Intro to R

Data Visualization

Introduction to R for Public Health Researchers

Read in the Data

```
library(readr)
mort = read_csv(
  "https://jhudatascience.org/intro_to_r/data/mortality.csv")
mort[1:2, 1:5]
```

```
# A tibble: 2 x 5
  X1          `1760` `1761` `1762` `1763`
  <chr>        <dbl>  <dbl>  <dbl>  <dbl>
1 Afghanistan     NA     NA     NA     NA
2 Albania         NA     NA     NA     NA
```

Read in Data: `jhur`

```
jhur::read_mortality()
```

```
# A tibble: 197 x 255
   X1      `1760` `1761` `1762` `1763` `1764` `1765` `1766` `1767` `1768` `176
   <chr>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <db
 1 Afghan…     NA     NA     NA     NA     NA     NA     NA     NA     NA
 2 Albania     NA     NA     NA     NA     NA     NA     NA     NA     NA
 3 Algeria     NA     NA     NA     NA     NA     NA     NA     NA     NA
 4 Angola      NA     NA     NA     NA     NA     NA     NA     NA     NA
 5 Argent…     NA     NA     NA     NA     NA     NA     NA     NA     NA
 6 Armenia     NA     NA     NA     NA     NA     NA     NA     NA     NA
 7 Aruba       NA     NA     NA     NA     NA     NA     NA     NA     NA
 8 Austra…     NA     NA     NA     NA     NA     NA     NA     NA     NA
 9 Austria     NA     NA     NA     NA     NA     NA     NA     NA     NA
10 Azerba…     NA     NA     NA     NA     NA     NA     NA     NA     NA
# … with 187 more rows, and 244 more variables: 1770 <dbl>, 1771 <dbl>,
#   1772 <dbl>, 1773 <dbl>, 1774 <dbl>, 1775 <dbl>, 1776 <dbl>, 1777 <dbl>,
#   1778 <dbl>, 1779 <dbl>, 1780 <dbl>, 1781 <dbl>, 1782 <dbl>, 1783 <dbl>,
#   1784 <dbl>, 1785 <dbl>, 1786 <dbl>, 1787 <dbl>, 1788 <dbl>, 1789 <dbl>,
#   1790 <dbl>, 1791 <dbl>, 1792 <dbl>, 1793 <dbl>, 1794 <dbl>, 1795 <dbl>,
#   1796 <dbl>, 1797 <dbl>, 1798 <dbl>, 1799 <dbl>, 1800 <dbl>, 1801 <dbl>,
#   1802 <dbl>, 1803 <dbl>, 1804 <dbl>, 1805 <dbl>, 1806 <dbl>, 1807 <dbl>,
#   1808 <dbl>, 1809 <dbl>, 1810 <dbl>, 1811 <dbl>, 1812 <dbl>, 1813 <dbl>,
#   1814 <dbl>, 1815 <dbl>, 1816 <dbl>, 1817 <dbl>, 1818 <dbl>, 1819 <dbl>,
#   1820 <dbl>, 1821 <dbl>, 1822 <dbl>, 1823 <dbl>, 1824 <dbl>, 1825 <dbl>,
#   1826 <dbl>, 1827 <dbl>, 1828 <dbl>, 1829 <dbl>, 1830 <dbl>, 1831 <dbl>,
#   1832 <dbl>, 1833 <dbl>, 1834 <dbl>, 1835 <dbl>, 1836 <dbl>, 1837 <dbl>,
#   1838 <dbl>, 1839 <dbl>, 1840 <dbl>, 1841 <dbl>, 1842 <dbl>, 1843 <dbl>,
#   1844 <dbl>, 1845 <dbl>, 1846 <dbl>, 1847 <dbl>, 1848 <dbl>, 1849 <dbl>,
#   1850 <dbl>, 1851 <dbl>, 1852 <dbl>, 1853 <dbl>, 1854 <dbl>, 1855 <dbl>,
```

Data are not Tidy!

ggplot2

Let's try this out on the childhood mortality data used above. However, let's do some manipulation first, by using `pivor_longer` on the data to convert to long.

```
library(tidyverse)
long = mort
long = long %>% pivot_longer(!country,
                             names_to = "year", values_to = "morts")
head(long, 2)
```

```
# A tibble: 2 x 3
  country     year  morts
  <chr>       <chr> <dbl>
1 Afghanistan 1760     NA
2 Afghanistan 1761     NA
```

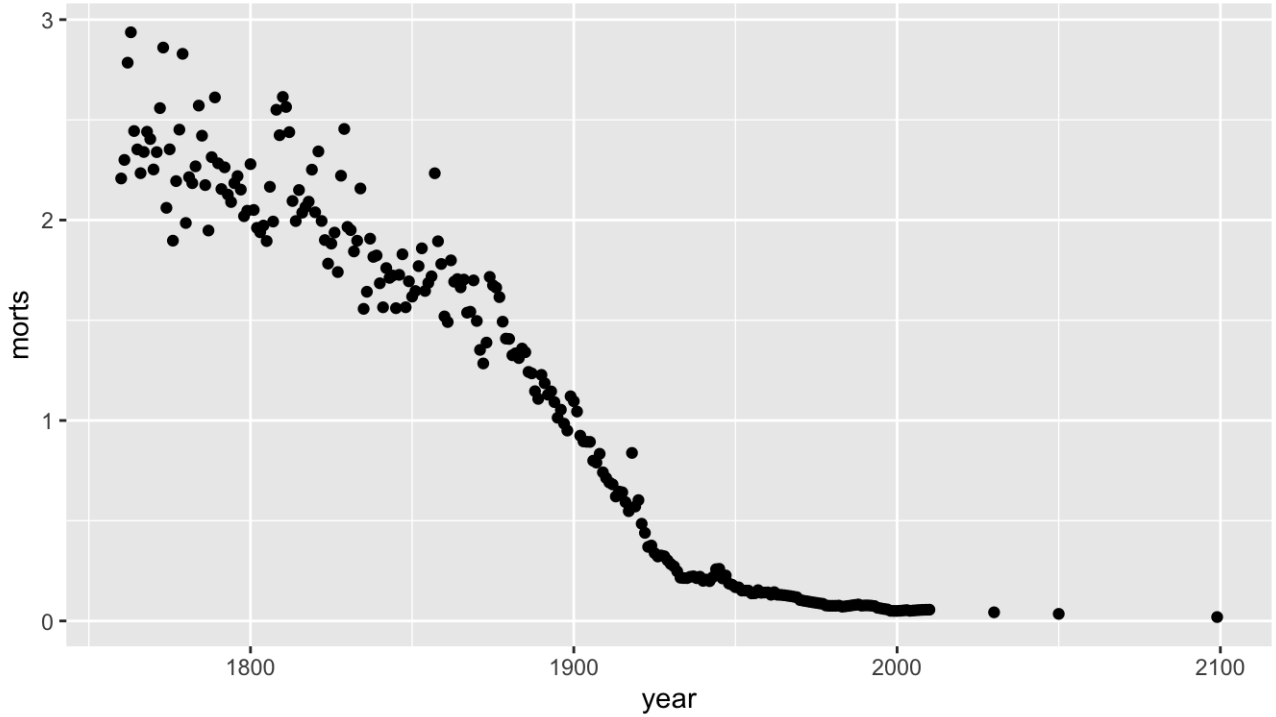
ggplot2

Let's also make the year numeric, as we did above in the stand-alone `year` variable.

```
library(stringr)
library(dplyr)
long$year = long$year %>% str_replace("^X", "") %>% as.numeric
long = long %>% filter(!is.na(morts))
```

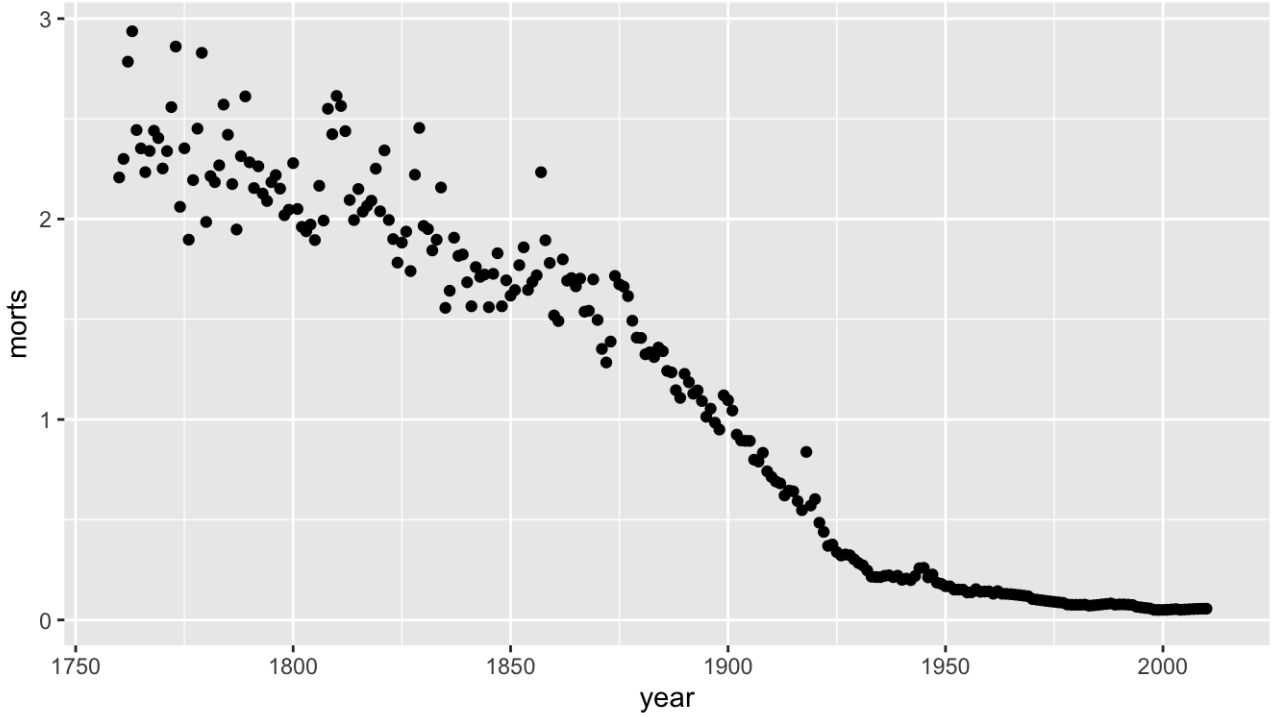
Plot the long data

```
swede_long = long %>% filter(country == "Sweden")
qplot(x = year, y = morts, data = swede_long)
```



Plot the long data only up to 2012

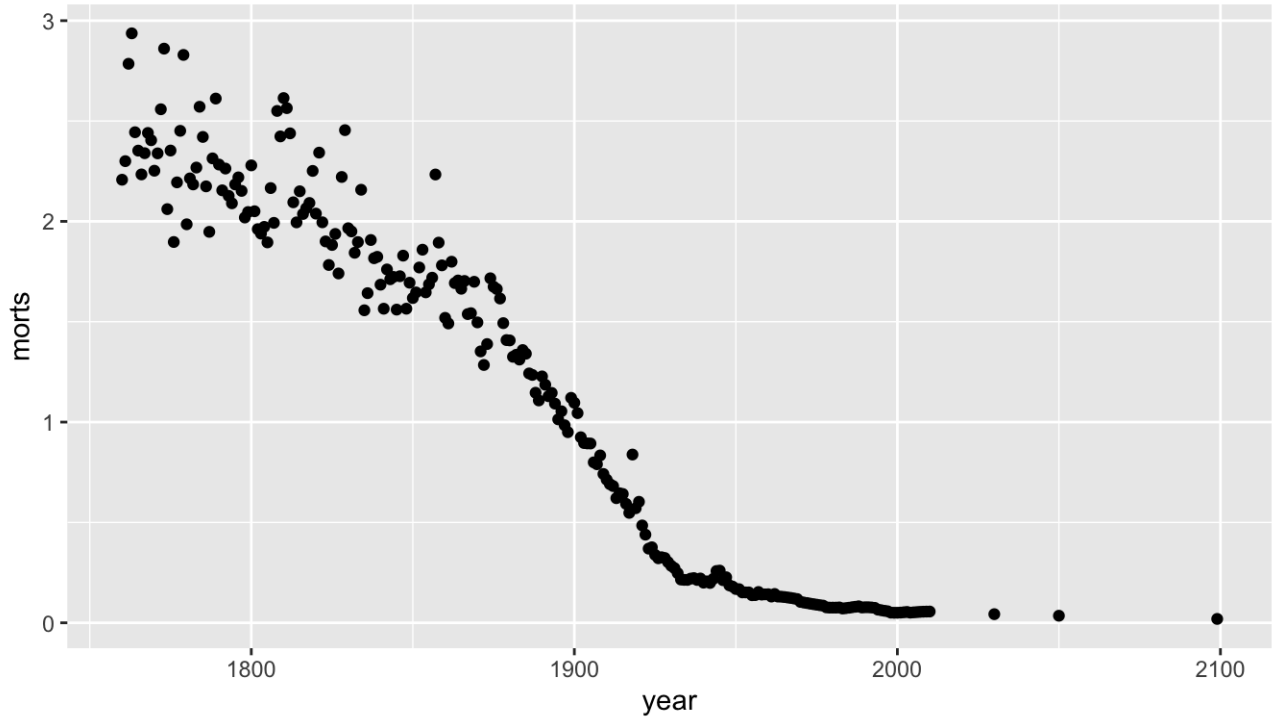
```
qplot(x = year, y = morts, data = swede_long, xlim = c(1760,2012))
```



ggplot2

`ggplot2` is a package of plotting that is very popular and powerful (using the **g**rammar of **g**raphics). `qplot` ("quick plot"), similar to `plot`

```
library(ggplot2)
qplot(x = year, y = morts, data = swede_long)
```



ggplot2

The generic plotting function is `ggplot`, which uses **aes**thetics:

```
ggplot(data, aes(args))
```

```
g = ggplot(data = swede_long, aes(x = year, y = morts))
```

`g` is an object, which you can adapt into multiple plots!

ggplot2

Common aesthetics:

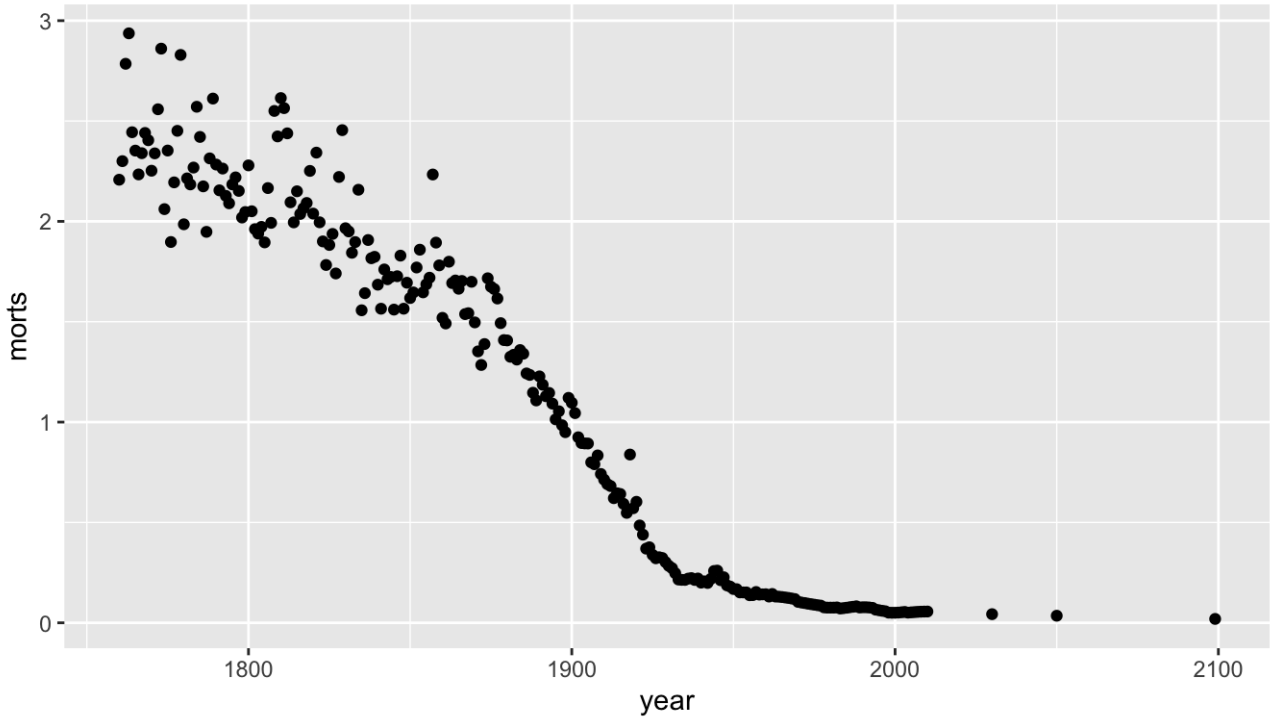
- **x**
- **y**
- **colour/color**
- **size**
- **fill**
- **shape**

If you set these in `aes`, you set them to a variable. If you want to set them for all values, set them in a `geom`.

ggplot2

You can do this most of the time using `qplot`, but `qplot` will assume a scatterplot if `x` and `y` are specified and histogram if `x` is specified:

```
q = qplot(data = swede_long, x = year, y = morts)
q
```



`g` is an object, which you can adapt into multiple plots!

ggplot2: what's a geom?

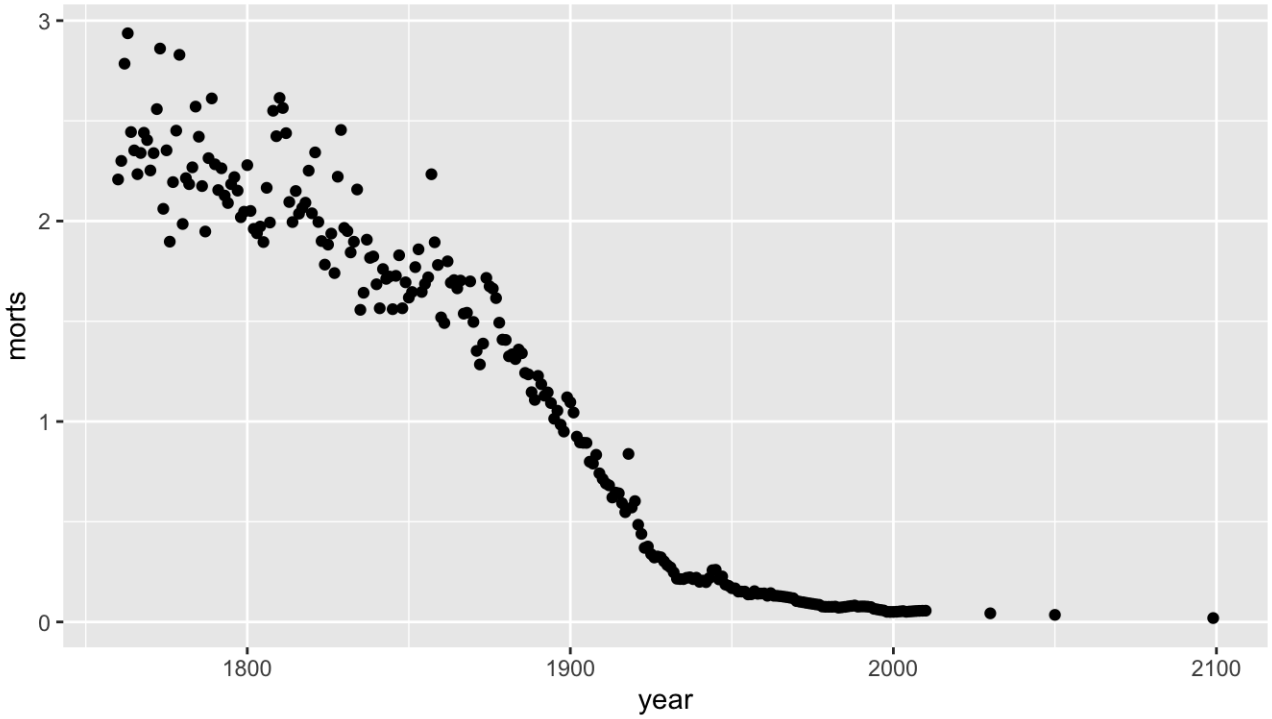
`g` on it's own can't be plotted, we have to add `layers`, usually with `geom_` commands:

- `geom_point` - add points
- `geom_line` - add lines
- `geom_density` - add a density plot
- `geom_histogram` - add a histogram
- `geom_smooth` - add a smoother
- `geom_boxplot` - add a boxplots
- `geom_bar` - bar charts
- `geom_tile` - rectangles/heatmaps

ggplot2: adding a geom and assigning

You “add” things to a plot with a + sign (not pipe!). If you assign a plot to an object, you must call `print` to print it.

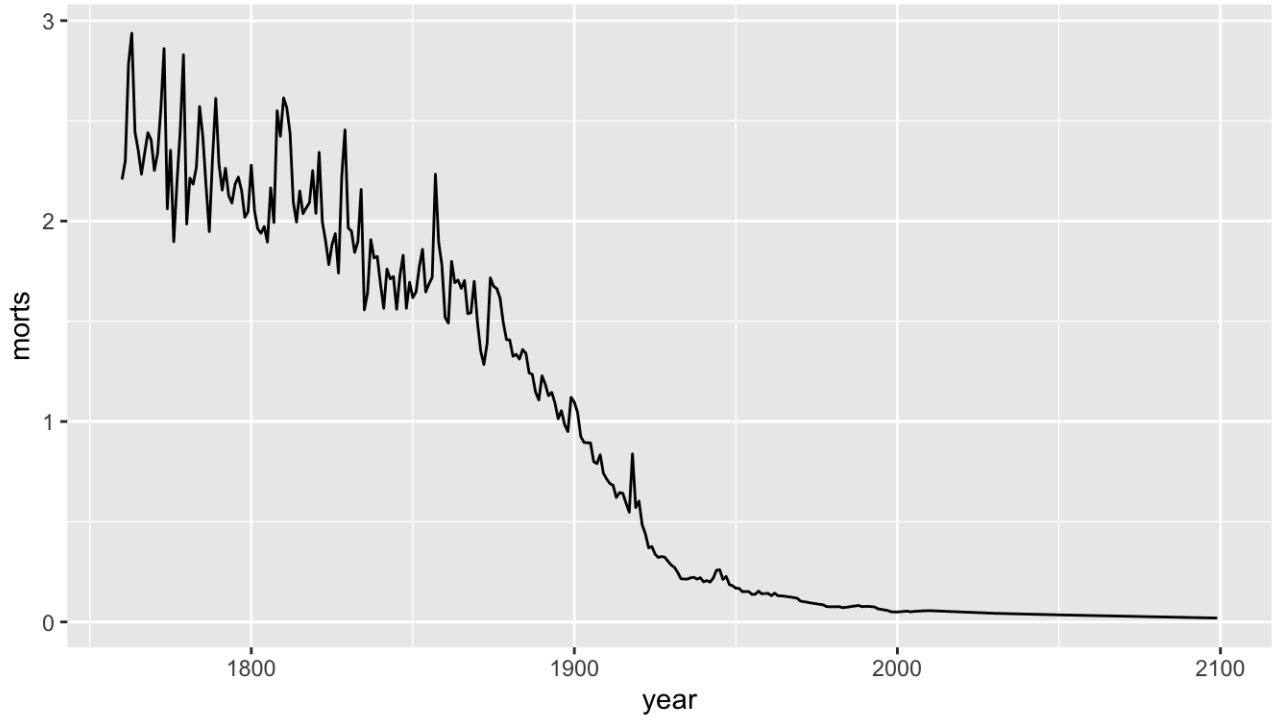
```
gpoints = g + geom_point(); print(gpoints) # one line for slides
```



ggplot2: adding a geom

Otherwise it prints by default - this time it's a line

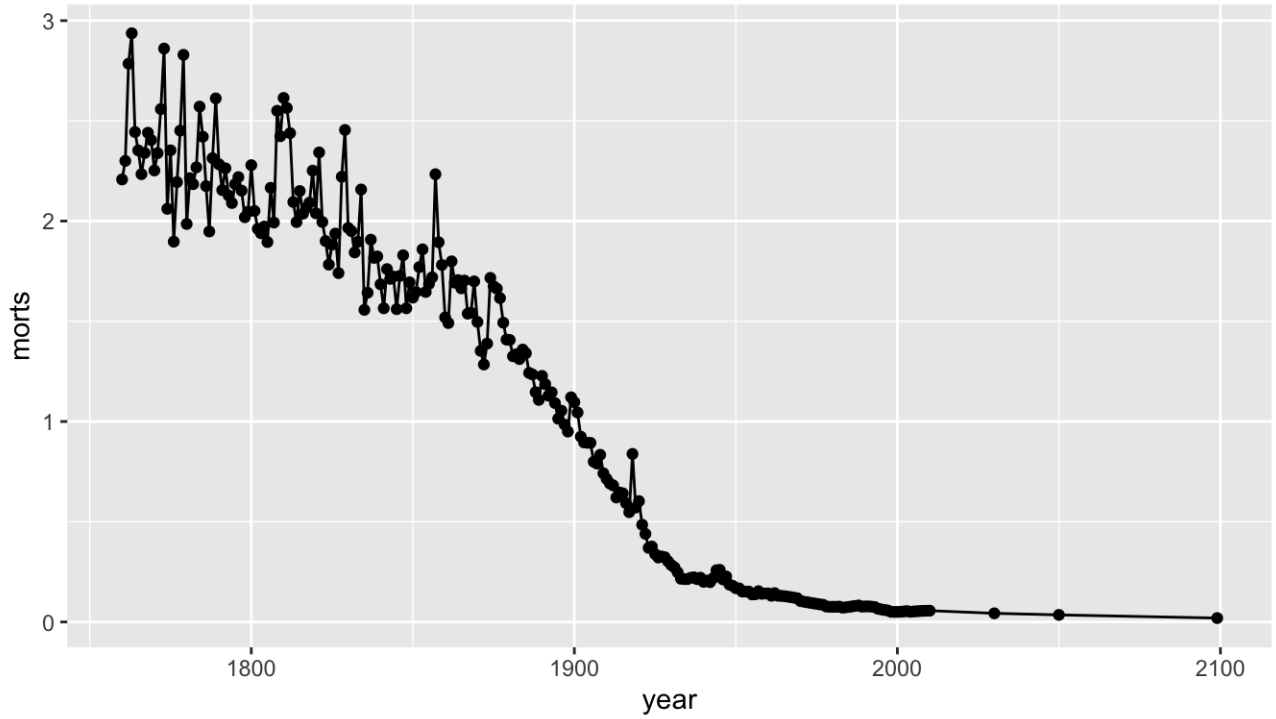
```
g + geom_line()
```



ggplot2: adding a geom

You can add multiple geoms:

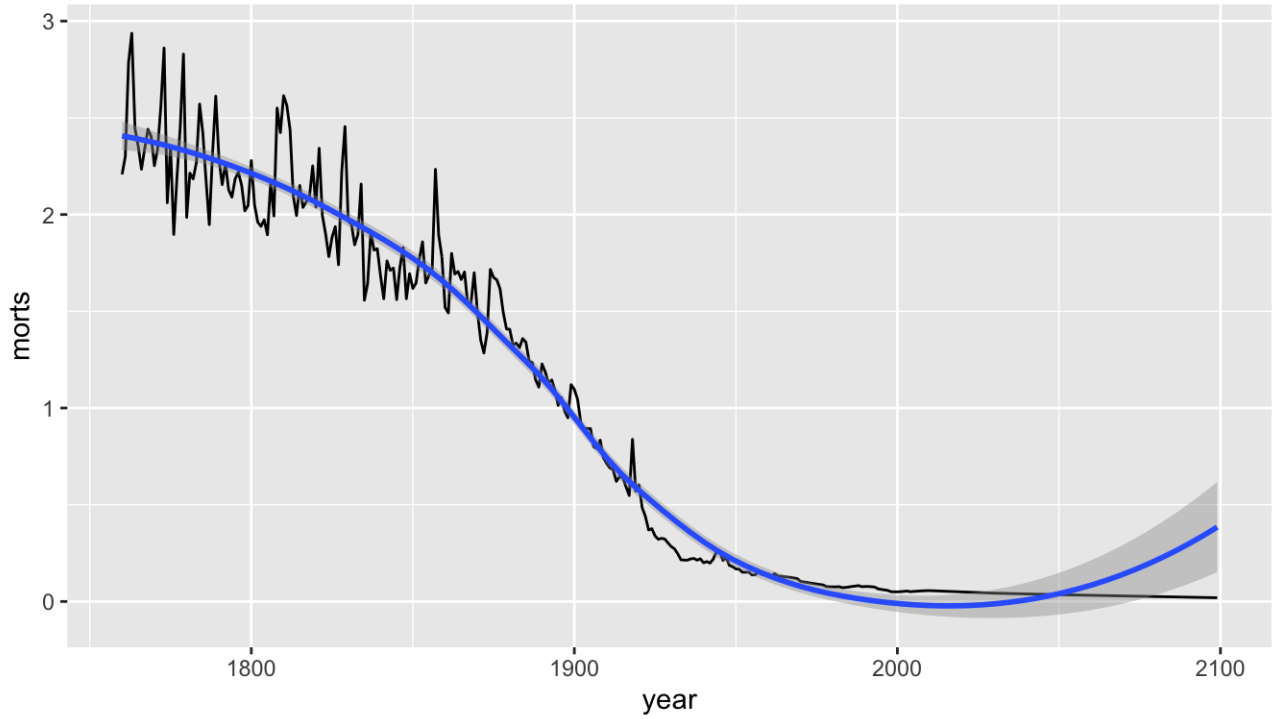
```
g + geom_line() + geom_point()
```



ggplot2: adding a smoother

Let's add a smoother through the points:

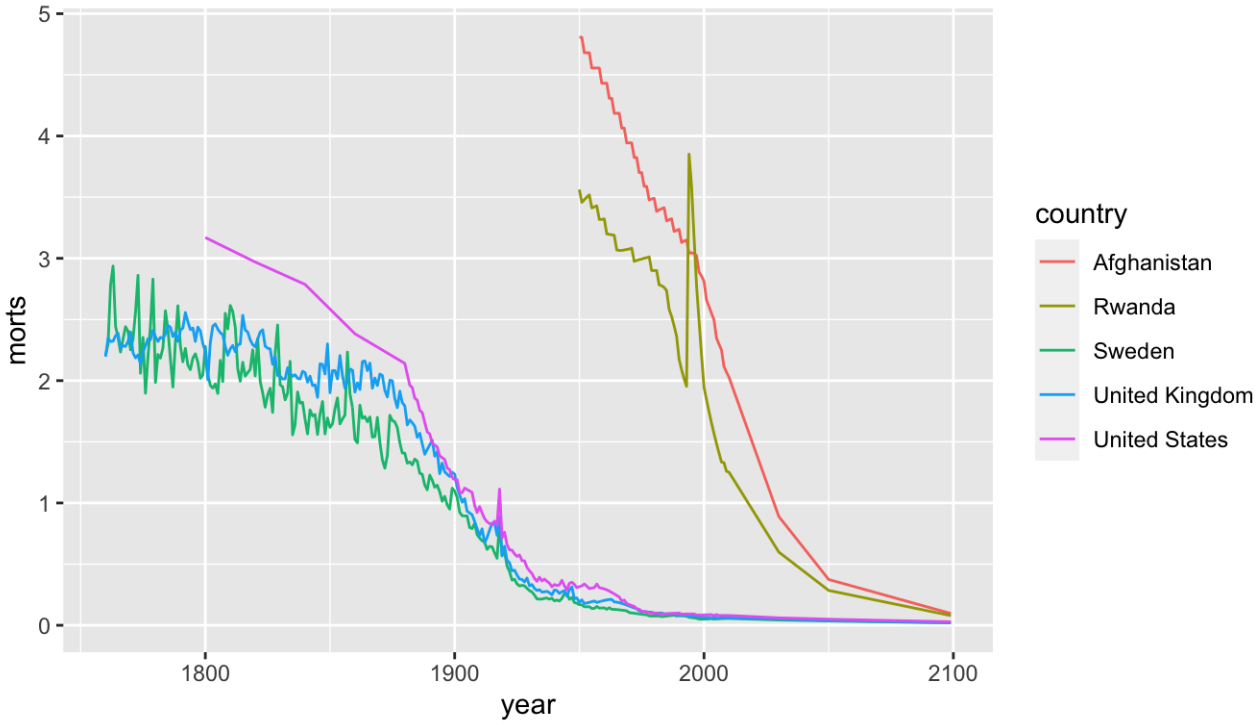
```
g + geom_line() + geom_smooth()
```



ggplot2: grouping - using colour

If we want a plot with new data, call `ggplot` again. Group plots by `country` using `colour` (piping in the data):

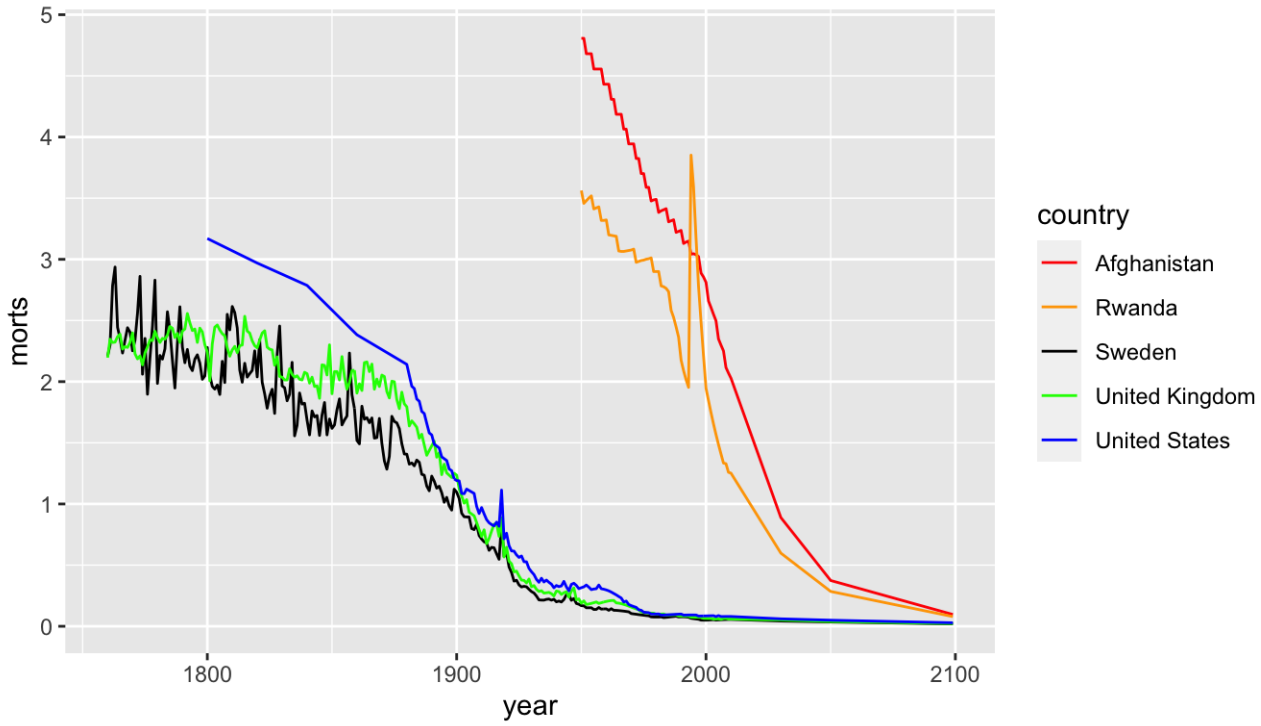
```
sub = long %>% filter(country %in% c("United States", "United Kingdom",
    "Sweden", "Afghanistan", "Rwanda"))
g = sub %>% ggplot(aes(x = year, y = morts, colour = country))
g + geom_line()
```



Coloring manually

There are many `scale_AESTHETICS_*` functions and `scale_AESTHETICS_manual` allows to directly specify the colors:

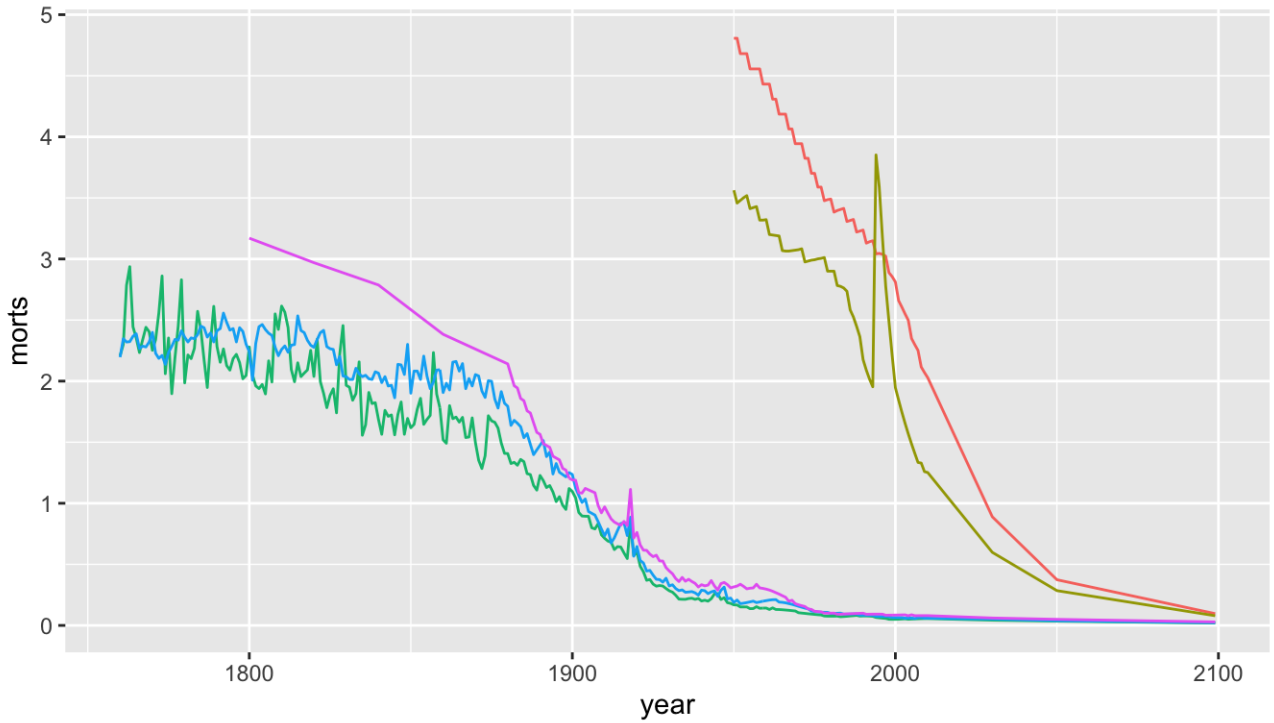
```
g + geom_line() + scale_colour_manual(values =
    c("United States" = "blue", "United Kingdom" = "green",
      "Sweden" = "black", "Afghanistan" = "red", "Rwanda" = "orange"))
```



ggplot2: grouping - using colour

Let's remove the legend using the `guide` command:

```
g + geom_line() + guides(colour = FALSE)
```

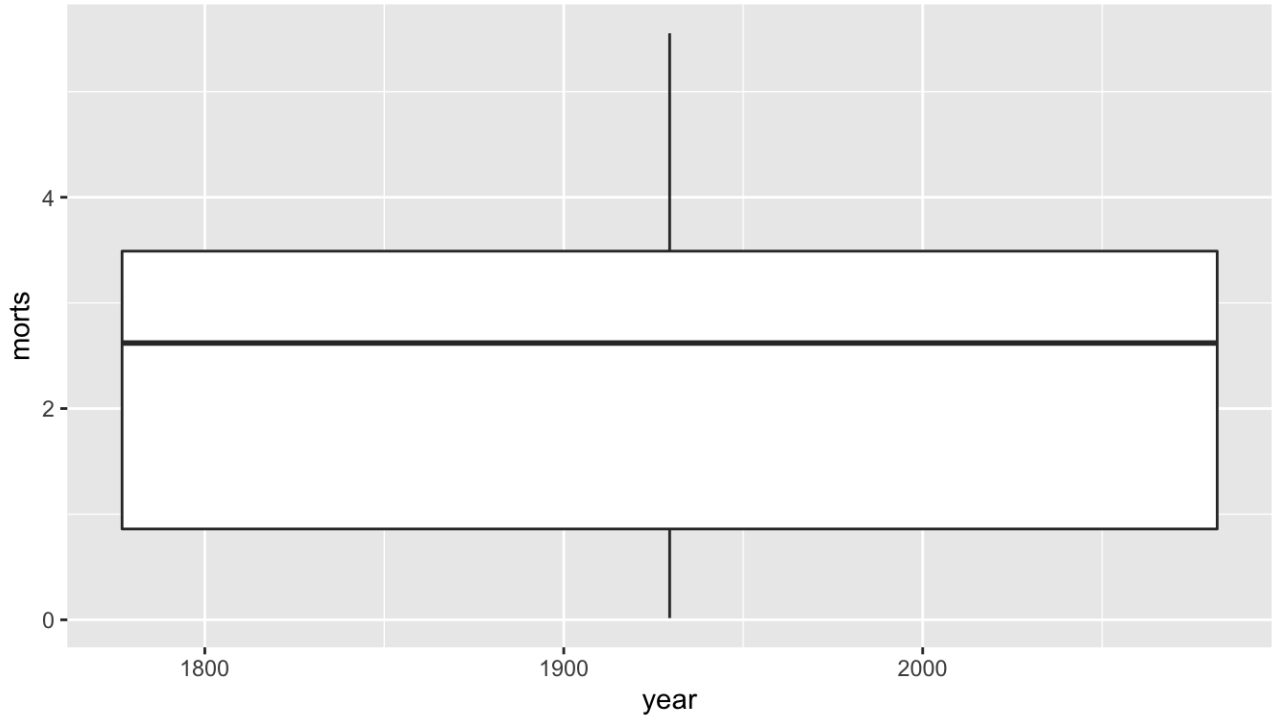


Lab Part 1

Website

ggplot2: boxplot

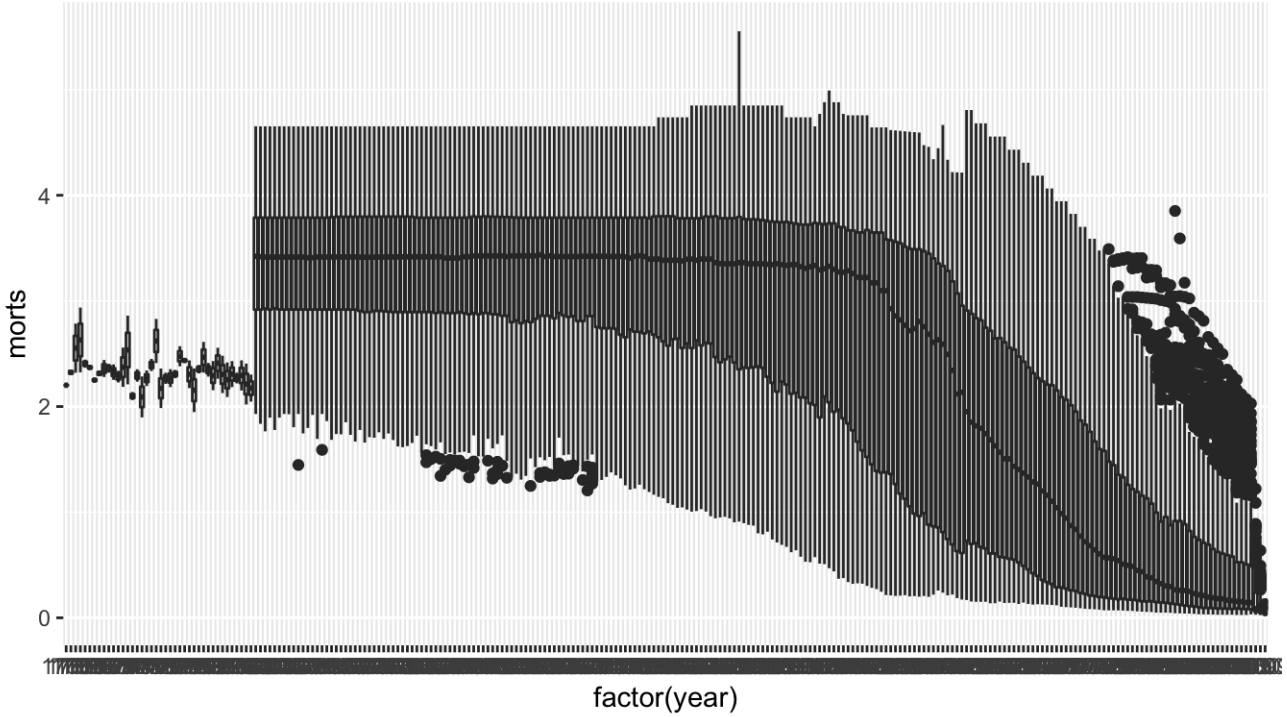
```
ggplot(long, aes(x = year, y = morts)) + geom_boxplot()
```



ggplot2: boxplot

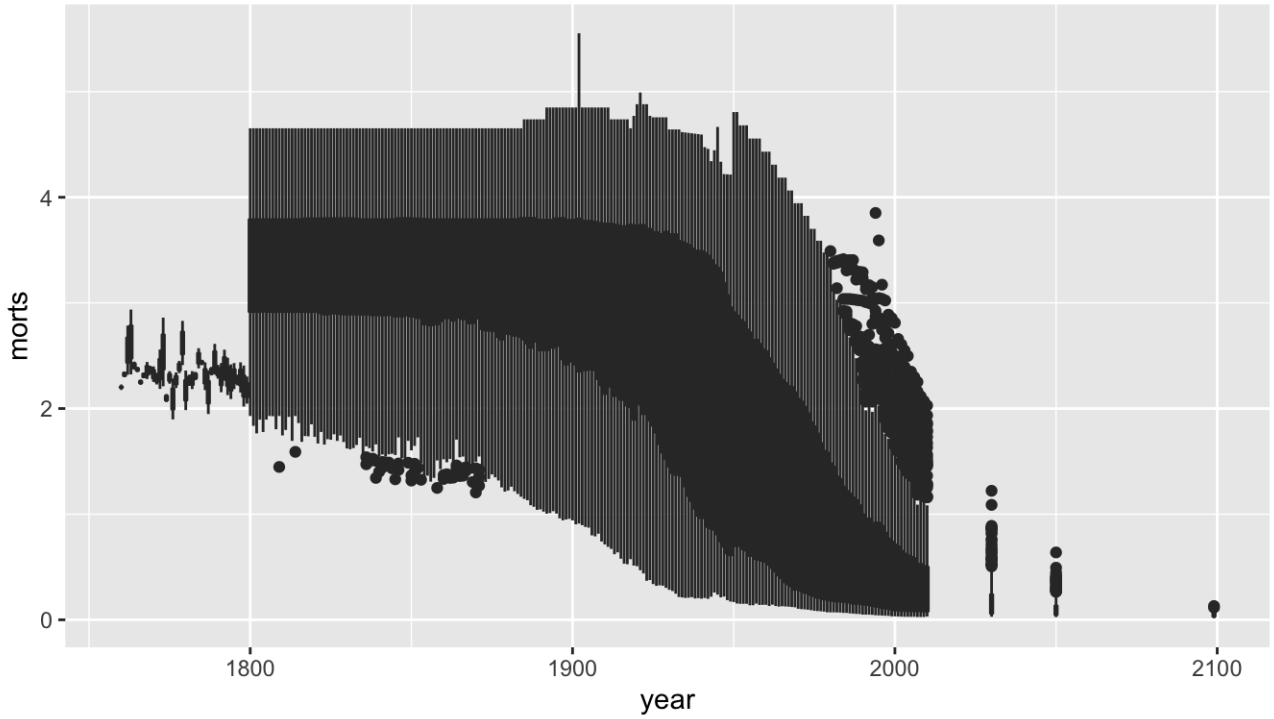
For different plotting per year - must make it a factor - but x-axis is wrong!

```
ggplot(long, aes(x = factor(year), y = morts)) + geom_boxplot()
```



ggplot2: boxplot

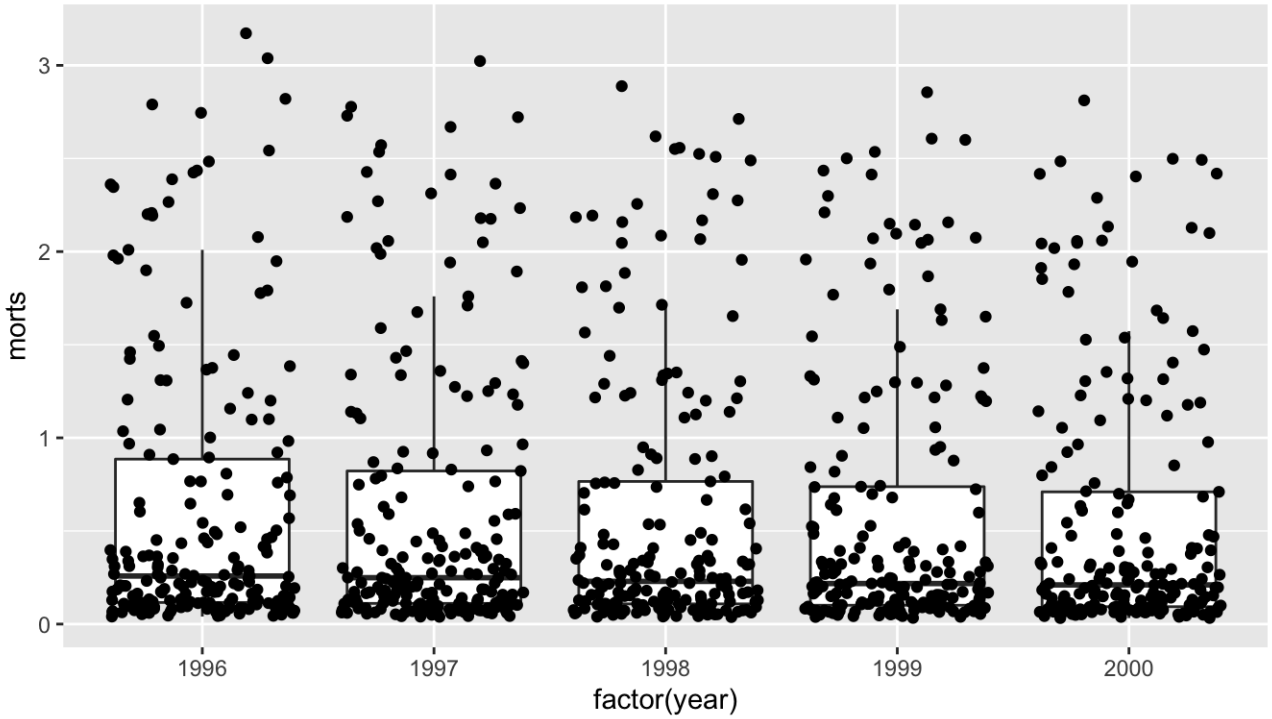
```
ggplot(long, aes(x = year, y = morts, group = year)) + geom_boxplot()
```



ggplot2: boxplot with points

- `geom_jitter` plots points "jittered" with noise so not overlapping

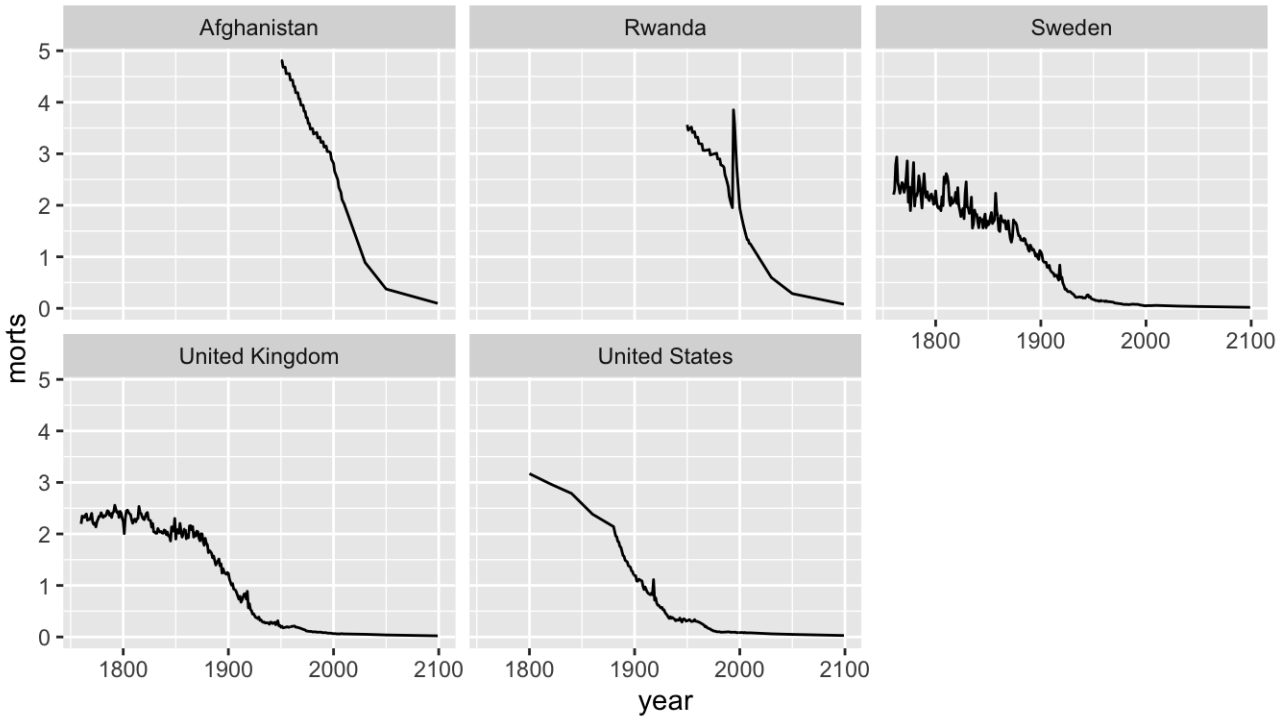
```
sub_year = long %>% filter( year > 1995 & year <= 2000)
ggplot(sub_year, aes(x = factor(year), y = morts)) +
  geom_boxplot(outlier.shape = NA) + # don't show outliers - will below
  geom_jitter(height = 0)
```



facets: plotting multiple panels

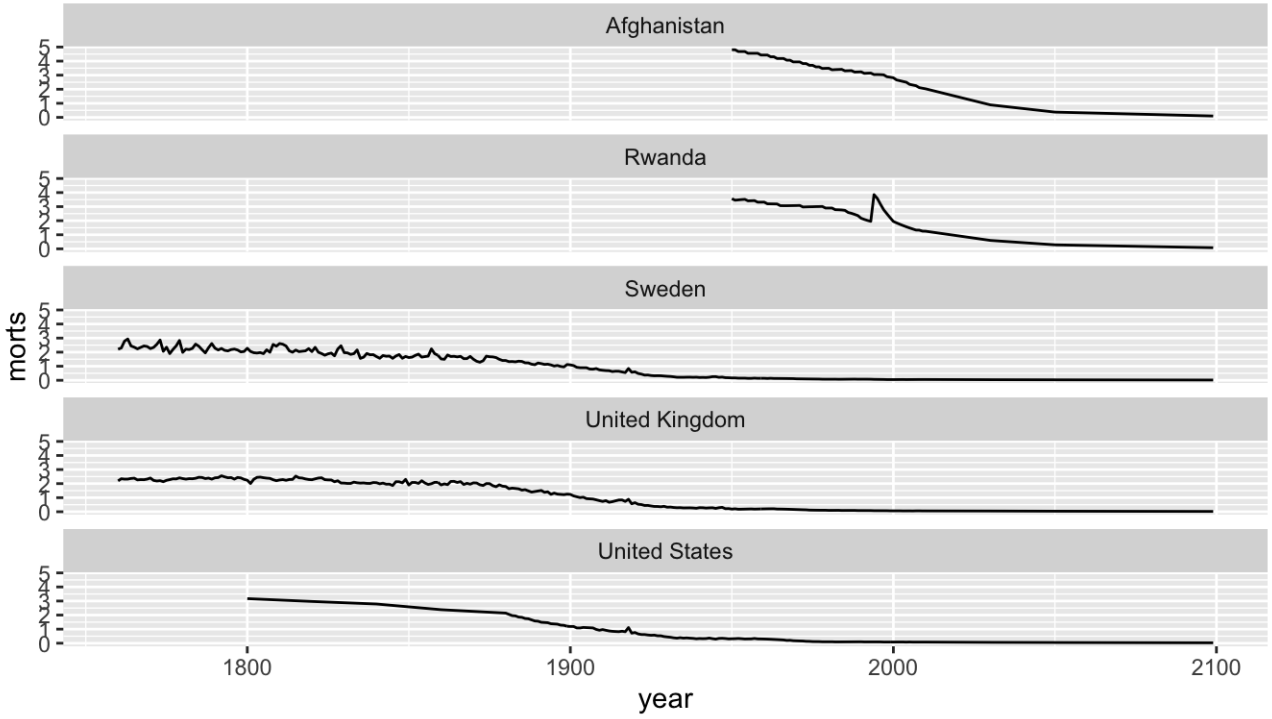
A `facet` will make a plot over variables, keeping axes the same (out can change that):

```
sub %>% ggplot(aes(x = year, y = morts)) +
  geom_line() +
  facet_wrap(~ country)
```



facets: plotting multiple panels

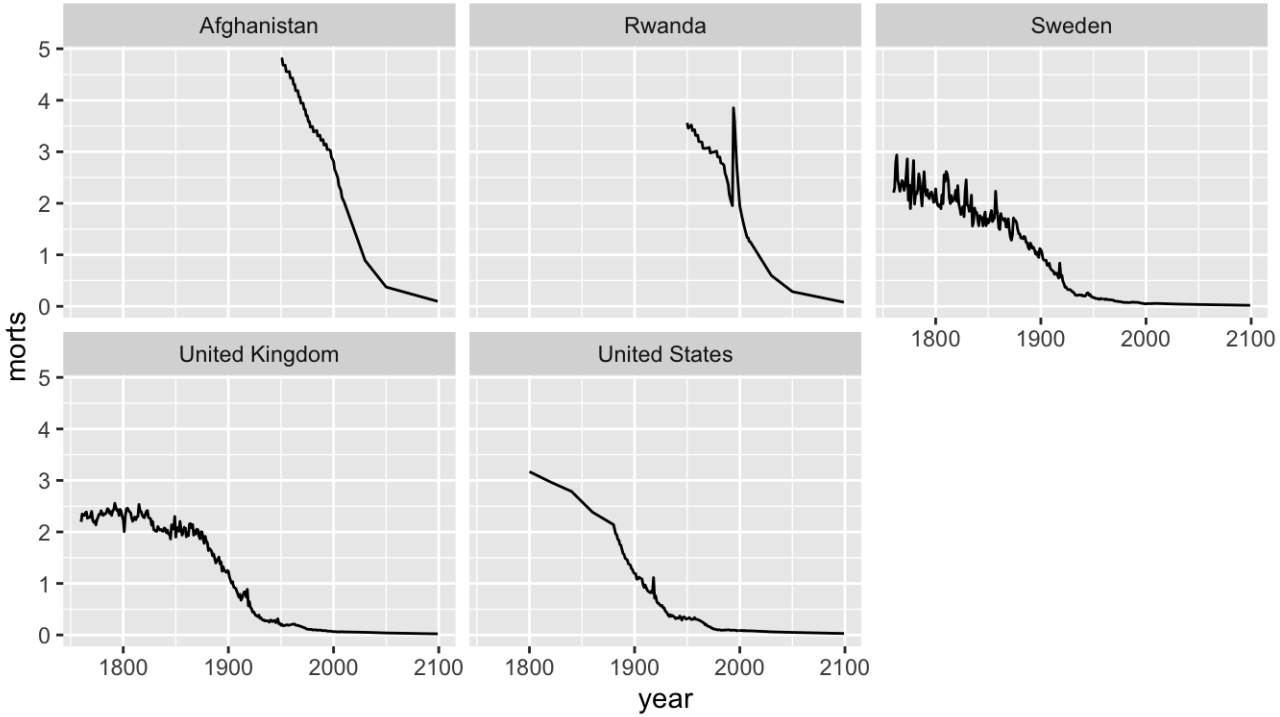
```
sub %>% ggplot(aes(x = year, y = morts)) +
  geom_line() +
  facet_wrap(~ country, ncol = 1)
```



facets: plotting multiple panels

You can use `facets` in `qplot`

```
qplot(x = year, y = morts, geom = "line", facets = ~ country, data = sub)
```



facets: plotting multiple panels

You can also do multiple factors with + on the right hand side

```
sub %>% ggplot(aes(x = year, y = morts)) +
  geom_line() +
  facet_wrap(~ country + x2 + ... )
```

Lab Part 2

Website

Devices

By default, R displays plots in a separate panel. From there, you can export the plot to a variety of image file types, or copy it to the clipboard.

However, sometimes its very nice to save many plots made at one time to one pdf file, say, for flipping through. Or being more precise with the plot size in the saved file.

R has 5 additional graphics devices: `bmp()`, `jpeg()`, `png()`, `tiff()`, and `pdf()`

Devices

The syntax is very similar for all of them:

```
pdf("filename.pdf", width=8, height=8) # inches
plot() # plot 1
plot() # plot 2
# etc
dev.off()
```

Basically, you are creating a pdf file, and telling R to write any subsequent plots to that file. Once you are done, you turn the device off. Note that failing to turn the device off will create a pdf file that is corrupt, that you cannot open.

Saving the output:

```
png("morts_over_time.png")
print(q)
dev.off()
```

```
quartz_off_screen
                2
```

```
file.exists("morts_over_time.png")
```

```
[1] TRUE
```

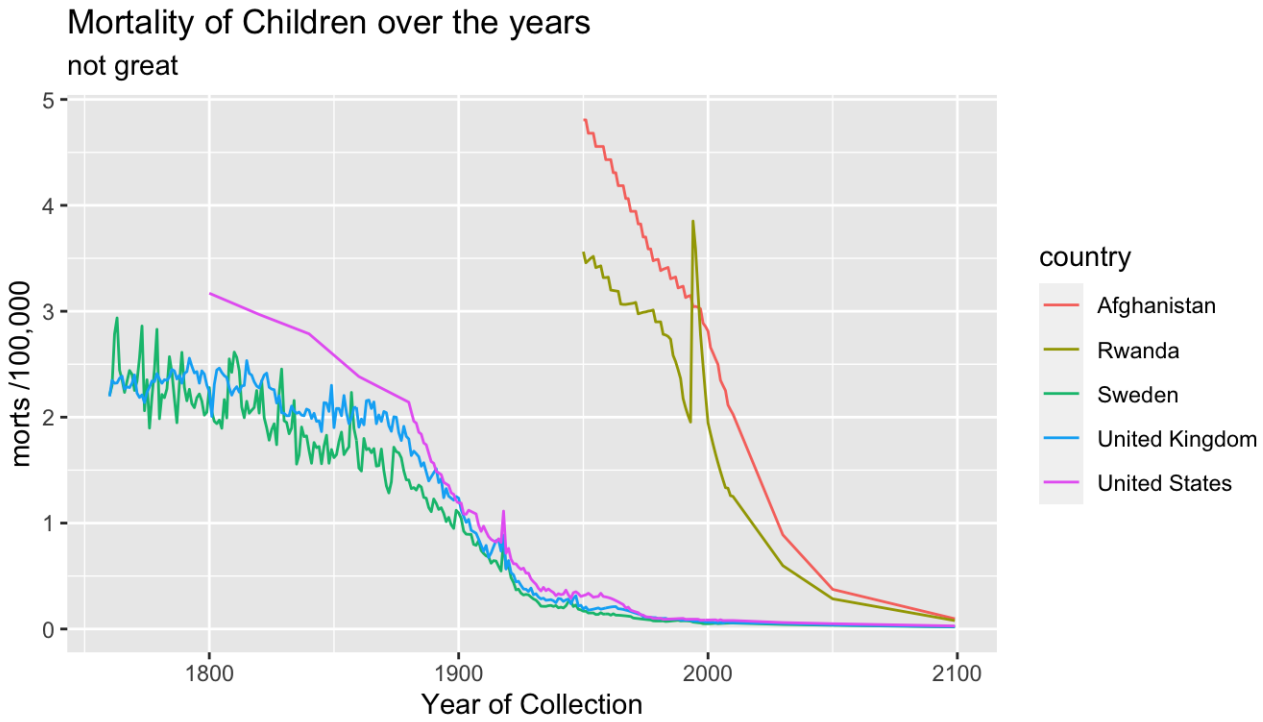
Saving the output

There's also a `ggsave` function that is useful for saving a single `ggplot` object.

Labels and such

- `xlab`/`ylab` - functions to change the labels; `ggtitle` - change the title

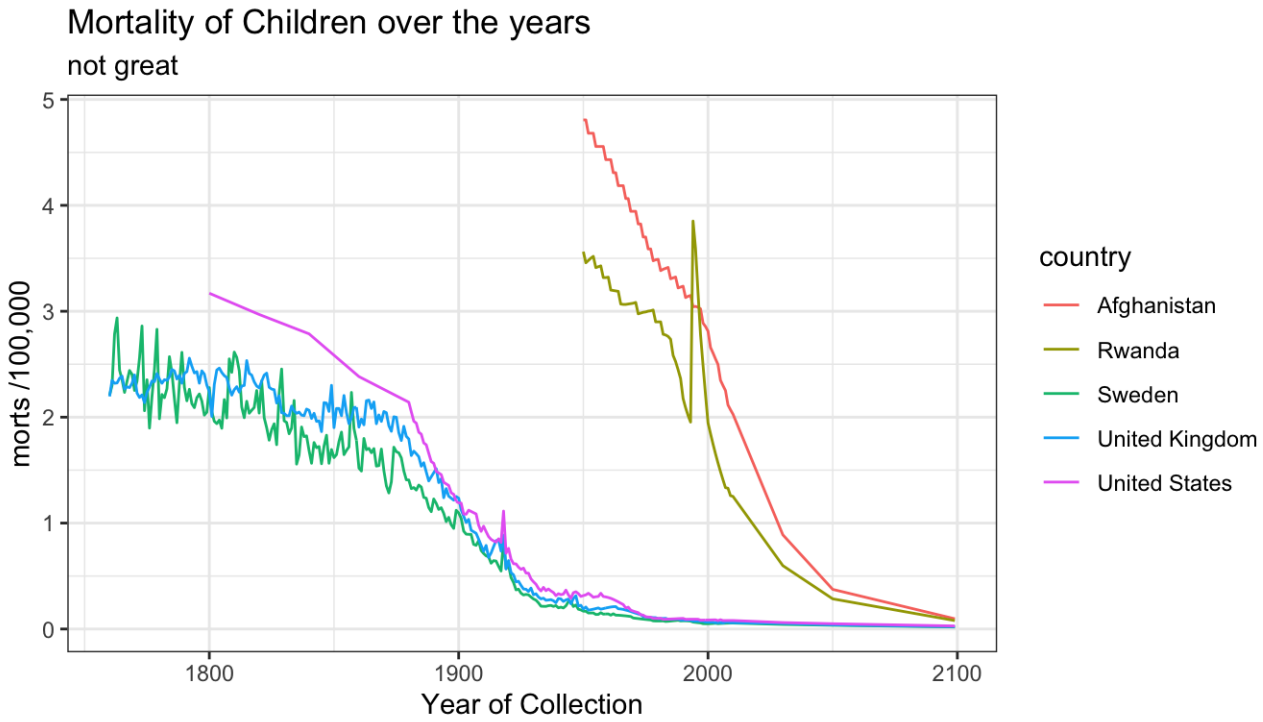
```
q = qplot(x = year, y = morts, colour = country, data = sub,
          geom = "line") +
  xlab("Year of Collection") + ylab("morts /100,000") +
  ggtitle("Mortality of Children over the years", subtitle = "not great")
q
```



Themes

- see `?theme_bw` - for ggthemes - black and white

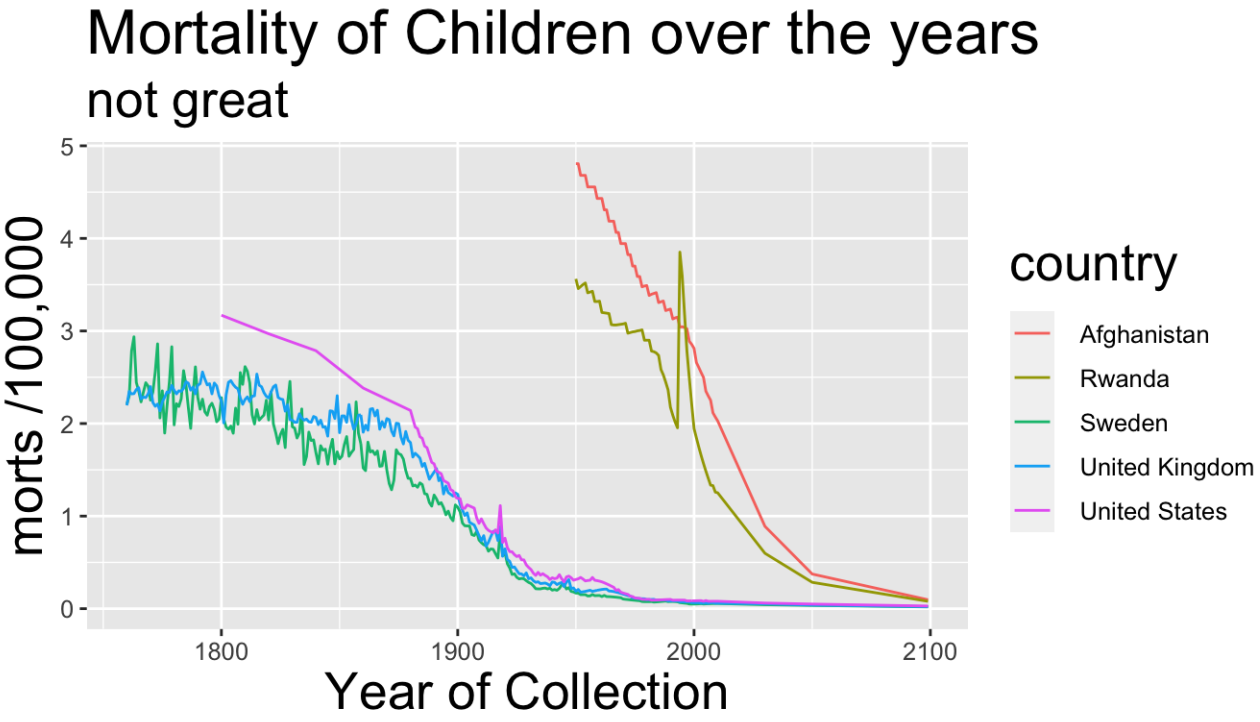
```
q + theme_bw()
```



Themes: change plot parameters

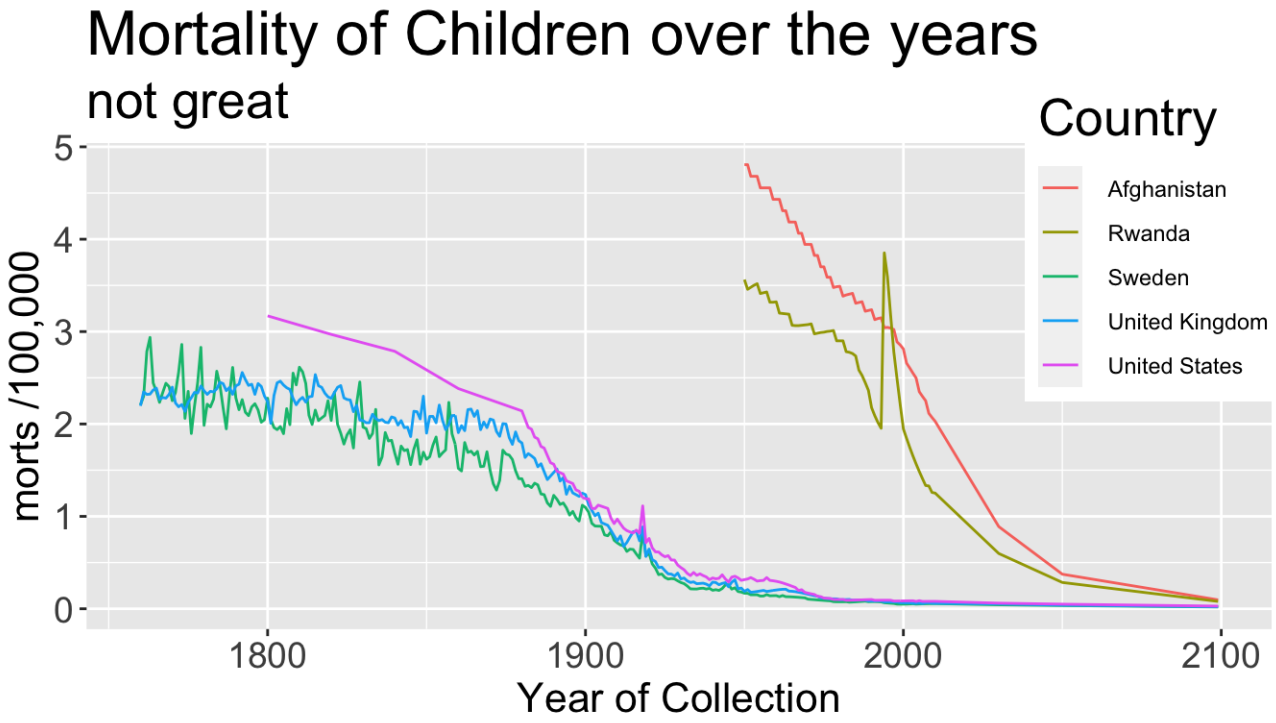
- `theme` - global or specific elements/increase text size

```
q + theme(text = element_text(size = 12), title = element_text(size = 20))
```



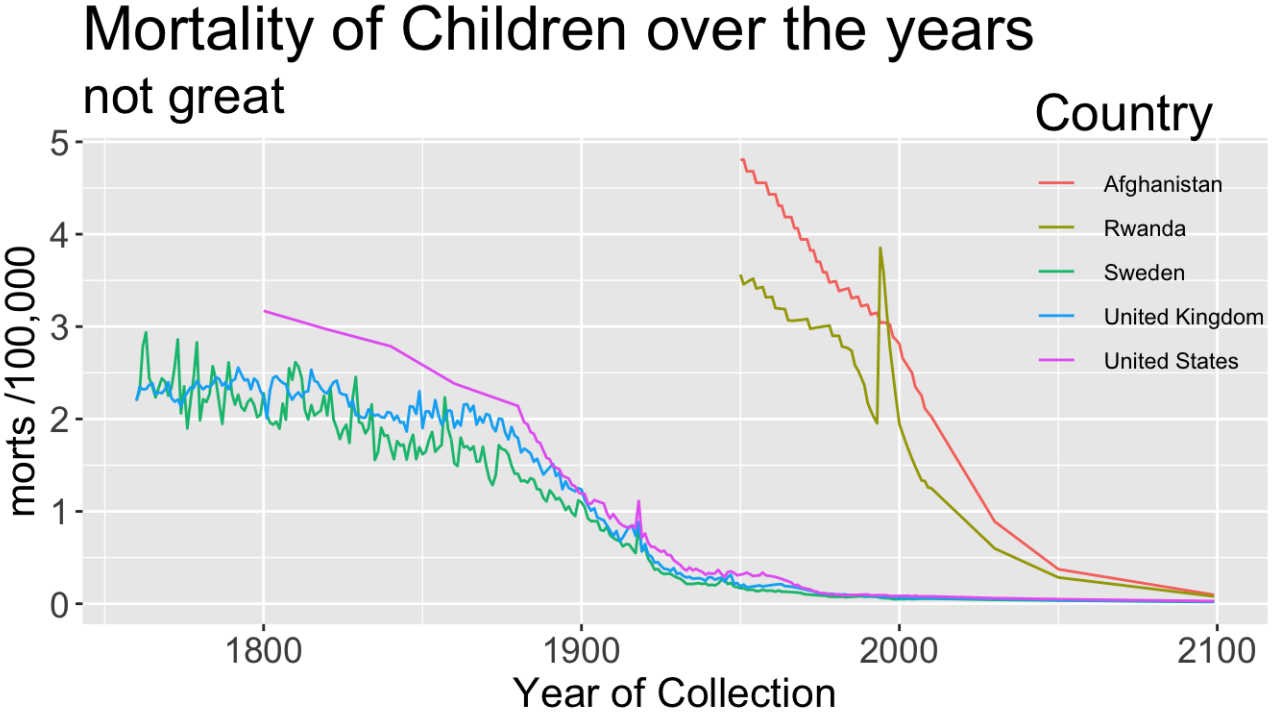
Themes

```
q = q + theme(axis.text = element_text(size = 14),
              title = element_text(size = 20),
              axis.title = element_text(size = 16),
              legend.position = c(0.9, 0.8)) +
  guides(colour = guide_legend(title = "Country"))
q
```



Code for a transparent legend

```
transparent_legend =  theme(legend.background = element_rect(
    fill = "transparent"),
  legend.key = element_rect(fill = "transparent",
                            color = "transparent") )
q + transparent_legend
```

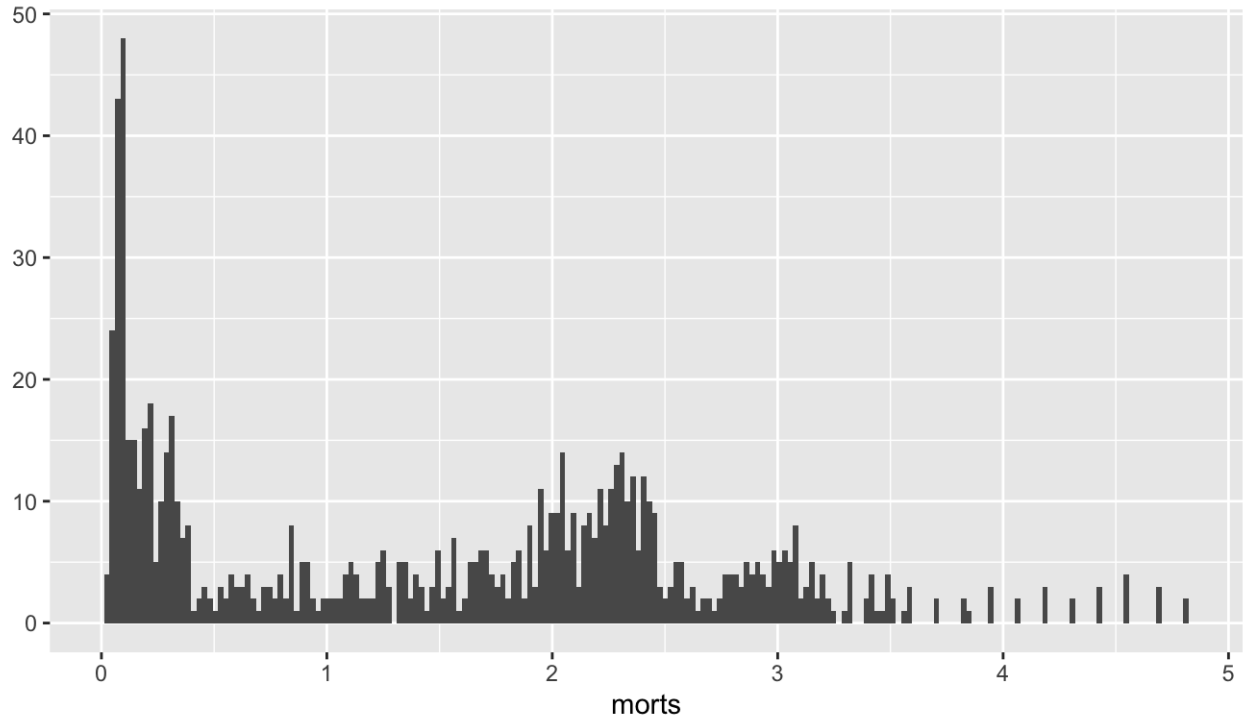


Lab Part 3

Website

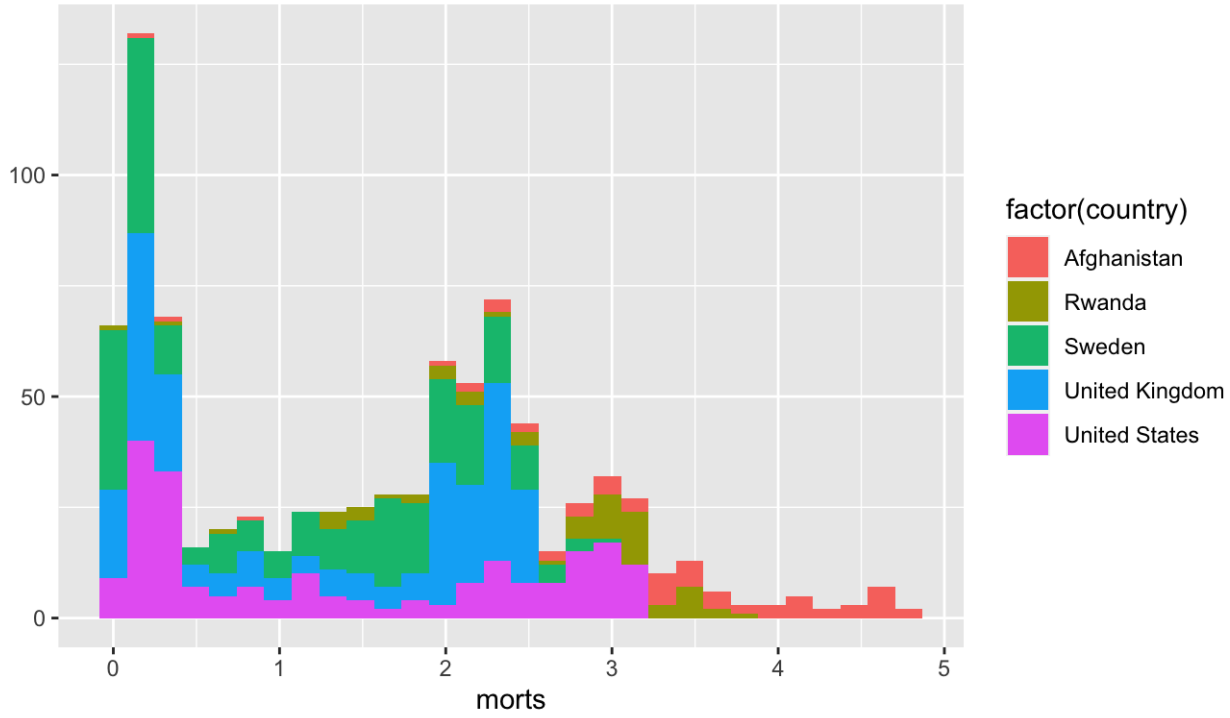
Histograms again: Changing bins

```
qplot(x = morts, data = sub, bins = 200)
```



Multiple Histograms

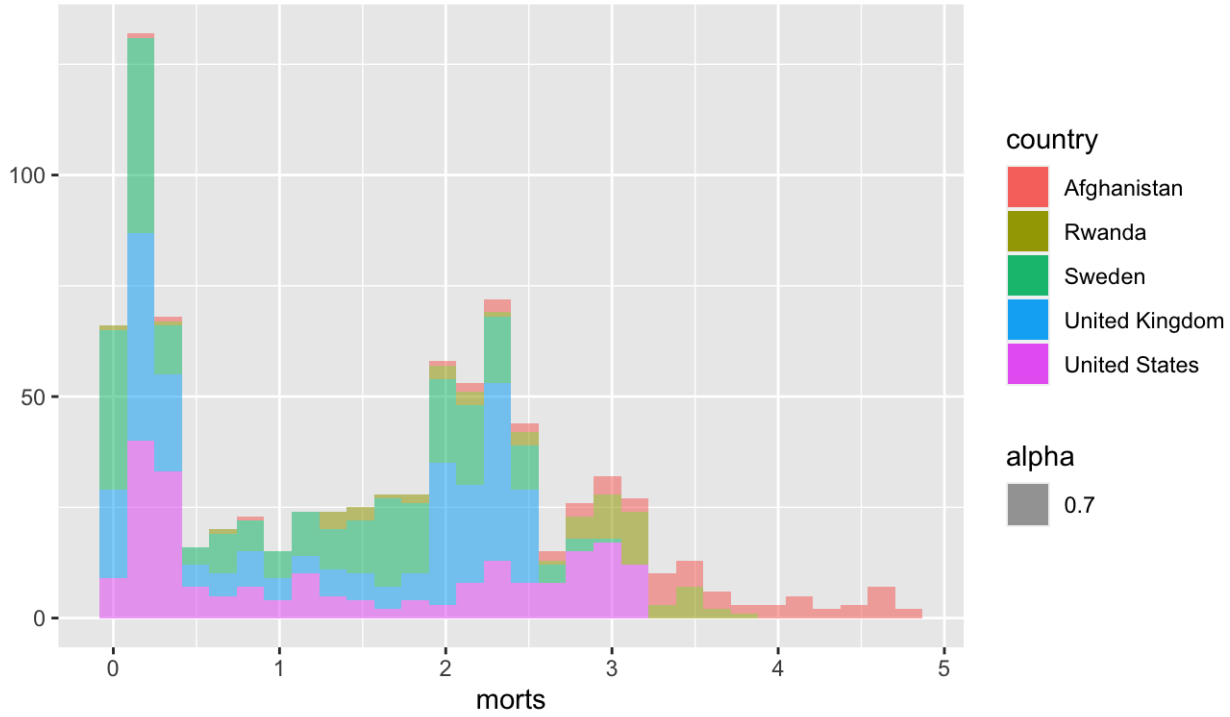
```
qplot(x = morts, fill = factor(country),
      data = sub, geom = c("histogram"))
```



Multiple Histograms

Alpha refers to the opacity of the color, less is more opaque

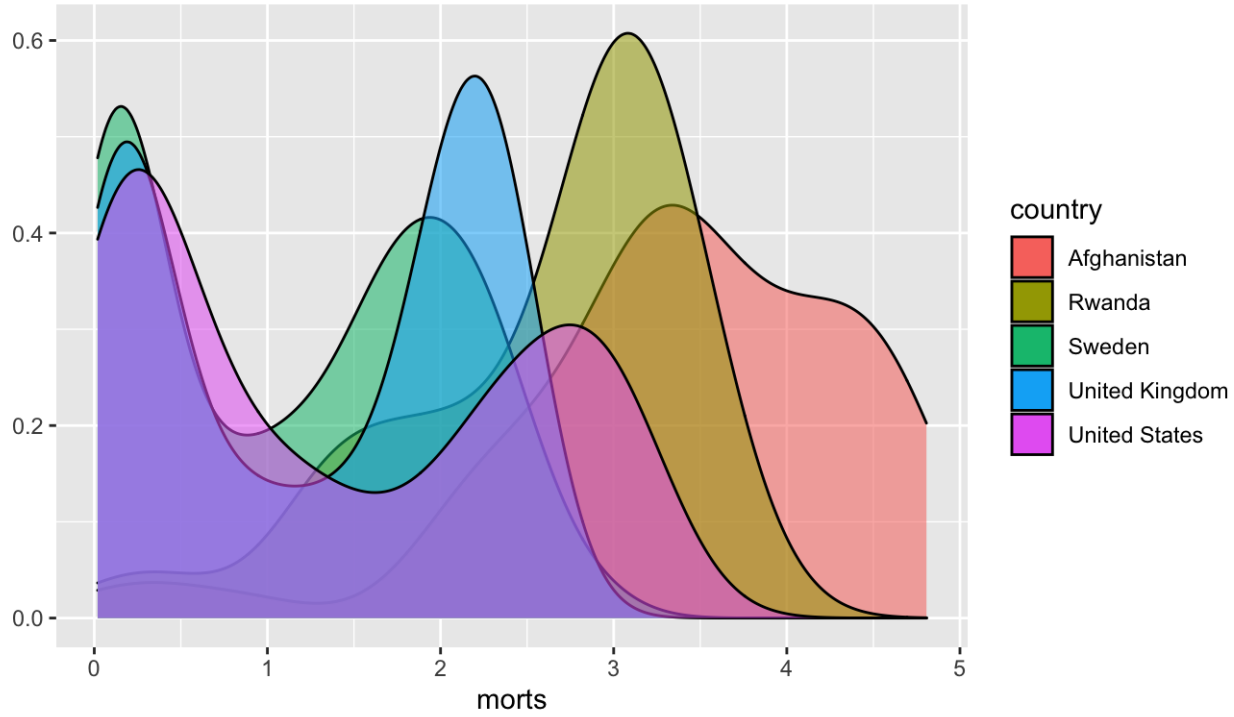
```
qplot(x = morts, fill = country, data = sub,
      geom = c("histogram"), alpha=.7)
```



Multiple Densities

We cold also do densities:

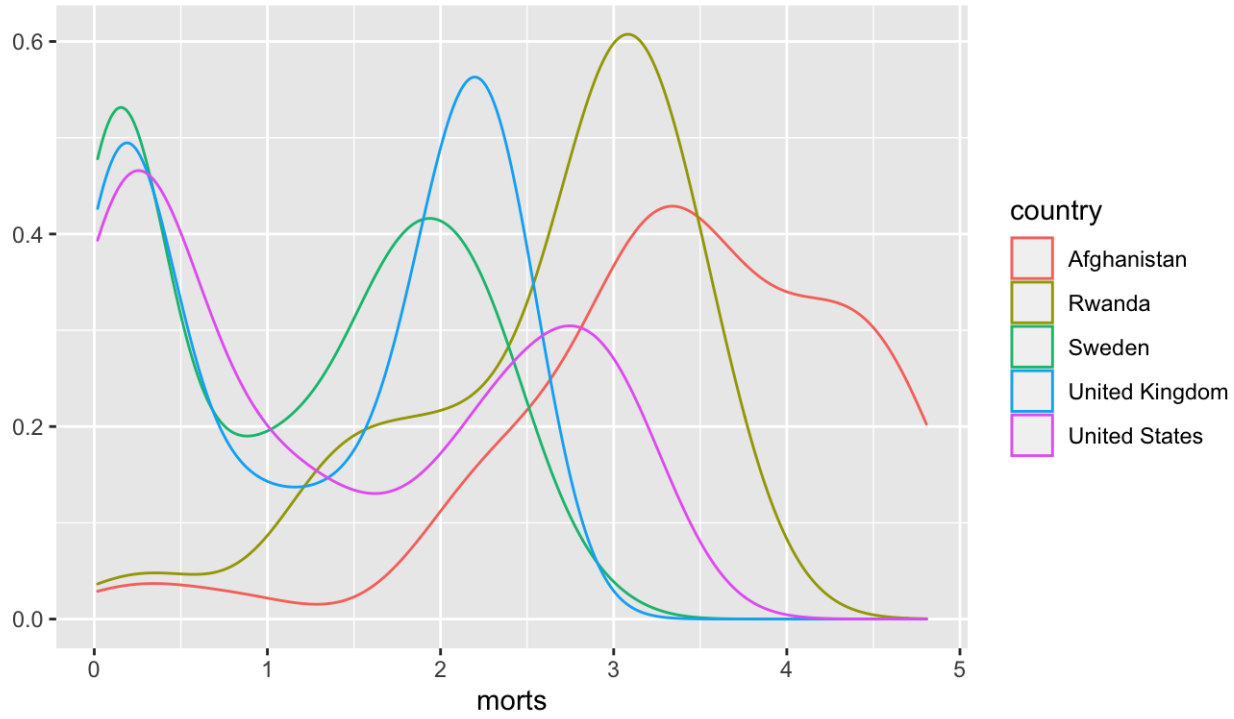
```
qplot(x= morts, fill = country, data = sub,
      geom = c("density"), alpha= .7)  + guides(alpha = FALSE)
```



Multiple Densities

- using `colour` not `fill`:

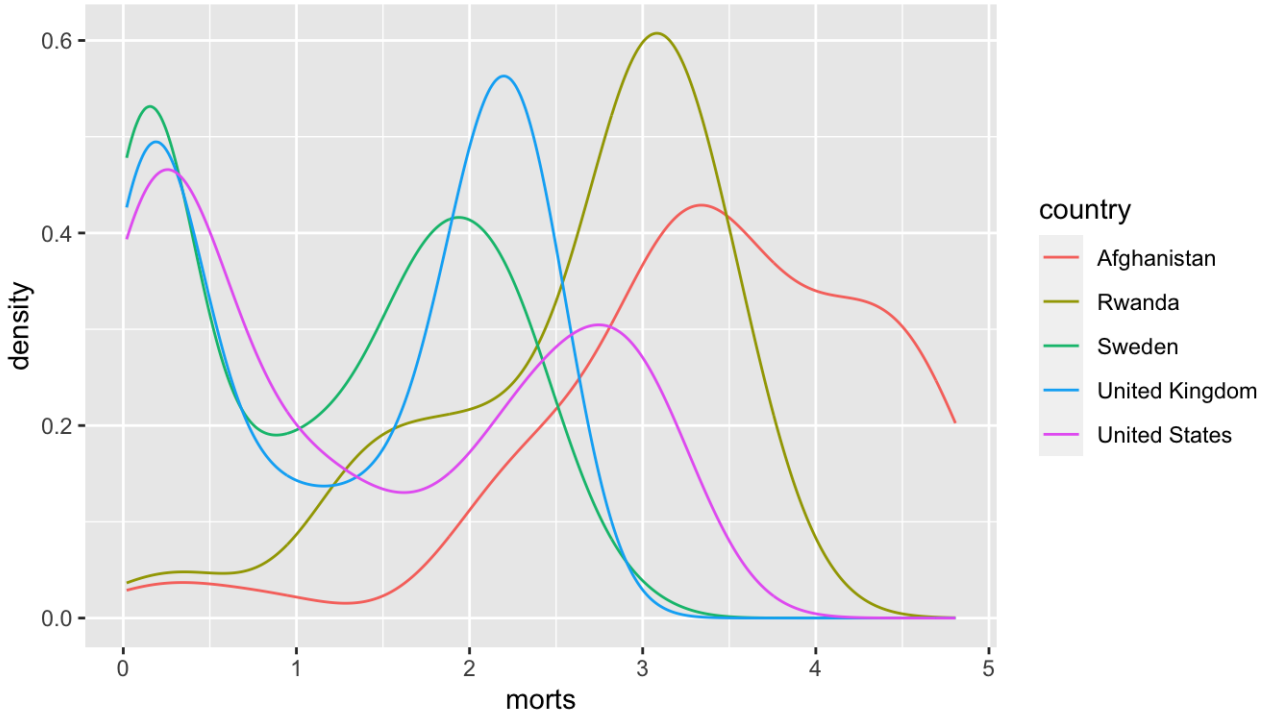
```
qplot(x = morts, colour = country, data = sub,
      geom = c("density"), alpha= .7) + guides(alpha = FALSE)
```



Multiple Densities

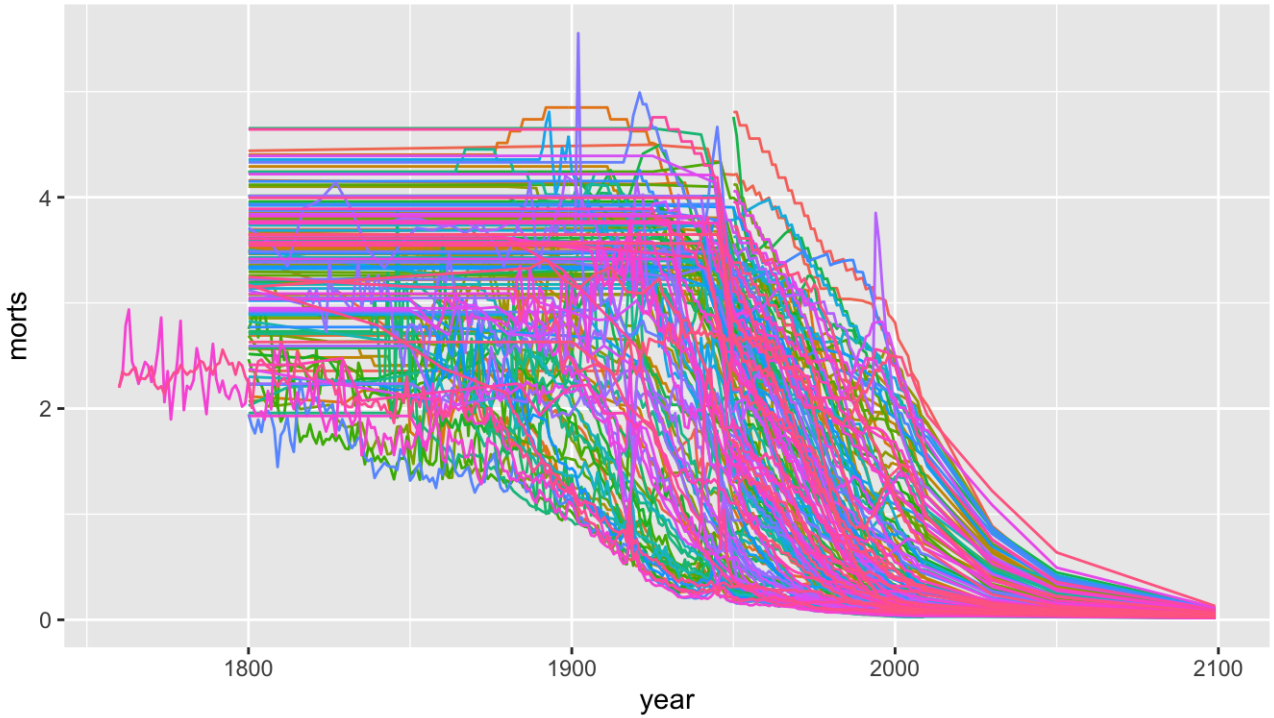
You can take off the lines of the bottom like this

```
ggplot(aes(x = morts, colour = country), data = sub) +
  geom_line(stat = "density")
```



ggplot2

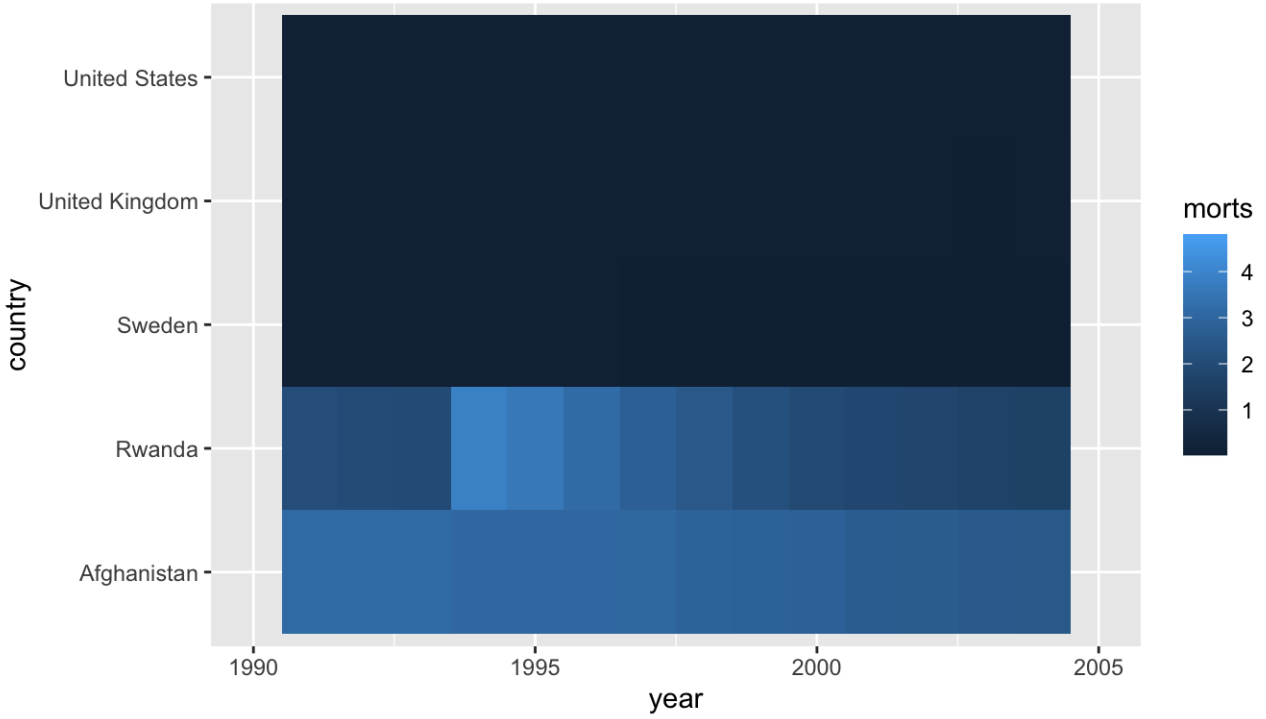
```
qplot(x = year, y = morts, colour = country,
    data = long, geom = "line") + guides(colour = FALSE)
```



ggplot2

Let's try to make it different like base R, a bit. We use `tile` for the geom:

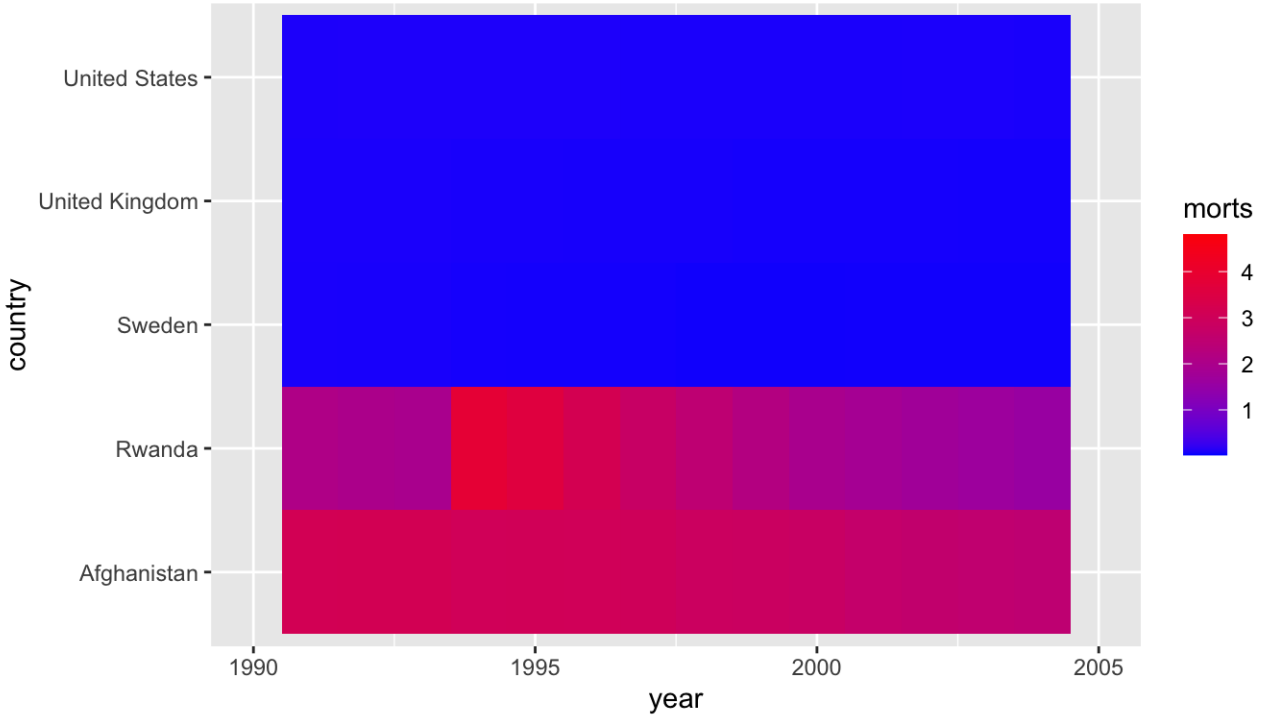
```
qtile = qplot(x = year, y = country, fill = morts, data = sub,
              geom = "tile") + xlim(1990, 2005) + guides(colour = FALSE)
```



ggplot2: changing colors

`scale_fill_gradient` let's us change the colors for the fill:

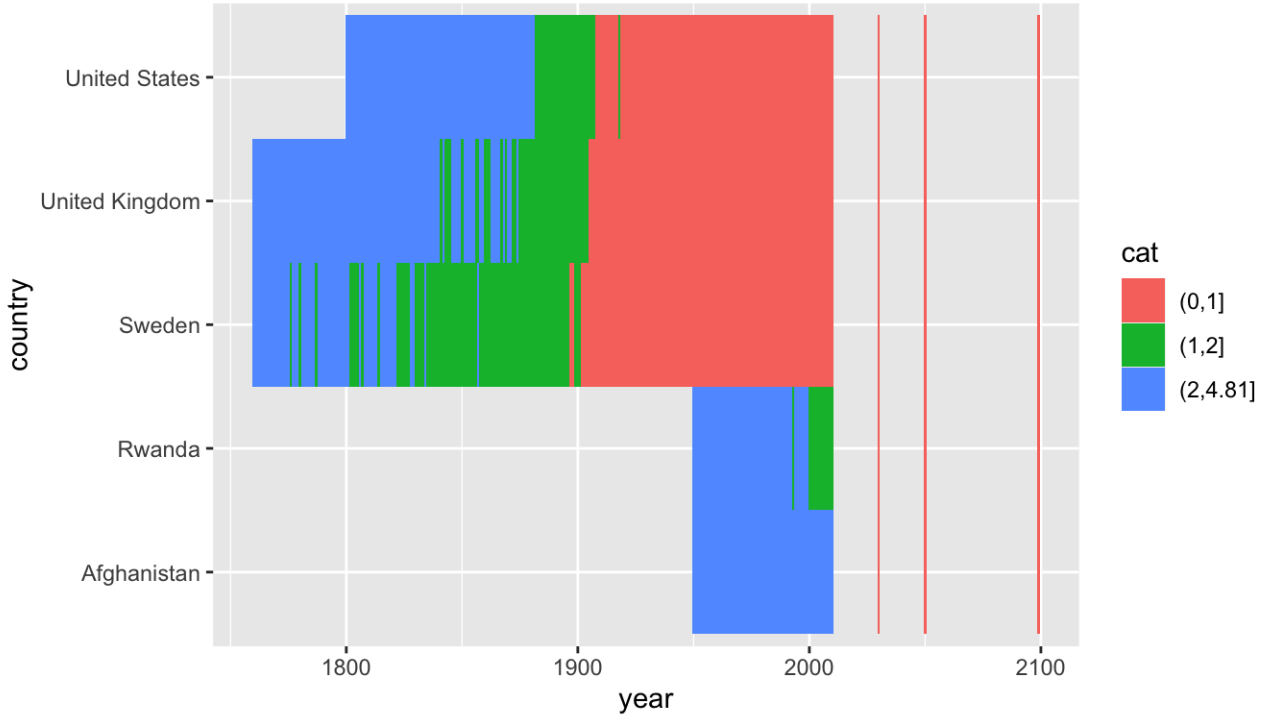
```
qtile + scale_fill_gradient( low = "blue", high = "red")
```



ggplot2

Let's try categories.

```
sub$cat = cut(sub$morts, breaks = c(0, 1, 2, max(sub$morts)))
q2 = qplot(x = year, y = country, fill = cat, data = sub, geom = "tile") +
  guides(colour = FALSE)
```



Colors

It's actually pretty hard to make a good color palette. Luckily, smart and artistic people have spent a lot more time thinking about this. The result is the `RColorBrewer` package

`RColorBrewer::display.brewer.all()` will show you all of the palettes available. You can even print it out and keep it next to your monitor for reference.

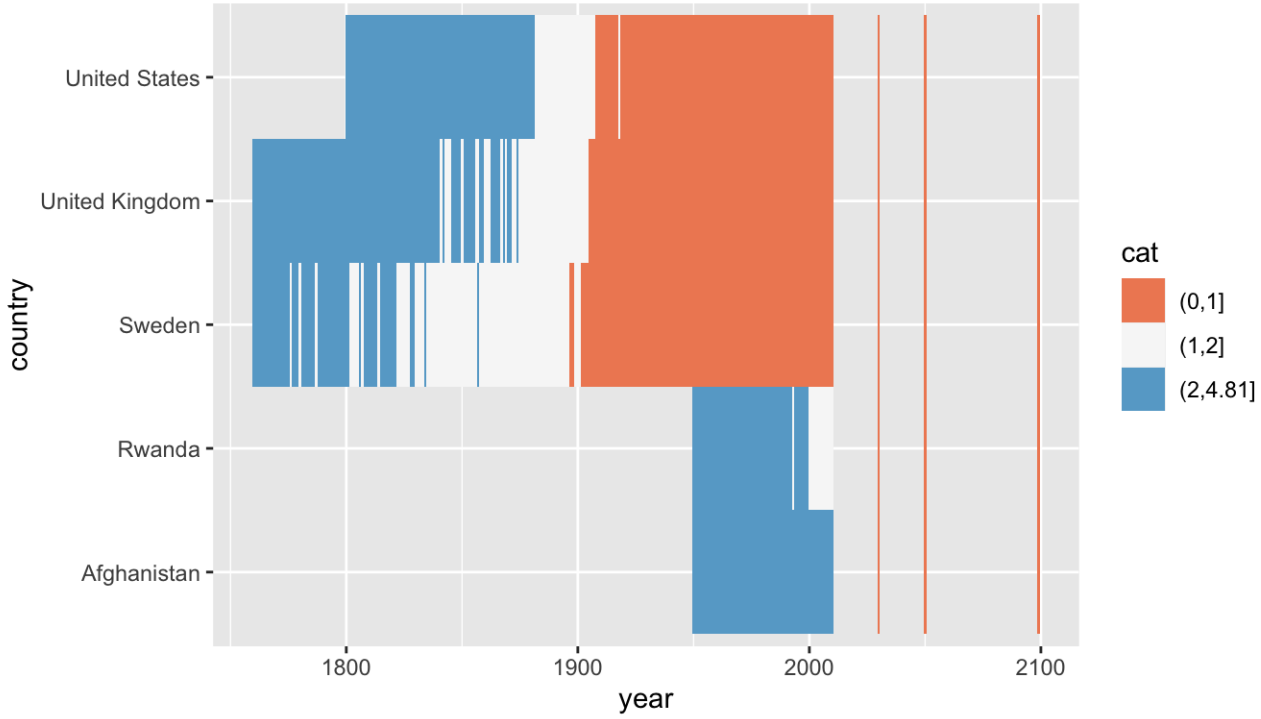
The help file for `brewer.pal()` gives you an idea how to use the package.

You can also get a "sneak peek" of these palettes at: http://colorbrewer2.org/ . You would provide the number of levels or classes of your data, and then the type of data: sequential, diverging, or qualitative. The names of the `RColorBrewer` palettes are the string after 'pick a color scheme:'

ggplot2: changing colors

`scale_fill_brewer` will allow us to use these palettes conveniently

```
q2 + scale_fill_brewer( type = "div", palette =  "RdBu" )
```



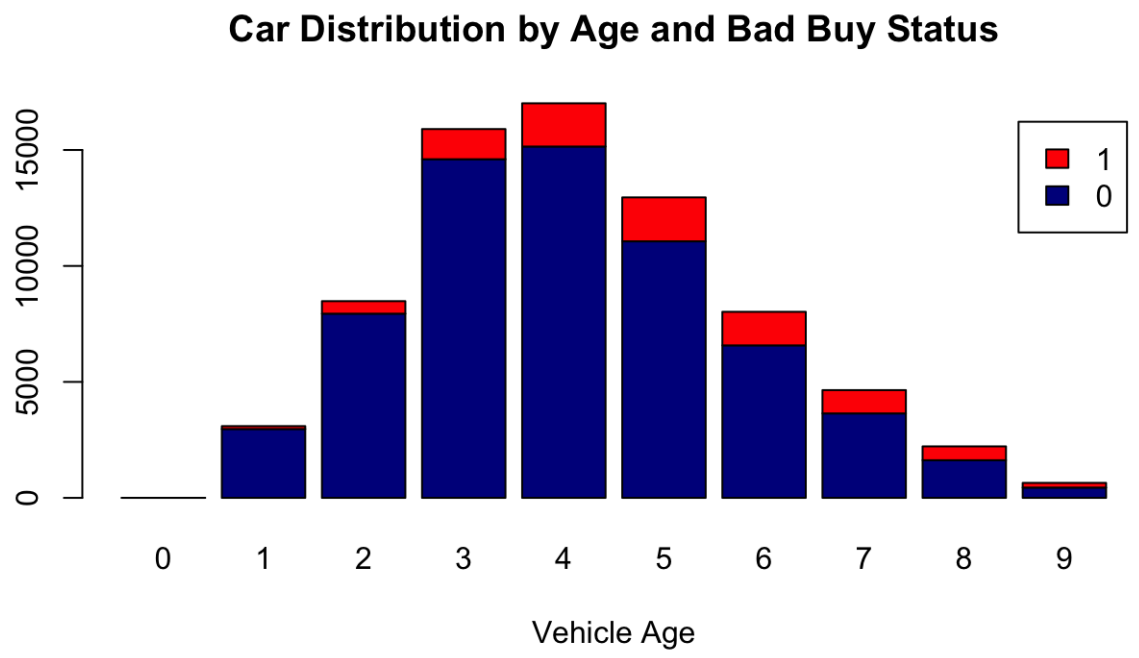
Bar Plots with a table

```
cars = read_csv(
  "https://jhudatascience.org/intro_to_r/data/kaggleCarAuction.csv",
  col_types = cols(VehBCost = col_double()))
counts <- table(cars$IsBadBuy, cars$VehicleAge)
```

Bar Plots

- Stacked Bar Charts are sometimes wanted to show distributions of data

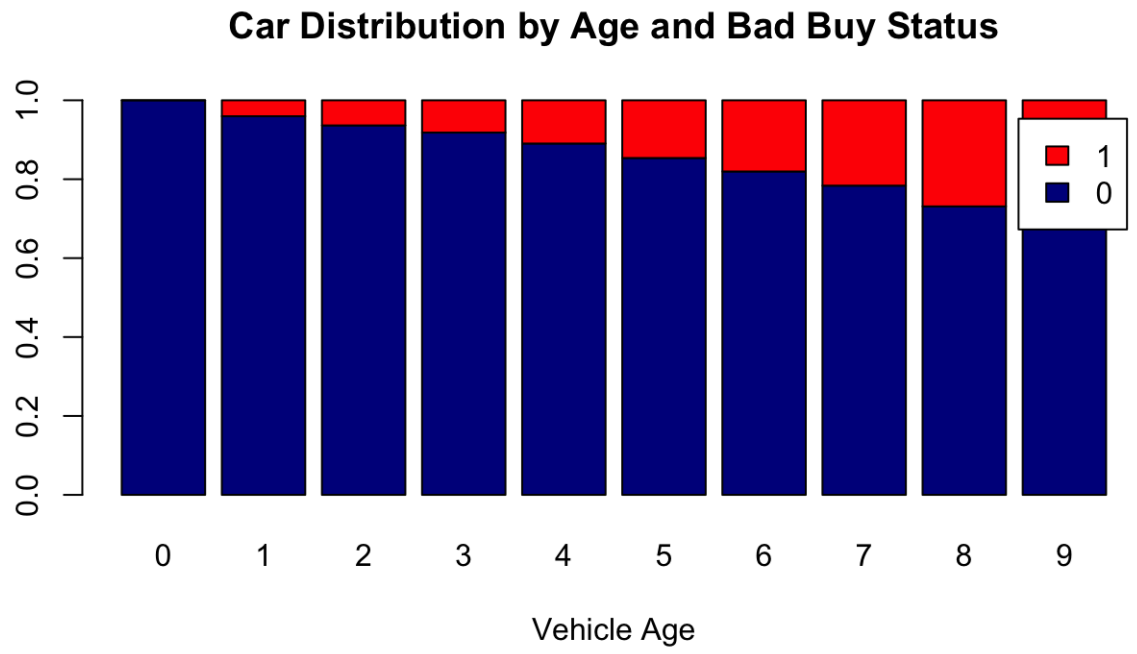
```
barplot(counts, main="Car Distribution by Age and Bad Buy Status", xlab="Vehic
```



Bar Plots

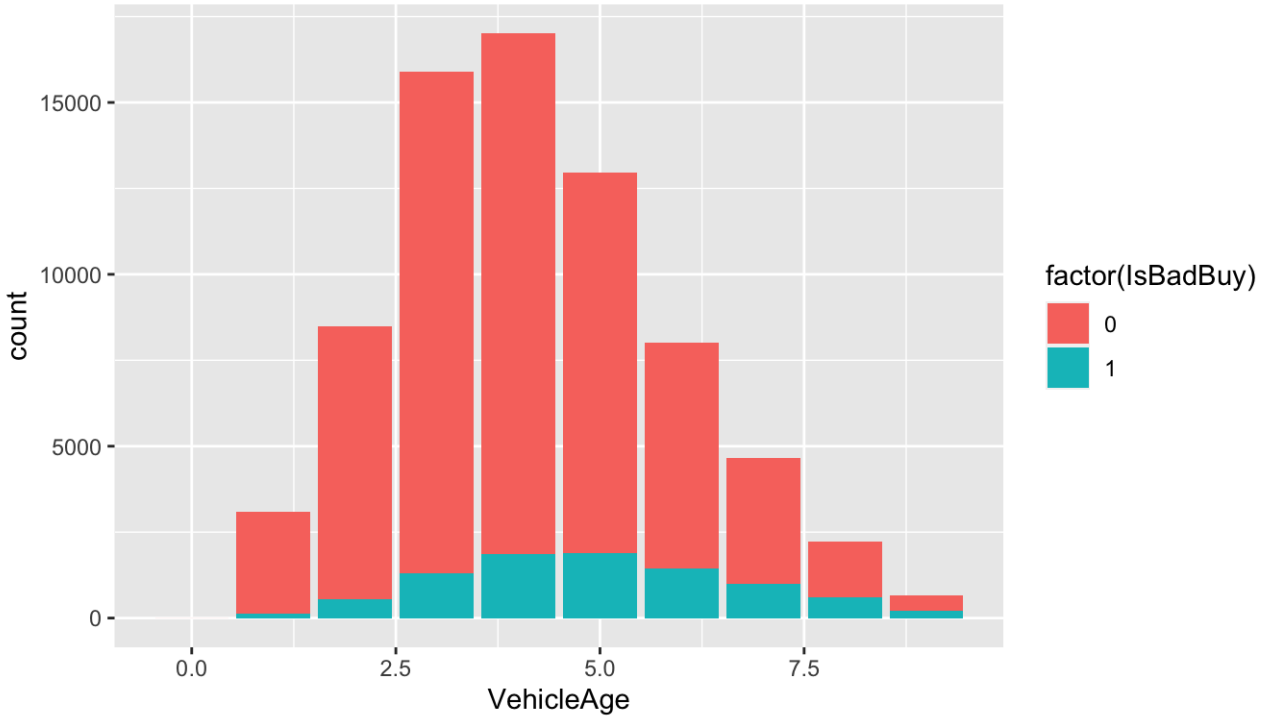
`prop.table` allows you to convert a table to proportions (depends on margin - either row percent or column percent)

```
## Use percentages (column percentages)
barplot(prop.table(counts, 2),
        main = "Car Distribution by Age and Bad Buy Status",
  xlab="Vehicle Age", col=c("darkblue","red"),
    legend = rownames(counts))
```



Bar Plots

```
ggplot(aes(fill = factor(IsBadBuy), x = VehicleAge),
       data = cars) + geom_bar()
```



Normalized Stacked Bar charts

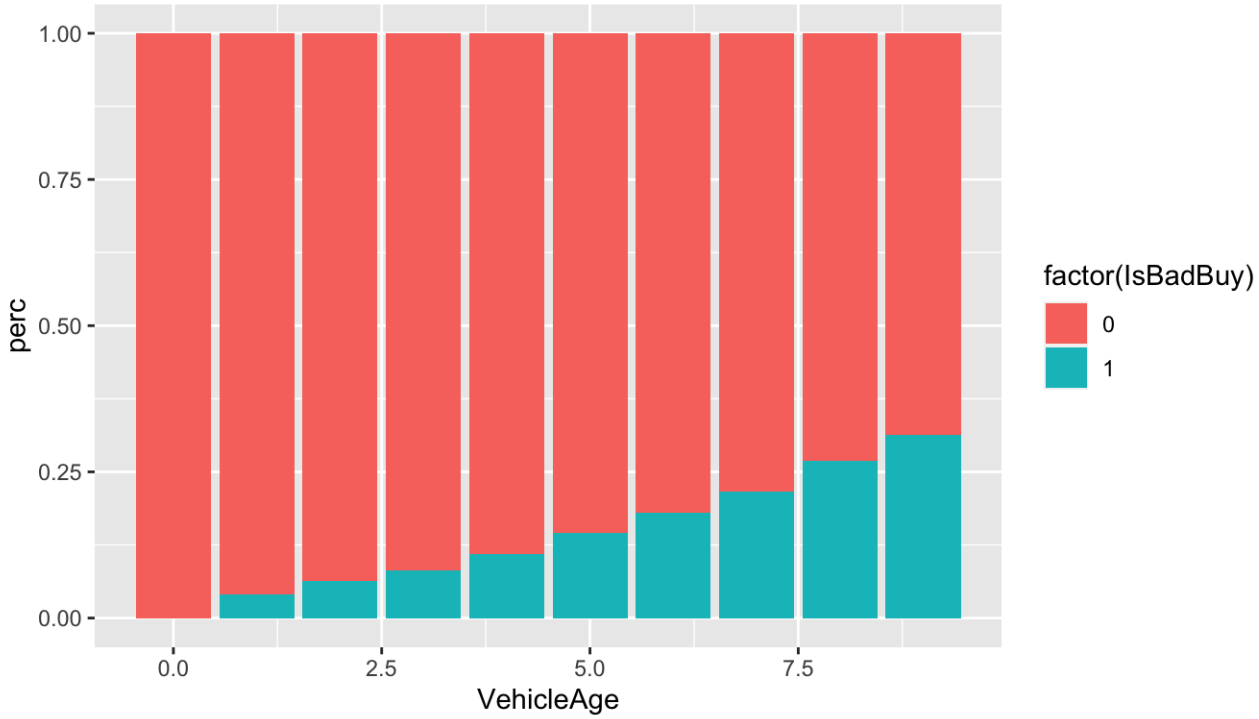
- we must calculate percentages on our own

```
perc = cars %>%
  group_by(IsBadBuy, VehicleAge) %>%
  tally() %>% ungroup
head(perc)
```

```
# A tibble: 6 x 3
  IsBadBuy VehicleAge     n
     <dbl>      <dbl> <int>
1        0          0     2
2        0          1  2969
3        0          2  7942
4        0          3 14601
5        0          4 15149
6        0          5 11061
```

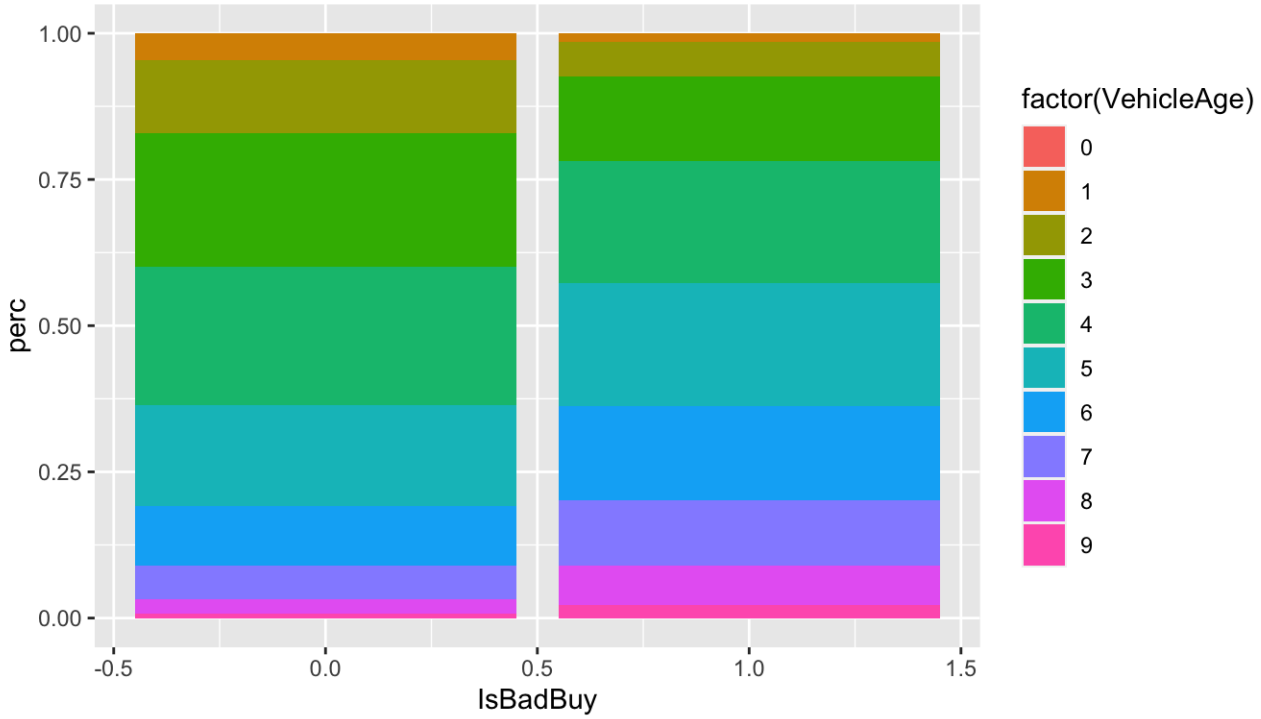
Each Age adds to 1

```
perc_is_bad = perc %>%
  group_by(VehicleAge) %>% mutate(perc = n / sum(n))
ggplot(aes(fill = factor(IsBadBuy),
           x = VehicleAge,
           y = perc),
       data = perc_is_bad) + geom_bar(stat = "identity")
```



Each Bar adds to 1 for bad buy or not

```
perc_yr = perc %>%
  group_by(IsBadBuy) %>% mutate(perc = n / sum(n))
ggplot(aes(fill = factor(VehicleAge),
           x = IsBadBuy,
           y = perc),
       data = perc_yr) + geom_bar(stat = "identity")
```



ggplot2

Useful links:

- http://docs.ggplot2.org/0.9.3/index.html
- http://www.cookbook-r.com/Graphs/

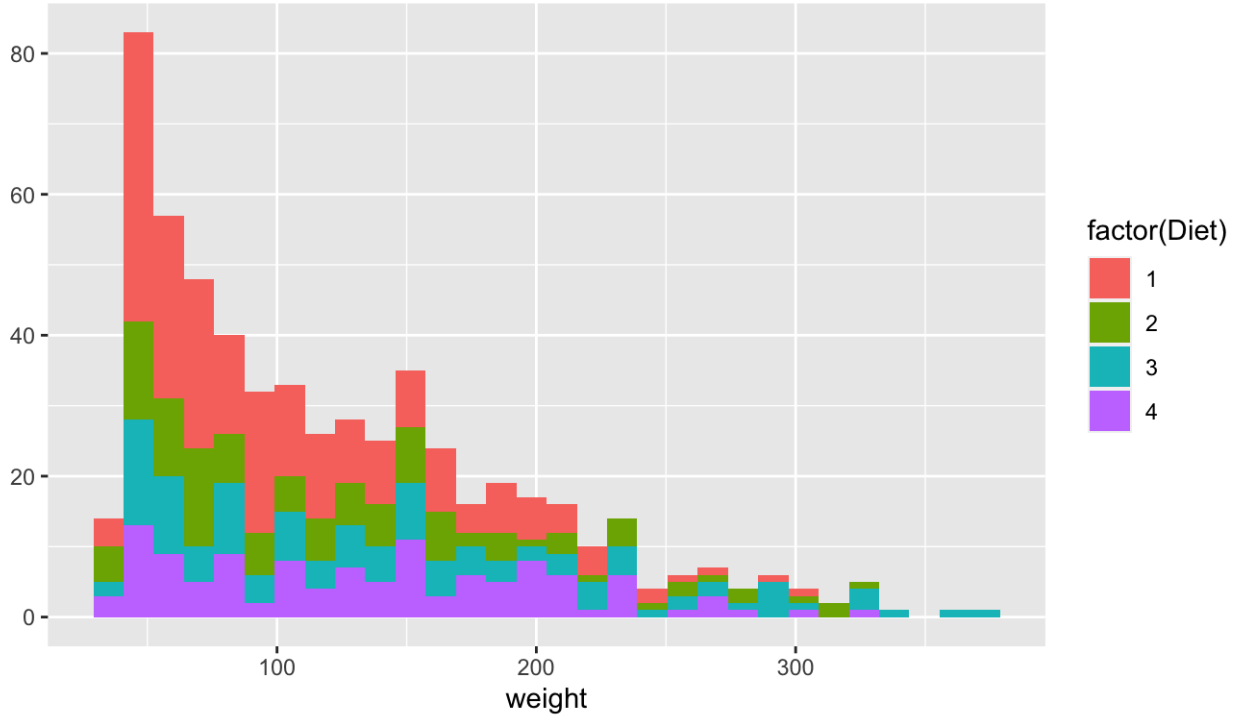
Website

Website

ggplot examples on a second dataset

Multiple Histograms

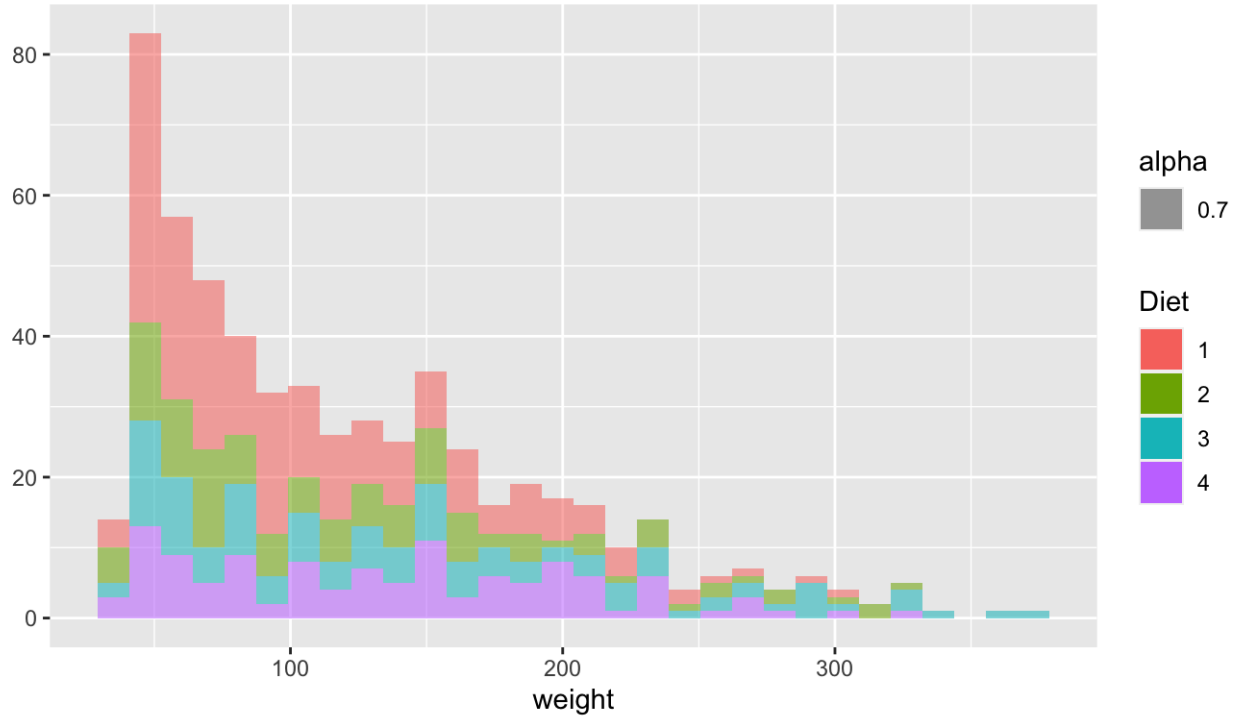
```
qplot(x = weight,
      fill = factor(Diet),
      data = ChickWeight,
      geom = c("histogram"))
```



Multiple Histograms

Alpha refers tot he opacity of the color, less is

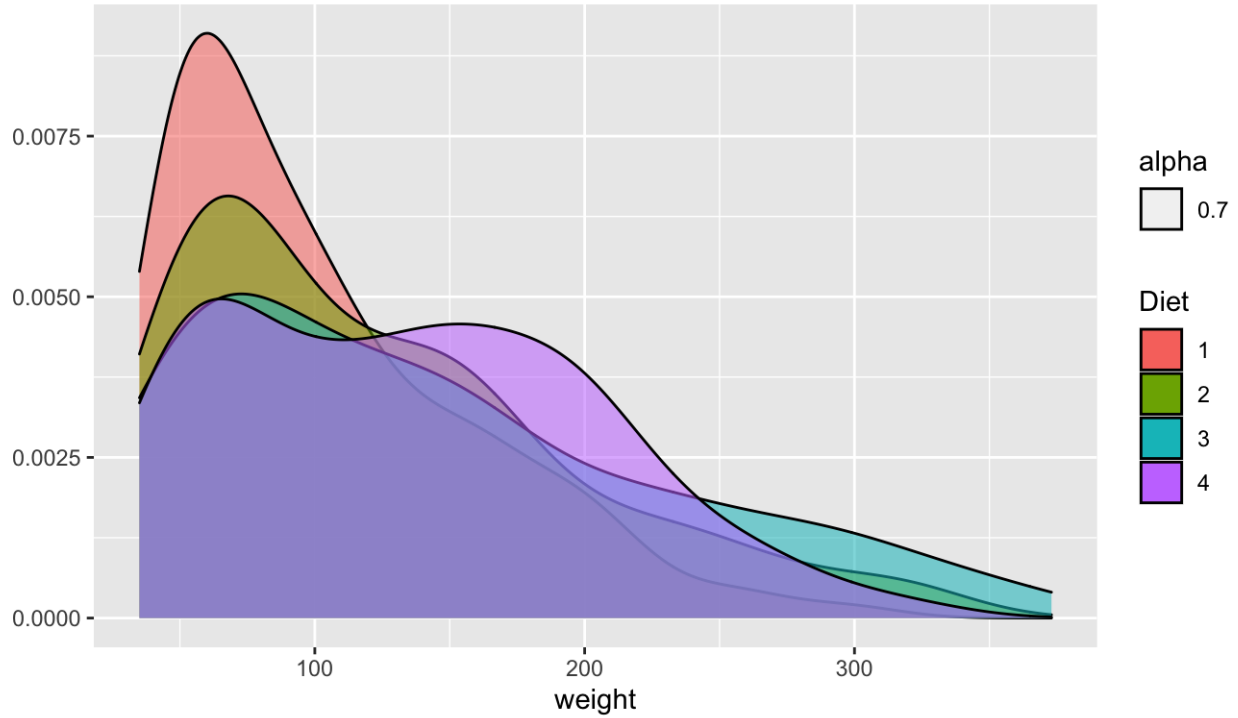
```
qplot(x = weight, fill = Diet, data = ChickWeight,
      geom = c("histogram"), alpha=.7)
```



Multiple Densities

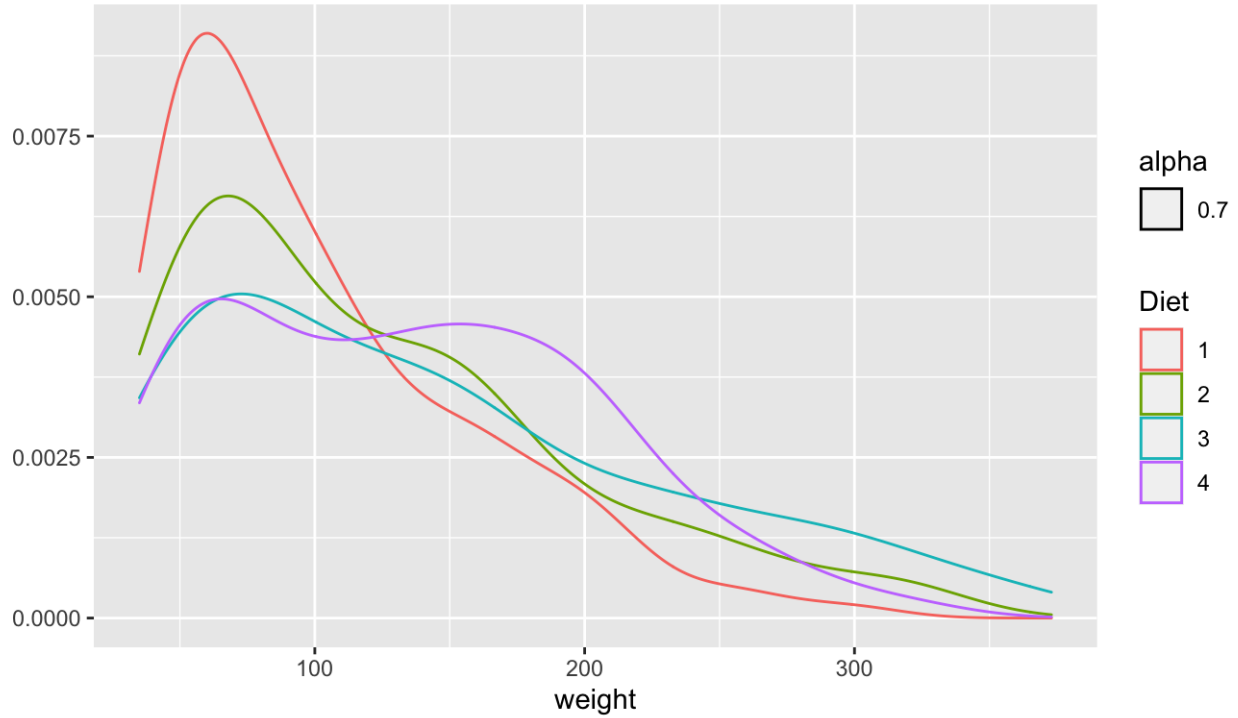
We cold also do densities

```
qplot(x= weight, fill = Diet, data = ChickWeight,
      geom = c("density"), alpha= .7)
```



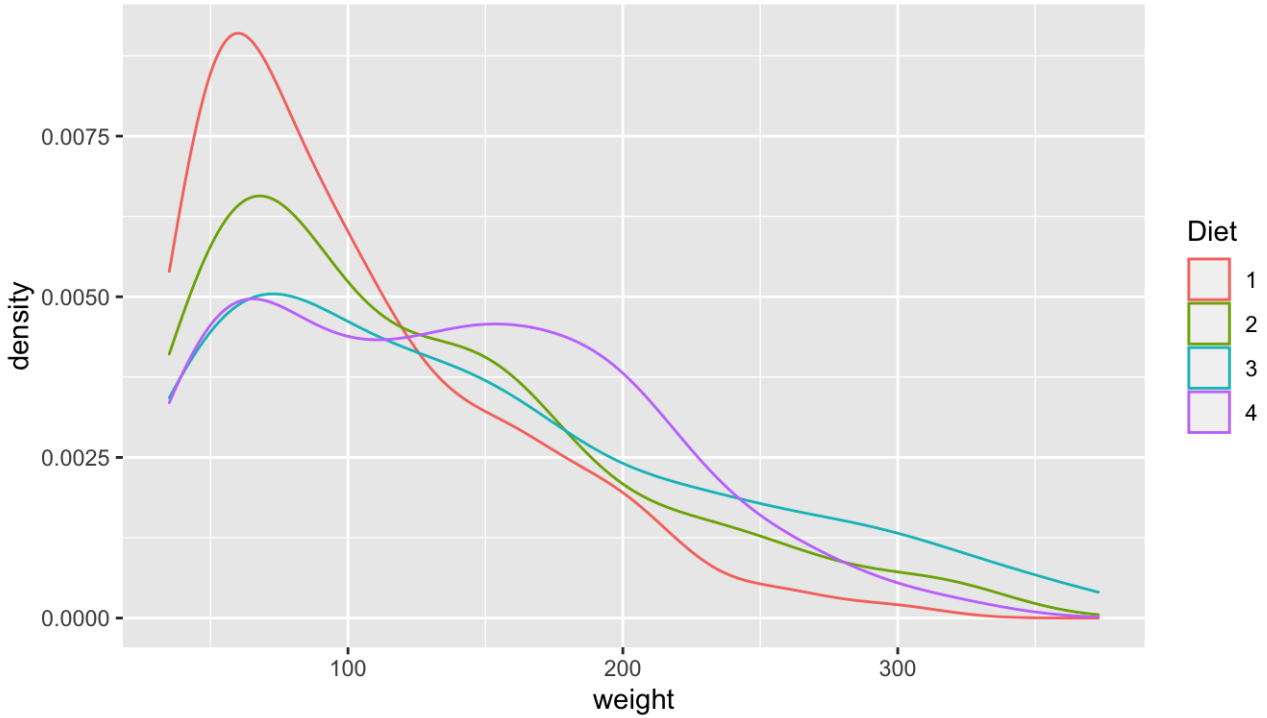
Multiple Densities

```
qplot(x= weight, colour = Diet, data = ChickWeight,
      geom = c("density"), alpha=.7)
```



Multiple Densities

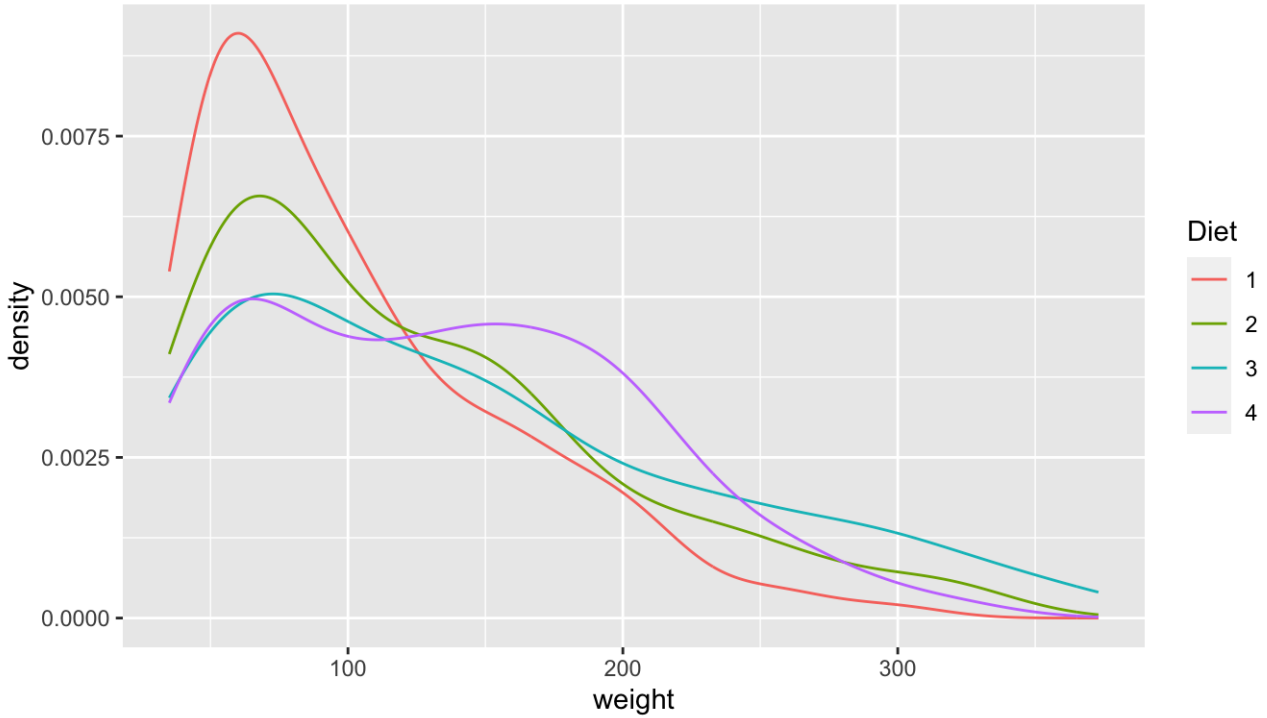
```
ggplot(aes(x= weight, colour = Diet),
  data = ChickWeight) + geom_density(alpha=.7)
```



Multiple Densities

You can take off the lines of the bottom like this

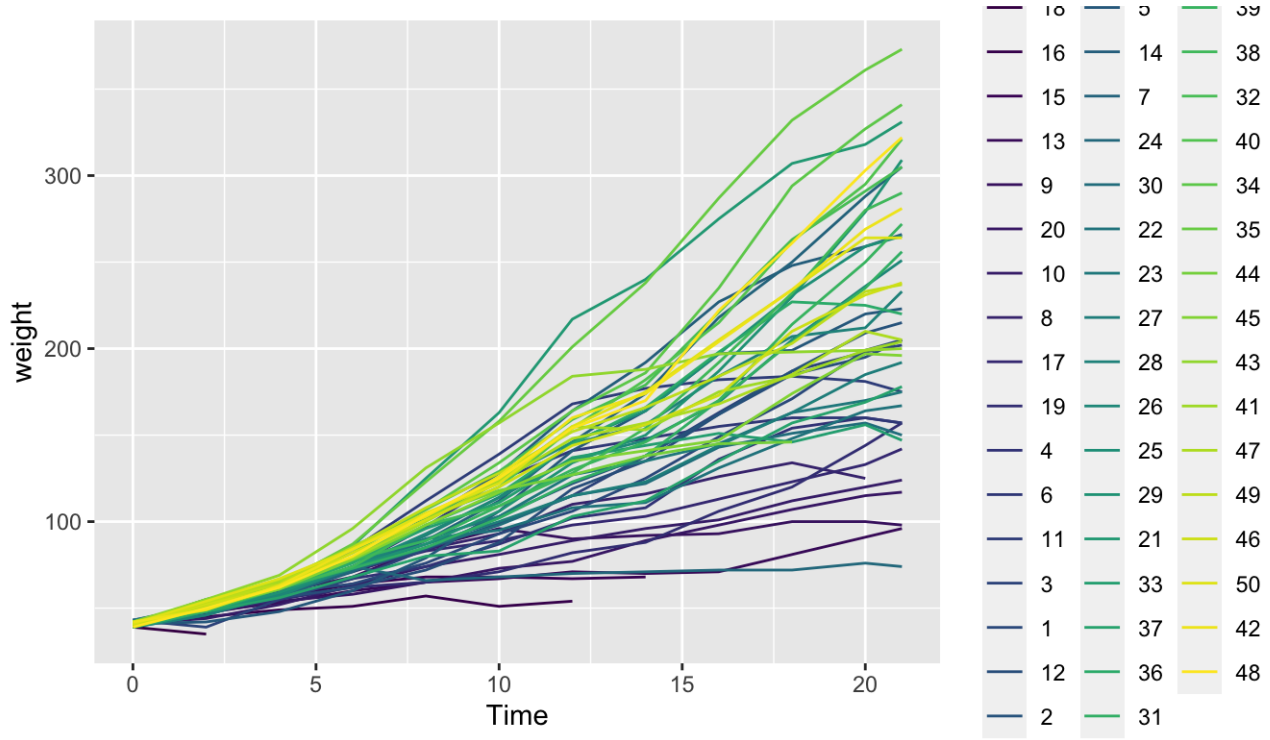
```
ggplot(aes(x = weight, colour = Diet), data = ChickWeight) +
  geom_line(stat = "density")
```



Spaghetti plot

We can make a spaghetti plot by telling ggplot we want a “line”, and each line is colored by Chick.

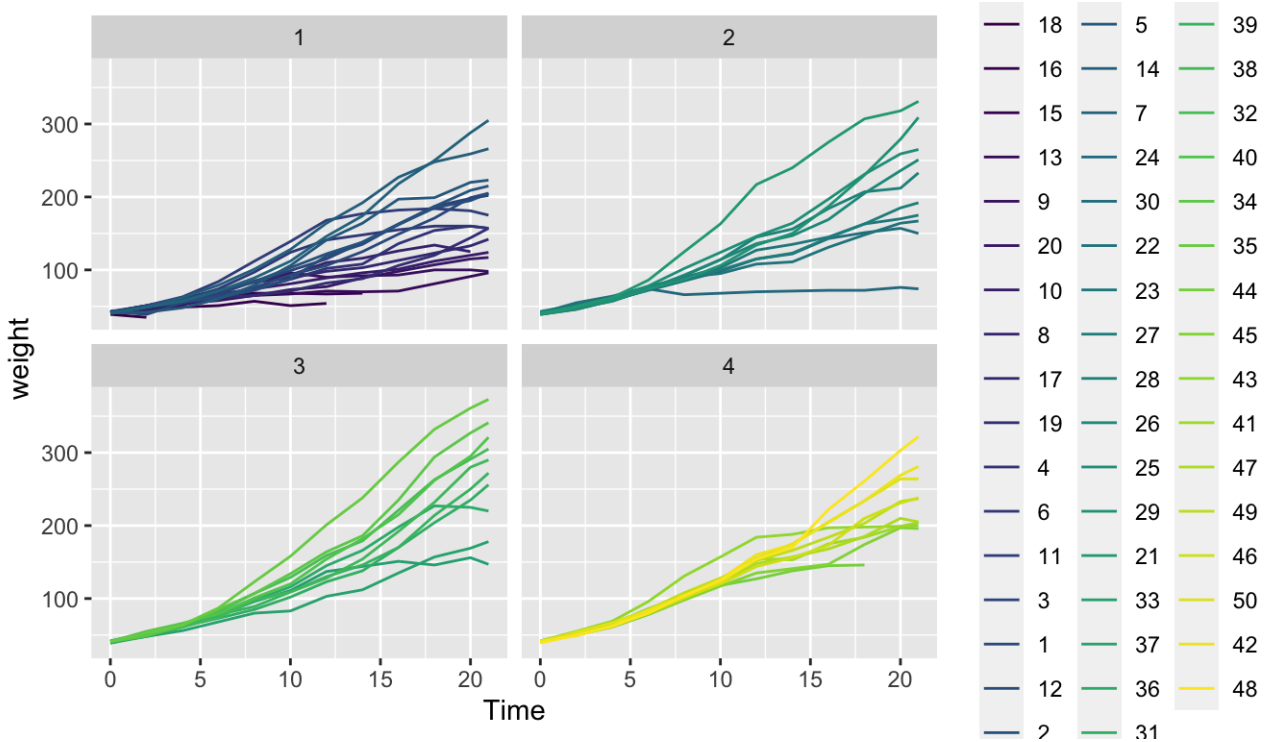
```
qplot(x=Time, y=weight, colour = factor(Chick),
      data = ChickWeight, geom = "line")
```



Spaghetti plot: Facets

In ggplot2, if you want separate plots for something, these are referred to as facets.

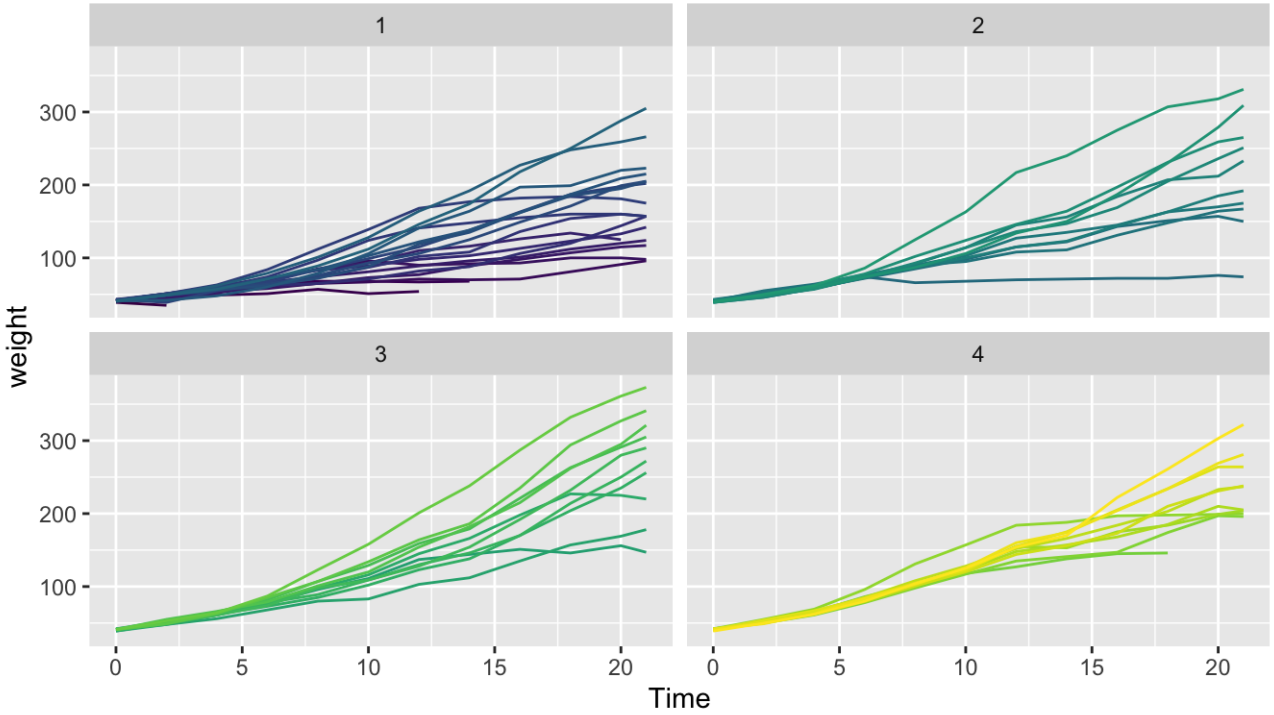
```
qplot(x = Time, y = weight, colour = factor(Chick),
      facets = ~Diet, data = ChickWeight, geom = "line")
```



Spaghetti plot: Facets

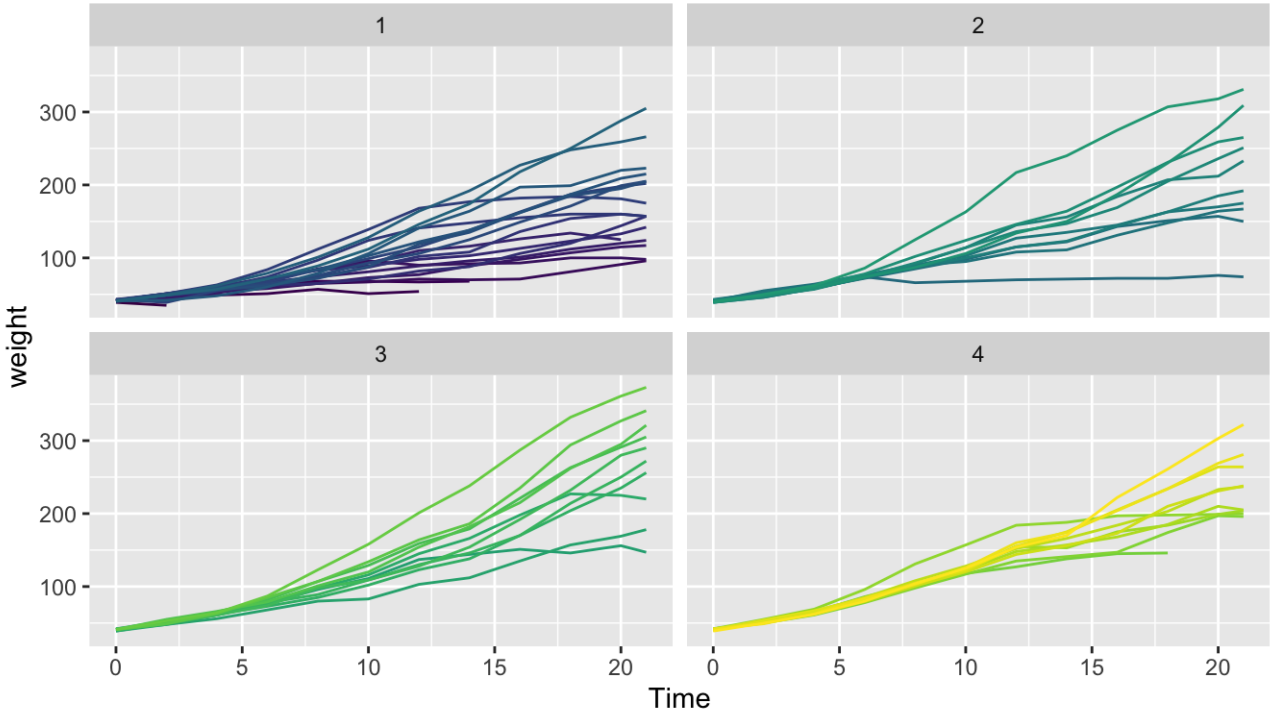
We can turn off the legend (referred to a “guide” in ggplot2). (Note - there is different syntax with the +)

```
qplot(x=Time, y=weight, colour = factor(Chick),
      facets = ~ Diet,  data = ChickWeight,
        geom = "line") + guides(colour=FALSE)
```



Spaghetti plot: Facets

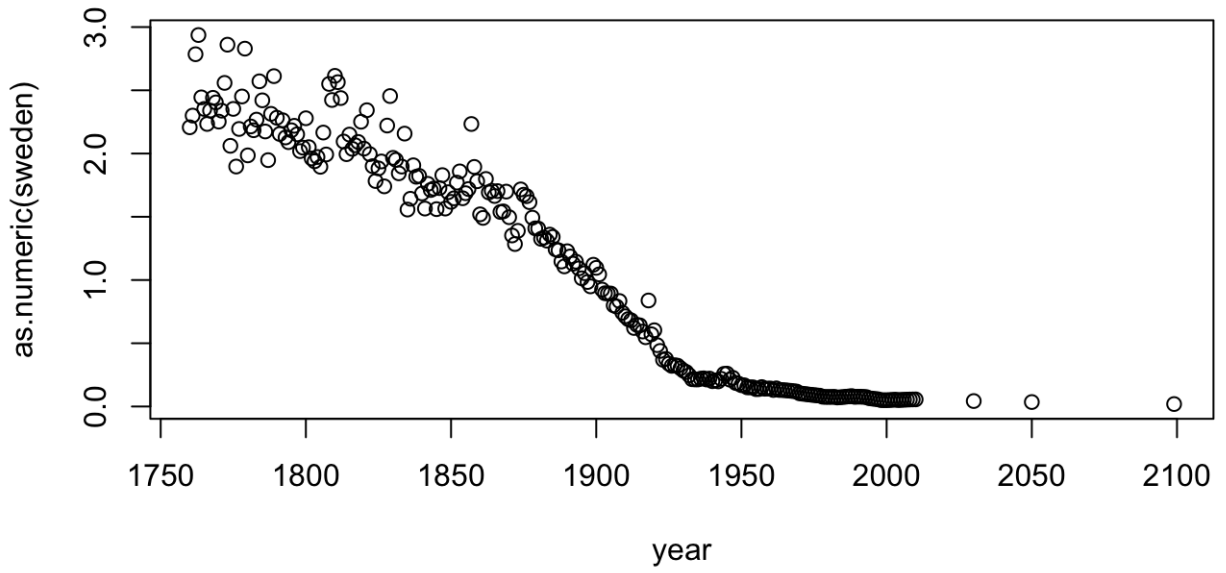
```
ggplot(aes(x = Time, y = weight, colour = factor(Chick)),
    data = ChickWeight) + geom_line() +
    facet_wrap(facets = ~Diet) + guides(colour = FALSE)
```



Base Graphics - explore on your own

Basic Plots

```
library(dplyr)
sweden = mort %>%
  filter(country == "Sweden") %>%
  select(-country)
year = as.numeric(colnames(sweden))
plot(as.numeric(sweden) ~ year)
```



Base Graphics parameters

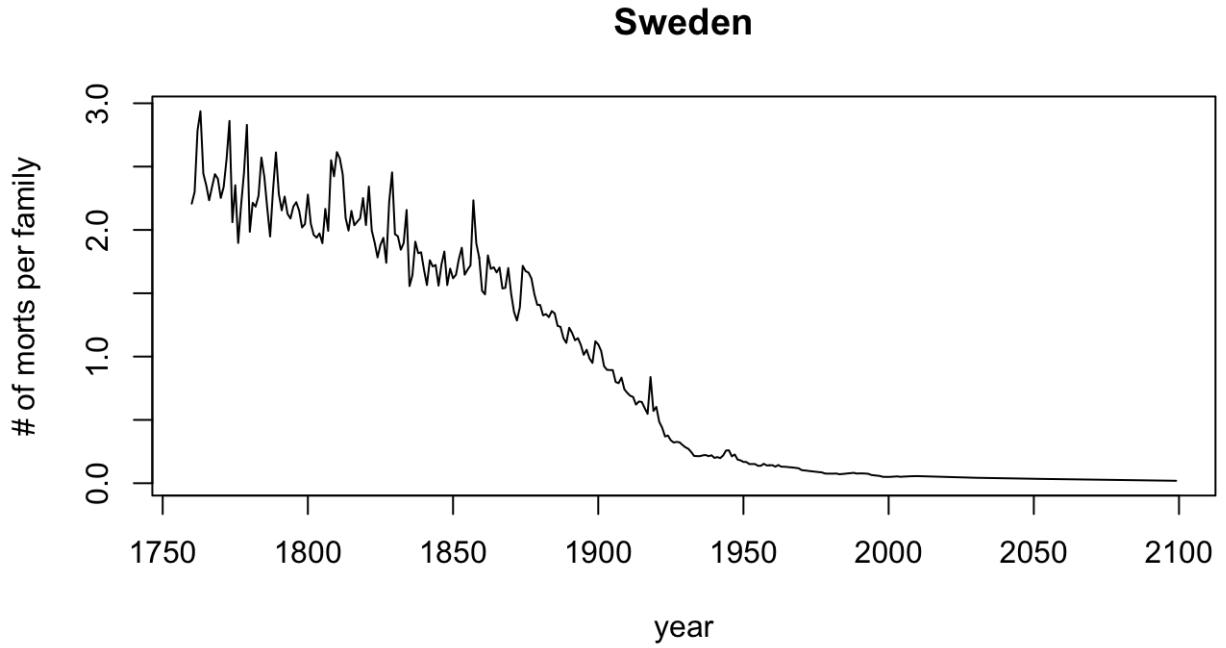
Set within most plots in the base 'graphics' package:

- pch = point shape, http://voteview.com/symbols_pch.htm
- cex = size/scale
- xlab, ylab = labels for x and y axes
- main = plot title
- lwd = line density
- col = color
- cex.axis, cex.lab, cex.main = scaling/sizing for axes marks, axes labels, and title

Basic Plots

The y-axis label isn't informative, and we can change the label of the y-axis using `ylab` (`xlab` for x), and `main` for the main title/label.

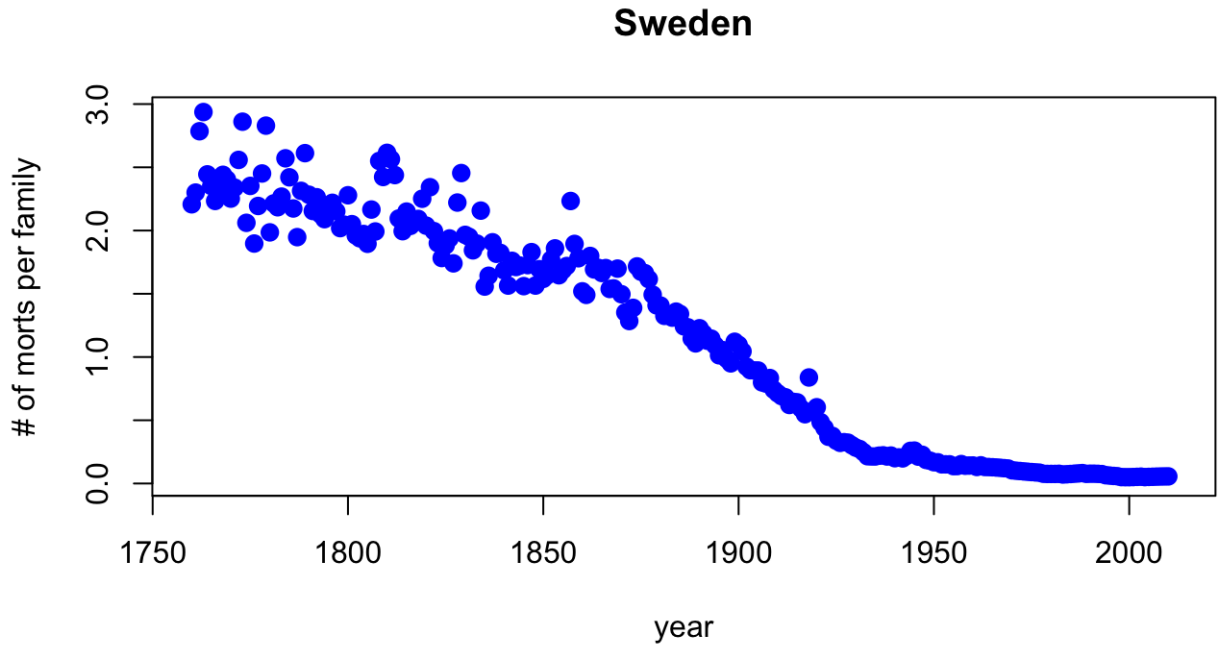
```
plot(as.numeric(sweden) ~ year,
      ylab = "# of morts per family", main = "Sweden", type = "l")
```



Basic Plots

Let's drop any of the projections and keep it to year 2012, and change the points to blue.

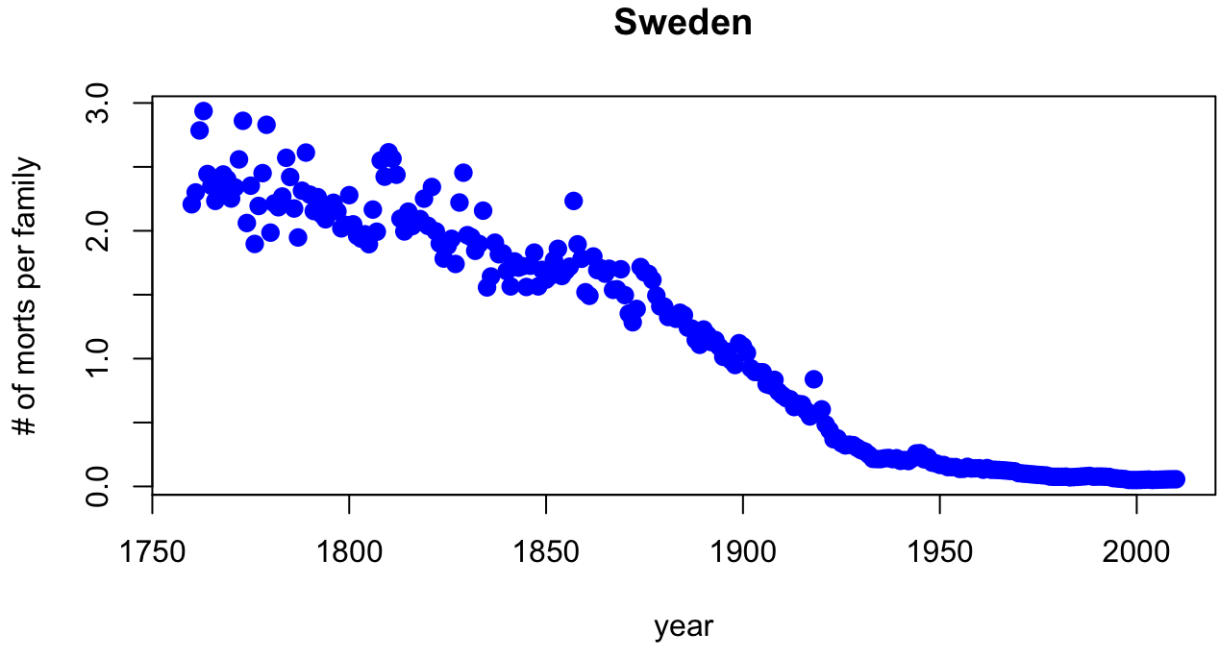
```
plot(as.numeric(sweden) ~ year,
      ylab = "# of morts per family", main = "Sweden",
     xlim = c(1760,2012), pch = 19, cex=1.2,col="blue")
```



Basic Plots

You can also use the `subset` argument in the `plot()` function, only when using formula notation:

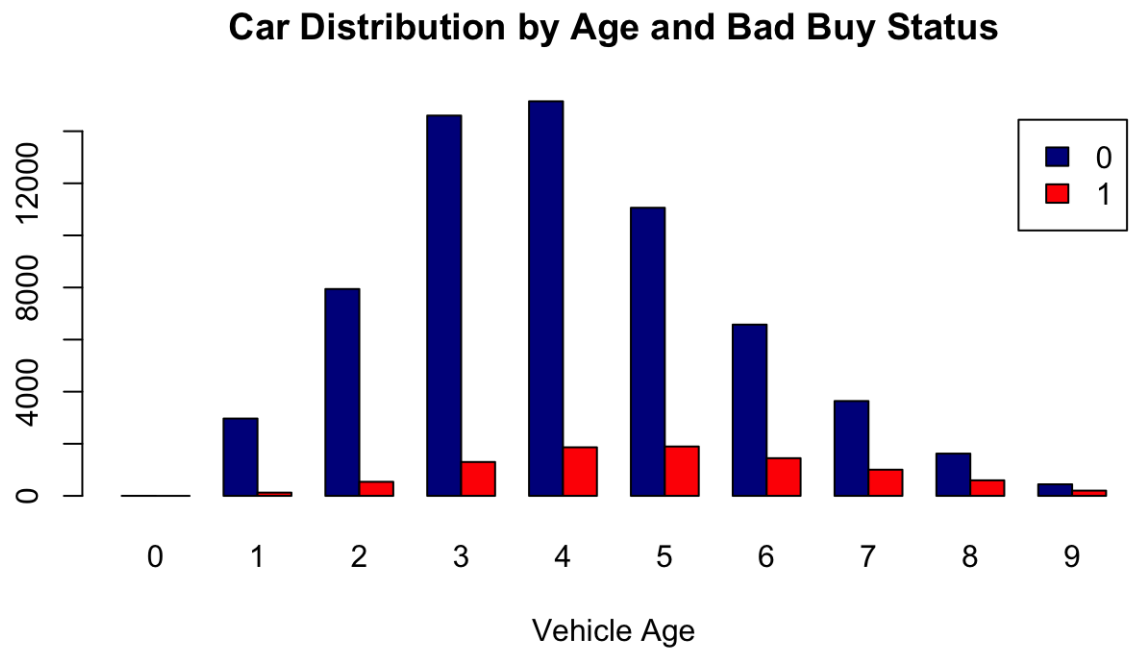
```
plot(as.numeric(sweden) ~ year,
      ylab = "# of morts per family", main = "Sweden",
     subset = year < 2015, pch = 19, cex=1.2,col="blue")
```



Bar Plots

Using the `beside` argument in `barplot`, you can get side-by-side barplots.

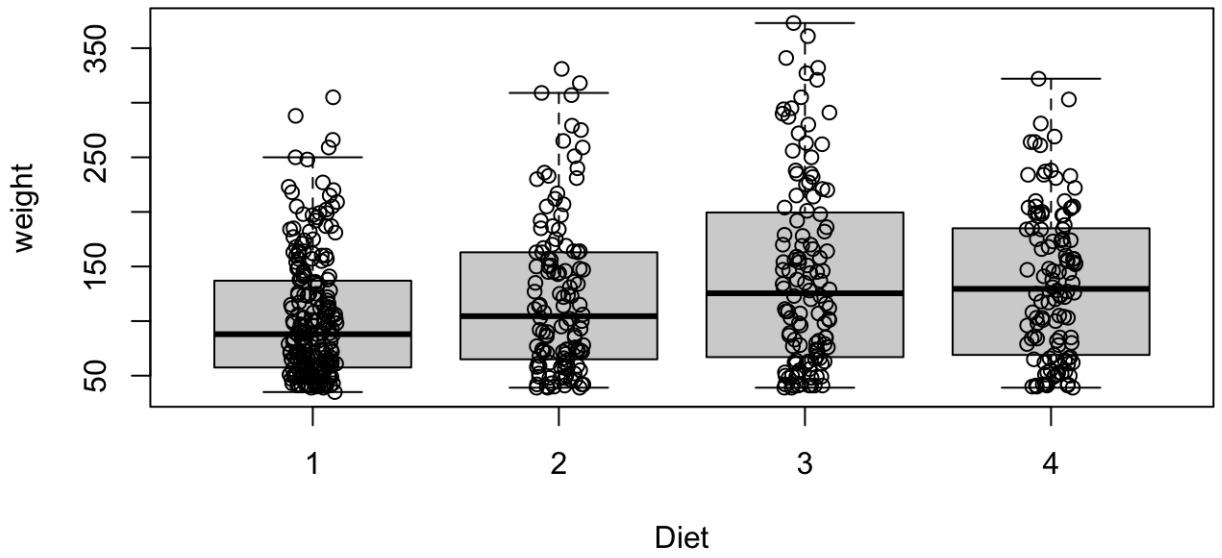
```
# Stacked Bar Plot with Colors and Legend
barplot(counts, main="Car Distribution by Age and Bad Buy Status",
  xlab="Vehicle Age", col=c("darkblue","red"),
    legend = rownames(counts), beside=TRUE)
```



Boxplots, revisited

These are one of my favorite plots. They are way more informative than the barchart + antenna…

```
boxplot(weight ~ Diet, data=ChickWeight, outline=FALSE)
points(ChickWeight$weight ~ jitter(as.numeric(ChickWeight$Diet),0.5))
```



Formulas

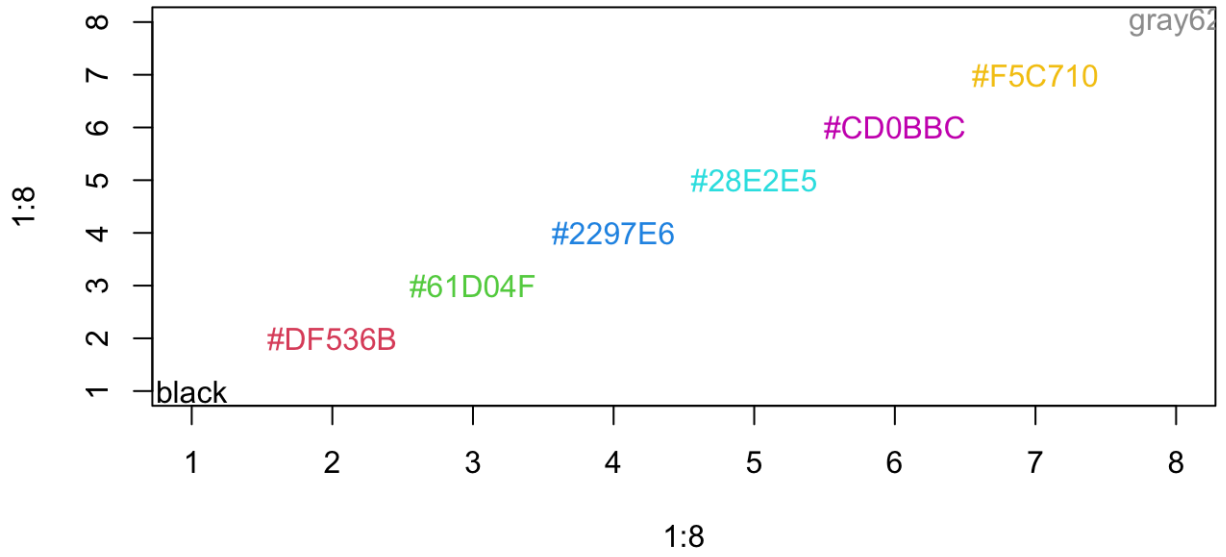
Formulas have the format of `y ~ x` and functions taking formulas have a `data` argument where you pass the data.frame. You don't need to use `$` or referencing when using formulas:

```
boxplot(weight ~ Diet, data=ChickWeight, outline=FALSE)
```

Colors

R relies on color 'palettes'.

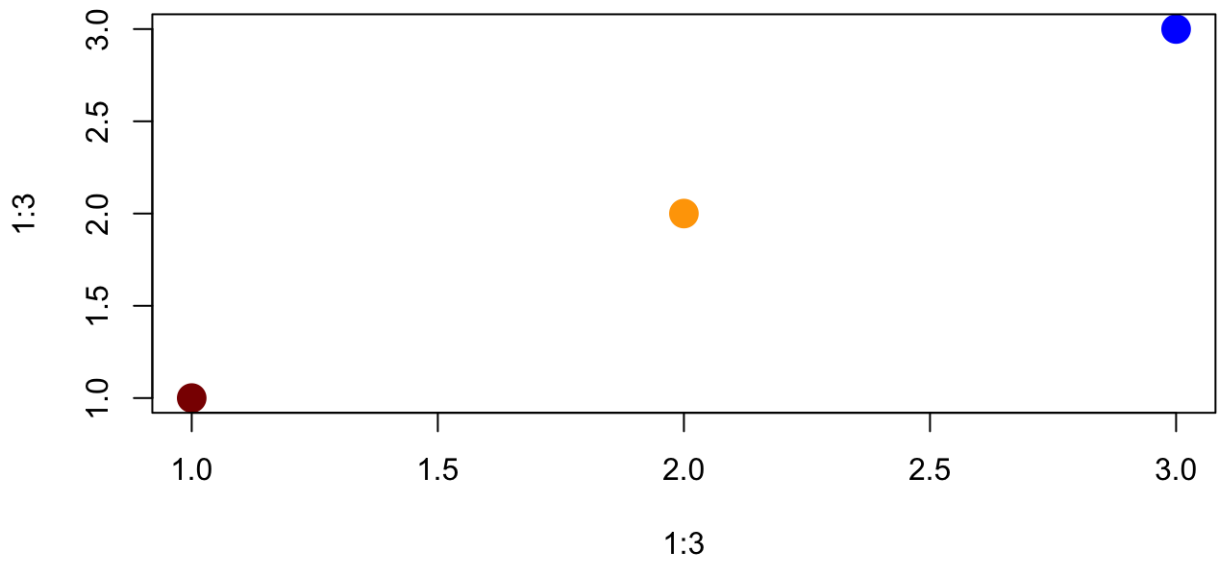
```
palette("default")
plot(1:8, 1:8, type="n")
text(1:8, 1:8, lab = palette(), col = 1:8)
```



Colors

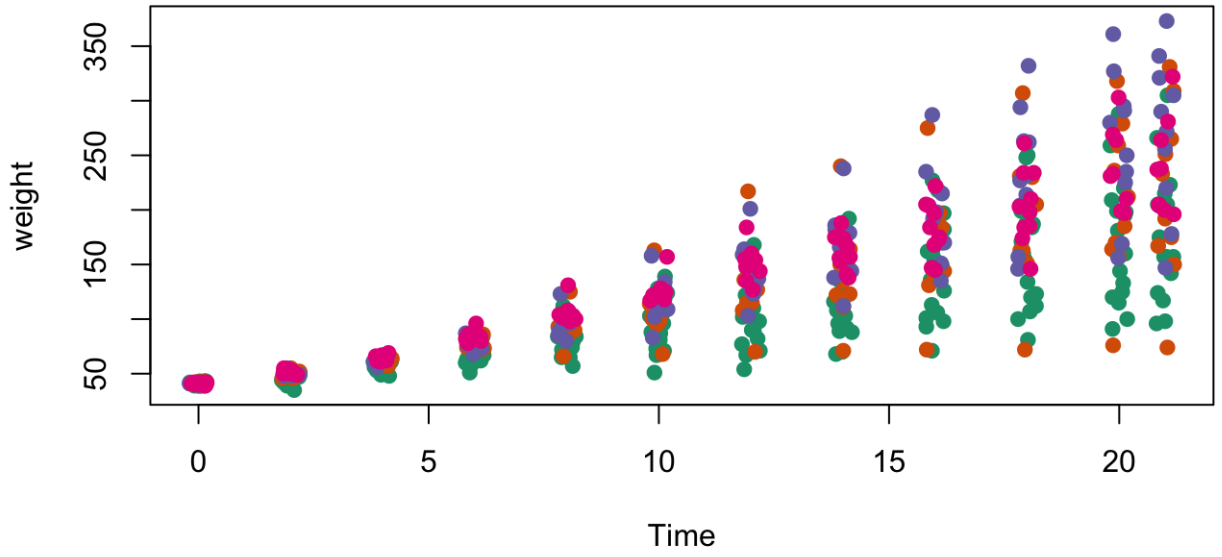
The default color palette is pretty bad, so you can try to make your own.

```
palette(c("darkred","orange","blue"))
plot(1:3,1:3,col=1:3,pch =19,cex=2)
```



Colors

```
library(RColorBrewer)
palette(brewer.pal(5,"Dark2"))
plot(weight ~ jitter(Time,amount=0.2),data=ChickWeight,
     pch = 19,  col = Diet,xlab="Time")
```



Adding legends

The legend() command adds a legend to your plot. There are tons of arguments to pass it.

x, y=NULL: this just means you can give (x,y) coordinates, or more commonly just give x, as a character string:
“top”,“bottom”,“topleft”,“bottomleft”,“topright”,“bottomright”.

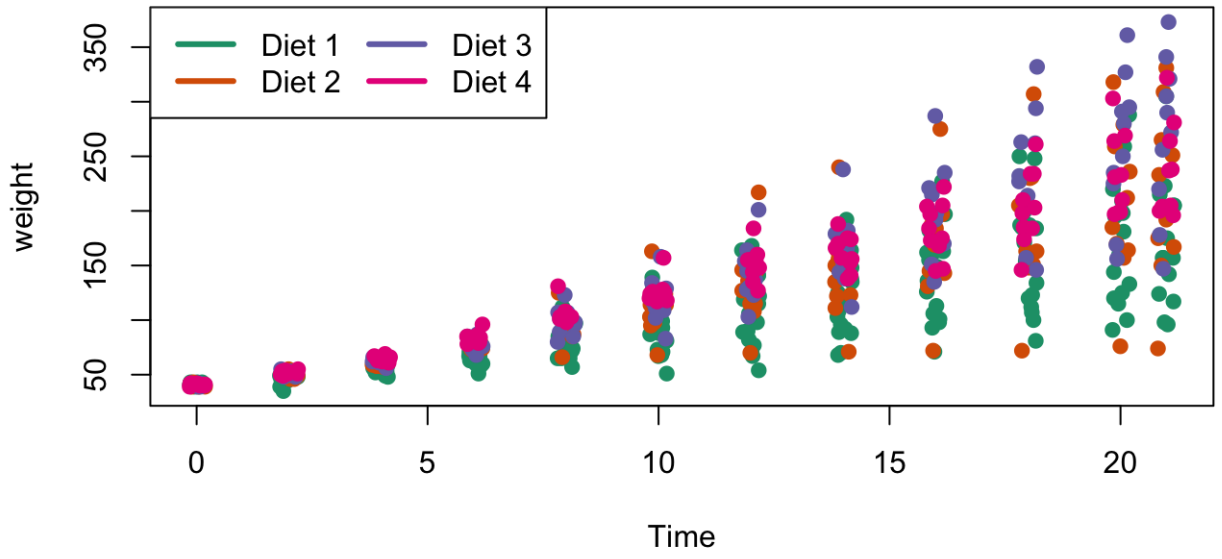
legend: unique character vector, the levels of a factor

pch, lwd: if you want points in the legend, give a pch value. if you want lines, give a lwd value.

col: give the color for each legend level

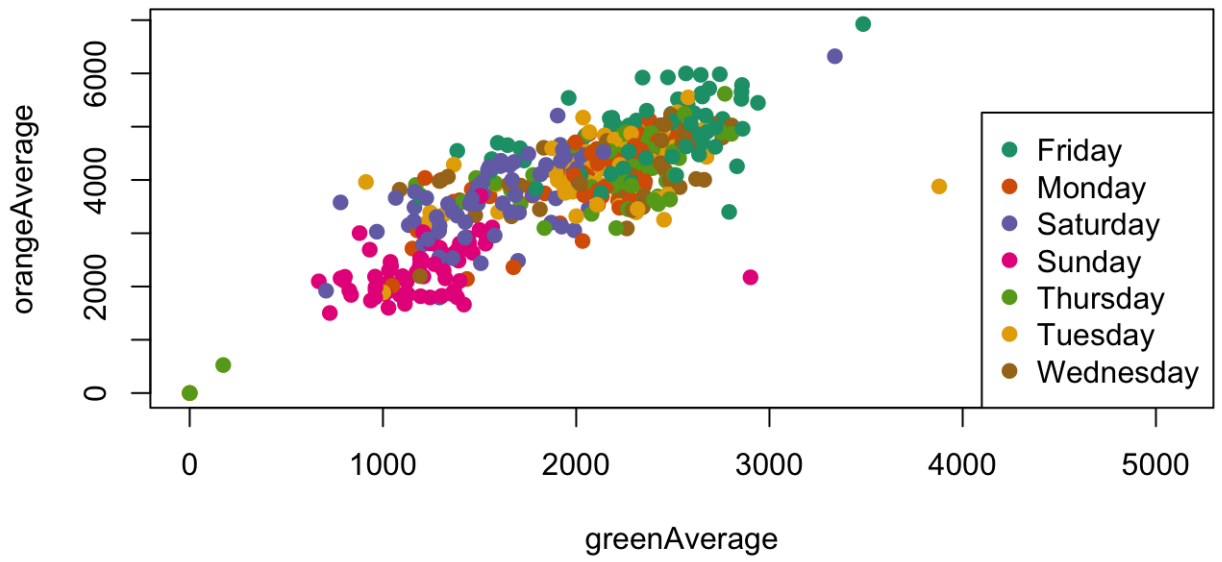
Adding legends

```
palette(brewer.pal(5,"Dark2"))
plot(weight ~ jitter(Time,amount=0.2),data=ChickWeight,
                  pch = 19,  col = Diet,xlab="Time")
legend("topleft", paste("Diet",levels(ChickWeight$Diet)),
        col = 1:length(levels(ChickWeight$Diet)),
       lwd = 3, ncol = 2)
```



Coloring by variable

```
circ = read_csv("https://jhudatascience.org/intro_to_r/data/Charm_City_Circula
palette(brewer.pal(7,"Dark2"))
dd = factor(circ$day)
plot(orangeAverage ~ greenAverage, data=circ,
     pch=19, col = as.numeric(dd))
legend("bottomright", levels(dd), col=1:length(dd), pch = 19)
```



Coloring by variable

```
dd = factor(circ$day, levels=c("Monday","Tuesday","Wednesday",
            "Thursday","Friday","Saturday","Sunday"))
plot(orangeAverage ~ greenAverage, data=circ,
     pch=19, col = as.numeric(dd))
legend("bottomright", levels(dd), col=1:length(dd), pch = 19)
```