# Visulzig data

*Marc Kissel*

*2/8/2019*

PACKAGES WE NEED

1. tidyverse
2. gapminder
3. scales
4. nycflights

## 3. EDA

##What is EDA? - idea is to generate questions about the data – such as what types of variation do i see – how do the varibles covary - search for answers (by visualizing, transforming, and modeling) - then, use what we learn to refine our questions

### first step is often looking a the data.

```
#install.packages("tidyverse")
#install.packages("modelr")
library(tidyverse)
```

```
## -- Attaching packages ---------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## Warning: package 'ggplot2' was built under R version 3.5.1
```

```
## Warning: package 'tidyr' was built under R version 3.5.2
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
## -- Conflicts ------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(modelr)
```

one part of the **Tidyverse** is a package called ggplot2. we are going to use this to explore the diamonds dataset.

First, lets look at the info that comes with the dataset

```
?diamonds
```

```
## starting httpd help server ... done
```

take a minute and read the help. what info is here?

Now we want to take a look at the actual dataset. there are a few ways we can do this, each with their own pros/cons

1. str

```
str(diamonds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  10 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

2. glimpse

```
glimpse(diamonds)
```

```
## Observations: 53,940
## Variables: 10
## $ carat   <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, ...
## $ cut     <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very G...
## $ color   <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, ...
## $ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI...
## $ depth   <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, ...
## $ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54...
## $ price   <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339,...
## $ x       <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, ...
## $ y       <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, ...
## $ z       <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, ...
```

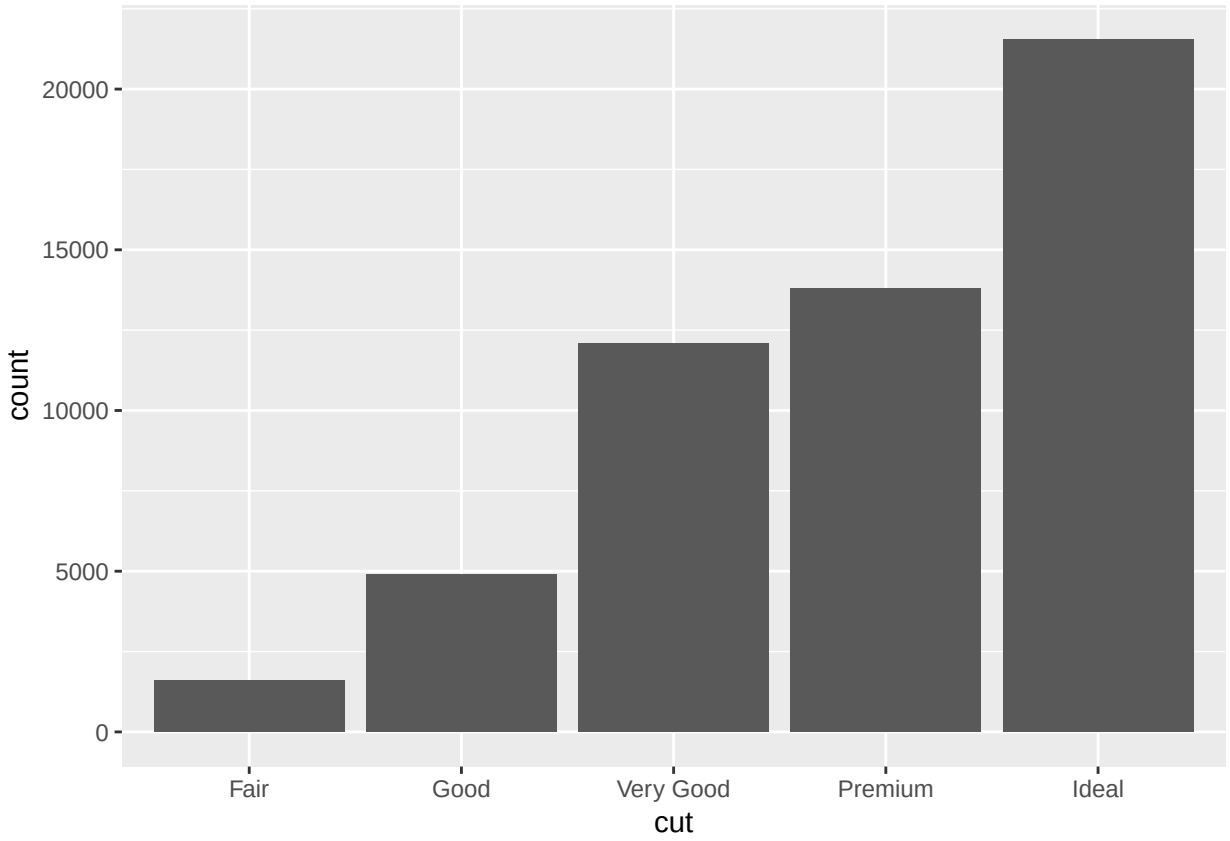
3. print

```
print(diamonds)
```

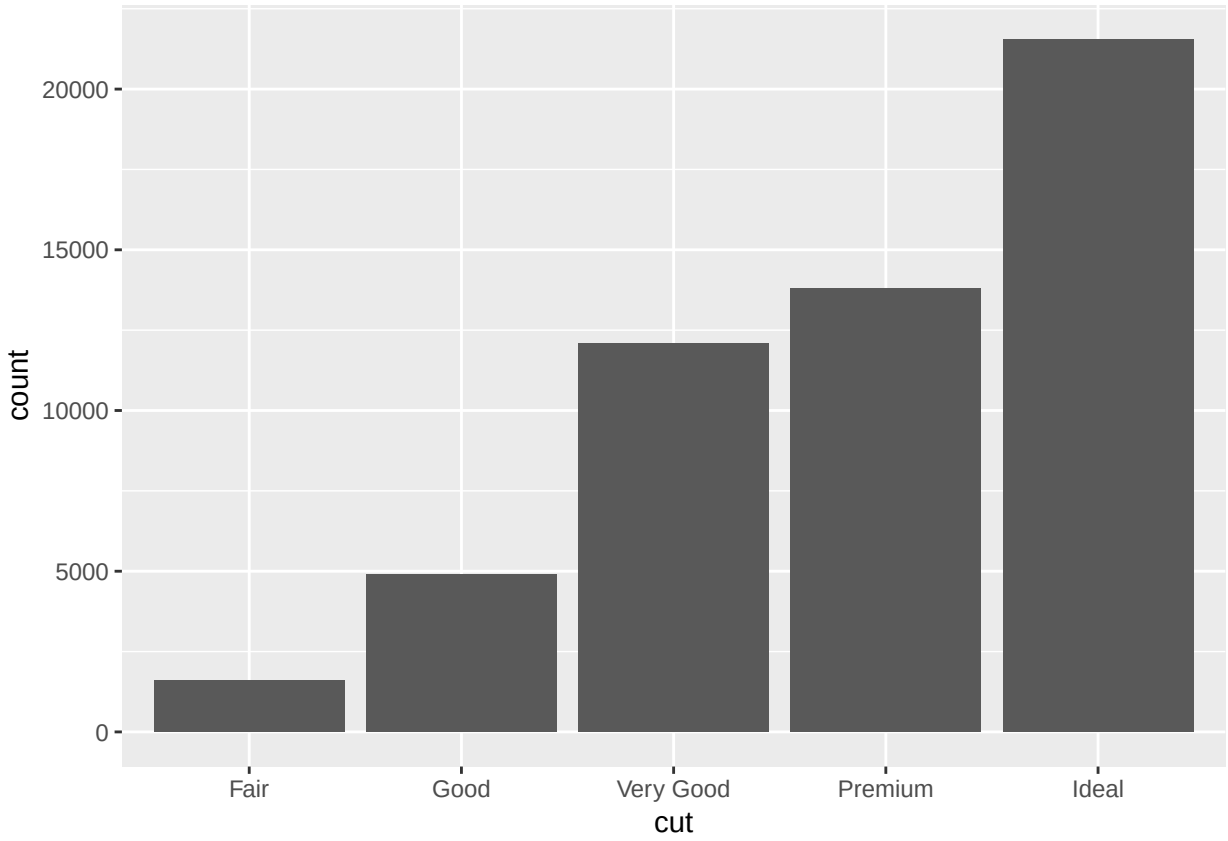
```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4 0.290 Premium   I     VS2      62.4    58   334  4.2   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
##  7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8 0.26  Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10 0.23  Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

# look at the data

```
ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut))
```




```
ggplot(diamonds) + geom_bar(aes(x = cut)) #same thing, just simpler
```




#if we want the actual count, we can find that using some of the dyply commands we learned before

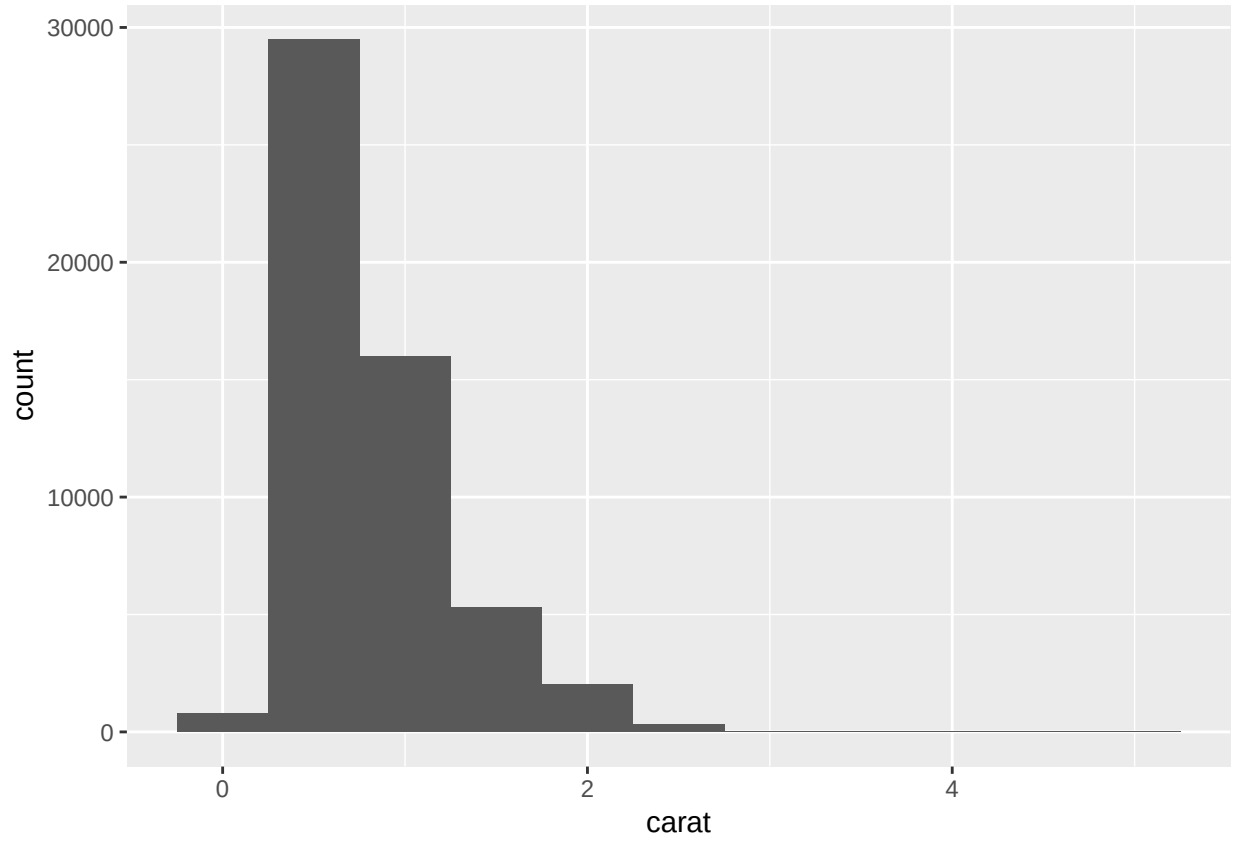
```
diamonds %>% count(cut)
```

```
## Warning: The `printer` argument is soft-deprecated as of rlang 0.3.0.
## This warning is displayed once per session.
```

```
## # A tibble: 5 x 2
##   cut           n
##   <ord>     <int>
## 1 Fair       1610
## 2 Good       4906
## 3 Very Good 12082
## 4 Premium   13791
## 5 Ideal     21551
```

#continuous data is better seen as a histogram.

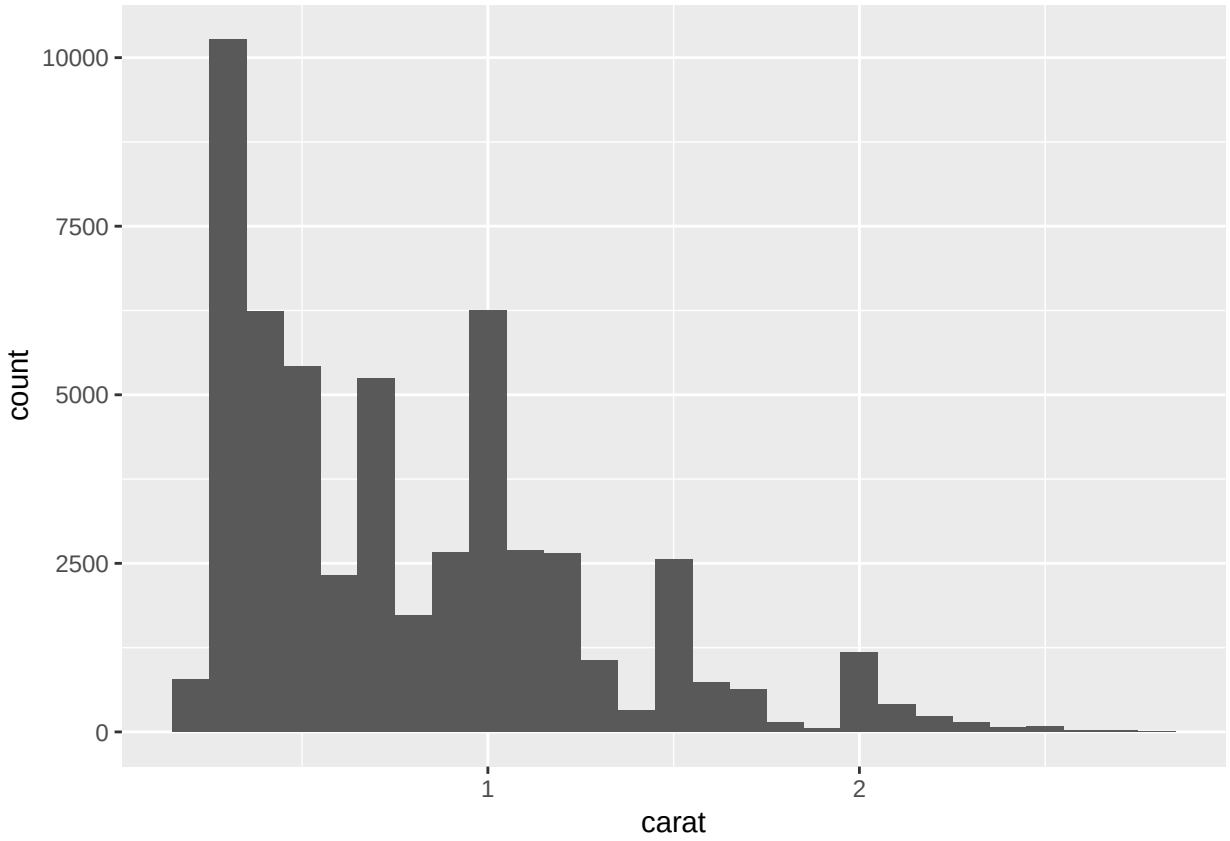
```
ggplot(diamonds) + geom_histogram(aes(x = carat), binwidth = 0.5)
```




```
diamonds %>%  count(cut_width(carat, 0.5)) #cut_width = cut up numeric vector into useful groups...make
```

```
## # A tibble: 11 x 2
##    `cut_width(carat, 0.5)`     n
##    <fct>                   <int>
##  1 [-0.25,0.25]              785
##  2 (0.25,0.75]             29498
##  3 (0.75,1.25]             15977
##  4 (1.25,1.75]              5313
##  5 (1.75,2.25]              2002
##  6 (2.25,2.75]               322
##  7 (2.75,3.25]                32
##  8 (3.25,3.75]                 5
##  9 (3.75,4.25]                 4
## 10 (4.25,4.75]                 1
## 11 (4.75,5.25]                 1
```

```
smaller <- diamonds %>%  filter(carat < 3)
ggplot(data = smaller, mapping = aes(x = carat)) +   geom_histogram(binwidth = 0.1)
```




#tall bars show the common values of a variable, and shorter bars show less-common values. #Places that do not have bars reveal values that were not seen in your data. To turn this #information into useful questions, look for anything unexpected: #you can use ggplot to overlay multiple histograms, but be careful. it is better to use a frequency #plot as this uses lines

ggplot(data = smaller, mapping = aes(x = carat, colour = cut)) + geom_freqpoly(binwidth = 0.1) #interpret the same way, but can be used to compare sets of data #i.e. compared to this one ggplot(data = smaller, mapping = aes(x = carat, colour = cut)) + geom_histogram(binwidth = 0.1)

ggplot(data = smaller, mapping = aes(x = carat, fill=cut)) + geom_density(alpha =.5) #In a density plot, areas can be interpreted as probabilities and the area under the entire #histogram is equal to 1.

#lets look at this plot and think about questions. How would you use EDA?

ggplot(data = smaller, mapping = aes(x = carat)) + geom_histogram(binwidth = 0.01)

#what values are most common? #what values are rare? why? #any weird patterns? ########## #why are there more counts on full carats... why no bigger than 3... why are they skewed to the right

#looking for outliers....good way to do this is to do a histogram and see what ggplot shows

ggplot(diamonds) + geom_histogram(mapping = aes(x = y), binwidth = 0.5) #y is col in the dataset

#hmmm... why is x axis so long? maybe we can't see the measurmenrts on count since it is so high #super fun function: coord_cartesian(ylim =c(0,50)) #what do you think this does ggplot(diamonds) + geom_histogram(mapping = aes(x = y), binwidth = 0.5) + coord_cartesian(ylim = c(0, 50)) # so, we limit y-axs to 50 to see the smaller values

unusual <- diamonds %>% filter(y < 2 | y > 20) %>% arrange(y) unusual #what is up with these outliers. #point about removing outliers #there are tests you can do, but also need to think through data #can't just

remove without reasons. if they affect your results, may have to remove them but only #if you have legit reason. and have to spell it out in your paper@!

#how to deal with values that are wrong # you could delete the whole row, but that gets rid of data that might be right #replace the 'wrong' value with NA #how to do that: we need to select the ones to replace and then tell R what to do #how ifelse works

diamonds2 <- diamonds %>% mutate(y = ifelse(y < 3 | y > 20, NA, y)) diamonds2 %>% filter(y < 2 | y > 20) %>% arrange(y) #they are no longer being found. . . since NA

#var. looks at behavior within a variable, covariation looks at behavior between vars #. the tendency for vars to vary together

#first, categorical data

ggplot(data = diamonds, mapping = aes(x = price)) + geom_freqpoly(mapping = aes(colour = cut), binwidth = 500) #but, this is hard to read. there are too many examples of ideal diamonds #density plots could help. . . .this sets area under each polygon to one

ggplot(data = diamonds, mapping = aes(x = price, y = ..density..)) + geom_freqpoly(mapping = aes(colour = cut), binwidth = 500)

#hmmm. . . looks like the Fair is the most money. . . that's weird. . . what else can we do #boxplots are great ways to look at this stuff

ggplot(data = diamonds, mapping = aes(x = cut, y = price)) + geom_boxplot()

#how do you read a boxplot? #IQR = interquarttile range diamonds %>% group_by(cut) %>% summarise (mean_price = mean(price))

#A percentile tells where a given value falls in a distribution

diamonds %>% group_by(cut) %>% summarise(25%=quantile(price, probs=0.25), 50%=quantile(price, probs=0.5), 75%=quantile(price, probs=0.75), avg=mean(price), n=n()) #The interquartile range is the width of the 50 percent coverage interval: the #difference between the 75th and 25th percentiles

#IMPORTANT: you can change the order of the boxplot cat with 'reorder'

ggplot(data = diamonds, mapping = aes(x = reorder(cut, price), y = price)) + geom_boxplot() #note how reorder works, we order the cut variable by the price variable ?reorder

#note u can flip the boxplot with 'coord_flip()' ggplot(data = diamonds, mapping = aes(x = reorder(cut, price), y = price)) + geom_boxplot() + coord_flip()

#there is also a package called 'ggstance' that makes horizontal boxplots that look very nice and can easily manipulate order. Might be nice to learn this one!

#often, you want to look at covar between categorical data„,

#quick way is using geom_count ggplot(data = diamonds) + geom_count(mapping = aes(x = cut, y = color))

#maybe a better way. . . . diamonds %>% count(color, cut)

diamonds %>% count(color, cut) %>%
ggplot(mapping = aes(x = color, y = cut)) + geom_tile(mapping = aes(fill = n))

#ok, how to look for patterns. . .

ggplot(data = diamonds) + geom_point(mapping = aes(x = carat, y = price)) #but hard to see with so much data #we can use bins for 2d data as well! #two new functions geom_bin2d and geom_hex() #these divide the plane into 2d bins and then use a fill colour to display #how many points fall within a bin # geom_bin2d() creates rectangular bins. geom_hex() creates hexagonal bins. You will need to install the hexbin package to use geom_hex().

install.packages("hexbin") library(hexbin)

ggplot(data = smaller) + geom_bin2d(mapping = aes(x = carat, y = price)) ggplot(data = smaller) + geom_hex(mapping = aes(x = carat, y = price))

#what is the dif between ter 2 plots #Hexagon bins avoid the visual artefacts sometimes generated by the very regular alignment of geom_bin2d.

#patterns #what does a pattern that you find in your data mean? could it be random chance? #good advice from wickham on what to do when you see a pattern

#Could this pattern be due to coincidence (i.e. random chance)?

#How can you describe the relationship implied by the pattern?

#How strong is the relationship implied by the pattern?

#What other variables might affect the relationship?

#Does the relationship change if you look at individual subgroups of the data?

#IMPORAT: models are a tool for extracting patterns out of data

#but we can look at more complex things. diamond dataset. what is relationship between cut and price? this is hard, since cut & carat and carat & price are related #so, we model to remove strong relationships between price and carat and see what remains

#predicts price from carat and then computes the residuals (the difference between the predicted value and the actual value). The residuals give us a view of the price of the diamond, once the effect of carat has been removed.

ggplot(data = diamonds) + geom_point(mapping = aes(x = carat, y = price))

###MORE ON modelr? ##### ####has lots of useful functions that allow us to use Tidy data in R w #one thing is that add predictions and residuals as additional columns to an existing data frame:

#example of this from the modelR gtihub

df <- tibble::data_frame( x = sort(runif(100)), y = 5 * x + 0.5 * x ^ 2 + 3 + rnorm(length(x)) #runif makes a uniform dist(n, min, max) )

df mod <- lm(y ~ x, data = df) mod df %>% add_predictions(mod)

df %>% add_residuals(mod)

mod <- lm(log(price) ~ log(carat), data = diamonds) mod summary(mod)

diamonds2 <- diamonds %>% add_residuals(mod) %>% mutate(resid = exp(resid)) #what does this line do? fixs log issue diamonds2

ggplot(data = diamonds2) + geom_point(mapping = aes(x = carat, y = resid)) #so we are now looking #he residuals give us a view of the price of the diamond, once the effect of carat has been removed.

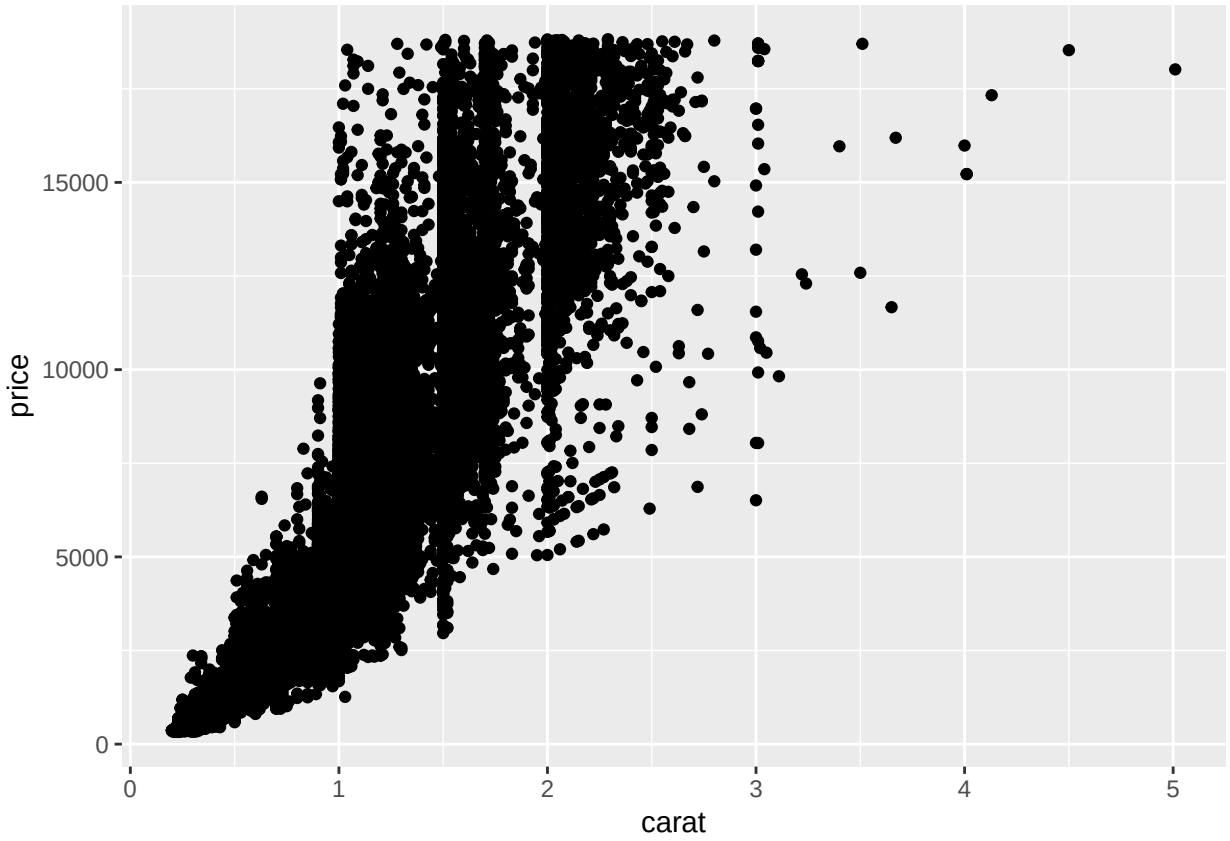
ggplot(data = diamonds2) + geom_boxplot(mapping = aes(x = cut, y = resid))

---

**much of this comes from examples from R4DS and the visulzation book**

#start with a basic plot just to get the idea

```
ggplot(diamonds, aes(x = carat, y= price)) +
geom_point()
```




this called the functio ggplot and tells it 1. use the diamonds dataset 2. set the aesthetic to be x = carat and y = price 3. set the type of plot to be a scatter plot

To save this plot, we can use a function called ggsave (whihc is part of the ggplot2 package)

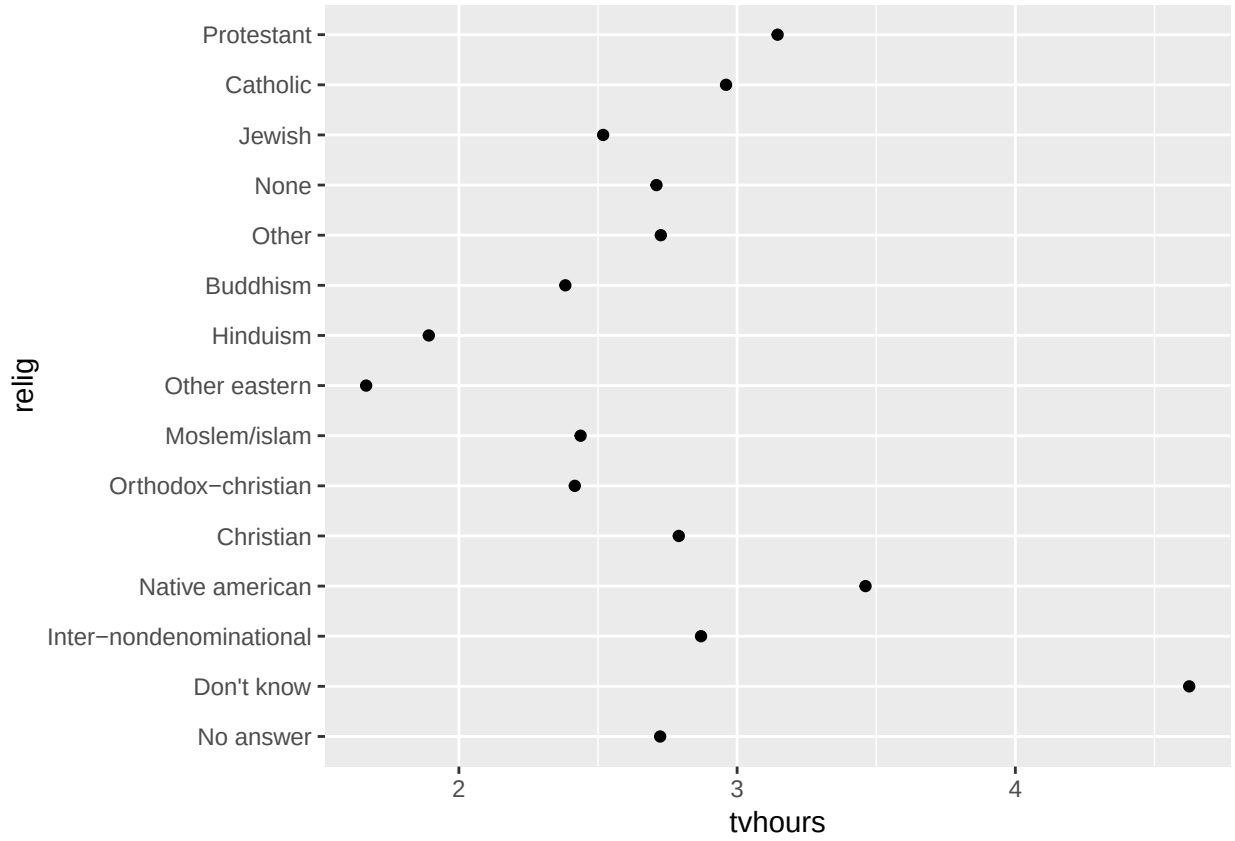
```
ggsave("diamond_plot.pdf")
```

```
## Saving 6.5 x 4.5 in image
```

#factors XXXXXXXXXXXXXXXX

often when making plots we want to change the order they appear in. one was is shown above. another way is to use the forcats package that comes with the tidyverse or load it by itself library(forcats) this comes with a datasset called *gss_cat*

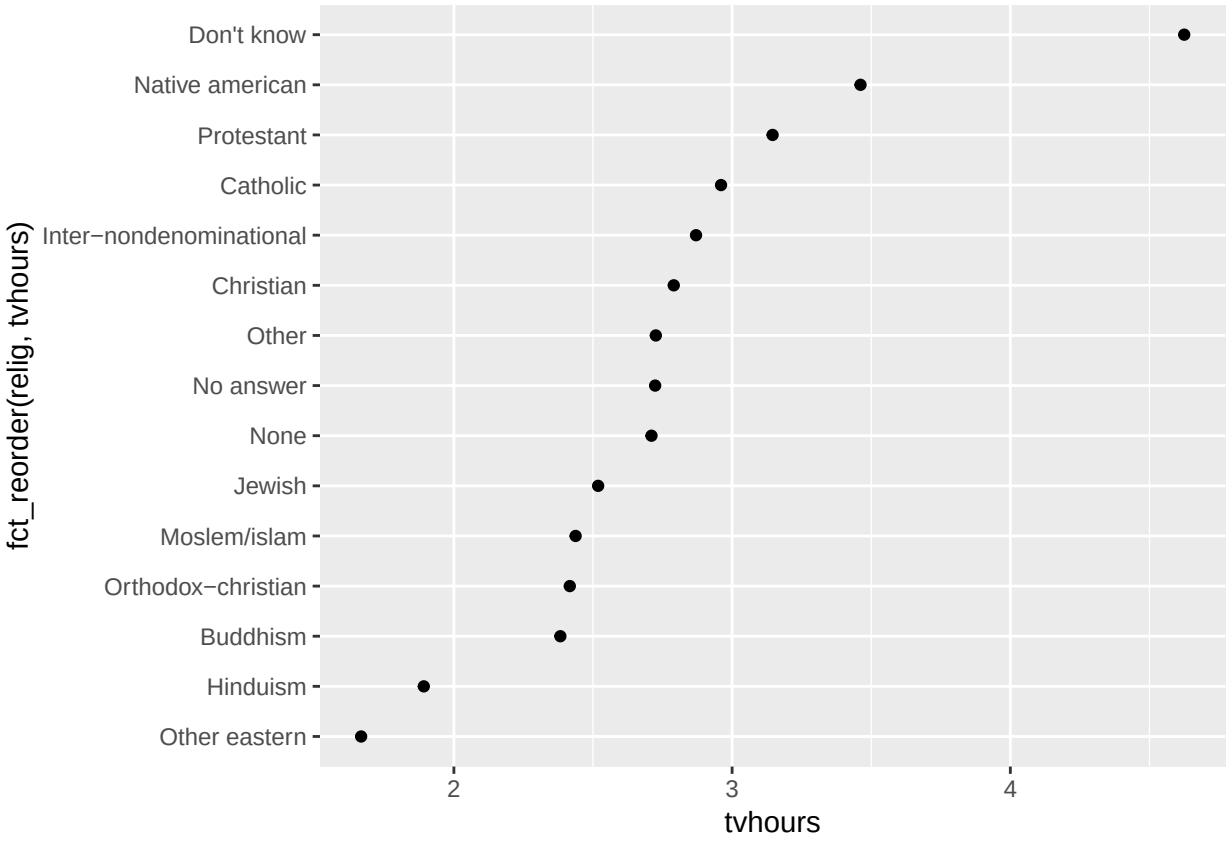
```
relig_summary <- gss_cat %>%
group_by(relig) %>%
summarise(
age = mean(age, na.rm = TRUE),
tvhours = mean(tvhours, na.rm = TRUE),
n = n()
)
ggplot(relig_summary, aes(tvhours, relig)) + geom_point()
```




ok, so this tells us something but hard to read? what relgion has the most tv hrs? not clear at first look?

when we plot this the factors are not organized by tv hours, which makes it hard to examine We can improve it by reordering the levels of relig using fct_reorder(). fct_reorder() takes A few arguments: #1 the factor we want to reorder #2 the numeric vector we want to reorder by

```
ggplot(relig_summary, aes(tvhours, fct_reorder(relig, tvhours))) +
geom_point()
```




#GGPLOT2

For this section we are going to use the gapminder package, which i load below

```
library(gapminder)
```

there are numerous plotting packages. Ggplot2 has become the standard for a number of reasons. it is based on an idea called the Grammar of Graphics by Leland Wilkinson while this package was developed by Hadley Wickham. His book is pretty useful:

https://www.amazon.com/ggplot2-Elegant-Graphics-Data-Analysis/dp/331924275X/ref=as_li_ss_tl?ie=UTF8&linkCode=sl1&tag=ggplot2-20&linkId=4b4de5146fdafd09b8035e8aa656f300

From the Wickham book we learn that Every ggplot2 plot has three key components:

1. data,
2. A set of aesthetic mappings between variables in the data and visual properties
3. At least one layer which describes how to render each observation. Layers are usually created with a geom function.

Another great resource is this book http://socviz.co/index.html#preface I learned a lot from it and as of now it is free online

In ggplot, the connections between your data and the plot elements are called *aesthetic mappings* or just *aesthetics*.

1. we begin every plot by telling the ggplot() function what your data is, and how the variables map onto the aesthetics.
2. Then we tell it what kind of plot (called a **geom**) we want. can be a scatterplot, a histogram, etc. geom_point() makes scatterplots and geom_boxplot() makes boxplots

3. we then combine these two pieces, the ggplot() object and the geom, by adding them together in an expression, using the + symbol.

general steps from Data Viz book

1.Tell the ggplot() function what our data is. The data = step. 2.Tell ggplot() what relationships we want to see. The mapping = aes step. Put the results of the first two steps in an object.

3.Tell ggplot how we want to see the relationships in our data. Choose a geom.

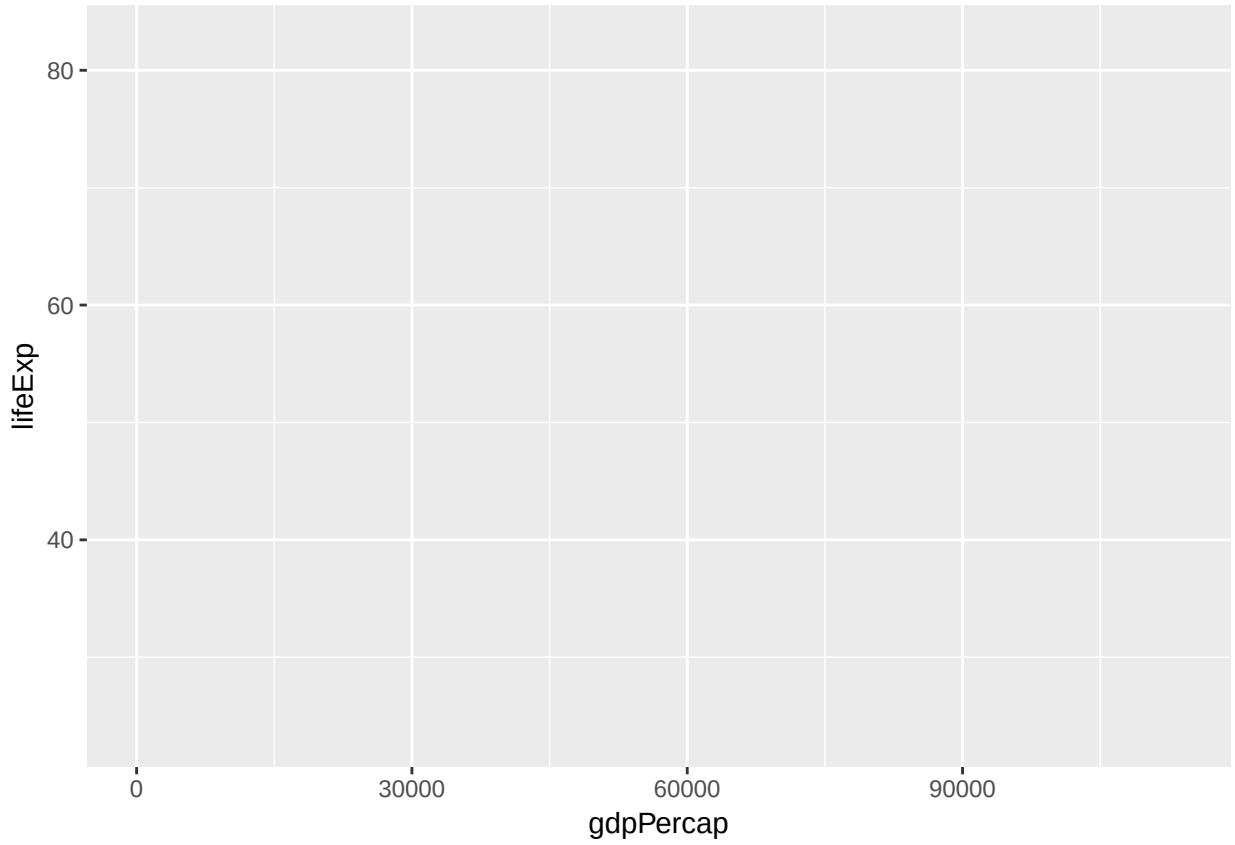
4. Layer on geoms as needed, by adding them to the object one at a time.
5. Use The scale_, family, labs() and guides() functions. some additional functions to adjust scales, labels, tick marks, titles.

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp))
```

the above code says "make a r object called p. this is going to use the data in the gapminder data table and we are going to want the x-axis to be gdpPercap and the y-axis to be lifeExp"

if you then told R to print p you'd see an empty plot

```
p
```

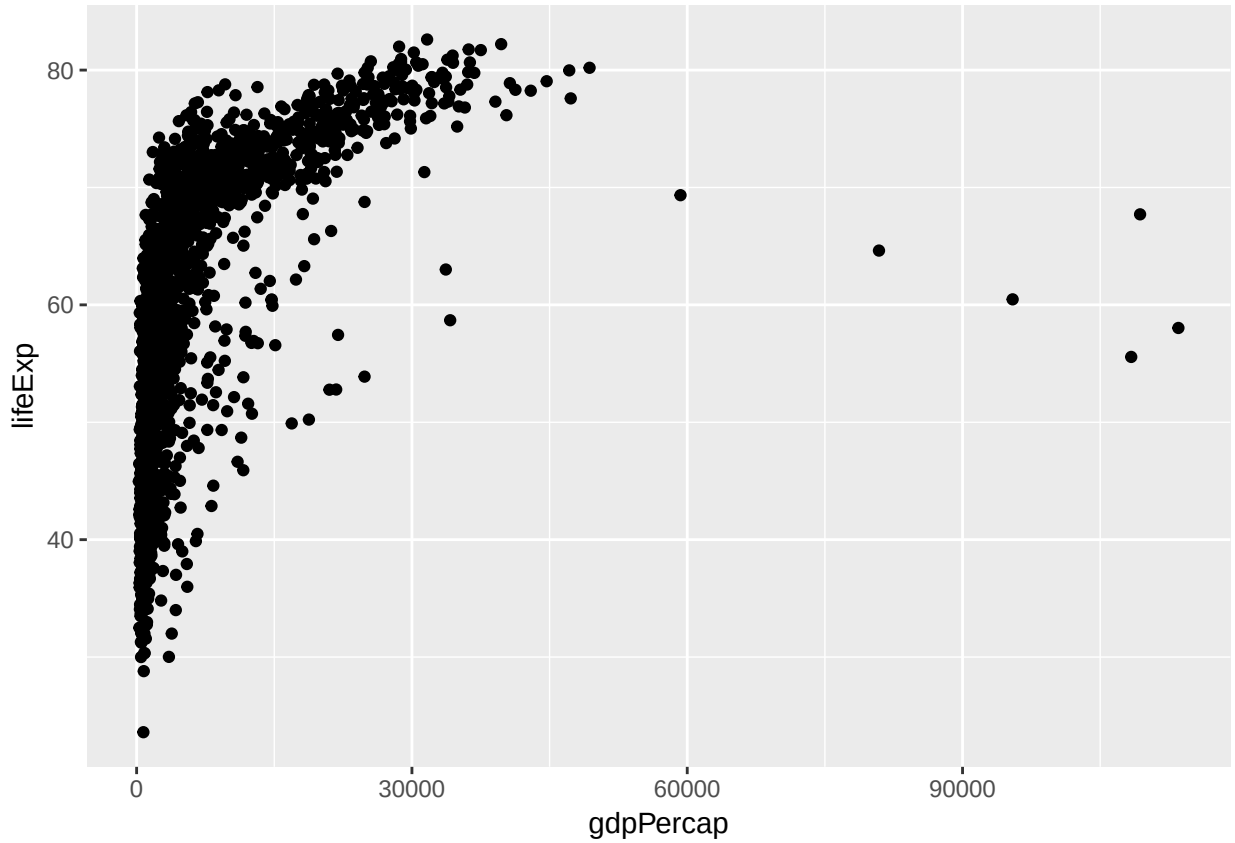



This is a happy thing! it has all the data there, but it doesn't know what we want to do. if you were to check the structure of p (can do so with str(p) ) you would see all kinds of thing behind the scenes.

the next step is to add layers to plots

when adding layers we dont need to tell ggplot whee the data is coming from since it inherits it from the original object. but we can change where the data is coming from if we wish!

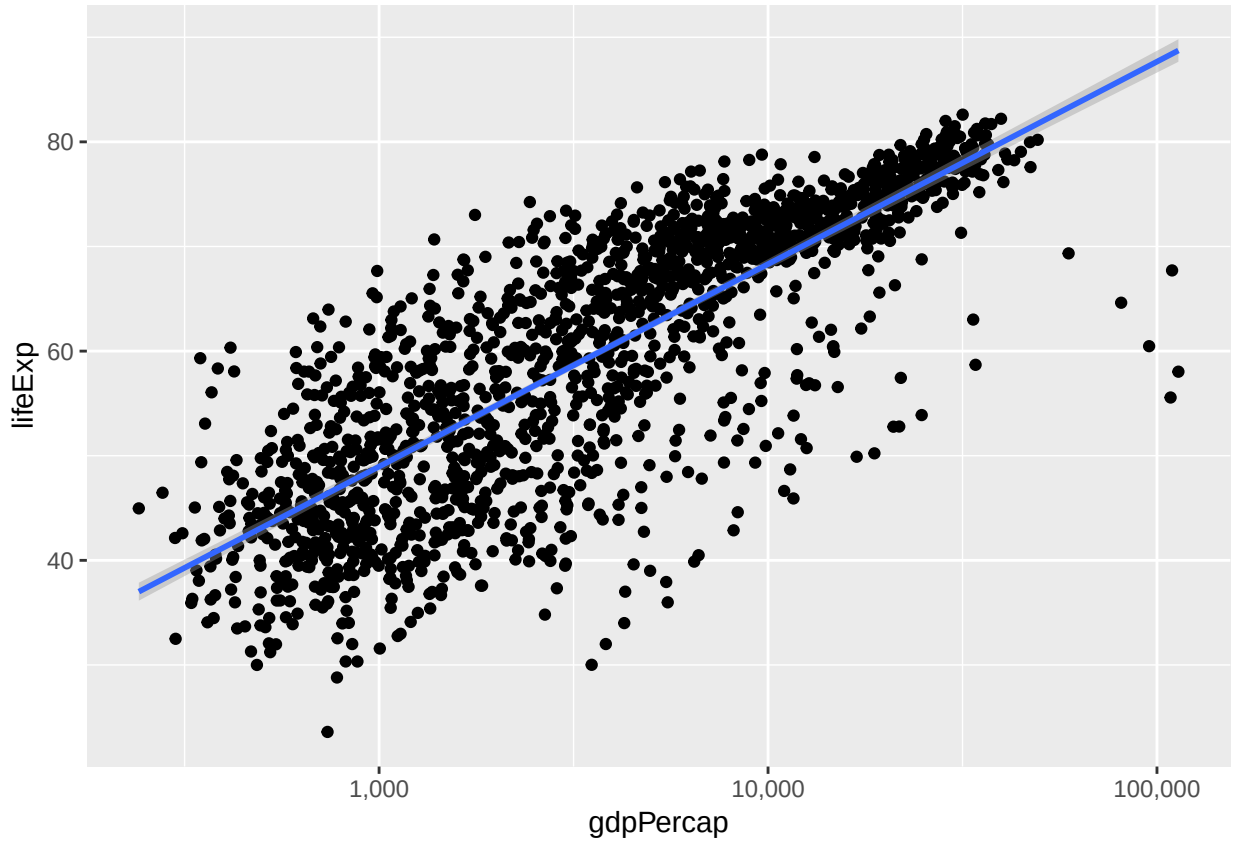
```
p + geom_point()
```




###SCALES can transform the x and y axis directly with scale_x_log10 etc. this will transform axis and tick marks will be in scientific notions. we can also give scale_X_log a label that reformat the text under the axis

protip: look at the line "scale_x_log10(labels =scales::comma)" the labels = scales::comma is kinda odd. what is going on here is that we are calling a function, in this case 'comma', from the scales package, without loading that package. this is sometimes useful and a good move to keep in your back pocket

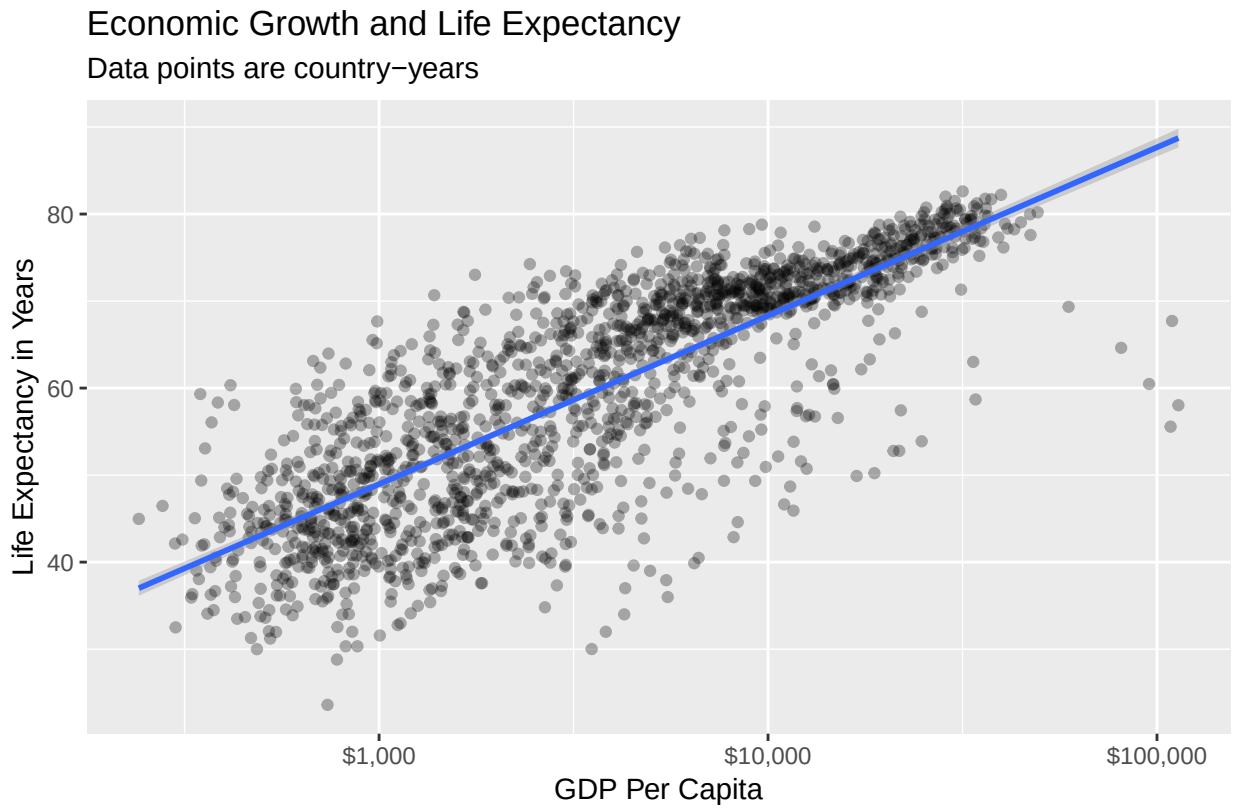
```
p <- ggplot(data = gapminder, mapping = aes(x = gdpPercap, y=lifeExp))
p + geom_point() + geom_smooth(method = "gam") + scale_x_log10(labels =scales::comma)
```




```
p <-  ggplot(data = gapminder,
             mapping = aes(x = gdpPercap,
                           y = lifeExp,
                           color = continent))
```

###Adding lables to the graph

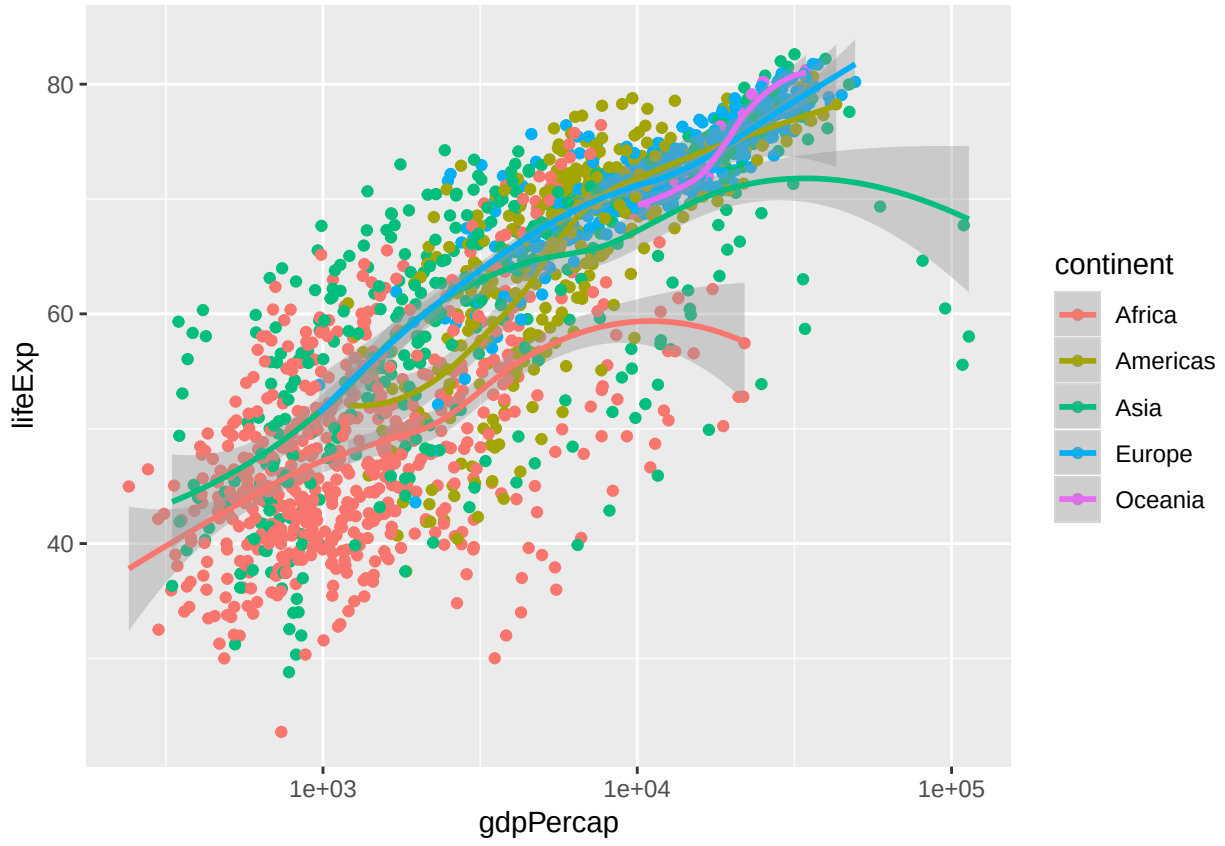
```
p <- ggplot(data = gapminder, mapping = aes(x = gdpPercap, y=lifeExp))
p + geom_point(alpha = 0.3) +
    geom_smooth(method = "gam") +
    scale_x_log10(labels = scales::dollar) +
    labs(x = "GDP Per Capita", y = "Life Expectancy in Years",
         title = "Economic Growth and Life Expectancy",
         subtitle = "Data points are country-years",
         caption = "Source: Gapminder.")
```




###Adding groups

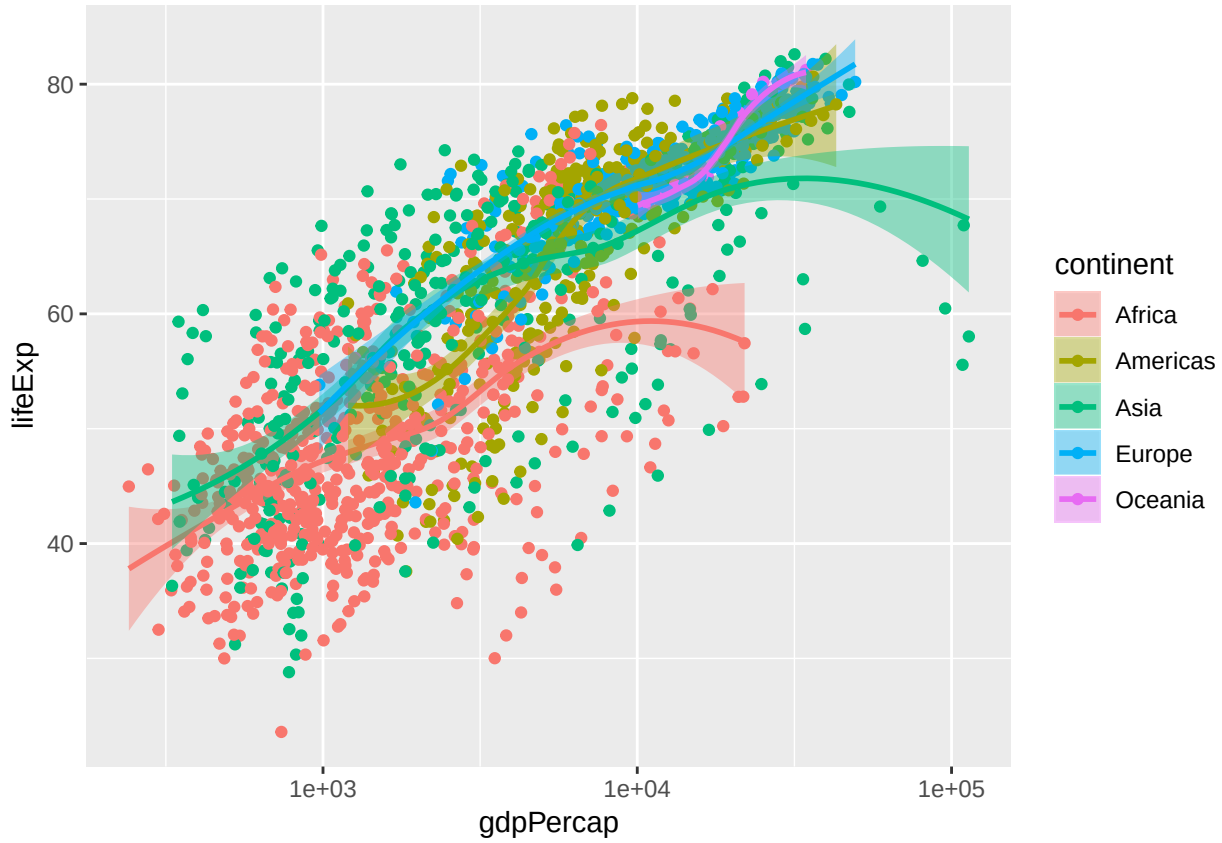
When setting aesthetic we can also make a new mapping. often we want to group things together. can group by color, shape, and size. fun to play with these things

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp,
                          color = continent))
p + geom_point() +
    geom_smooth(method = "loess") +
    scale_x_log10()
```




let's say we want the color of the standard error ribbon to match that of the dominant color. this color is controlled by the fill aesthetic like this

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp,
                          color = continent,
                          fill = continent))
p + geom_point() +
    geom_smooth(method = "loess") +
    scale_x_log10()
```




As you play around with ggplot you can learn all kinds of tricks and tips!

####ok, lets look at some data as a team!

FIND A GOOD TIDY TUSEDAY DATASET

##########################################################################