Data visualization in R

Mikhail Dozmorov
Fall 2017

Why visualize data?

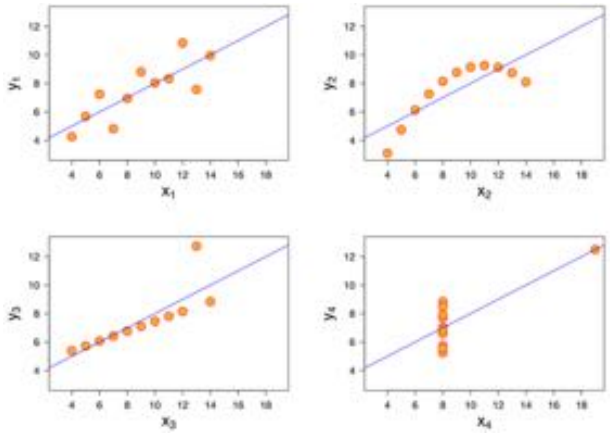
- Anscombe's quartet comprises four datasets that have nearly identical simple descriptive statistics, yet appear very different when graphed. (See Wikipedia link below)
- 11 observations (x, y) per group

Property	Value
Mean of *x* in each case	9 (exact)
Sample variance of *x* in each case	11 (exact)
Mean of *y* in each case	7.50 (to 2 decimal places)
Sample variance of *y* in each case	4.122 or 4.127 (to 3 decimal places)
Correlation between *x* and *y* in each case	0.816 (to 3 decimal places)
Linear regression line in each case	$y = 3.00 + 0.500x$ (to 2 and 3 decimal places, respectively)

https://en.wikipedia.org/wiki/Anscombe%27s_quartet

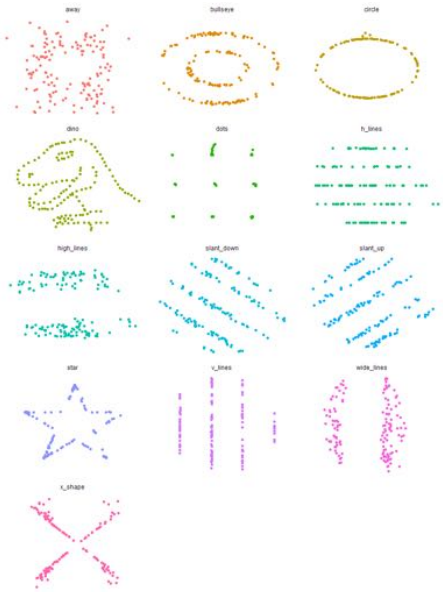
Why visualize data?

- Four groups
- 11 observations (x, y) per group

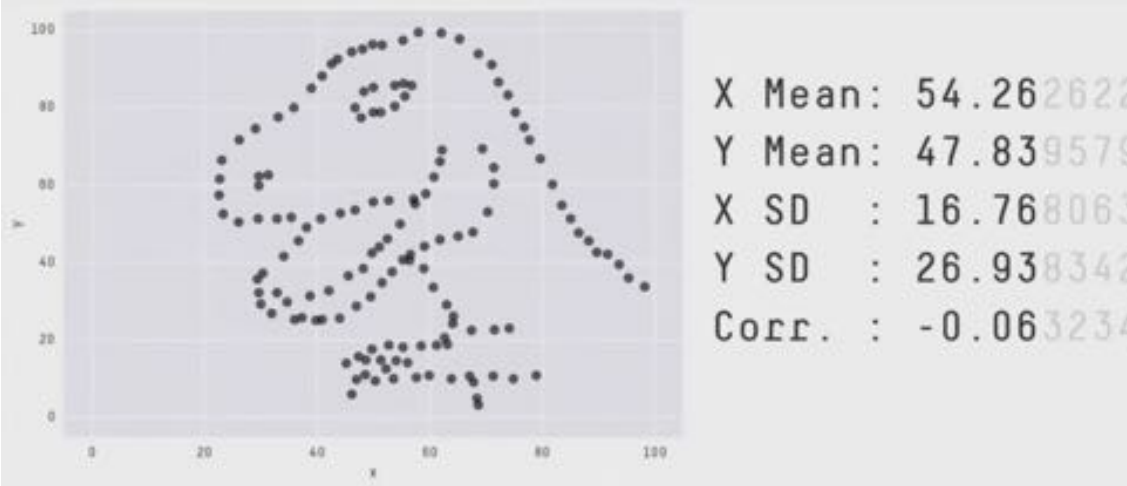


https://en.wikipedia.org/wiki/Anscombe%27s_quartet

Why visualized data?



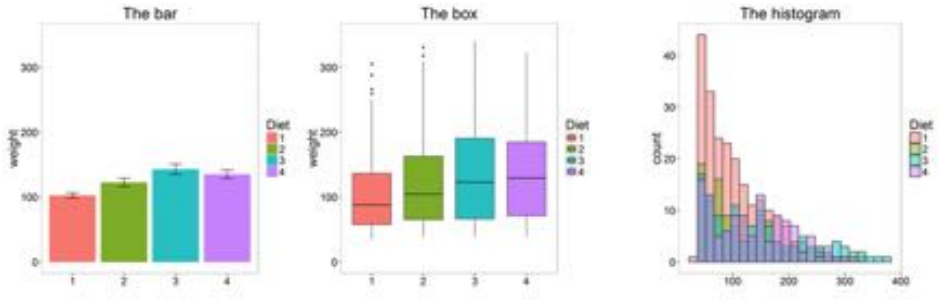
Why visualized data?



https://github.com/stephlocke/datasauRus

R base graphics

- `plot()` generic x-y plotting
- `barplot()` bar plots
- `boxplot()` box-and-whisker plot
- `hist()` histograms



http://manuals.bioinformatics.ucr.edu/home/R_BioCondManual#TOC-Some-Great-R-Functions

Don't use barplots

Weissgerber T et.al., "Beyond Bar and Line Graphs: Time for a New Data Presentation Paradigm", PLOS Biology,2015
http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1002128
https://cogtales.wordpress.com/2016/06/06/congratulations-barbarplots/

R base graphics

- `stats::heatmap()` - basic heatmap

Alternatives:

- `gplots::heatmap.2()` - an extension of heatmap
- `heatmap3::heatmap3()` - another extension of heatmap
- `ComplexHeatmap::Heatmap()` - highly customizable, interactive heatmap

Other options:

- `pheatmap::pheatmap()` - grid-based heatmap
- `NMF::aheatmap()` - another grid-based heatmap

More heatmaps

- `fheatmap::fheatmap()` - heatmap with some ggplot2
- `gapmap::gapmap()` - gapped heatmap (ggplot2/grid)

Interactive heatmaps:

- `d3heatmap::d3heatmap()` - interactive heatmap in d3
- `heatmaply::heatmaply()` - interactive heatmap with better dendrograms

Compare clusters

- `dendextend` package - make better dendrograms, compare them with ease

https://channel9.msdn.com/Events/useR-international-R-User-conference/useR2016/Heatmaps-in-R-Overview-and-best-practices

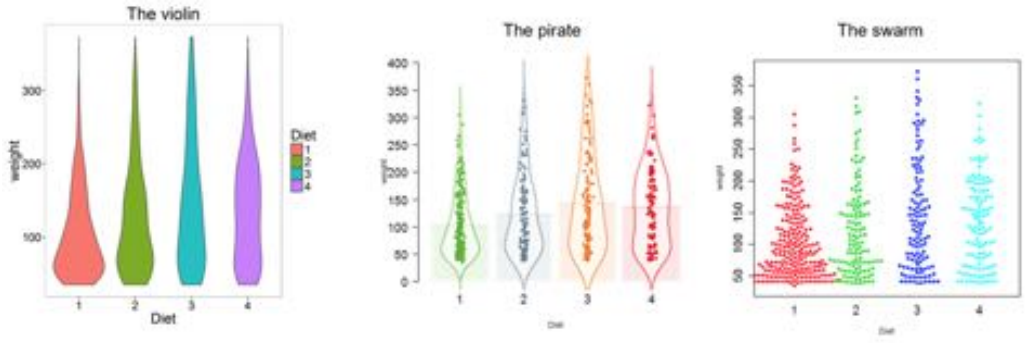
Other useful plots

- `qqnorm()`, `qqline()`, `qqplot()` - distribution comparison plots
- `pairs()` - pair-wise plot of multivariate data

http://manuals.bioinformatics.ucr.edu/home/R_BioCondManual#TOC-Some-Great-R-Functions

Special plots

- `vioplot()`: Violin plot, https://cran.r-project.org/web/packages/vioplot/
- `PiratePlot()`: violin plot enhanced. `install_github("ndphillips/yarrr")`, http://nathanieldphillips.com/
- `beeswarm()`: The Bee Swarm Plot, an Alternative to Stripchart, https://cran.r-project.org/web/packages/beeswarm/index.html



Saving plots

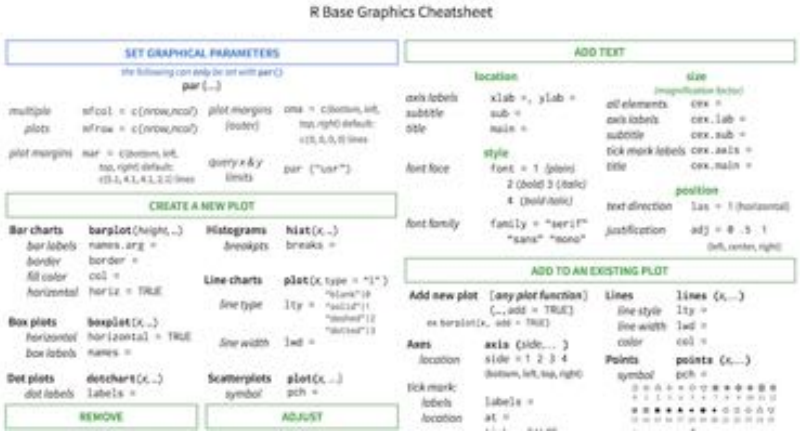
- Save to PDF

```
pdf("filename.pdf", width = 7, height = 5)
plot(1:10, 1:10)
dev.off()
```

- Other formats: `bmp()`, `jpg()`, `pdf()`, `png()`, or `tiff()`
- Click Export in the Plots window in RStudio
- Learn more `?Devices`

R base graphic cheat-sheet

https://github.com/nbrgraphs/mro/blob/master/BaseGraphicsCheatsheet.pdf



Data manipulation

dplyr: data manipulation with R

80% of your work will be data preparation

- getting data (from databases, spreadsheets, flat-files)
- performing exploratory/diagnostic data analysis
- reshaping data
- visualizing data

http://www.gettinggeneticsdone.com/2014/08/do-your-data-janitor-work-like-boss.html

dplyr: data manipulation with R

80% of your work will be data preparation

- Filtering rows (to create a subset)
- Selecting columns of data (i.e., selecting variables)
- Adding new variables
- Sorting
- Aggregating
- Joining

http://www.gettinggeneticsdone.com/2014/08/do-your-data-janitor-work-like-boss.html

Dplyr: A grammar of data manipulation

https://github.com/hadley/dplyr

```
install.packages("dplyr")
```

Basic `dplyr` verbs

- `filter()`
- `arrange()`
- `select()`
- `mutate()`
- `summarize()`

The pipe %>% operator

- Pipe output of one command into an input of another command - chain commands together. (Think about the "|" operator in Linux)
- Read as "then". Take the dataset (or object), *then* do …

```
library(dplyr)
round( sqrt(1000), 3)
```

```
## [1] 31.623
```

```
1000 %>% sqrt %>% round()
```

```
## [1] 32
```

```
1000 %>% sqrt %>% round(., 3)
```

```
## [1] 31.623
```


The pipe %>% operator

- For example, we can view the head of the `diamonds` data.frame using either of the last two lines of code here:

```
library(dplyr)
library(ggplot2)
data(diamonds)
head(diamonds)
diamonds %>% head
```

```
## # A tibble: 6 x 10
##   carat       cut color clarity depth table price     x     y     z
##   <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
## 2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
## 3  0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
## 4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
## 5  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
```

The pipe %>% operator

- For example, read the last line of code as: "Take the `price` column of the `diamonds` data.frame and *then* summarize it"

```
library(dplyr)
data(diamonds)
head(diamonds)
diamonds %>% head
summary(diamonds$price)
diamonds$price %>% summary(object = .)
```


dplyr::filter()

- Filter (select) rows based on the condition of a column
- Syntax: `filter(data, condition)`

dplyr::filter()

For example, keep only the entries with Ideal cut

```
df.diamonds_ideal <- filter(diamonds, cut == "Ideal")
df.diamonds_ideal
```

```
## # A tibble: 21,551 x 10
##    carat   cut color clarity depth table price     x     y     z
##    <dbl> <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
##  2  0.23 Ideal     J     VS1  62.8    56   340  3.93  3.90  2.46
##  3  0.31 Ideal     J     SI2  62.2    54   344  4.35  4.37  2.71
##  4  0.30 Ideal     I     SI2  62.0    54   348  4.31  4.34  2.68
##  5  0.33 Ideal     I     SI2  61.8    55   403  4.49  4.51  2.78
##  6  0.33 Ideal     I     SI2  61.2    56   403  4.49  4.50  2.75
##  7  0.33 Ideal     J     SI1  61.1    56   403  4.49  4.55  2.76
##  8  0.23 Ideal     G     VS1  61.9    54   404  3.93  3.95  2.44
##  9  0.32 Ideal     I     SI1  60.9    55   404  4.45  4.48  2.72
## 10  0.30 Ideal     I     SI2  61.0    59   405  4.30  4.33  2.63
## # ... with 21,541 more rows
```


dplyr::filter()

We can achieve this same result using the `%>%` operator

```
diamonds %>% head
df.diamonds_ideal <- filter(diamonds, cut == "Ideal")
df.diamonds_ideal <- diamonds %>% filter(cut == "Ideal")
```

dplyr::select()

- Select columns from the dataset by names
- Syntax: `select(data, columns)`

```
df.diamonds_ideal %>% head
select(df.diamonds_ideal, carat, cut, color, price, clarity)
df.diamonds_ideal <- df.diamonds_ideal %>% select(., carat, cut, color, price, clarity)
```


dplyr::mutate()

- Add new columns to your dataset that are functions of old columns
- Syntax: `mutate(data, new_column = function(old_columns))`

```
df.diamonds_ideal %>% head
mutate(df.diamonds_ideal, price_per_carat = price/carat)
df.diamonds_ideal <- df.diamonds_ideal %>% mutate(price_per_carat = price/carat)
```

dplyr::arrange()

- Sort your data by columns
- Syntax: `arrange(data, column_to_sort_by)`

```
df.diamonds_ideal %>% head
arrange(df.diamonds_ideal, price)
df.diamonds_ideal %>% arrange(price, price_per_carat)
```


dplyr::summarize()

- Summarize columns by custom summary statistics
- Syntax: `summarize(function_of_variables)`

```
summarize(df.diamonds_ideal, length = n(), avg_price = mean(price))
df.diamonds_ideal %>% summarize(length = n(), avg_price = mean(price))
```

dplyr::group_by()

- Summarize *subsets of* columns by custom summary statistics
- Syntax: `group_by(data, column_to_group)`

```
group_by(diamonds, cut) %>% summarize(mean(price))
group_by(diamonds, cut, color) %>% summarize(mean(price))
```


The power of pipe %>%

- Summarize *subsets of* columns by custom summary statistics

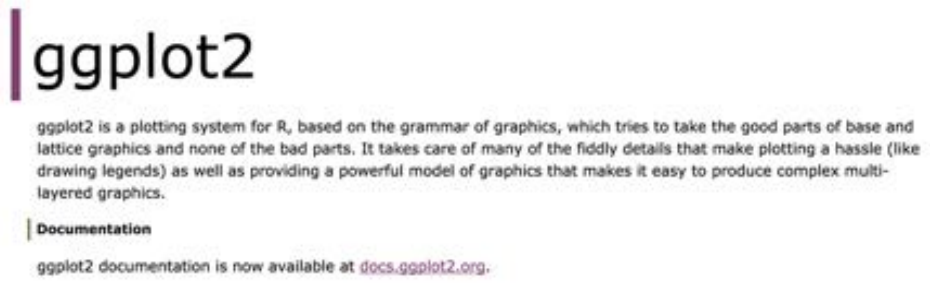
```
arrange(mutate(arrange(filter(tbl_df(diamonds), cut == "Ideal"), price),
  price_per_carat = price/carat), price_per_carat)
arrange(
  mutate(
    arrange(
      filter(tbl_df(diamonds), cut == "Ideal"),
    price),
  price_per_carat = price/carat),
price_per_carat)
diamonds %>% filter(cut == "Ideal") %>% arrange(price) %>%
  mutate(price_per_carat = price/carat) %>% arrange(price_per_carat)
```

ggplot2 - the grammar of graphics

ggplot2 package

http://ggplot2.org/

```
install.packages("ggplot2")
```



ggplot2

ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics.

Documentation

ggplot2 documentation is now available at docs.ggplot2.org.

The basics of ggplot2 graphics

- Data mapped to graphical elements
- Add graphical layers and transformations
- Commands are chained with "+" sign

Object		Description
Data		The raw data that you want to plot
Aethetics	aes()	How to map your data on x, y axis, color, size, shape (aesthetics)
Geometries	geom_	The geometric shapes that will represent the data

data +

aesthetic mappings of data to plot coordinates +

geometry to represent the data

Basic ggplot2 syntax

Specify data, aesthetics and geometric shapes

`ggplot(data, aes(x=, y=, color=, shape=, size=, fill=)) +` `geom_point()`, or `geom_histogram()`, or `geom_boxplot()`, etc.

- This combination is very effective for exploratory graphs.
- The data must be a data frame in a **long** (not wide) format
- The `aes()` function maps **columns** of the data frame to aesthetic properties of geometric shapes to be plotted.
- `ggplot()` defines the plot; the `geoms` show the data; layers are added with +

Examples of ggplot2 graphics

```
diamonds %>% filter(cut == "Good", color == "E") %>%
  ggplot(aes(x = price, y = carat)) +
  geom_point()  # aes(size = price) +
```

Try other geoms

```
  geom_smooth() # method = lm
  geom_line()
  geom_boxplot()
  geom_bar(stat="identity")
  geom_histogram()
```

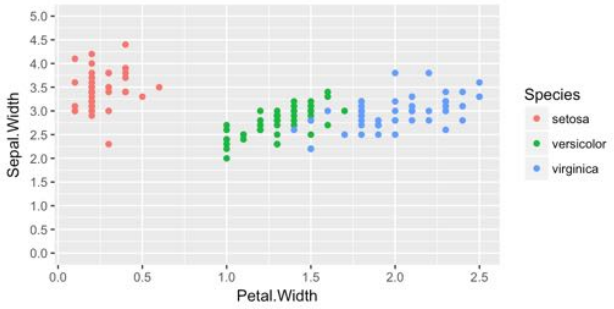

Moving beyond `ggplot` + `geoms`

Customizing scales

- Scales control the mapping from data to aesthetics and provide tools to read the plot (ie, axes and legends).
- Every aesthetic has a default scale. To add or modify a scale, use a `scale` function.
- All scale functions have a common naming scheme: `scale` _ name of aesthetic _ name of scale
- Examples: `scale_y_continuous`, `scale_color_discrete`, `scale_fill_manual`

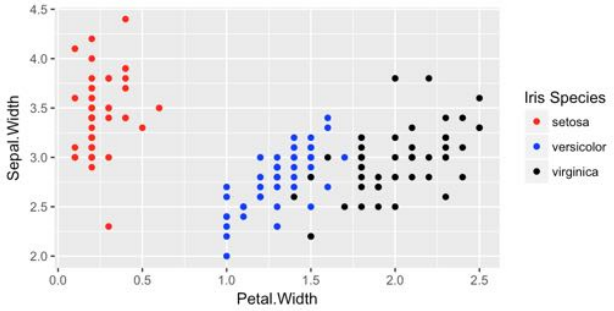
ggplot2 example - update scale for y-axis

```
ggplot(iris, aes(x = Petal.Width, y = Sepal.Width,
                 color=Species)) + geom_point() +
  scale_y_continuous(limits=c(0,5), breaks=seq(0,5,0.5))
```



ggplot2 example - update scale for color

```
ggplot(iris, aes(x = Petal.Width, y = Sepal.Width,
                 color=Species)) + geom_point() +
  scale_color_manual(name="Iris Species",
                     values=c("red","blue","black"))
```



Moving beyond `ggplot` + `geoms`

Split plots

- A natural next step in exploratory graphing is to create plots of subsets of data. These are called facets in ggplot2.
- Use `facet_wrap()` if you want to facet by one variable and have `ggplot2` control the layout. Example:

```
- + facet_wrap( ~ var)
```

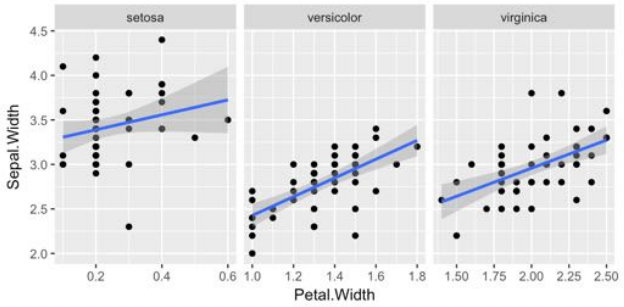
- Use `facet_grid()` if you want to facet by one and/or two variables and control layout yourself.

Examples:
+ `facet_grid(. ~ var1)` - facets in columns
+ `facet_grid(var1 ~ .)` - facets in rows
+ `facet_grid(var1 ~ var2)` - facets in rows and columns

ggplot2 example - `facet_wrap`

Note free x scales

```
ggplot(iris, aes(x = Petal.Width, y = Sepal.Width)) +
  geom_point() + geom_smooth(method="lm") +
  facet_wrap(~ Species, scales = "free_x")
```



stat functions

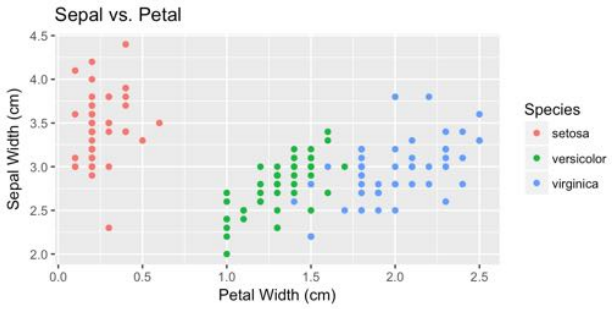
- All `geoms` perform a default statistical transformation.
- For example, `geom_histogram()` bins the data before plotting. `geom_smooth()` fits a line through the data according to a specified method.
- In some cases the transformation is the "identity", which just means plot the raw data. For example, `geom_point()`
- These transformations are done by `stat` functions. The naming scheme is `stat_` followed by the name of the transformation. For example, `stat_bin`, `stat_smooth`, `stat_boxplot`
- **Every geom has a default stat, every stat has a default geom.**

Update themes and labels

- The default ggplot2 theme is excellent. It follows the advice of several landmark papers regarding statistics and visual perception. (Wickham 2009, p. 141)
- However you can change the theme using ggplot2's themeing system. To date, there are seven built-in themes: `theme_gray` (*default*), `theme_bw`, `theme_linedraw`, `theme_light`, `theme_dark`, `theme_minimal`, `theme_classic`
- You can also update axis labels and titles using the `labs` function.

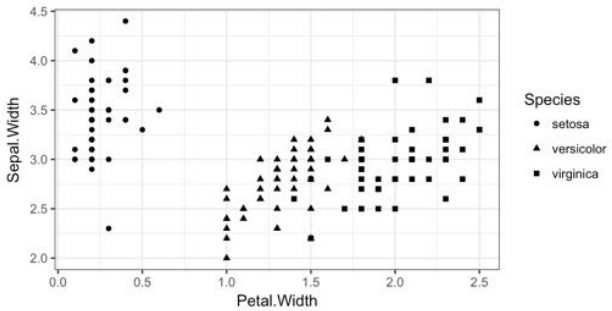
ggplot2 example - update labels

```
ggplot(iris, aes(x = Petal.Width, y = Sepal.Width,
                 color=Species)) + geom_point() +
  labs(title="Sepal vs. Petal",
       x="Petal Width (cm)", y="Sepal Width (cm)")
```



ggplot2 example - change theme

```
ggplot(iris, aes(x = Petal.Width, y = Sepal.Width,
                 shape=Species)) + geom_point() +
  theme_bw()
```



Summary: Fine tuning ggplot2 graphics

Parameter		Description
Facets	facet_	Split one plot into multiple plots based on a grouping variable
Scales	scale_	Maps between the data ranges and the dimensions of the plot
Visual Themes	theme	The overall visual defaults of a plot: background, grids, axe, default typeface, sizes, colors, etc.
Statistical transformations	stat_	Statistical summaries of the data that can be plotted, such as quantiles, fitted curves (loess, linear models, etc.), sums etc.
Coordinate systems	coord_	Expressing coordinates in a system other than Cartesian

Putting it all together

```
diamonds %>%                  # Start with the 'diamonds' dataset
  filter(cut == "Ideal") %>% # Then, filter rows where cut == Ideal
  ggplot(aes(price)) +        # Then, plot using ggplot
  geom_histogram() +          # and plot histograms
  facet_wrap(~ color) +       # in a 'small multiple' plot, broken out by 'color'
  ggtitle("Diamond price distribution per color") +
  labs(x="Price", y="Count") +
  theme(panel.background = element_rect(fill="lightblue")) +
  theme(plot.title = element_text(family="Trebuchet MS", size=28, face="bold", hjust=0, color='
  theme(axis.title.y = element_text(angle=0)) +
  theme(panel.grid.minor = element_blank())
```

Other resources

- **Plotly** for R, https://plot.ly/r/
- **GoogleVis** for R, https://cran.r-project.org/web/packages/googleVis/vignettes/googleVis_examples.html
- **ggbio** - grammar of graphics for genomic data, http://www.tengfei.name/ggbio/