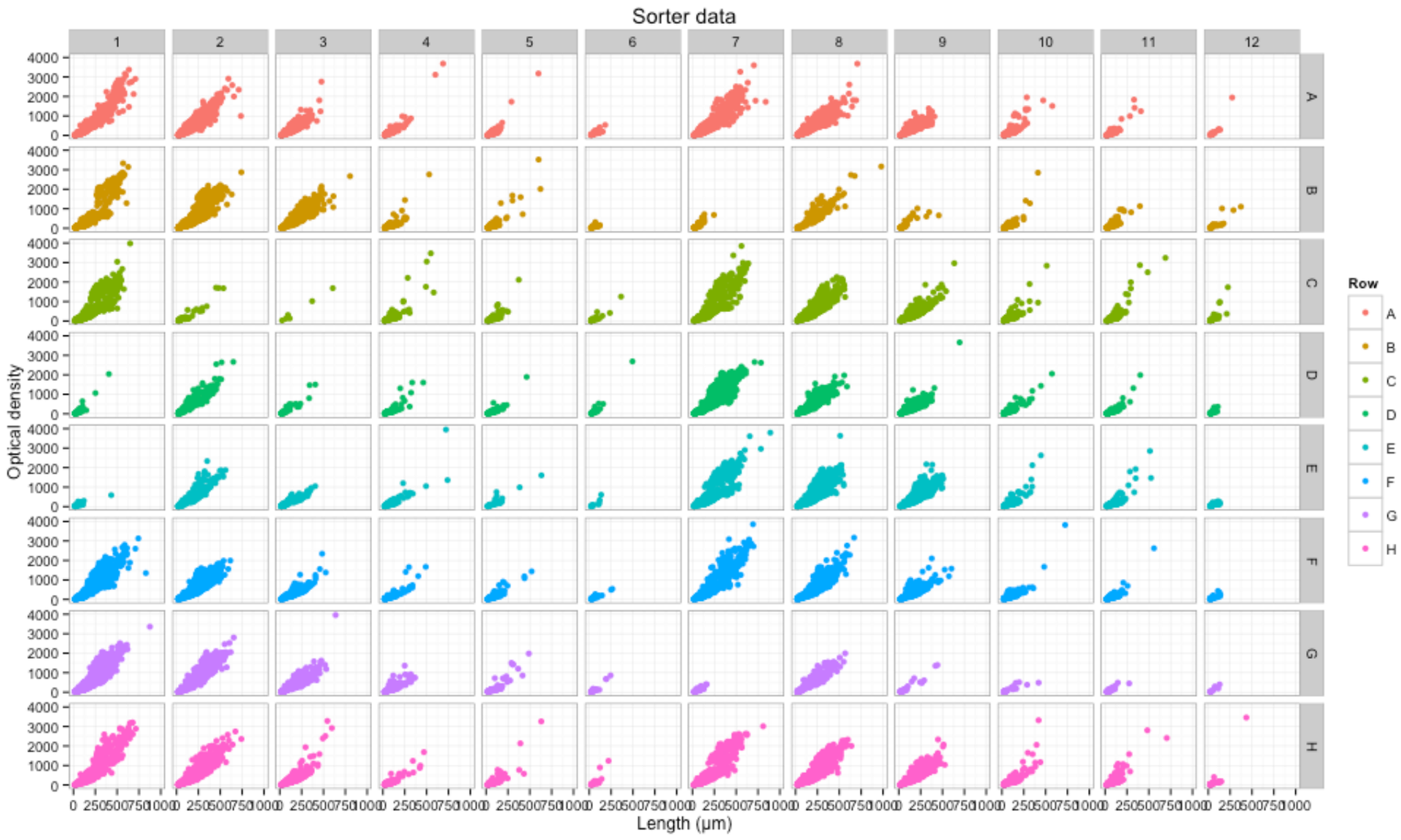
```
ggplot(data=df) + aes(x=TOF, y=EXT, color=Row) + geom_point(alpha=0.25) + xlim(0, 1000) + ylim(0,4000) + facet_grid(Row~Column)
```

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



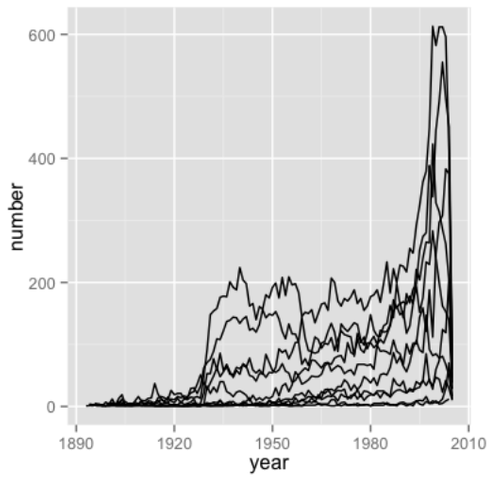
How to make the plot look prettier

```
ggplot(data=df) + aes(x=TOF, y=EXT, color=Row) +
     geom_point(alpha=0.25) +
     xlim(0, 1000) + ylim(0,4000) +
     labs(x="Length (μm)", y="Optical Density", title="Sorter Data") +
     facet_grid(Row~Column) +
     theme_bw()
```

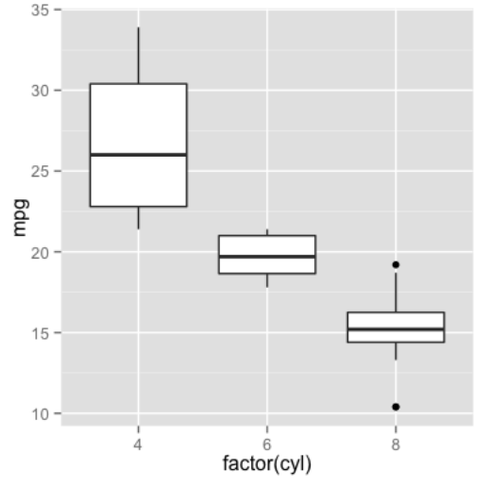


http://ggplot2.org

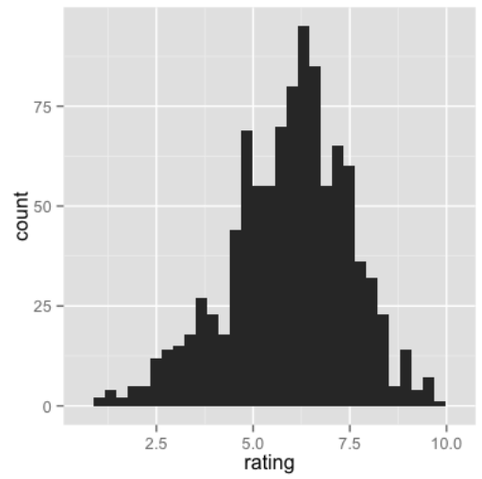
Other geometric objects



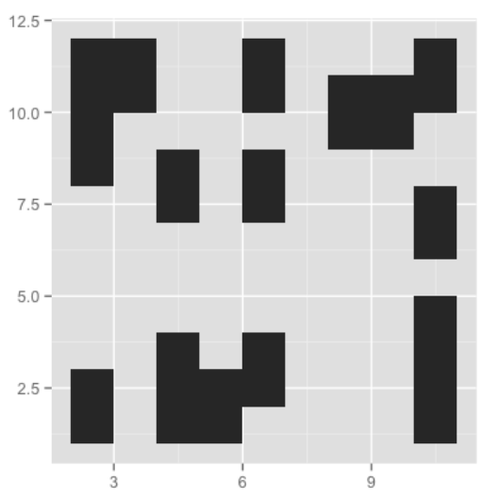
`geom_line`



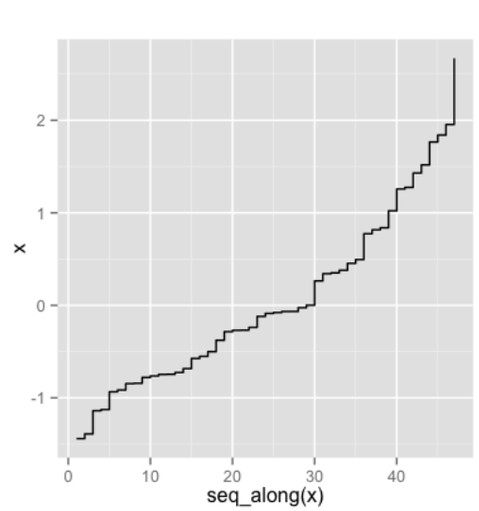
`geom_boxplot`



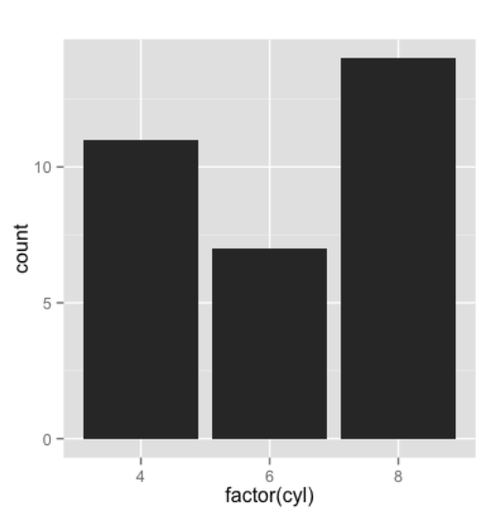
`geom_histogram`



`geom_rect`



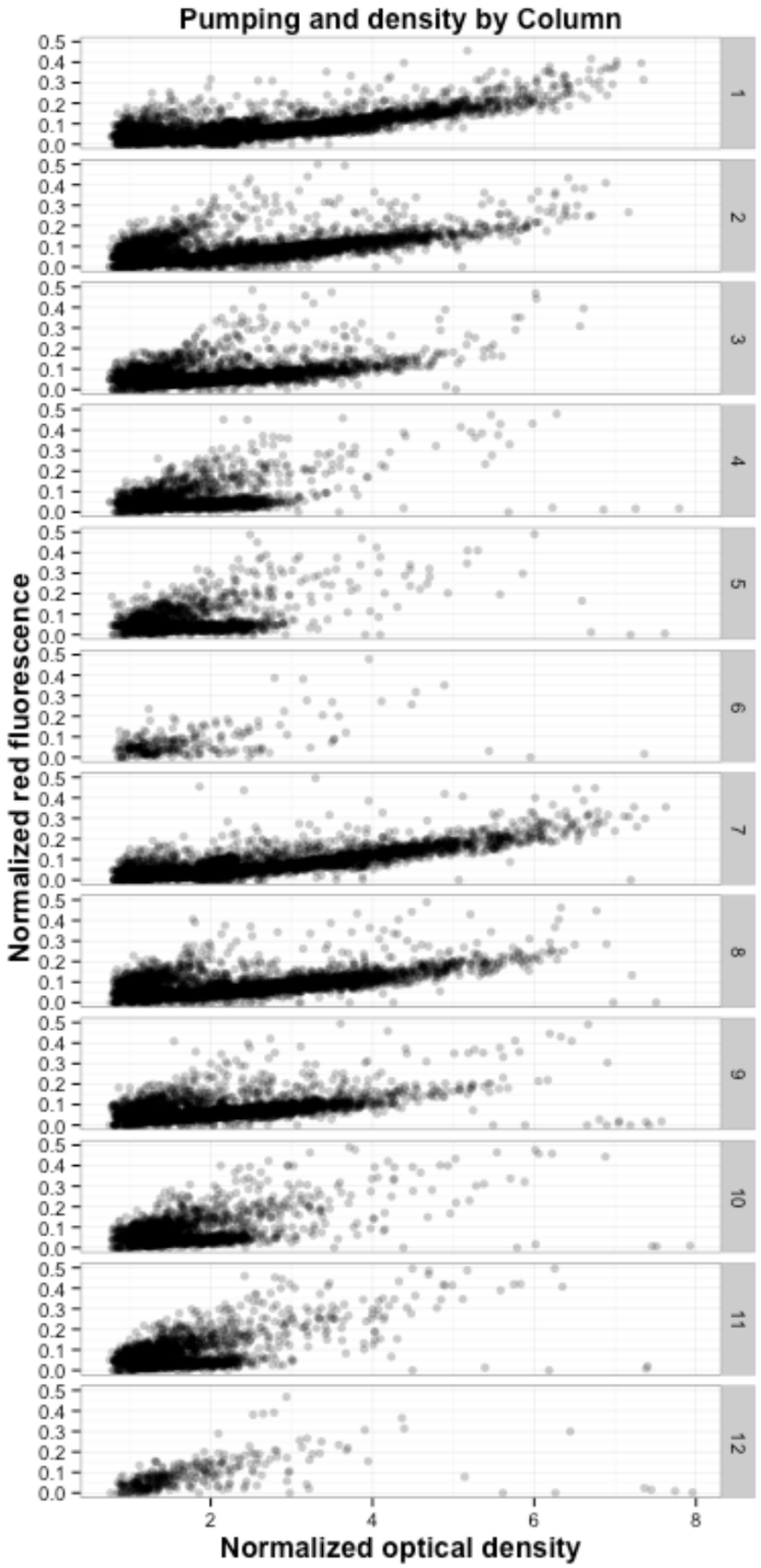
`geom_step`



`geom_bar`

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

ggplot2 Challenge



1. Make a new column in the df called norm.red, where norm.red = Red / TOF
2. Make another new column in the df called norm.EXT, where norm.EXT = EXT / TOF
3. Plot norm.EXT by norm.red as a scatterplot faceted by Column vertically
4. Add descriptive x, y, and title labels in bold
5. Limit the axes and other aesthetics to show much of the data
6. Use the black and white theme

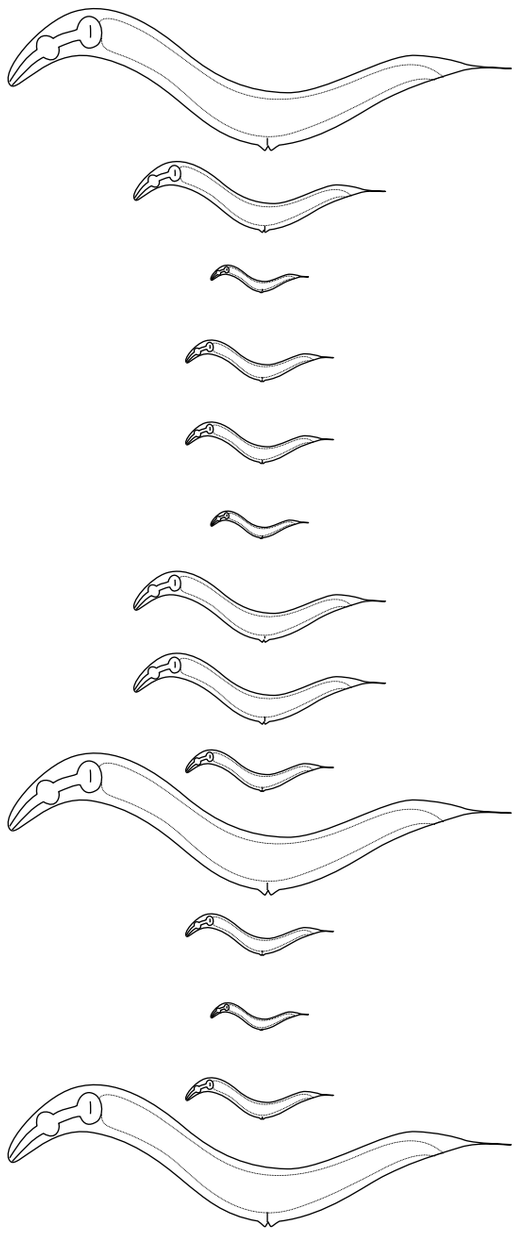
```
df$norm.red <- df$Red/df$TOF
df$norm.EXT <- df$EXT/df$TOF
```

```
ggplot(data=df) + aes(x=norm.EXT, y=norm.red) +
  geom_point(alpha=0.25) +
  ylim(0,0.5) +
  labs(x="Normalized optical density", y="Normalized red fluorescence",
       title="Pumping and density by Column") +
  facet_grid(Column~.) +
  theme_bw() +
  theme(plot.title=element_text(face="bold", size=14),
        axis.title.x=element_text(face="bold", size=14),
        axis.title.y=element_text(face="bold", size=14))
```

Factors

Factors are categorical variables that can be numeric or strings

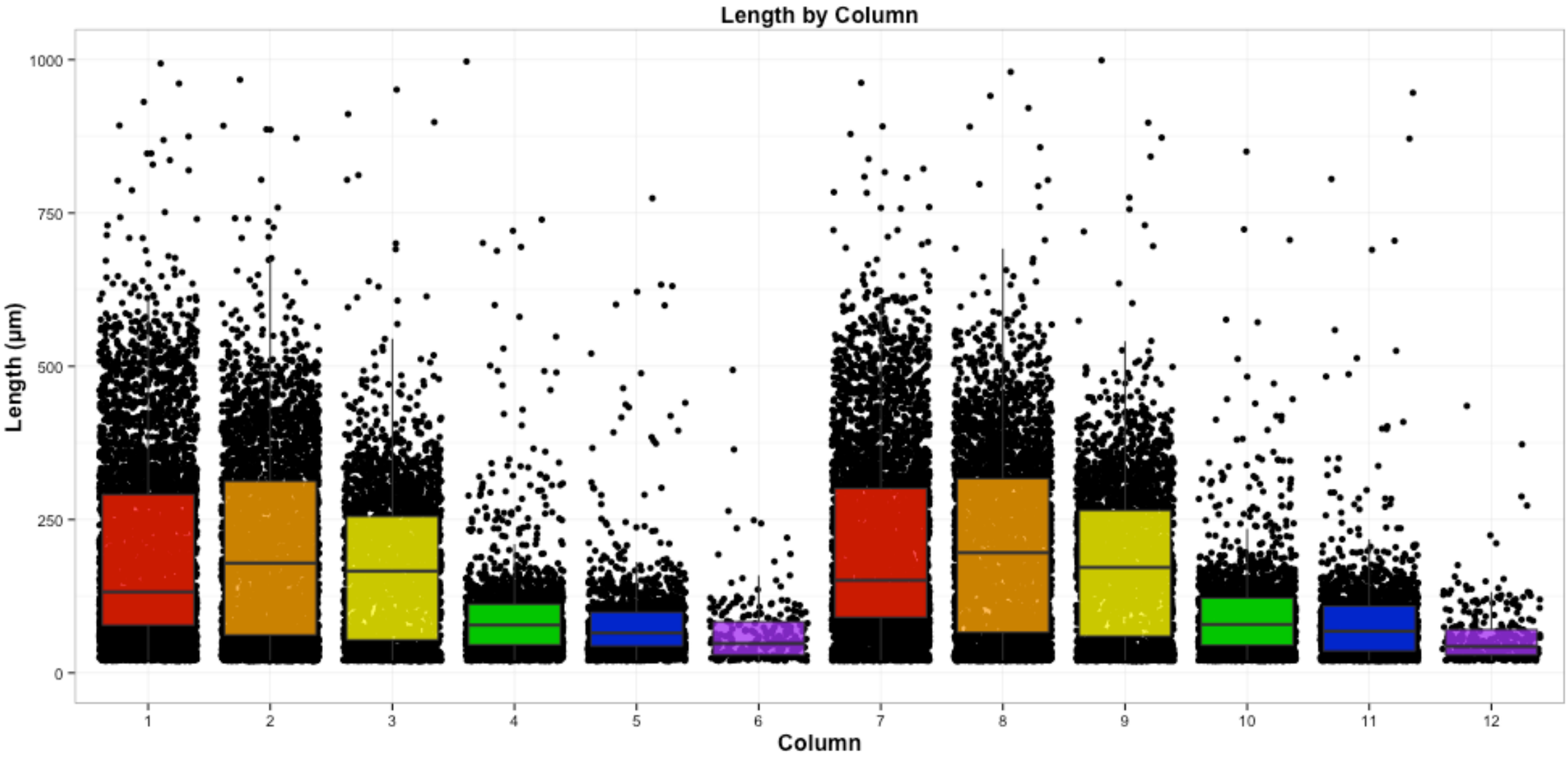
They are useful for dividing data into groups or categories



Length (*μ*m)	Bin	Factor
952	4	Adult
700	3	L4
110	1	L1
232	2	L2/L3
245	2	L2/L3
106	1	L1
688	3	L4
712	3	L4
250	2	L2/L3
1022	1	Adult
225	2	L2/L3
100	1	L1
213	2	L2/L3
1059	4	Adult

ggplot2 Challenge

1. Plot boxplots of TOF separated by Column along with jittered points
2. Add descriptive x, y, and title labels in bold
3. Limit the axes and other aesthetics to show much of the data
4. Use the black and white theme
5. Color the box plots in ROY G BIV repeated



```
ggplot(data=df) + aes(x=Column, y=TOF, fill=Column) +
  geom_jitter() +
  geom_boxplot(alpha=0.75, outlier.size=0) +
  labs(x="Column", y="Length (µm)", title="Length by Column") +
  ylim(0,1000) +
  scale_fill_manual(values=c("red", "orange", "yellow", "green", "blue", "purple",
                             "red", "orange", "yellow", "green", "blue", "purple")) +
  theme_bw() +
  theme(plot.title=element_text(face="bold", size=14),
        axis.title.x=element_text(face="bold", size=14),
        axis.title.y=element_text(face="bold", size=14),
        legend.position="none")
```

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

http://flowingdata.com/

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
```

Data

arrange select
mutate filter
summarise

Simple operations

Simple, efficient storage, fast algorithms, database integration

dplyr: simple verbs

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

arrange arrange the order of rows

```
> head(df)

> df <- arrange(df, Row, TOF)

> head(df)
```

select select columns to keep

```
> head(df)
> reduced.df <- select(df, Row, Column, TOF, EXT, Red, Yellow)
> head(reduced.df)
```

mutate mutate columns into new columns

```
> head(reduced.df)

> reduced.df <- mutate(reduced.df, norm.yellow = Yellow/TOF)

> head(reduced.df)
```

group_by group data by specific column(s)

summarise summarize data by group

```
> head(reduced.df)
> reduced.df <- group_by(reduced.df, Row, Column)
> processed.df <- summarise(reduced.df, mean.TOF = mean(TOF, na.rm=TRUE))
> head(processed.df)
```

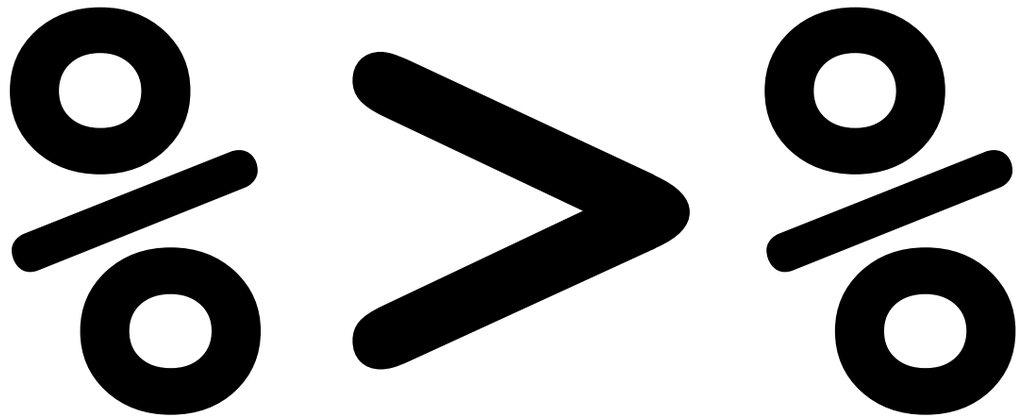
filter filter rows to keep

```
> head(processed.df)

> processed.df <- filter(processed.df, Column %in% c(1:5, 7:11))

> head(processed.df)
```

Let's put it all together



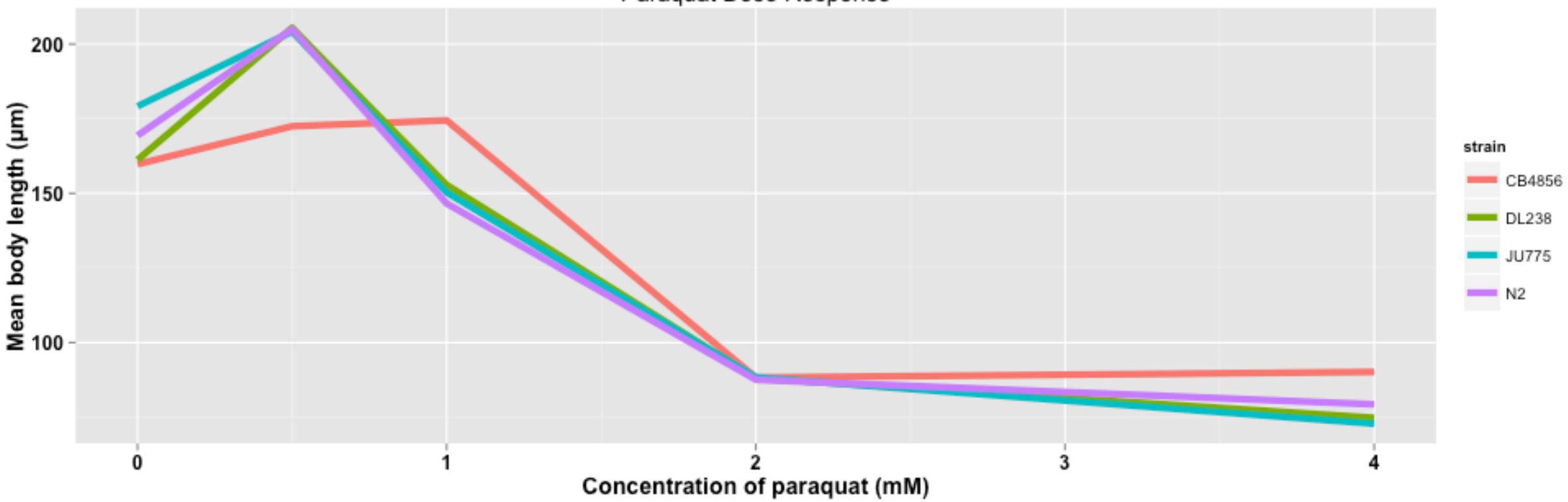
```
> processed.df <- df %>%
      select(Row, Column, TOF, EXT, Red, Yellow) %>%
      mutate(norm.yellow = Yellow/TOF) %>%
      group_by(Row, Column) %>%
      summarise(mean.TOF = mean(TOF, na.rm=TRUE)) %>%
      filter(Column %in% c(1:5, 7:11))
```

Final plot with sorter data

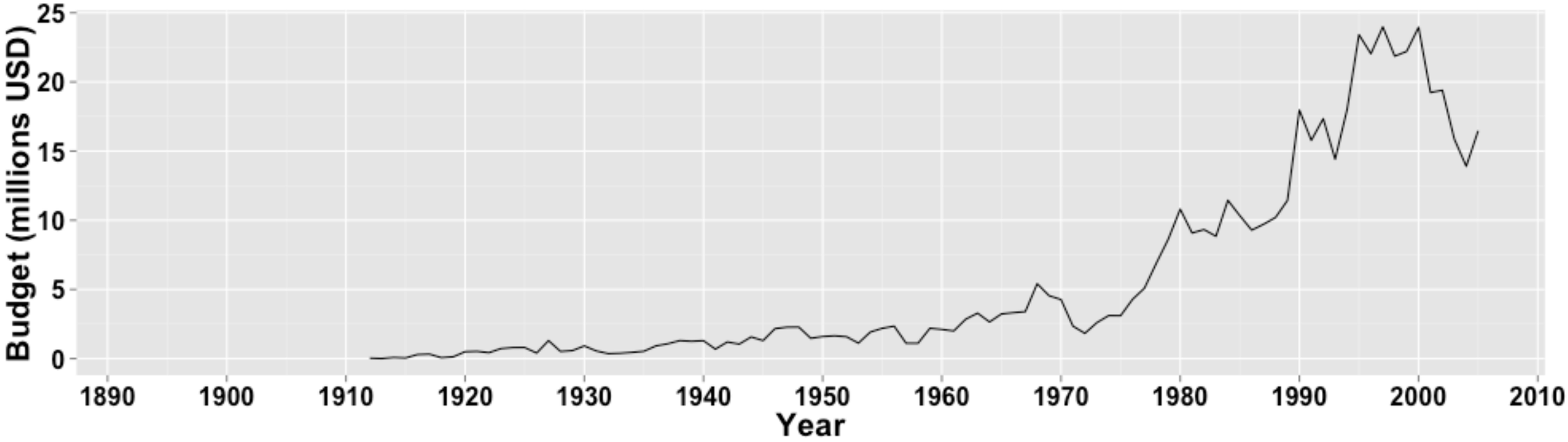
```
> processed.df$strain <- factor(rep(c("CB4856", "DL238", "N2", "JU775"), each=5))

> processed.df$concentration <- rep(c(0, 0.5, 1, 2, 4), 16)

> processed.df2 <- processed.df %>%
        group_by(strain, concentration) %>%
        summarise(m.TOF = mean(mean.TOF, na.rm=T))
```



How have movie budgets changed over time?

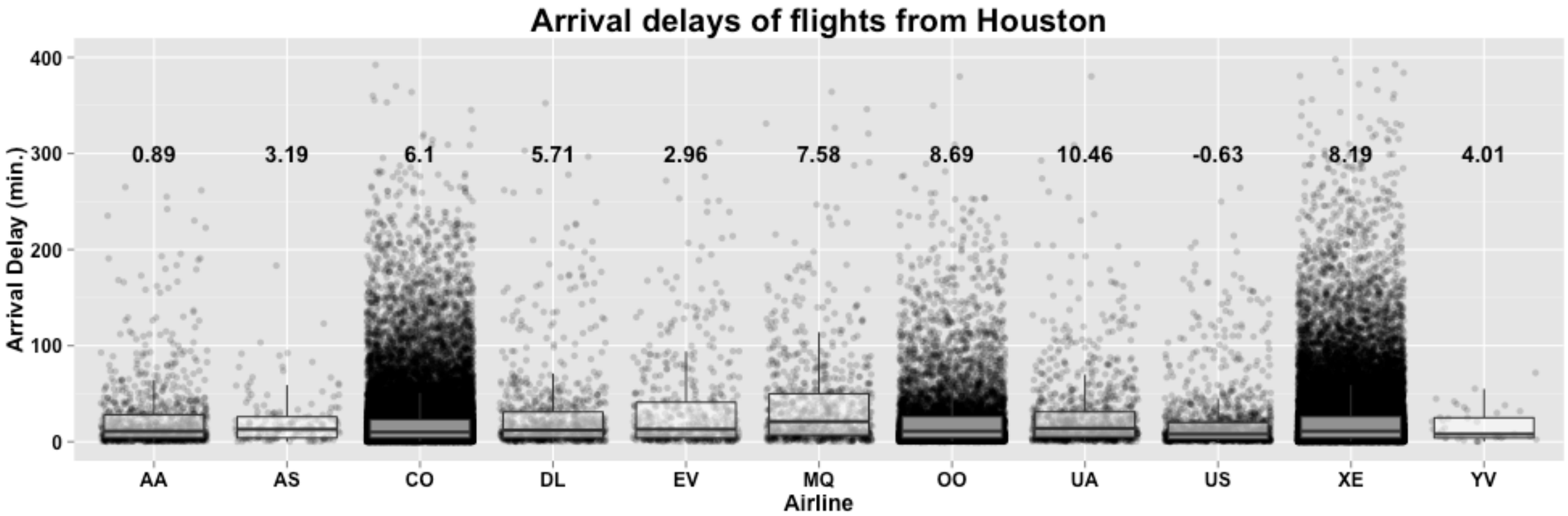


Use **`data(movies)`** to make this plot with dplyr and ggplot2

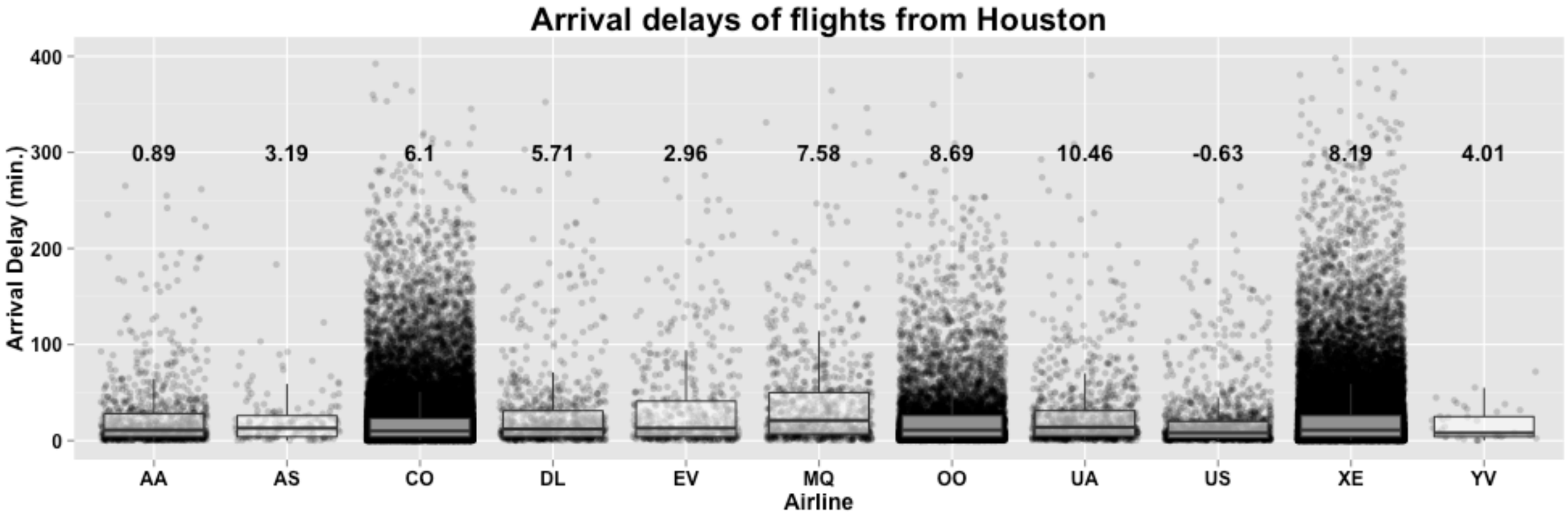
```
> proc.mov  <- movies %>% group_by(year) %>% summarise(m.budg= mean(budget, na.rm=T))

> ggplot(proc.mov) + aes(x=year, y=m.budg/1000000) + geom_line() +
  labs(x="Year", y="Budget (millions USD)") +
  scale_x_continuous(breaks=c((189:201)*10)) +
  presentation
```

How have movie budgets changed over time?



Use `data(hflights)` to make this plot with dplyr and ggplot2




```
> proc.fl <- hflights %>% filter(Origin == "IAH") %>%
    group_by(UniqueCarrier) %>%
    summarise(mean.ArrDelay = round(mean(ArrDelay, na.rm=T), digits=2),
              sd.ArrDelay = sd(ArrDelay, na.rm=T))

> df <- filter(hflights, Origin == "IAH")

> ggplot(df) + aes(x=UniqueCarrier, y=ArrDelay) + geom_jitter(alpha=0.2) +
    geom_boxplot(alpha=0.5, outlier.size=0) +
    geom_text(data=proc.fl, aes(label=mean.ArrDelay, x=UniqueCarrier, y=300), fontface="bold") +
    ylim(0, 400) +
    labs(x="Airline", y="Arrival Delay (min.)", title="Arrival delays of flights from Houston") +
    theme(axis.text.x = element_text(face="bold", size=12, color="black"),
        axis.text.y = element_text(face="bold", size=12, color="black"),
        axis.title.x = element_text(face="bold", size=14),
        axis.title.y = element_text(face="bold", size=14),
        plot.title = element_text(face="bold", size=20))
```

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

ggplot2 and dplyr work better in long-format data frames.
More generally, long-format data is the way to go when using R.
There are however uses for wide-format data.

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

Other useful functions in R

`match, %in%` = match elements in object to another

`grep` = same as in CL, look for pattern in strings

`do` = dplyr way to run functions

`is.na` = gives TRUE or FALSE for NA data values

`unique` = same as in CL, 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

How to get help?

Code Club

Fridays 1:30-3:30 Cook 4123

Unsolicited advice about computational work

Don’t be afraid

It will be rewarding

Stop using Excel

Homework

Fill out course survey ASAP

http://goo.gl/Y4eTOD