Introduction to R for Social and Behavioral Sciences

Corey Sparks, PhD - UTSA Department of Demography

October 19, 2017

Contents

Welcome to R.

R is an interpreted languages, not a compiled one. This means, you type something into R and it does it.

If you're coming to R from SAS, there is no data step. There are no procs. The SAS and R book is very useful for going between the two programs.

If you're coming from SPSS and you've been using the button clicking method, be prepared for a steep learning curve. If you've been writing syntax in SPSS, you're at least used to having to code. There's a good book for SAS and SPSS users by Bob Meunchen at the Univ. of Tennessee here, which may be of some help.

R and Rstudio

The Rgui is the base version of R, but is not very good to program in. Rstudio is much better, as it gives you a true integrated development environment (IDE), where you can write code in on window, see results in others, see locations of files, see objects you've created

R file types

.R files R uses a basic script file with the .R extension. This type of file is useful if you're going to write a function or do some analysis and don't want to have formatted output or text.

.Rmd files Rstudio uses a form of the markdown formatting language, called Rmarkdown, for creating formatted documents that include code chunks, tables, figures and statistical output. **This entire example is written in Rmarkdown!**

Rmarkdown is nice for lots of reasons, such as the abiltity to insert latex equations into documents

$$y_i \sim Normal(x`\beta, \sigma_2)$$

or to include output tables directly into a document:

term	estimate	std.error	statistic	p.value
(Intercept)	18.78	9.803	1.916	0.05723
tfr	13.56	2.227	6.089	8.807e-09
percurban	-0.263	0.06675	-3.94	0.0001236
percpoplt15	0.1949	0.2951	0.6604	0.51
percmarwomcontraall	-0.2579	0.09202	-2.803	0.005724

without having to make tables in Word or some other program. You can basically do your entire anlaysis and slideshow or paper writeup, including bibliography in Rstudio.

R Notebooks In recent versions of Rstudio, another kind of document, called a R Notebook has been created. This is nice because, like a Rmarkdown document, R notebooks include code, text and formatted output, but you can also show and hide code chunks throughout the document.

Getting help in R

I wish I had a nickel for every time I ran into a problem trying to do something in R, that would be a lot of nickles. Here are some good tips for finding help:

1) If you know the name of a function you want to use, but just need help using it, try `?`

```
?lm
```

2) If you need to find a function to do something, try `??`

```
??"linear model"
```

3) If you want to search among other R users' questions to the R list serve, try `RSiteSearch()`

```
RSiteSearch("heteroskedasticity")
```

4) Speaking of which, there are multiple R user email list serves that you can ask questions on, but they typically want an example of what you're trying to do. I wish I also had nickles for each question i've asked and answered on these forums.

5) A good source for all things programming is the statistics branch of Stack Exchange, which has lots of contributed questions and answers, although many answers are either very snarky or wrong or for an old version of a library, so *caveat emptor*.

6) Your local R guru or R user group UTSA is

Using R

Getting around in R

When you begin an R session (open R) you will begin in your home directory. This is traditionally on Windows at 'C:/Users/yourusername/Documents' and on Mac, is '/Users/yourusername'.

If you're not sure where you are you can type `getwd()`, for get working directory, and R will tell you:

```
getwd()
```

```
## [1] "C:/Users/ozd504/Documents/GitHub/r_courses"
```

If you don't like where you start, you can change it, by using `setwd()`, to set your working directory to a new location.

```
setwd("C:/Users/ozd504/Google Drive/")
getwd()
```

R libraries

R uses libraries to do different types of analysis, so we will need to install lots of different libraries to do different things. There are around 10,000 different packages currently for R. These are also organized into *Task Views*, where packages are organized into thematic areas.

Libraries/packages need to be downloaded from the internet, using the `install.packages()` command. You only need to install a package once. E.g.

```
install.packages("car")
```

will install the `car` library. To use the functions within it, type

```
library(car)
```

Now you have access to those functions. You don't need to install the package again, unless you update your R software.

I strongly recommend you install several packages prior to us beginning. I've written a short script on Github you can use it by running:

```
source("https://raw.githubusercontent.com/coreysparks/Rcode/master/install_first_short.R")
```

This will install a few dozen R packages that are commonly used for social science analysis.

R examples

Below we will go through a simple R session where we introduce some concepts that are important for R.

R is a calculator

```
#addition and subtraction
3+7
```

```
## [1] 10
```

```r
3-7
```

```
## [1] -4
```

```r
#multiplication and division
3*7
```

```
## [1] 21
```

```r
3/7
```

```
## [1] 0.4285714
```

```r
#powers
3^2
```

```
## [1] 9
```

```r
3^3
```

```
## [1] 27
```

```r
#sommon math functions
log(3/7)
```

```
## [1] -0.8472979
```

```r
exp(3/7)
```

```
## [1] 1.535063
```

```r
sin(3/7)
```

```
## [1] 0.4155719
```

```r
#custom functions
myfun<-function(x){
  sqrt(x)^x
}

myfun(5)
```

```
## [1] 55.9017
```

Variables and objects

In R we assign values to objects (object-oriented programming). These can generally have any name, but some names are reserved for R. For instance you probably wouldn't want to call something 'mean' because there's a 'mean()' function already in R. For instance:

```r
x<-3
y<-7
x+y
```

```
## [1] 10
```

```r
x*y
```

```
## [1] 21
```

```r
log(x*y)
```

```
## [1] 3.044522
```

vectors

R thinks everything is a matrix, or a vector, meaning a row or column of numbers, or characters. One of R's big selling points is that much of it is completely vectorized. Meaning, I can apply an operation along all elements of a vector without having to write a loop. For example, if I want to multiply a vector of numbers by a constant, in SAS, I could do:

```
for (i in 1 to 5) x[i]<-y[i]*5 end
```

but in R, I can just do:

```
x<-c(3, 4, 5, 6, 7)
#c() makes a vector
y<-7

x*y
```

```
## [1] 21 28 35 42 49
```

R is also very good about using vectors, let's say I wanted to find the third element of x:

```
x[3]
```

```
## [1] 5
```

```
#or if I want to test if this element is 10
x[3]==10
```

```
## [1] FALSE
```

```
x[3]!=10
```

```
## [1] TRUE
```

```
#of is it larger than another number:
x[3]>3
```

```
## [1] TRUE
```

```
#or is any element of the whole vector greater than 3
x>3
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE
```

If you want to see what's in an object, use `str()`, for `str`ucture

```
str(x)
```

```
##  num [1:5] 3 4 5 6 7
```

and we see that x is numeric, and has those values.

We can also see different characteristics of x

```
#how long is x?
length(x)
```

```
## [1] 5
```

```
#is x numeric?
is.numeric(x)
```

```
## [1] TRUE
```

```r
#is x full of characters?
is.character(x)
```

```
## [1] FALSE
```

```r
#is any element of x missing?
is.na(x)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

```r
xc<-c("1","2")

#now i'll modify x
x<-c(x, NA) #combine x and a missing value ==NA
x
```

```
## [1]  3  4  5  6  7 NA
```

```r
#Now ask if any x's are missing
is.na(x)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

replacing elements of vectors

Above, we had a missing value in X, let's say we want to replace it with another value:

```r
x<-ifelse(test = is.na(x)==T, yes =  sqrt(7.2), no =  x)
x
```

```
## [1] 3.000000 4.000000 5.000000 6.000000 7.000000 2.683282
```

Done!

Dataframes

Traditionally, R organizes variables into data frames, these are like a spreadsheet. The columns can have names, and the dataframe itself can have data of different types. Here we make a short data frame with three columns, two numeric and one character:

```r
mydat<-data.frame(
  x=c(1,2,3,4,5),
  y=c(10, 20, 35, 57, 37),
  group=c("A", "A" ,"A", "B", "B")
)

#See the size of the dataframe
dim(mydat)
```

```
## [1] 5 3
```

```r
length(mydat$x)
```

```
## [1] 5
```

```r
#Open the dataframe in a viewer and just print it
View(mydat)
print(mydat)
```

```
##   x  y group
## 1 1 10     A
## 2 2 20     A
## 3 3 35     A
## 4 4 57     B
## 5 5 37     B
```

Real data

Now let's open a 'real' data file. This is the 2008 World population data sheet from the Population Reference Bureau. It contains summary information on many demographic and population level characteristics of nations around the world in 2008.

I've had this entered into a **Comma Separated Values** file by some poor previous GRA of mine and it lives happily on Github now for all the world to see. CSV files are a good way to store data coming out of a spreadsheet. R can read Excel files, but it digests text files easier. Save something from Excel as CSV.

I can read it from github directly by using a function in the `readr` library:

That's handy. If the file lived on our computer, I could read it in like so: *note, please make a folder on your computer so you can store things for this class in a single location!!!! Organization is Key to Success in Graduate School*

```
prb<-read_csv("C:/Users/ozd504/Documents/GitHub/r_courses/PRB2008_All.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   Country = col_character(),
##   Continent = col_character(),
##   Region = col_character(),
##   Population. = col_double(),
##   Rate.of.natural.increase = col_double(),
##   ProjectedPopMid2025 = col_double(),
##   ProjectedPopMid2050 = col_double(),
##   IMR = col_double(),
##   TFR = col_double(),
##   PercPop1549HIVAIDS2001 = col_double(),
##   PercPop1549HIVAIDS2007 = col_double(),
##   PercPpUnderNourished0204 = col_double(),
##   PopDensPerSqMile = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

Same result.

The `haven` library can read files from other statistical packages easily, so if you have data in Stata, SAS or SPSS, you can read it into R using those functions, for example, the `read_dta()` function reads stata files, `read_sav()` to read spss data files.

I would not recommend you store data in Excel files for many reasons, but if you do, save the files as a CSV file and use the `read_csv()` function above to read it in.

```
library(haven)
prb_stata<-read_dta("C:/Users/ozd504/Documents/GitHub/r_courses/prb2008.dta")

prb_spss<-read_sav("C:/Users/ozd504/Documents/GitHub/r_courses/prb_2008.sav")
```

Don't know what a function's called use ??

`??stata ??csv`

and Rstudio will show you a list of functions that have these strings in them.

What if you know the function name, like `read_csv()` but you want to see all the function arguments?

`?read_csv`

will open up the help file for that specific function

Save a file

Want to save something as a R data file? Use `save()`

```
save(prb, file="C:/Users/ozd504/Documents/GitHub/r_courses/prb_2008_saved.Rdata")
```

If you have an R data file, use `load()` to open it:

```
load("C:/Users/ozd504/Documents/GitHub/r_courses/prb_2008.Rdata")
```

Descriptive analysis

Let's have a look at some descriptive information about the data:

```
#Frequency Table of # of Contries by Continent
table(prb$Continent)
```

```
##
##        Africa          Asia        Europe North America       Oceania
##            56            51            45            27            17
## South America
##            13
```

```
#basic summary statistics for the fertility rate
summary(prb$TFR)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   1.000   1.775   2.500   3.032   4.000   7.100       1
```

From this summary, we see that the mean is 3.023, there is one country missing the Total fertility rate variable. The minimum is 1 and the maximum is 7.1 children per woman.

Now, we will cover some basic descriptive statistical analysis. We will describe measures of central tendency and variability and how these are affected by outliers in our data.

Measures of central tendency

We can use graphical methods to describe what data 'look like' in a visual sense, but graphical methods are rarely useful for comparative purposes. In order to make comparisons, you need to rely on a numerical summary of data vs. a graphical one (I'm not saying statistical graphics aren't useful, they are!)

Numerical measures tell us a lot about the form of a distribution without resorting to graphical methods. The first kind of summary statistics we will see are those related to the measure of *central tendency*. Measures of central tendency tell us about the central part of the distribution

Mean and median

Here is an example from the PRB data. R has a few different ways to get a variable from a dataset. One way is the `$` notation, used like `dataset$variable`

```
mean(prb$TFR)
```

```
## [1] NA
```

Whoops! What happened? This means that R can't calculate the mean because there's a missing value, which we saw before. We can tell R to automatically remove missing values by:

```
mean(prb$TFR, na.rm = T)
```

```
## [1] 3.032212
```

Which is correct.

Measures of variation

One typical set of descriptive statistics that is very frequently used is the so-called **five number summary** and it consists of : the Minimum, lower quartile, median, upper quartile and maximum values. This is often useful if the data are not symmetric or skewed. This is what you get when you use the `fivenum()` function, or we can include the mean if we use the `summary()` function.

```
?fivenum
```

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

```
fivenum(prb$TFR)
```

```
## [1] 1.00 1.75 2.50 4.00 7.10
```

```
summary(prb$TFR)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   1.000   1.775   2.500   3.032   4.000   7.100       1
```

Variance

To calculate the variance and standard deviation of a variable:

```
var(x)
```

```
## [1] 2.894531
```

```
sd(x)
```

```
## [1] 1.701332
```

```
sqrt(var(x))#same as using sd()
```

```
## [1] 1.701332
```

Really Real data example

Now let's open a 'really real' data file. This is a sample from the 2015 1-year American Community Survey microdata, meaning that each row in these data is a person who responded to the survey in 2015. I get these,

and you should too from the Minnesota Population Center IPUMS data. The IPUMS stands for "Integrated Public Use Microdata Series", and consists of individual person responses to decennial census returns going back to 1850, and the American Community Survey data from 2001 to the present.

I'm using data from the US, but there is an IPUMS International data series too, which has data from 85 countries and over 300 censuses.

I've done an extract (do example in class) and stored the data in a R data (.Rdata) format on my github data site. The file we are using is called census.Rdata. This extract is small by demographic data standards and is only about 300,000 people.

There is also a codebook that describes the data and all the response levels for each variable in the data. They are also on my github data page, and called Codebook_census_data.

I can read it from my github repository directly by using the `load()` function combined with the `url()` function, to tell R that the file is on the web. If the file were, say, in my documents folder, I could likewise load the data from disk.

```
load(file=url("https://github.com/coreysparks/r_courses/blob/master/census_data.Rdata?raw=true"))

#from disk
#load("C:/Users/ozd504/Documents/census_data.Rdata")

#print the column names
names(census)
```

```
##  [1] "year"      "datanum"   "serial"    "hhwt"      "statefip"
##  [6] "met2013"   "puma"      "gq"        "pernum"    "perwt"
## [11] "famsize"   "nchild"    "nchlt5"    "eldch"     "nsibs"
## [16] "relate"    "related"   "sex"       "age"       "marst"
## [21] "birthyr"   "fertyr"    "race"      "raced"     "hispan"
## [26] "hispand"   "bpl"       "bpld"      "citizen"   "yrsusa1"
## [31] "language"  "languaged" "speakeng"  "educ"      "educd"
## [36] "empstat"   "empstatd"  "labforce"  "occ"       "ind"
## [41] "inctot"    "incwage"   "poverty"   "hwsei"     "migrate1"
## [46] "migrate1d" "carpool"   "trantime"
```

Pipes

The `dplyr` library is a portion of a suite of libraries known as the *tidyverse*, which are oriented towards reproducible, intelligible coding for data science. There are too many things within the tidyverse to cover them all here, but I will introduce you to two aspects: 1) dplyr verbs and pipes 2) the ggplot2 library for producing graphs

Basic tidyverse verbs

The dplyr library has many verbs (action words) that are used to do various things. The neat thing about dplyr is it allows you to tell R what data source you want to do something to at the top of a *pipe*, then you can execute as many verbs as you need within the pipe without referring to the dataset by name again.

For instance, in the census data, let's say we want to calculate the median income for adult, men and women who are in the labor force, in Texas. Sounds easy, right? In base R we would have to do some subsetting of the data first (to limit our analysis to adults, in the labor force, who live in Texas), then use another function to calculate the median income for men and women.

dplyr allows us to do this in one fell swoop of code. We will use a few verbs, notably `filter()` to subset our cases, based on the conditions we describe, `mutate()` to recode our income variable, `group_by()` to let R know that the summaries we want will be for specific groups and `summarise()` to calculate the numeric summaries we want.

Now, most variables in the IPUMS can't be used out of the box. For example open the pdf codebook and find the variable "incwage", which is person's income from wages in the previous year.

We are specifically wanting to pay attention to the "Coder Instructions" *you're the coder*. Notice two codes (values) that are of special note. *Specific Variable Codes 999999 = N/A and 999998=Missing*. So if we want to use this variable, we need to do a basic recode of these values to tell R that they represent missing values.

Pipes

The second thing we need to know about are *pipes*. Pipes can be used to chain together verbs so that the code executes on the same dataset in sequence. They are identified by a `%>%` at the end of each verb statement. Here's our example in action:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
census%>%
  filter(age>18, statefip==48, labforce==2)%>%
  mutate(newwage= ifelse(incwage%in%c(999998,999999), NA, incwage),
         newsex=ifelse(sex==1, "male", "female" ))%>%
  group_by(newsex)%>%
  summarise(med_income= median(newwage, na.rm=T))
```

```
## # A tibble: 2 x 2
##   newsex med_income
##    <chr>      <dbl>
## 1 female      26100
## 2   male      38000
```

and we see a difference of about `$12,000` between men and women in Texas.

Notice in the code above, I did two three different filters in a single `filter()` statement and two recodes in a single `mutate()` statement, this is totally legal, and in general you can do several operations within a single verb statement. Otherwise I would have to do:

```
library(dplyr)

census%>%
  filter(age>18)%>%
  filter(statefip==48)%>%
  filter(labforce==2)%>%
  mutate(newwage= ifelse(incwage%in%c(999998,999999), NA, incwage))%>%
  mutate(newsex=ifelse(sex==1, "male", "female" ))%>%
```

```
  group_by(newsex)%>%
  summarise(med_income= median(newwage, na.rm=T))
```

```
## # A tibble: 2 x 2
##   newsex med_income
##    <chr>      <dbl>
## 1 female      26100
## 2   male      38000
```

So we get to the same place. It's up to you which way you do it, always go with the route that you understand better and that is more readable and explicable to someone else.

I always say that in R, there's **always** more than one way to do anything!

We could also see how incomes are different in San Antonio (variable met2013==41700) compared to Dallas (variable met2013==19100).

```
census%>%
  filter(labforce==2, met2013%in%c(41700, 19100), age>18) %>%
  mutate(newwage= ifelse(incwage%in%c(999998,999999), NA, incwage),
         sexrecode=ifelse(sex==1, "male", "female"),
         city=ifelse(met2013==41700, "San Antonio", "Dallas")) %>%
  group_by(sexrecode, city)%>%
  summarise(med_income=median(newwage, na.rm=T), n=n())
```

```
## # A tibble: 4 x 4
## # Groups:   sexrecode [?]
##   sexrecode        city med_income     n
##       <chr>       <chr>      <dbl> <int>
## 1    female      Dallas      32000  1437
## 2    female San Antonio      25000   424
## 3      male      Dallas      40000  1744
## 4      male San Antonio      36000   466
```

So, we see that men in Dallas make about $4000 more than men in San Antonio, and women in Dallas make about $7000 more than women in San Antonio

Basic ggplot()

Let's say that we want to compare the distributions of income from the above examples graphically. Since the ggplot2 library is part of the tidyverse, it integrates directly with dplyr and we can do plots within pipes too.

In generally, ggplot() has a few core statements.

1) ggplot() statement - This tells R the data and the basic aesthetic that will be plotted, think x and y axis of a graph
2) Define the geometries you want to use to plot your data, there are many types of plots you can do, some are more appropriate for certain types of data
3) Plot annotations - Titles, labels etc.

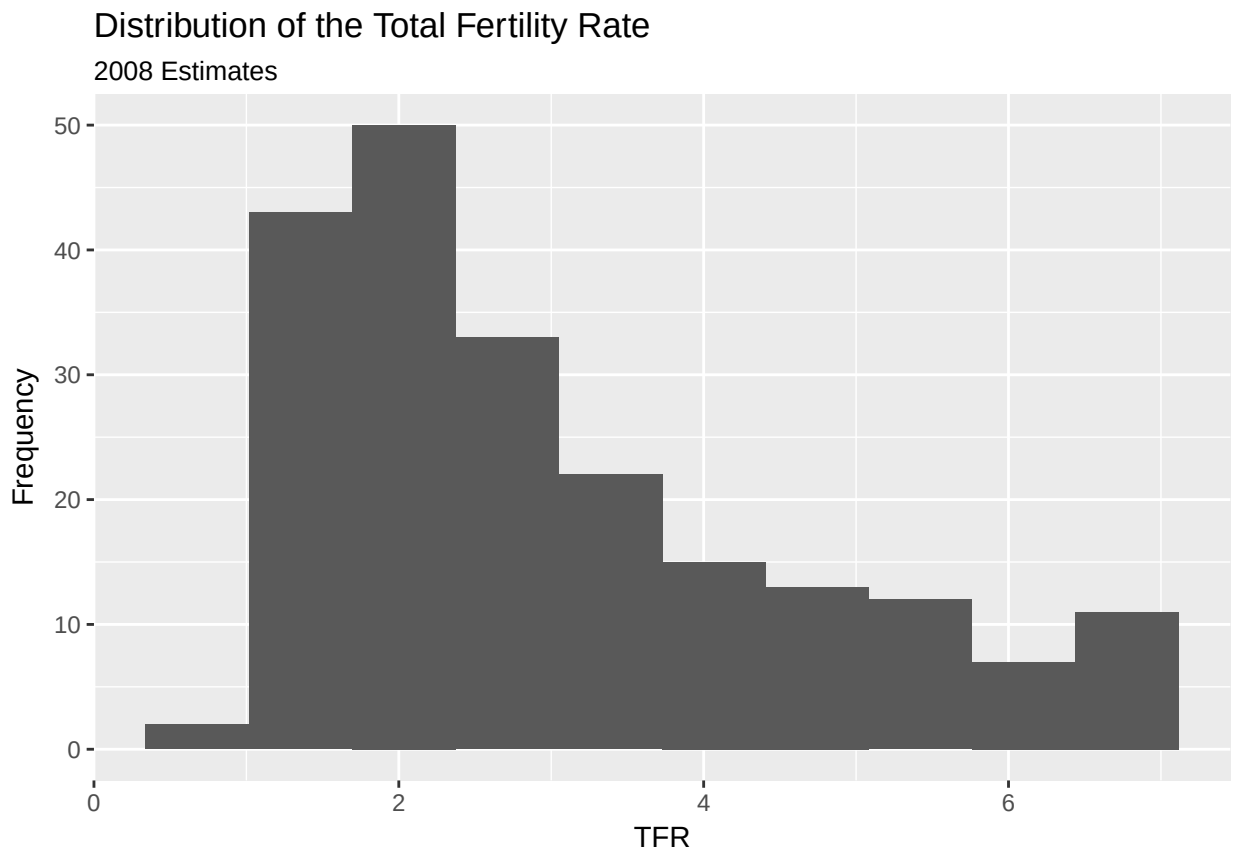
Now I will illustrate some basic ggplot examples, and I'm going to use the PRB data for now because it's much prettier than the ACS data for plotting.

```
library(ggplot2)

ggplot(data=prb, mapping=aes(TFR))+
  geom_histogram( bins=10)+
  ggtitle(label = "Distribution of the Total Fertility Rate ", subtitle = "2008 Estimates")+
```

```
  xlab(label = "TFR")+
  ylab(label="Frequency")
```

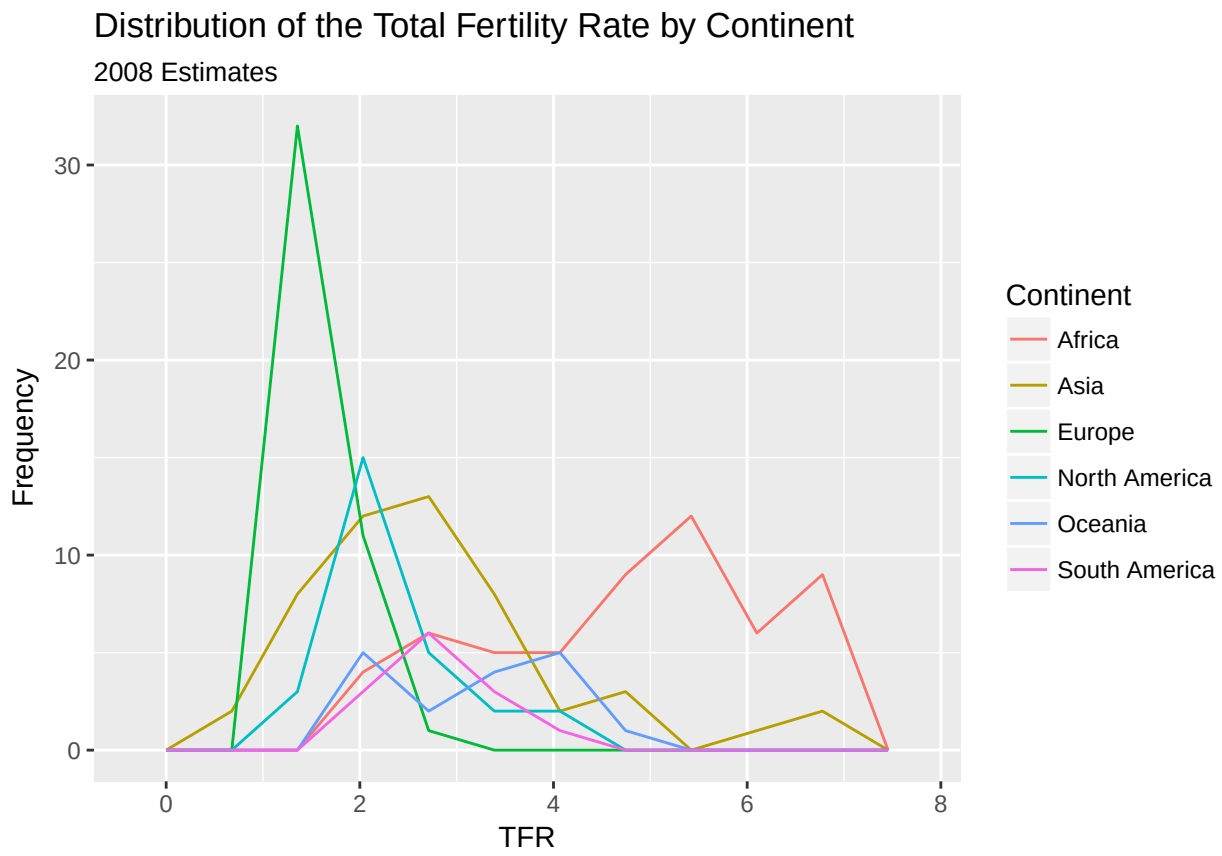
```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```



There is also a nice geometry called `freqpoly` that will draw polygons instead of bars for a histogram. I will use this to produce histograms for each continent.

```
ggplot(data=prb,mapping = aes(TFR, colour=Continent))+
  geom_freqpoly( bins=10)+
  ggtitle(label = "Distribution of the Total Fertility Rate by Continent", subtitle = "2008 Estimates")+
  xlab(label = "TFR")+
  ylab(label="Frequency")
```

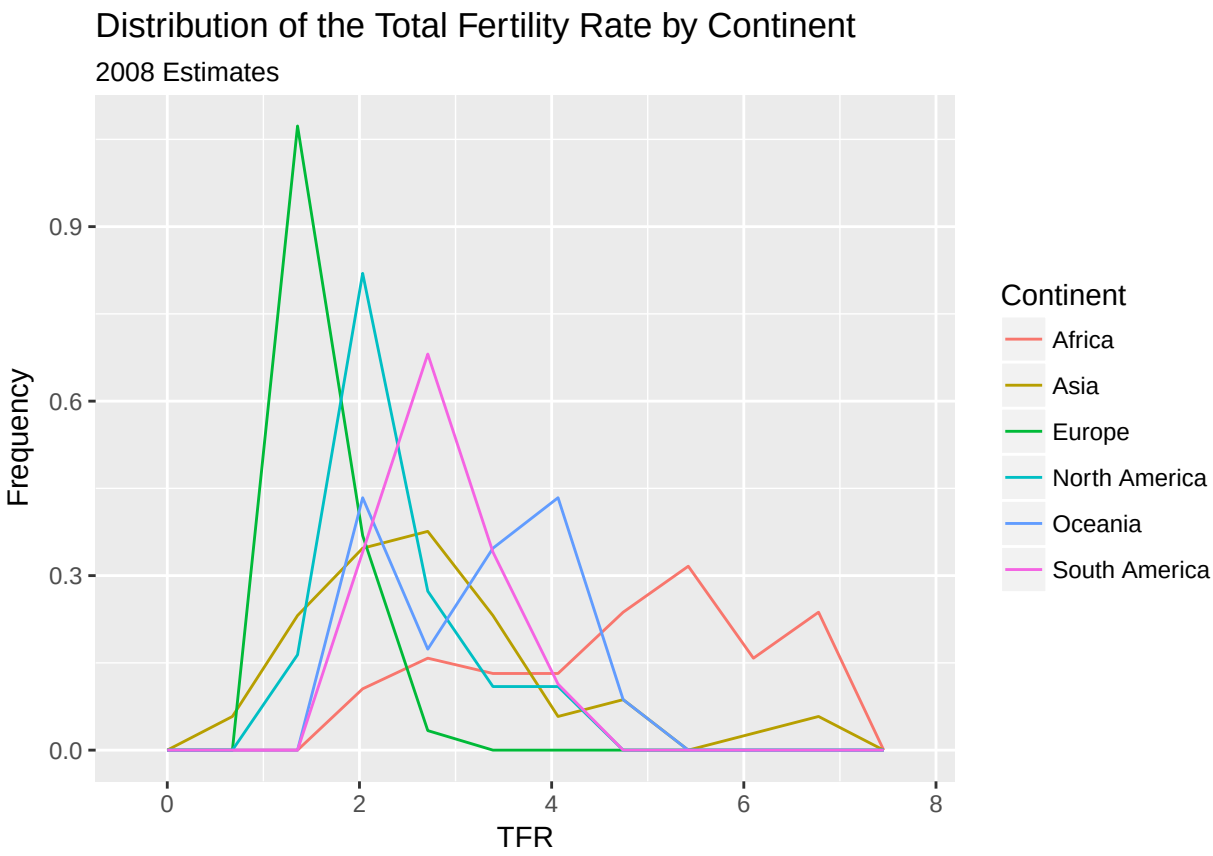
```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```



Also, we can plot the relative frequency , or density, instead of the count by including the `..density..` argument in the aesthetic `aes()`.

```
ggplot(data=prb,mapping = aes(TFR, colour=Continent, ..density..))+
  geom_freqpoly( bins=10)+
  ggtitle(label = "Distribution of the Total Fertility Rate by Continent", subtitle = "2008 Estimates")+
  xlab(label = "TFR")+
  ylab(label="Frequency")
```

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```

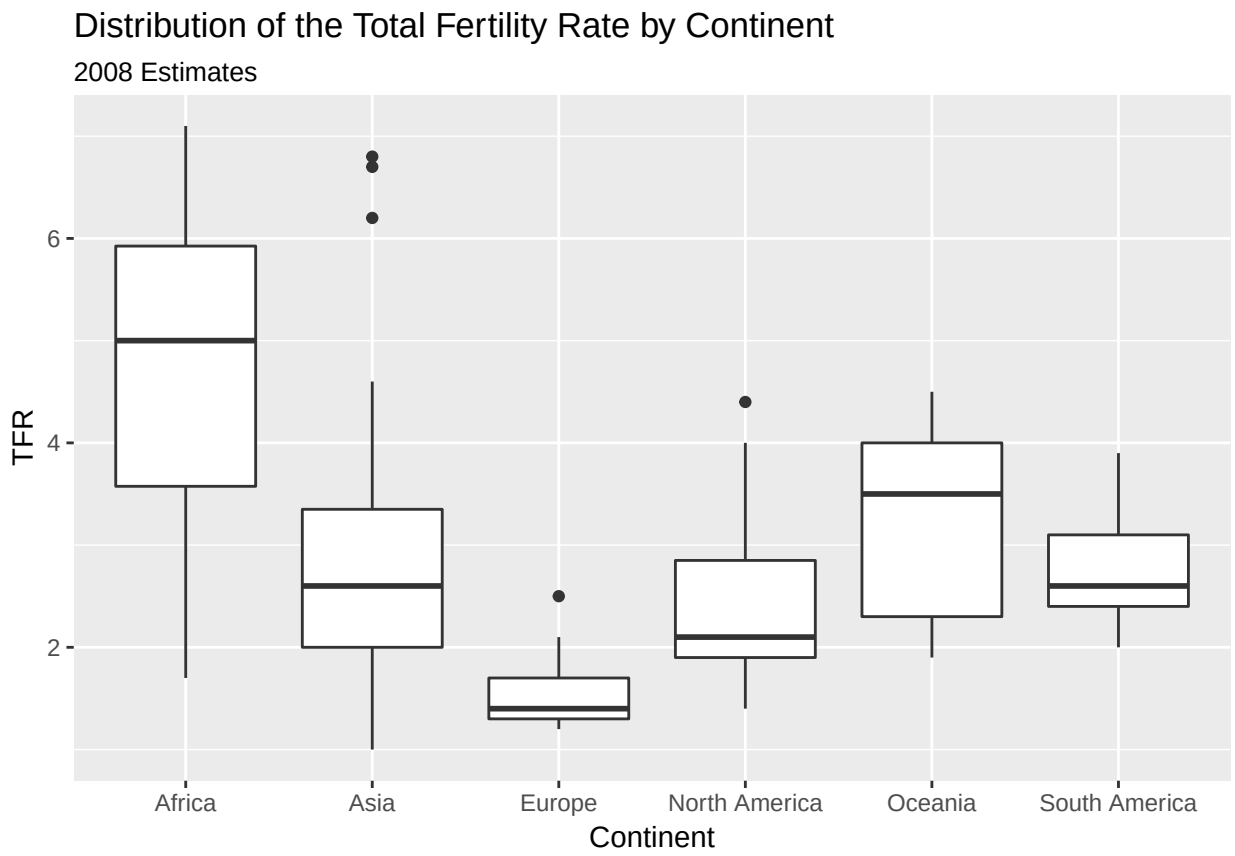


Stem and leaf plots/Box and Whisker plots

Another visualization method is the stem and leaf plot, or box and whisker plot. This is useful when you have a continuous variable you want to display the distribution of across levels of a categorical variable. This is basically a graphical display of Tukey's 5 number summary of data.

```
ggplot(prb, mapping = aes(x= Continent, y =TFR))+
  geom_boxplot()+
  ggtitle(label = "Distribution of the Total Fertility Rate by Continent", subtitle = "2008 Estimates")
```

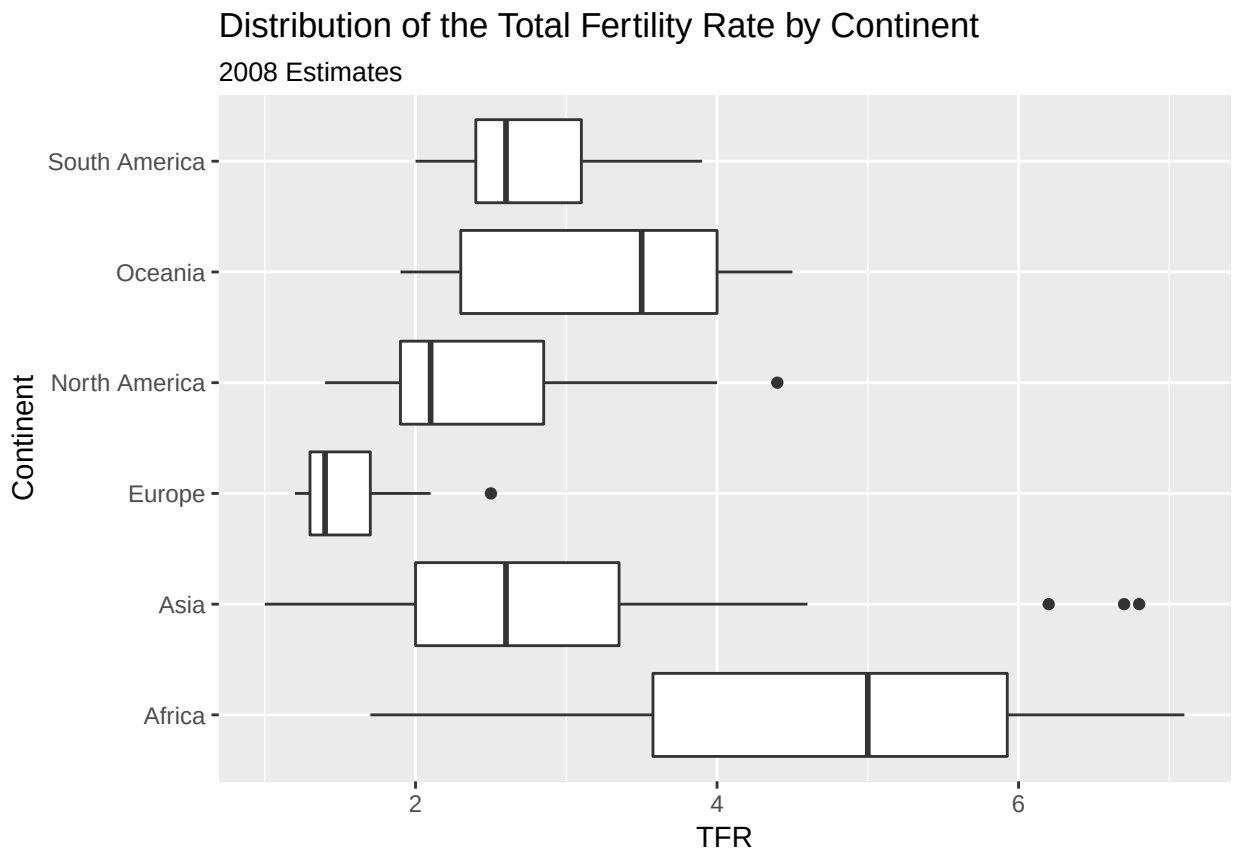
```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```



You can flip the axes, by adding `coord_flip()`

```
ggplot(prb, mapping = aes(x= Continent, y =TFR))+
  geom_boxplot()+
  ggtitle(label = "Distribution of the Total Fertility Rate by Continent", subtitle = "2008 Estimates")+
```

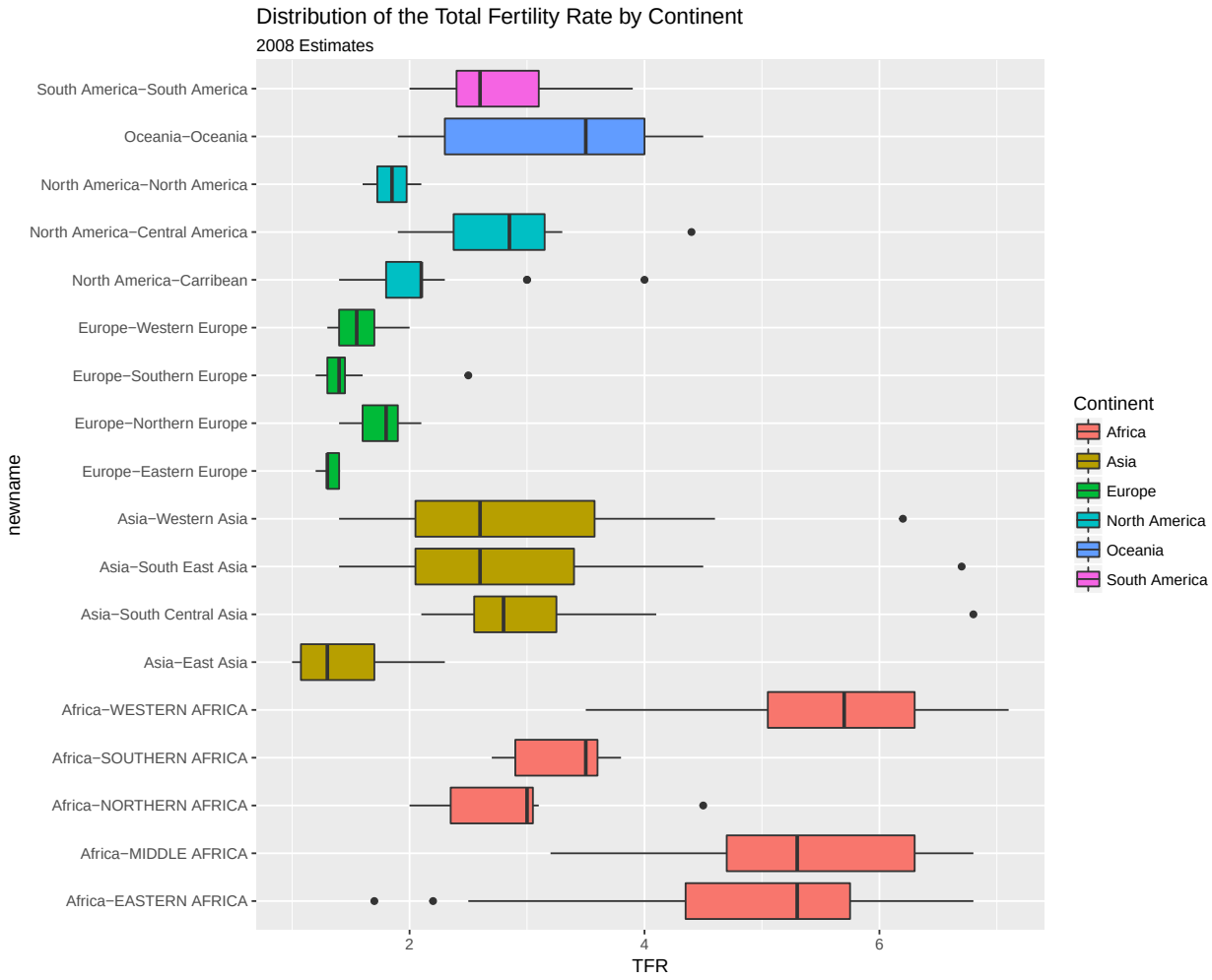
```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```



You can also color the boxes by a variable, Here, I will make a new variable that is the combination of the continent variable with the region variable, using the `paste()` function. It's useful for combining values of two strings.

```
prb%>%
  mutate(newname = paste(Continent, Region, sep="-"))%>%
  ggplot(aes(x= newname, y =TFR,fill=Continent))+
  geom_boxplot()+coord_flip()+
  ggtitle(label = "Distribution of the Total Fertility Rate by Continent", subtitle = "2008 Estimates")
```

```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```



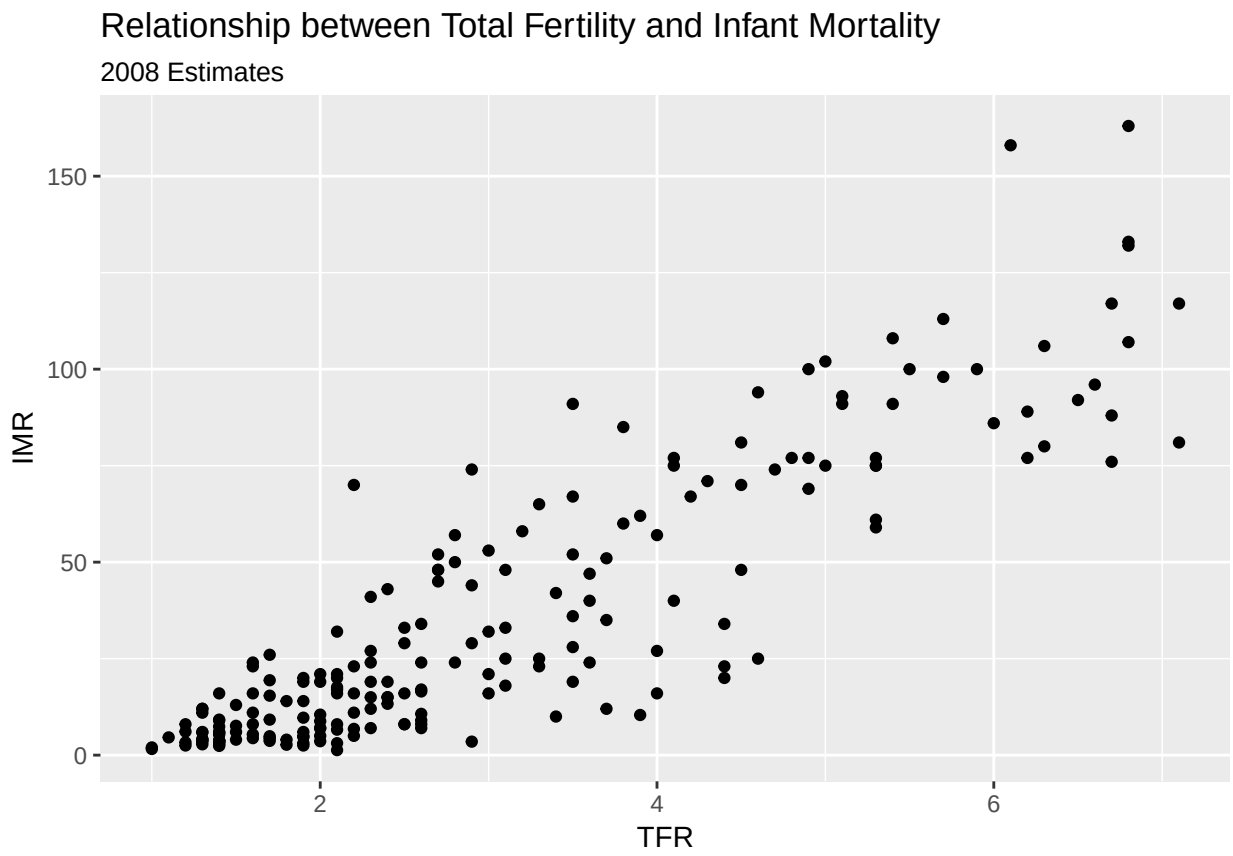
X-Y Scatter plots

These are useful for finding relationships among two or more continuous variables. `ggplot()` can really make these pretty.

Here are a few riffs using the PRB data:

```
ggplot(data=prb,mapping= aes(x=TFR, y=IMR))+
  geom_point()+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality", subtitle = "2008 Estimat
  xlab(label = "TFR")+
  ylab(label="IMR")
```

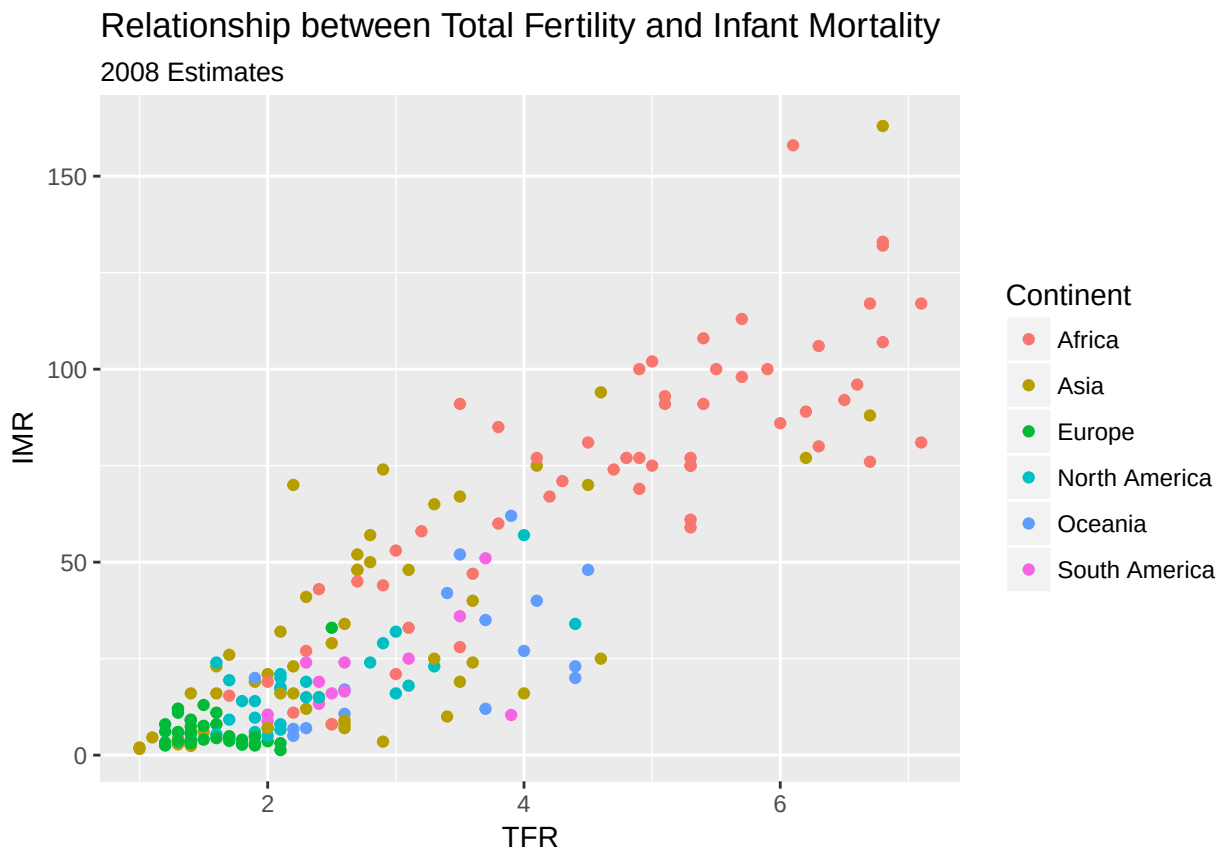
```
## Warning: Removed 2 rows containing missing values (geom_point).
```



Now we color varies by continent

```
ggplot(data=prb,mapping= aes(x=TFR, y=IMR, color=Continent))+
  geom_point()+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality", subtitle = "2008 Estimat
  xlab(label = "TFR")+
  ylab(label="IMR")
```

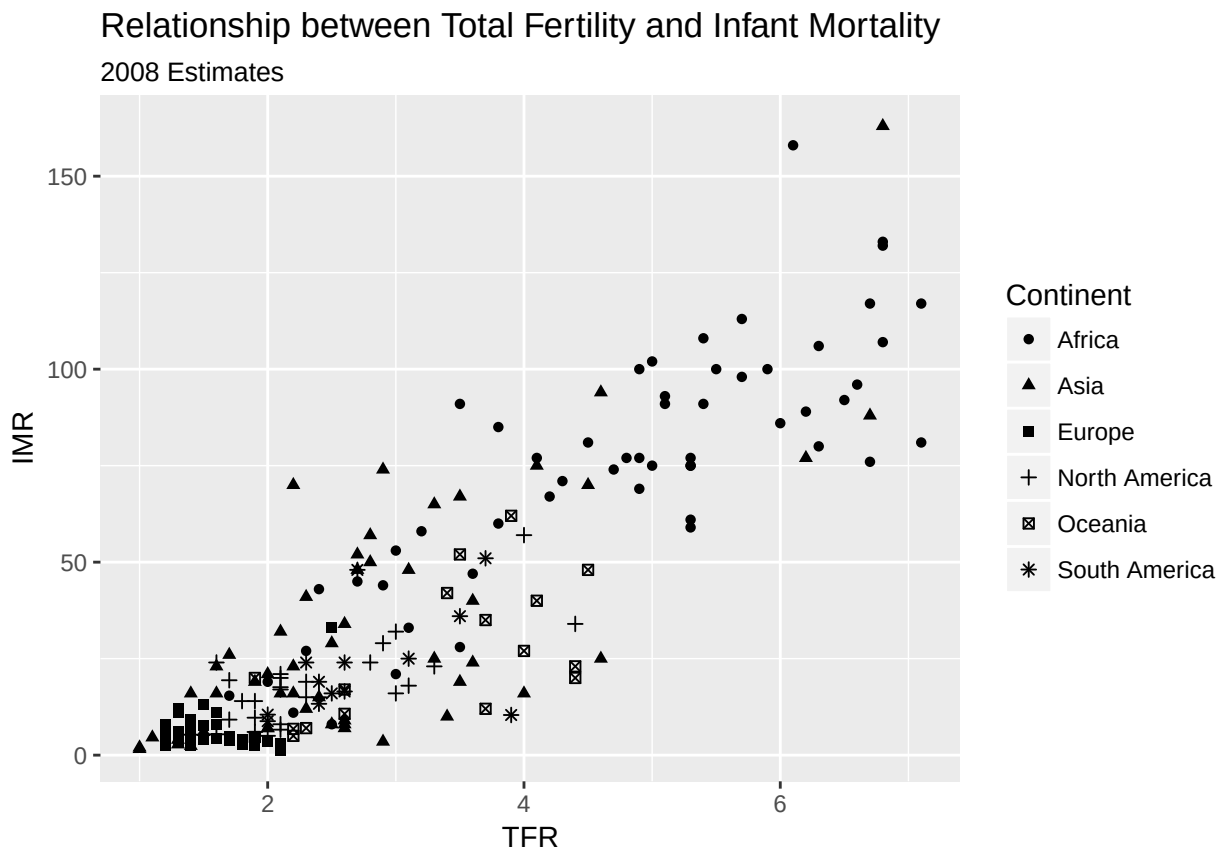
```
## Warning: Removed 2 rows containing missing values (geom_point).
```



Now we vary the shape of the point by continent

```
#shape varies by continent
ggplot(data=prb,mapping= aes(x=TFR, y=IMR, shape=Continent))+
  geom_point()+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality", subtitle = "2008 Estimat
  xlab(label = "TFR")+
  ylab(label="IMR")
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```

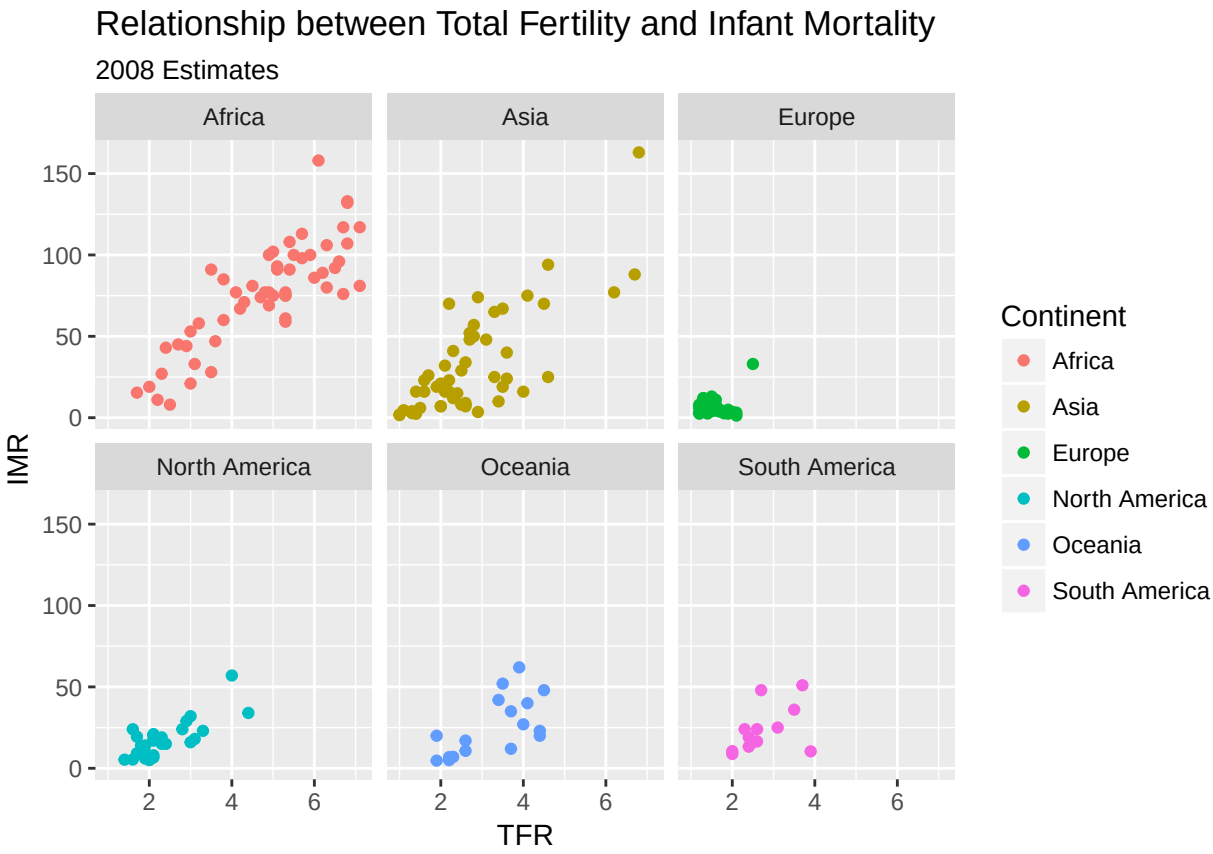


Facet plots

Facet plots are nice, if you want to create a plot separately for a series of groups. This allows you to visualize if the relationship is constant across those groups, well at least graphically.

```
ggplot(data=prb,mapping= aes(x=TFR, y=IMR, color=Continent))+
  geom_point()+
  facet_wrap(~Continent)+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality", subtitle = "2008 Estimat
  xlab(label = "TFR")+
  ylab(label="IMR")
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



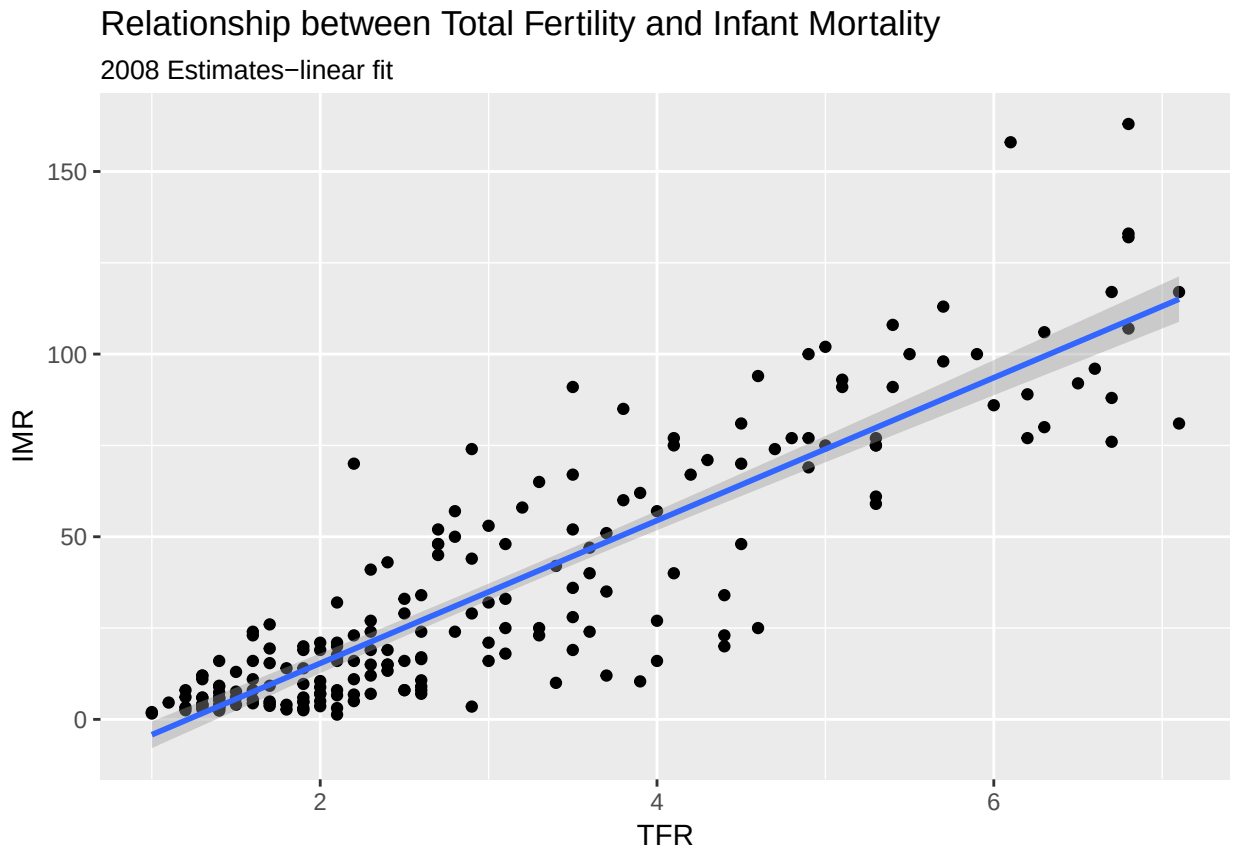
Plotting relationships with some line fits

`ggplot` allows you to make some very nice line-fit plots for scatter plots. While the math behind these lines is not what we are talking about, they do produce a nice graphical summary of the relationships.

```
ggplot(data=prb,mapping= aes(x=TFR, y=IMR))+
  geom_point()+
  geom_smooth( method = "lm")+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality", subtitle = "2008 Estimat
  xlab(label = "TFR")+
  ylab(label="IMR")
```

```
## Warning: Removed 2 rows containing non-finite values (stat_smooth).
```

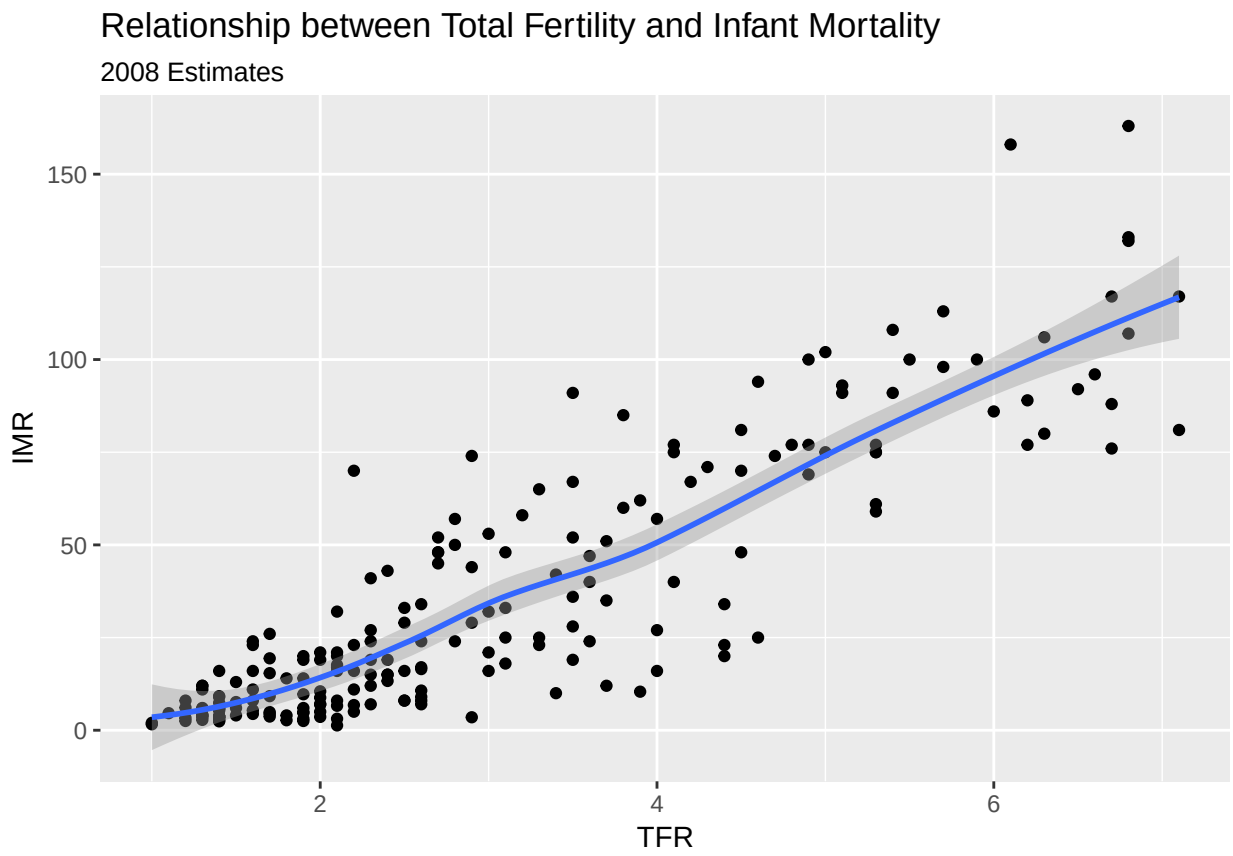
```
## Warning: Removed 2 rows containing missing values (geom_point).
```




```
ggplot(data=prb)+
  geom_point(mapping= aes(x=TFR, y=IMR))+
  geom_smooth(mapping= aes(x=TFR, y=IMR) , method = "loess")+
  ggtitle(label = "Relationship between Total Fertility and Infant Mortality", subtitle = "2008 Estimat
  xlab(label = "TFR")+
  ylab(label="IMR")
```

```
## Warning: Removed 2 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```

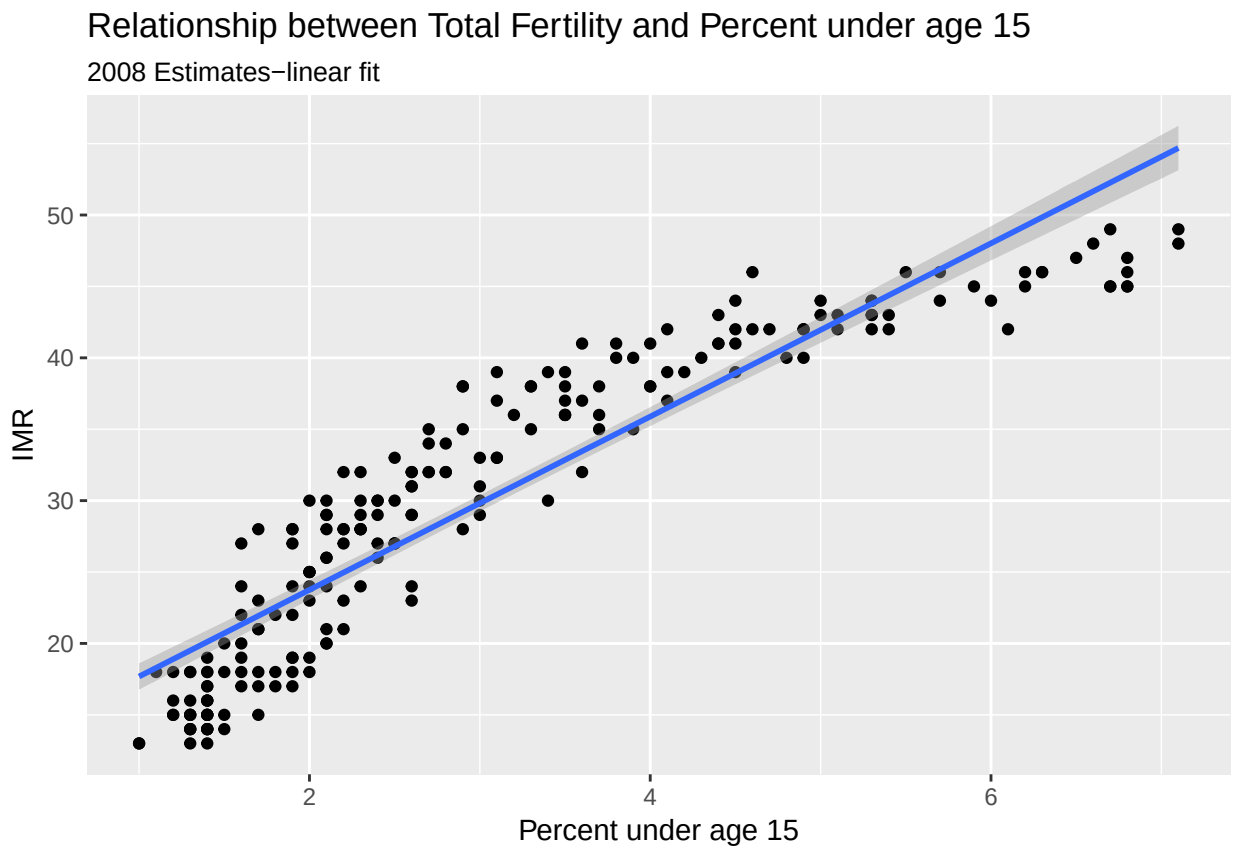


Another example, this time of a bad linear plot!

```
ggplot(data=prb,mapping= aes(x=TFR, y=PercPopLT15))+
  geom_point()+
  geom_smooth( method = "lm")+
  ggtitle(label = "Relationship between Total Fertility and Percent under age 15", subtitle = "2008 Esti
  xlab(label = "Percent under age 15")+
  ylab(label="IMR")
```

```
## Warning: Removed 1 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

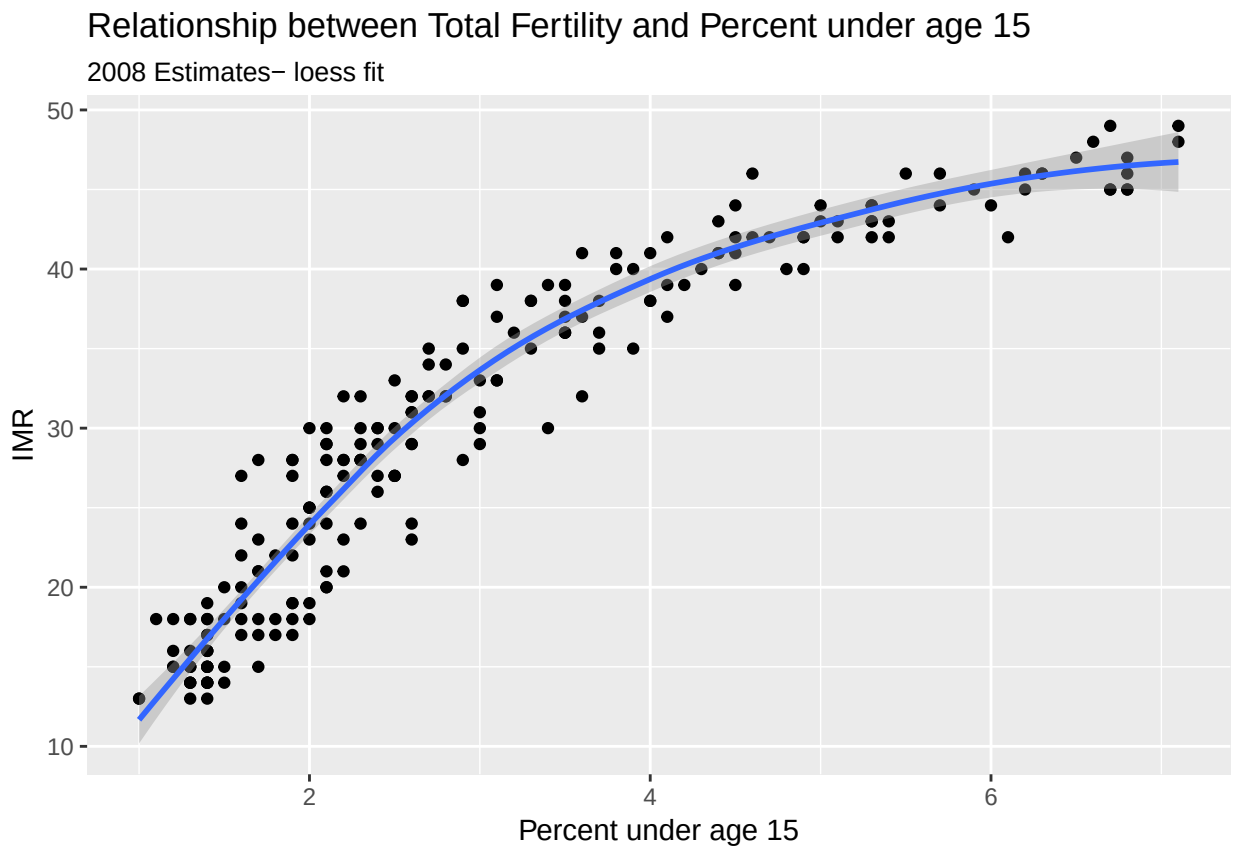


So instead, us a nonlinear fit, a la a loess regression:

```
ggplot(data=prb, mapping= aes(x=TFR, y=PercPopLT15))+
  geom_point()+
  geom_smooth( method = "loess")+
  ggtitle(label = "Relationship between Total Fertility and Percent under age 15", subtitle = "2008 Esti
  xlab(label = "Percent under age 15")+
  ylab(label="IMR")
```

```
## Warning: Removed 1 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Other Resources

Miktek

To build pdf's you'll need a version of Latex installed, Miktek is a good option

R-markdown cheat sheets

Rstudio keeps a variety of cheat sheets for various topics, they can be helpful in a pinch

UCLA Statistical computing help

This page has lots of examples of using R for various types of analysis.

Other examples

On my Rpubs page I have lots of examples of various types of analysis using R and you can get the data for these on my Github data page

R on Shamu

The UTSA High performance computing group maintains a computational cluster named Shamu, and R is installed on it. You can contact them for assistance. We are also working on starting a R user group for the UTSA community.