Data visualization in R

Mikhail Dozmorov

Summer 2018

tidyverse

The `tidyverse` is a collection of packages based on 4 principles for handling data:

1. Reuse existing data structures.
2. Compose simple functions with the pipe.
3. Embrace functional programming.
4. Design for humans.

The R project for Statistical Computing was built for a different age; the tidyverse is a collection of tools for *our* age

tidyverse

Tidyverse Packages Articles Learn Help Contribute

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

https://www.tidyverse.org/

Reading in data

readr

- There are some built-in functions for reading in data in text files. These functions are *read-dot-something* – for example, `read.csv()` reads in comma-delimited text data; `read.delim()` reads in tab-delimited text, etc.
- `readr` package provides fast and intelligent data reading capabilities. Very similar looking functions, named *read-underscore-something* – e.g., `read_csv()`
- They're good at guessing the types of data in the columns, they don't do some of the other silly things that the base functions do
- Play nicely with `dplyr` - data manipulation package

http://readr.tidyverse.org/

Inspecting data.frame objects

There are several built-in functions that are useful for working with data frames.

- Content:
  - `head()`: shows the first few rows
  - `tail()`: shows the last few rows
- Size:
  - `dim()`: returns a 2-element vector with the number of rows in the first element, and the number of columns as the second element (the dimensions of the object)
  - `nrow()`: returns the number of rows
  - `ncol()`: returns the number of columns

Inspecting data.frame objects

- Summary:
  - `colnames()` (or just `names()`): returns the column names
  - `str()`: structure of the object and information about the class, length and content of each column
  - `summary()`: works differently depending on what kind of object you pass to it. Passing a data frame to the `summary()` function prints out useful summary statistics about numeric column (min, max, median, mean, etc.)

tibbles

Data frames are great! Except for

- printing them
- working with both characters and factors
- manipulating multiple columns

tibbles are the data frame alternative of the tidyverse

tibbles

- A `tibble`, or `tbl_df`, is a modern reimagining of the `data.frame`, keeping what time has proven to be effective, and throwing out what is not
- Tibbles are `data.frames` that are lazy and surly: they do less (i.e. they don't change variable names or types, and don't do partial matching) and complain more (e.g. when a variable does not exist)
- This forces you to confront problems earlier, typically leading to cleaner, more expressive code. Tibbles also have an enhanced `print` method which makes them easier to use with large datasets containing complex objects.
    - Hadley Wickham, Chief Scientist at RStudio

Making the data tidy with `reshape2`

- Principles of tidy data
  - Each *column* is a *variable*
  - Each *row* is an *observation*
- `reshape2` - flexible data reshaping
  - `melt` - melt a data frame into a long format. See `?melt.data.frame`
  - `dcast` - cast a molten data frame into a wide format. See `?cast`

Example of gathering columns to create tidy data

site	1999	2000
Whitehorse	745	2666
Yellowknife	37737	80488
Inuvik	212258	213766

site	year	cases
Whitehorse	1999	745
Whitehorse	2000	2666
Yellowknife	1999	37737
Yellowknife	2000	80488
Inuvik	1999	212258

Better way to tidy the data - the `tidyr` package

- `tidyr` - easily tidy data with `spread()` and `gather()` functions
  - The `gather()` function takes multiple columns, and gathers them into key-value pairs: it makes "wide" data longer
  - The `separate()` function separates one column into multiple columns

https://cran.r-project.org/web/packages/tidyr/index.html

Data scraping from the web

Meet `rvest` - Simple web scraping for R

```
library(rvest)
```

```
## Loading required package: xml2
```

```
## Warning: package 'xml2' was built under R version 3.4.3
```

```
url <- "https://www.rottentomatoes.com/celebrity/mira_furlan"

dl_tab <- url %>%
  read_html() %>%
  html_node("#filmographyTbl") %>%
  html_table()

tail(dl_tab)
```

Data visualization

Why visualize data?

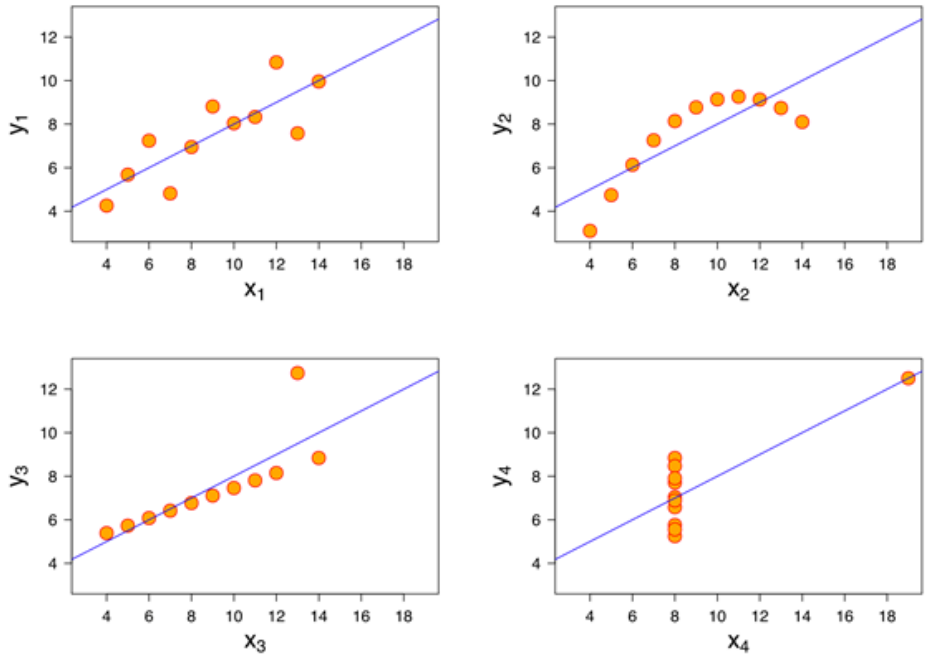
- Anscombe's quartet comprises four datasets that have nearly identical simple descriptive statistics, yet appear very different when graphed. (See Wikipedia link below)
- 11 observations (x, y) per group

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

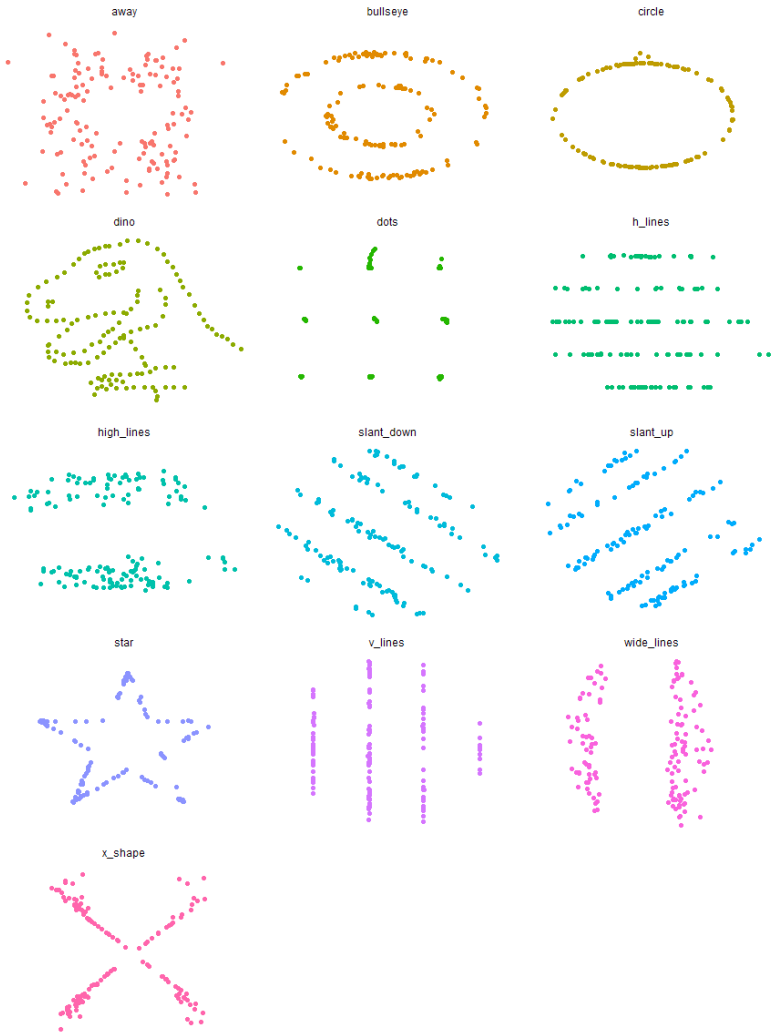
Why visualize data?

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



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

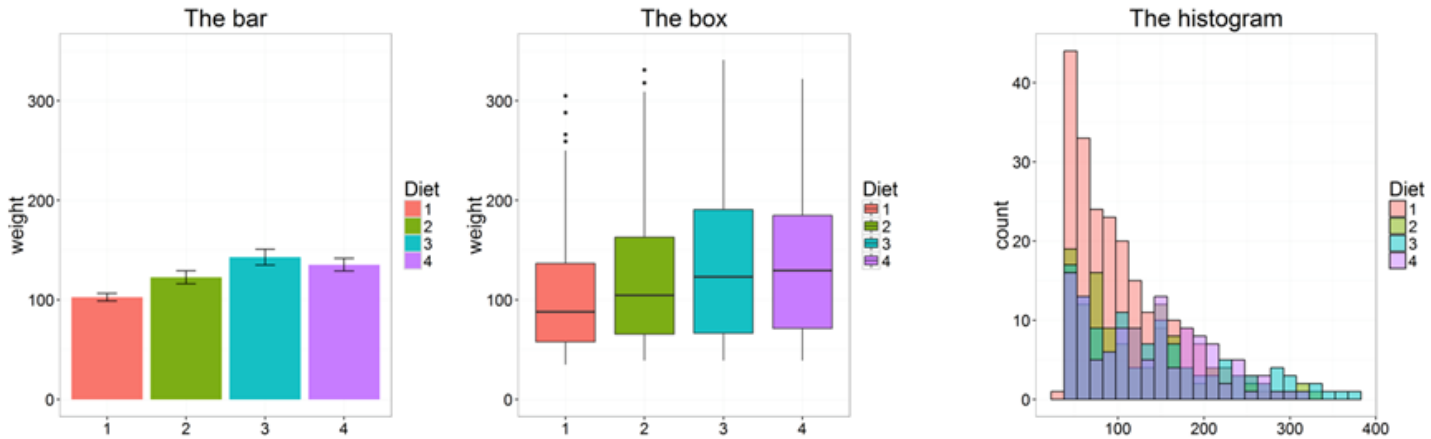
Why visualized data?



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-Graphical-Procedures

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

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

Interactive heatmaps

- `d3heatmap::d3heatmap()` - interactive heatmap in d3
- `heatmaply::heatmaply()` - interactive heatmap with better dendrograms
- `plotly` - make ggplot2 plots interactive

Compare clusters

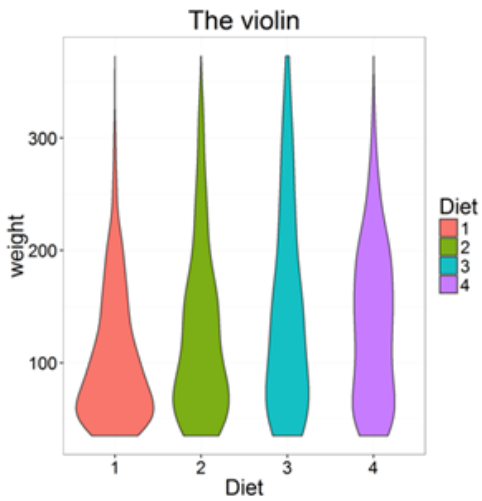
- `dendextend` package - make better dendrograms, compare them with ease

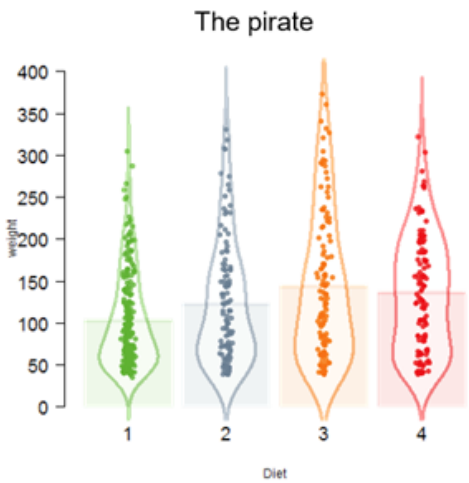
https://channel9.msdn.com/Events/useR-international-R-User-conference/useR2016/Heatmaps-in-R-Overview-and-best-practices

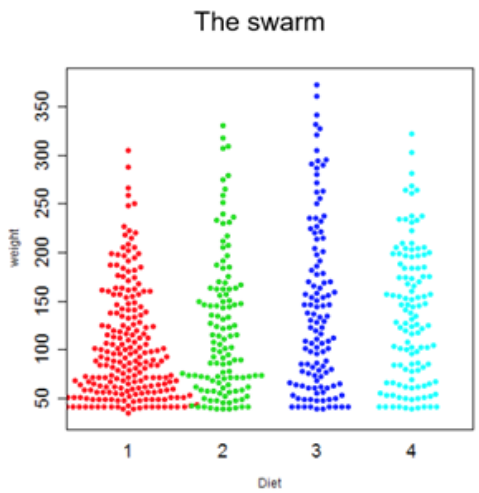
https://davetang.org/muse/2018/05/18/interactive-plots-in-r/

Special plots

- `vioplot()`: Violin plot
- `PiratePlot()`: violin plot enhanced
- `beeswarm()`: The Bee Swarm Plot, an Alternative to Stripchart







<https://cran.r-project.org/web/packages/vioplot/>

`install_github("ndphillips/yarrr")`, http://nathanieldphillips.com/

https://cran.r-project.org/web/packages/beeswarm/index.html

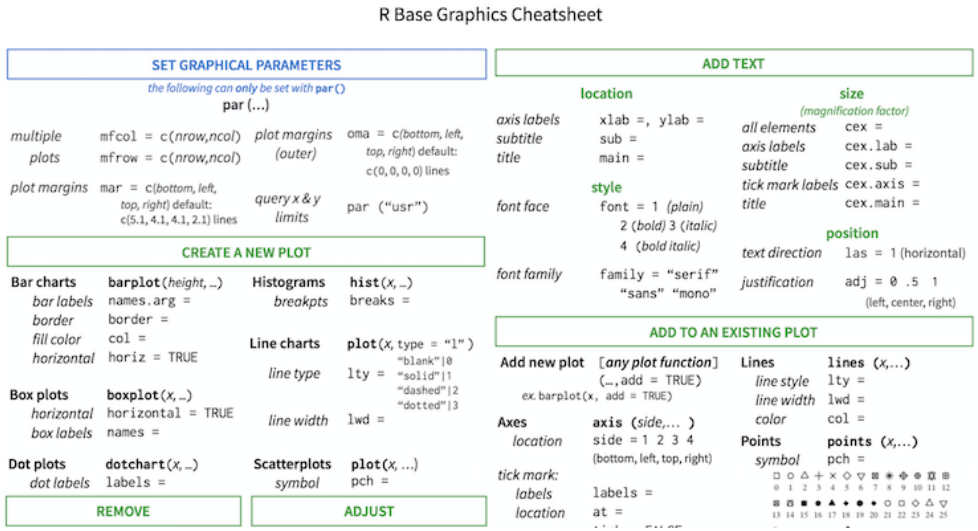
Saving plots

- 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

- The `dplyr` package gives you a handful of useful **verbs** for managing data. On their own they don't do anything that base R can't do
- Basic `dplyr` verbs
  - `filter()`
  - `select()`
  - `mutate()`
  - `arrange()`
  - `summarize()`
  - `group_by()`
- They all take a data frame or tibble as their input for the first argument, and they all return a data frame or tibble as output

<https://github.com/hadley/dplyr>

https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html

The pipe %>% operator

- Pipe %> output of one command into an input of another command - chain commands together. (Think about the "|" operator in Linux)
- Imported from `magrittr` package
- 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)
```

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
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <db
## 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.
## 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.
## 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.
## 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.
```

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 = .)
```

- There's a keyboard shortcut to insert the %>% sequence - you can see what it is by clicking the *Tools* menu in RStudio, then selecting *Keyboard Shortcut Help*
- On Mac, it's CMD-SHIFT-M

dplyr::filter()

If you want to filter **rows** of the data where some condition is true, use the `filter()` function.

1. The first argument is the data frame you want to filter, e.g. `filter(mydata, ...`.
2. The second argument is a condition you must satisfy, e.g. `filter(ydat, symbol == "LEU1")`. If you want to satisfy *all* of multiple conditions, you can use the "and" operator, `&`. The "or" operator `|` (the pipe character, usually shift-backslash) will return a subset that meet *any* of the conditions.

- `==`: Equal to
- `!=`: Not equal to
- `>`, `>=`: Greater than, greater than or equal to
- `<`, `<=`: Less than, less than or equal to

dplyr::filter()

For example, keep only the entries with Ideal cut

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

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
df.diamonds_ideal
```

```
## # A tibble: 21,551 x 10
##    carat cut   color clarity depth table price     x     y
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5
```

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()

- The `filter()` function allows you to return only certain *rows* matching a condition. The `select()` function returns only certain *columns*. The first argument is the data, and subsequent arguments are the columns you want.
  - 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, cu
```

dplyr::mutate()

- The `mutate()` function adds new columns to the data that are functions of old columns
- It doesn't actually modify the data frame you're operating on, and the result is transient unless you assign it to a new object or reassign it back to itself (generally, not a good practice)
  - 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_ca
```

dplyr::arrange()

- The `arrange()` function does what it sounds like - sorts things
- It takes a `data.frame` or `tbl_df` and arranges (or sorts) by column(s) of interest
- The first argument is the data, and subsequent arguments are columns to sort on. Use the `desc()` function to arrange by descending
  - 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()

- The `summarize()` function summarizes multiple values to a single value
- The power of `summarize()` comes from a few convenience functions called `n()` and `n_distinct()` that tell you the number of observations or the number of distinct values of a particular variable.
  - Syntax: `summarize(function_of_variables)`

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

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_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_
```

ggplot2 - the grammar of graphics

ggplot2 package

- `ggplot2` is a widely used R package that extends R's visualization capabilities. It takes the hassle out of things like creating legends, mapping other variables to scales like color, or faceting plots into small multiples
- *Where does the "gg" in ggplot2 come from?* The `ggplot2` package provides an R implementation of Leland Wilkinson's *Grammar of Graphics* (1999)
    - The *Grammar of Graphics* allows you to think beyond the garden variety plot types (e.g. scatterplot, barplot) and the consider the components that make up a plot or graphic, such as how data are represented on the plot (as lines, points, etc.), how variables are mapped to coordinates or plotting shape or color, what transformation or statistical summary is required, and so on

<http://ggplot2.org/>

Wilkinson, Leland. The grammar of graphics. Springer Science & Business Media, 2006.

The basics of ggplot2 graphics

Specifically, **ggplot2** allows you to build a plot layer-by-layer by specifying:

- a **geom**, which specifies how the data are represented on the plot (points, lines, bars, etc.),
- **aesthetics** that map variables in the data to axes on the plot or to plotting size, shape, color, etc.,
- a **stat**, a statistical transformation or summary of the data applied prior to plotting,
- **facets**, which we've already seen above, that allow the data to be divided into chunks on the basis of other categorical or continuous variables and the same plot drawn for each chunk.

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()
```

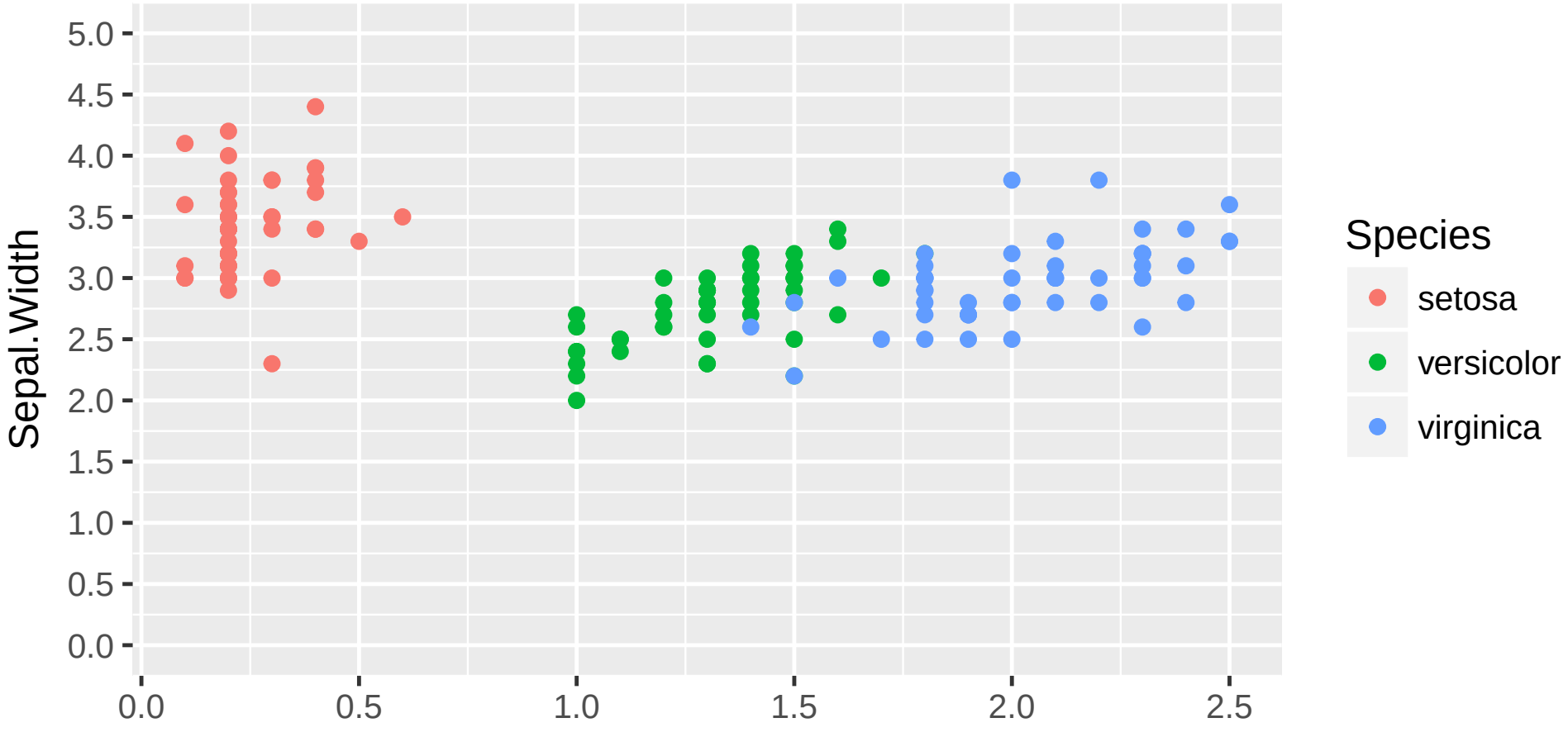
Moving beyond `ggplot + geoms`

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`

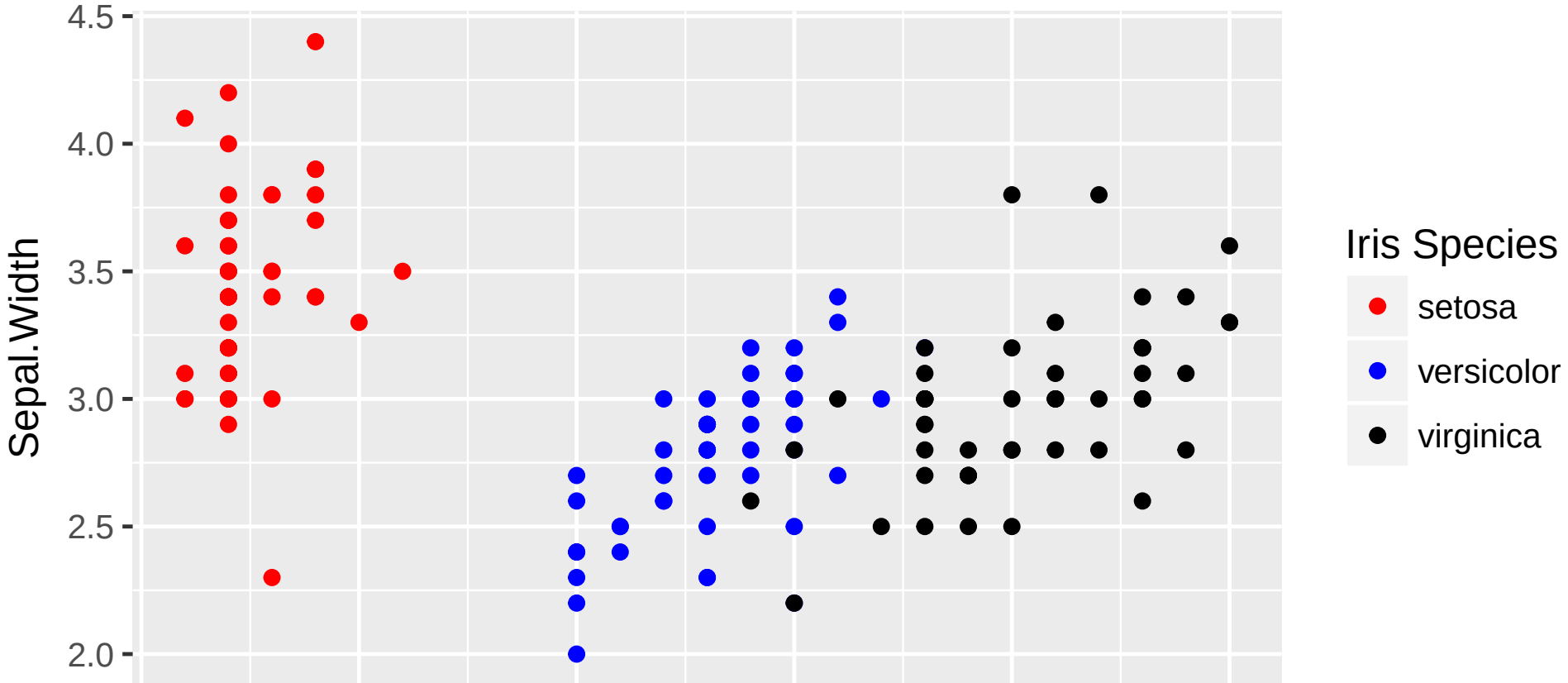
ggplot2 example - update scale for y-axis

```
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.

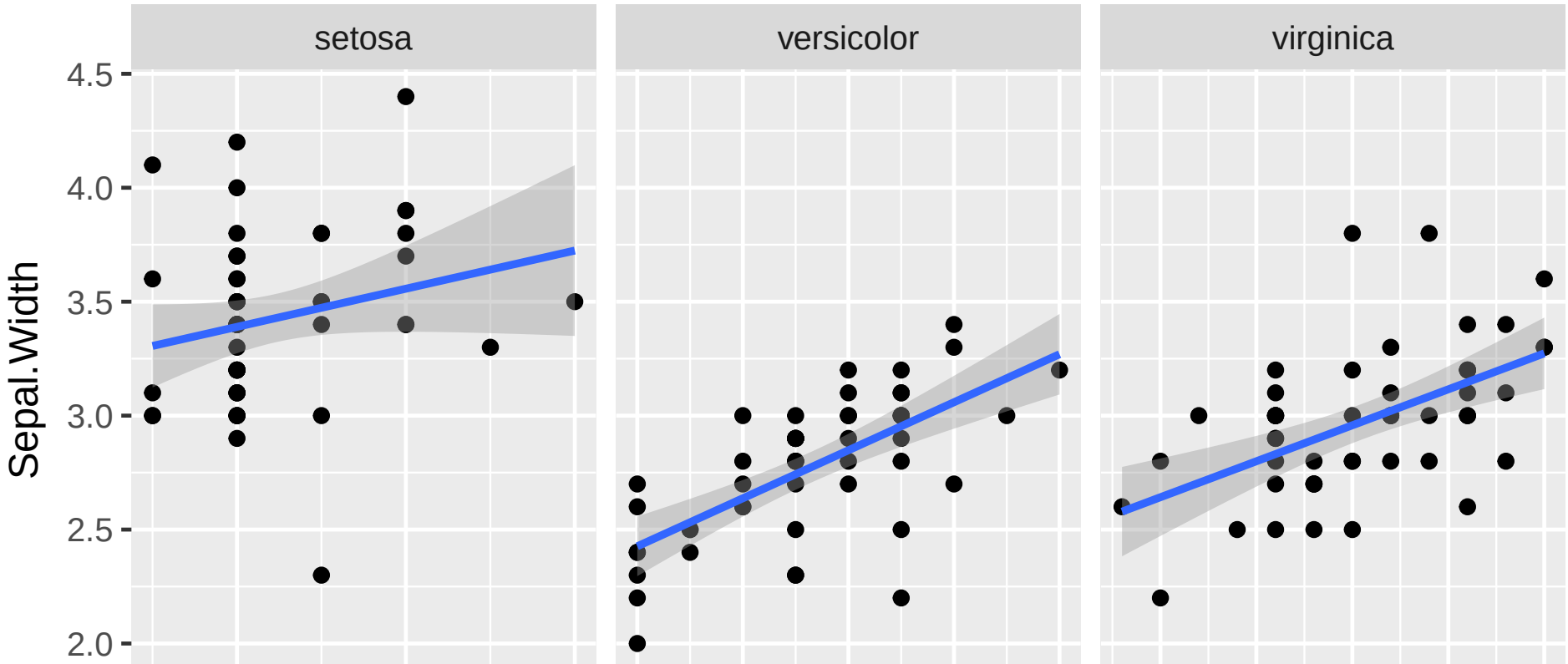
Examples:

+ `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

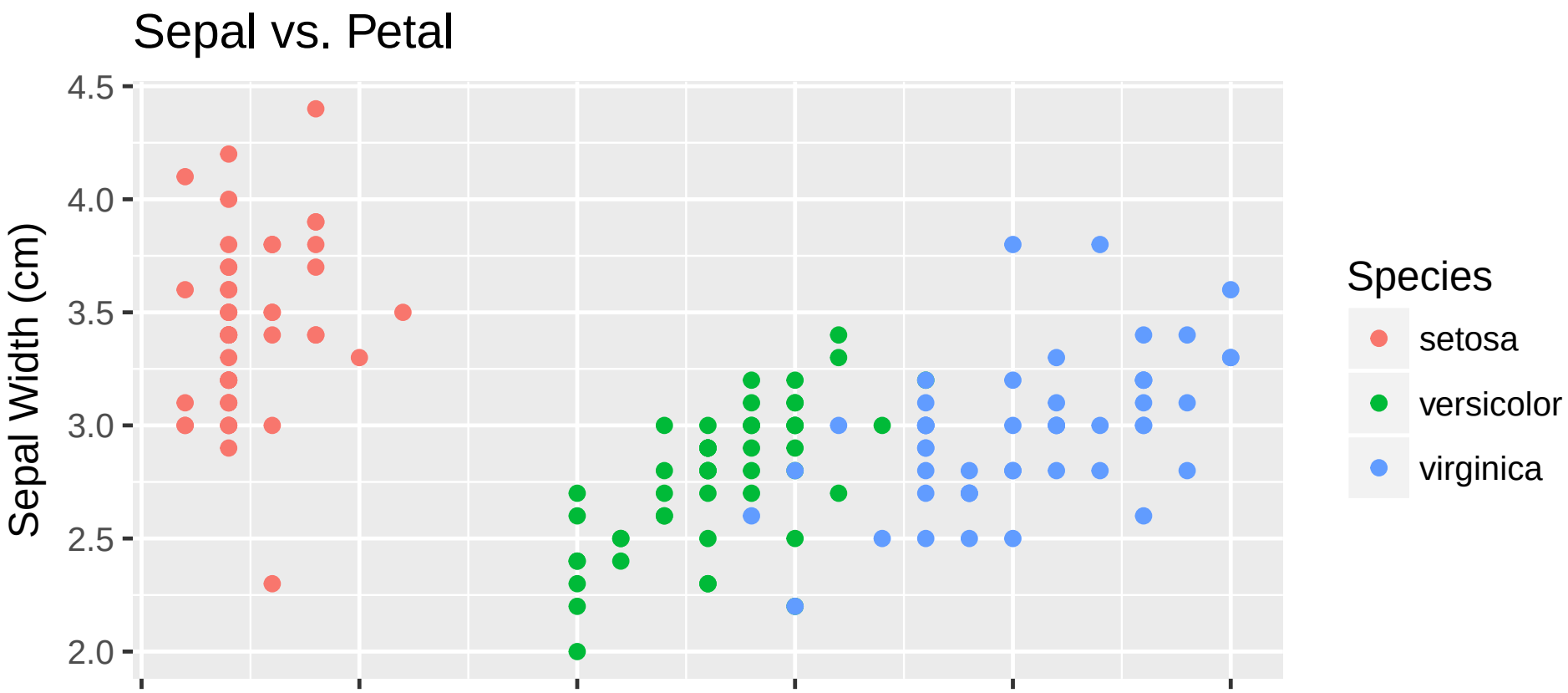
- All `geoms` perform a default statistical transformation.
- For example, `geom_histogram()` bins the data before plotting. `geom_smooth()` fits a line through the data according to a specified method.
- In some cases the transformation is the "identity", which just means plot the raw data. For example, `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.

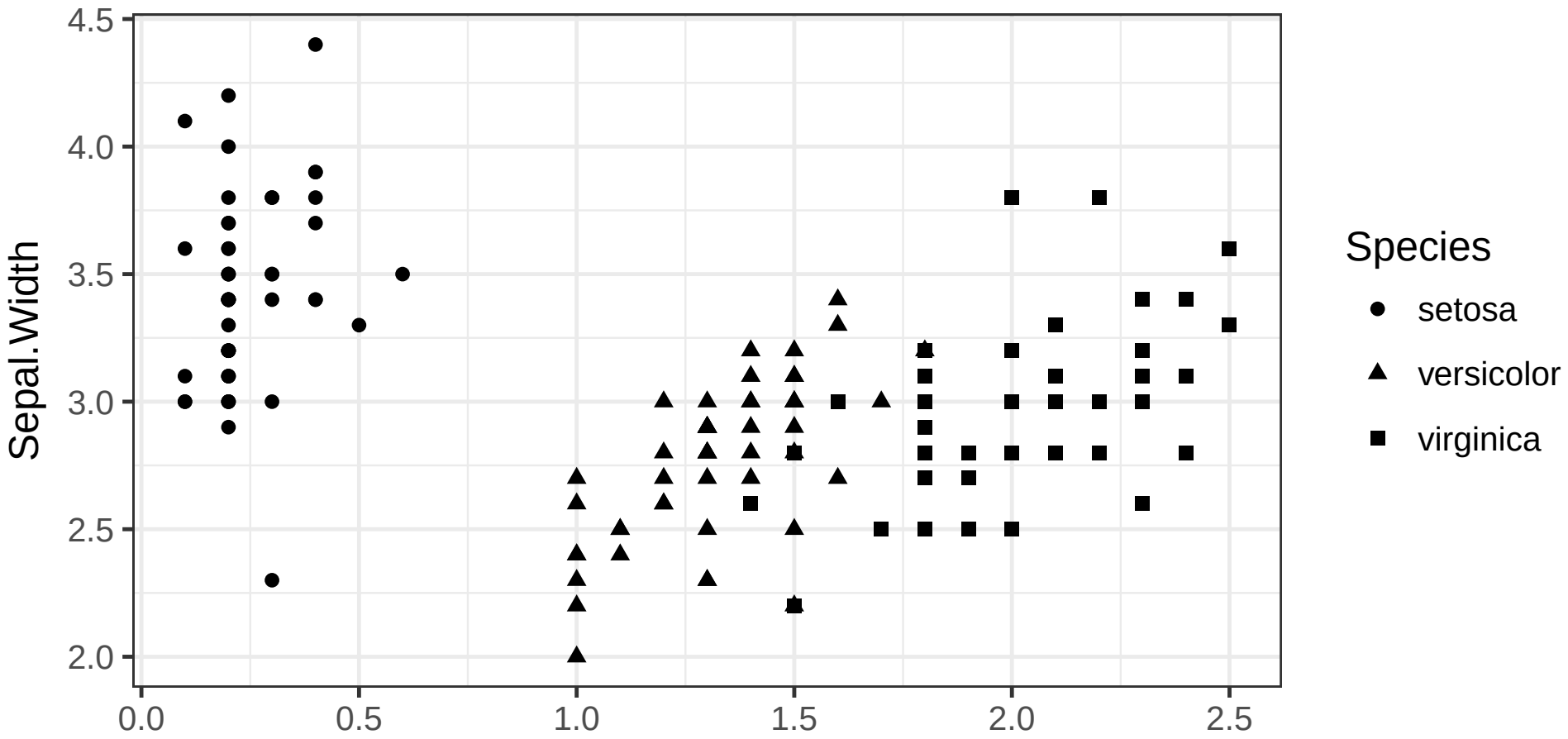
ggplot2 example - update labels

```
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' datas
  filter(cut == "Ideal") %>% # Then, filter rows where cut ==
  ggplot(aes(price)) +                # Then, plot using ggplot
  geom_histogram() +                  # and plot histograms
  facet_wrap(~ color) +               # in a 'small multiple' plot, bro
  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=
  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/