**R for Data Science**

**Chapter 1: Intro**

* When data is **tidy** (consistent structure/form that matches semantics of the dataset w/ the way its stored) 🡺 each column = a variable, each row = an observation.
* important b/c consistent structure lets you focus on questions about the data
* **wrangling =** tidying + transforming (manipulate variables, create new ones, summary stats) data
* **good visualizations** = shows things you did not expect, raises new questions about the data, hint that you’re asking the wrong question, or you need to collect different data.
* **Models** = complementary tools to visualisation.
* Make questions sufficiently precise🡪 then use a model to answer them.
* Models = fundamentally mathematical/computational tool 🡺 so they generally scale well.
* Even when they don’t, cheaper to buy more CPUs than more brains
* Every model makes assumptions + by its very nature a model cannot question its own assumptions = therefore a model cannot fundamentally surprise you.
* Doesn’t matter how well models + visualisation have led you to understand data unless you can also **communicate** results to others.
* If routinely working w/ larger data (10-100 Gb), learn more about **data.table**
* it has a very concise interface which makes it harder to learn since it offers fewer linguistic cues.
* But if working w/ large data, a performance payoff is worth the extra effort required to learn it
* If your data is bigger than this, *carefully consider if your big data problem might actually be a small data problem in disguise.*
* While the *complete* data might be big, often data needed to answer a specific question is small.
* Subsets, subsamples, or summaries that fit in memory can still allow you to answer a question you’re interested in.
* Challenge = finding the *right* small data, which often requires a lot of iteration.
* Big data problem might actually a *large number of small data problems*.
* Each individual problem might fit in memory, but you have millions of them.
* Ex: fit a model to each person in a dataset = trivial w/ 10 or 100 people, but you have a million.
* Fortunately each problem is independent of the others (**embarrassingly parallel** setup)
* Just need a system (Hadoop or Spark) that allows you to send different datasets to different CPUs for processing.
* Once you’ve figured out how to answer a question for a single subset using tools in this book, you learn new tools like **sparklyr**, **rhipe**, + **ddr** to solve it for the full dataset.
* It’s possible to divide data analysis into 2 camps: **hypothesis generation** + **hypothesis confirmation** (or **confirmatory analysis**).
* Focus of this book = hypothesis *generation*, or **data exploration 🡺** look deeply at data + in combination w/ subject knowledge, generate interesting hypotheses to help explain why data behaves the way it does.
* Evaluate hypotheses informally, using your scepticism to challenge the data in multiple ways.
* Complement of hypothesis generation = hypothesis confirmation which is hard for 2 reasons:
* Need a precise mathematical model to generate falsifiable predictions.
* often requires considerable statistical sophistication.
* Can only use an observation once to confirm a hypothesis.
* As soon as you use it more than once you’re back to doing EDA.
* To do hypothesis confirmation you need to **preregister** (write out in advance) an analysis plan + *not deviate from it even when you have seen the data*
* Common to think about modelling as a tool for hypothesis confirmation + visualisation as a tool for hypothesis generation = a false dichotomy
* Models are often used for exploration, + w/ a little care you can use visualisation for confirmation.
* Key difference = how often you look at each observation:
* If you look only once = confirmation If you look more than once = exploration.
* tidyverse core =  ggplot2, tibble, tidyr, readr, purrr, dplyr (updates **🡺 tidyverse\_update()**)
* 3 things you to include to make examples reproducible: required packages, data, code.
* **Packages** = loaded at the *top of the script*, so it’s easy to see which ones an example needs.
* good time to check you’re using the latest version of each package;
* Easiest way to include **data** in a question is to use **dput()** to generate R code to recreate it.
* Ex: Recreate mtcars dataset in R, I’d perform the following steps:
* dput(mtcars) in R 🡺 Copy the output 🡺 type “mtcars <-“ + then paste.
* Try and find the smallest subset of your data that still reveals the problem.
* Spend a little bit of time ensuring your **code** is easy for others to read:
* Use spaces + use variable names that’re concise, yet informative.
* Use comments to indicate where your problem lies.
* Do your best to remove everything that is not related to the problem.
* The shorter your code is, the easier it is to understand, and the easier it is to fix.
* Finish by checking you’ve actually made a reproducible example by starting a fresh R session + copying + pasting your script in.

**Chapter 2: Explore Intro**

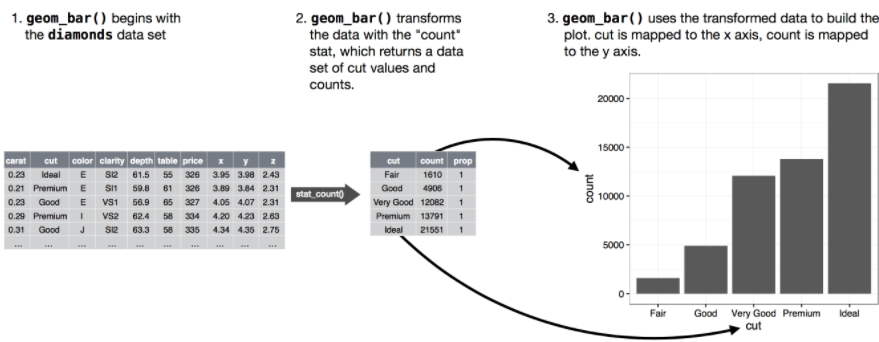
* **EDA** = art of looking at data, rapidly generating hypotheses, quickly testing them, then repeating again + again + again.
* Goal = to generate many promising leads you can later explore in more depth.

**Chapter 3 - Visualization**

* “The simple graph has brought more information to the data analyst’s mind than any other device.” — John Tukey
* **ggplot2** implements the **grammar of graphics**, a coherent system for describing + building graphs.
* “The greatest value of a picture is when it forces us to notice what we never expected to see.” — John Tukey
* Inside aes(), can **map** a 3rd variable to a 2D scatterplot w/ an **aesthetic** = visual property of objects in a plot (shape, size, color of points) that you can map to variables to display info about data
* This assigns a unique level (here, color) to the aesthetic = **scaling**
* Don’t use discrete (unordered) variable for size aesthetics (ordered)
* Shape only takes 6 levels at a time
* aes() takes a variable + associates an aesthetic w/ it by gathering each aesthetic mapping used by a layer + passing them to the layer’s mapping argument
* x + y points are themselves aesthetics
* ggplot2 takes care of reasonable scales for an aesthetic + constructs a legend to explain the mapping
* Set an aesthetic *manually* 🡺set the aesthetic by name as an arg of a geom function (outside aes())
* **Facets** = subplots that each display 1 subset of the data
* Single variable = facet\_wrap(~ ‘discrete\_variable’) where ~ = “by”
* ~ creates a **formula** = a data structure in R, NOT an equation
* Combo of 2 variables = facet\_wrap(‘x\_axis\_var’ ~ ‘y-axis-variable’)
* **geom =** geometric object a plot uses to represent data
* Every geom function in takes a mapping arg, but not every aesthetic works w/ every geom.
* shape of a point, no “shape” of a line (could set linetype of a line)

**Statistical Transformations**

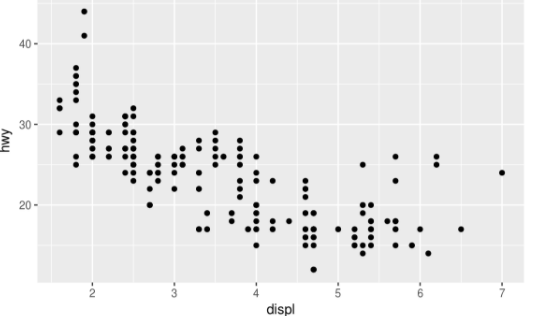
* Bar charts seem simple, but are interesting b/c they reveal something subtle about plots.
* Many graphs, like scatterplots, plot raw values of a dataset.
* Other graphs, like bar charts, calculate new values to plot:
* bar charts, histograms, frequency polygons 🡪 **bin** data + then plot **bin counts** (# of points that fall in each bin.
* smoothers fit a model to data + then plot predictions from the model.
* boxplots compute a robust summary of the distribution + display a specially formatted box.
* **stat** (short for statistical transformation) = the algorithm used to calculate new values for a graph



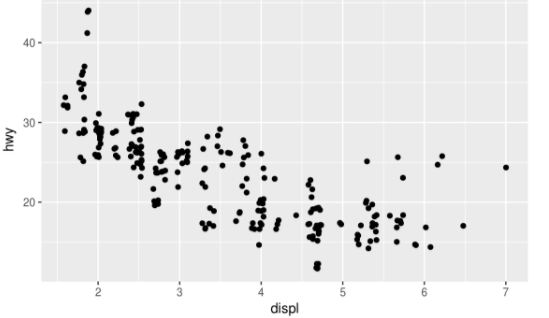
* Can learn which **stat** a geom uses by inspecting the default value for the **stat** argument.
* geom\_bar default value for stat = “count” 🡪 geom\_bar() uses **stat\_count() =** computes 2 new variables: **count** + **prop**.
* Can generally use geoms + stats *interchangeably* b/c every geom has a default stat + every stat has a default geom.
* There are 3 reasons you might need to use a stat explicitly:
* To override the default stat.
* Changing the stat of geom\_bar() from count (default) to **identity** lets us *map the height of the bars to RAW values of a y variable*.
* To override default mapping from transformed variables to aesthetics.
* Ex: display a bar chart of proportion, rather than count:
* To draw greater attention to a statistical transformation in the code
* Ex: use **stat\_summary()** tosummarizes the y-values for each unique x-value to draw attention to the summary you’re computing:
* ggplot2 provides over 20 stats to use + each stat is a function
* To see a complete list of stats, try the ggplot2 cheatsheet or ?stat\_bin

**3.8 Position adjustments**

* **Stacking in** stacked bar charts = performed automatically by **position adjustment** specified by the **position** argument.
* If don’t want a stacked bar chart, can use 1 of 3 other options:
* **Identity** = places each object exactly where it falls in the context of the graph.
* Not very useful for bars b/c it overlaps them.
* To see the overlapping either need to make bars slightly transparent by setting **alpha** to a small value or completely transparent by setting **fill** = NA
* More useful for 2D geoms, like points, where it is the default.
* **Dodge =** places overlapping objects directly *beside* one another
* Makes it easier to compare individual values.
* **Fill =** works like stacking, but makes each set of stacked bars the same height + each color a proportion
* Makes it easier to compare proportions across groups.
* Another type of adjustment not useful for bar charts, but for scatterplots is **jitter**



* This displays only 126 points, even though there are 234 observations in the dataset
* Values of hwy + displ are *rounded* so points appear on a grid + many points overlap = **overplotting**
* Makes it hard to see where the mass of the data is. Are DP’s spread equally throughout the graph, or is there 1 special combination of hwy + displ that contains 109 values?
* **Jitter** adds a small amount of random noise to each DP to spread DP’s out b/c no 2 points are likely to receive the same amount of random noise.



* makes graph less accurate at small scales, but more revealing at large scales

**3.9 Coordinate systems**

* **Coordinate systems** = probably most complicated part of ggplot2
* Default coordinate system = **Cartesian coordinate system** where x + y positions act *independently* to determine the location of each point. There are others
* **coord\_flip()**switches x + y axes.
* useful if you want horizontal boxplots or for long labels (hard to get them to fit w/out overlapping on the x-axis
* **coord\_quickmap()**sets aspect ratio correctly for maps.
* Very important if plotting **spatial data**
* **coord\_polar()**uses **polar coordinates** = interesting connection between a bar chart + a **Coxcomb chart**

**3.10 The Layered Grammar Of Graphics**

* Foundation to make any type of plot w/ ggplot2:

ggplot(data = <DATA>) +

<GEOM\_FUNCTION>(

mapping = aes(<MAPPINGS>),

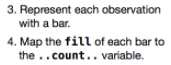
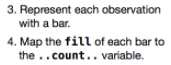
stat = <STAT>,

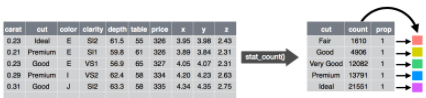
position = <POSITION>

) +

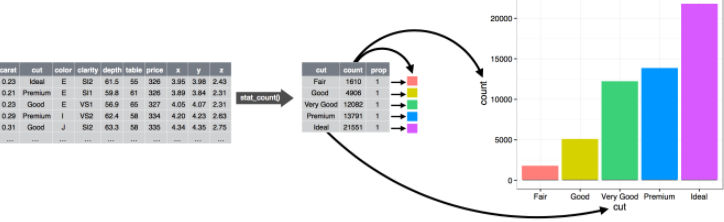
<COORDINATE\_FUNCTION> +

<FACET\_FUNCTION>

* Raw template takes 7 parameters = bracketed words
* In practice, rarely need to supply all 7 parameters to make a graph b/c ggplot2 provides useful defaults for everything except data, mappings, + a geom function.
* The 7 parameters compose the **Grammar Of Graphics** = formal system for building plots
* **The grammar of graphics** = based on insight that you can uniquely describe any plot as a combo of a dataset, geom, set of mappings, stat, position adjustment, coordinate system, + a faceting scheme.
* Consider building a basic plot from scratch: start w/ a dataset + transform it into info to display
*  
* Next, choose a geometric object (**geom**) to represent each observation in the transformed data + use the **aesthetic** properties of geoms to *represent variables in the data* + **map** the *values of each variable* to the *levels of an aesthetic.*
*  



* Then select a **coordinate system** to *place the geoms into* by using the location of the objects (itself an aesthetic property) to display the values of the x + y variables.
* Can further this complete graph by adjusting positions of geoms w/in the coordinate system (**position adjustment**) or split a graph into subplots (**faceting**). You
* Could also extend a plot by adding 1+ additional layers, where each additional layer uses a dataset, a geom, a set of mappings, a stat, and a position adjustment.
* 



* Can use this method to build *any* plot you imagine (hundreds of thousands of unique plots)