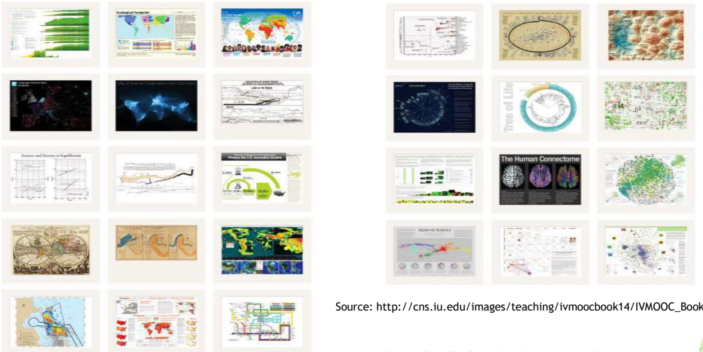
Data Analysis and Visualization with R



Source: http://cns.iu.edu/images/teaching/ivmoocbook14/IVMOOC_Book_Preview.html

ESCOLA POLITÉCNICA DA USP

Bio Comp

André Batista, Ph.D. Student

andrefmb@usp.br

2016

AGENDA

- R Programming Language
  - Fundamentals
    - Variables & Data Structures
  - Data Visualization with ggplot2
  - Data Analysis
    - Statistical Testing and Prediction
    - Exploratory Analysis

This content is inspired on David Robinson "Data Analysis and Visualization Using R" website, available at http://varianceexplained.org/RData/code/code_lesson1/

Others references are cited in the proper slides

Part I

R Fundamentals

R - FUNDAMENTALS

- R is a de facto standard language for data analysis
- Firstly, we need to set up our working environment
- Working directory
  - Default location on the computer that R is pointing at
  - If you want to save or load a file, you need to know what the current directory is

```
> getwd()
[1] "C:/Users/sn1029722/Documents"
>
> setwd("C:/temp/Rtutorial")
>
> getwd()
[1] "C:/temp/Rtutorial"
>
```

We use the functions getwd() and setwd()

R - VARIABLES

- Variables
  - Most basic and crucial element of R
  - Single numbers, vectors, matrix, data frame are the most used variables
- Examples

```
> my.number = 42
> my.number
[1] 42
> print(my.number)
[1] 42
>
```

```
> 6+4
[1] 10
> x = 6 + 4
> x
[1] 10
> y = 4
> x / y
[1] 2.5
> x^2
[1] 100
> log(x)
[1] 2.302585
>
```

Primitively, R can be used as a scientific calculator

R - VECTORS

- We can create a vector consisting of multiple numeric values by using a function **c( )**

```
> v1 = c(1, 5.5, 1e2)
> v1
[1]   1.0   5.5 100.0
> v2 = c(0.14, 0, -2)
> v2
[1]  0.14  0.00 -2.00
> v3 = c(v1,v2)
> v3
[1]   1.00   5.50 100.00   0.14   0.00  -2.00
>
```

- Subset the vector

```
> v3[2]
[1] 5.5
> v3[c(2,3)]
[1]   5.5 100.0
> v3_sub  = v3[c(2,3)]
> v3_sub
[1]   5.5 100.0
> |
```

and using APPEND( ) function

```
> x <- c(1,3,4,5)
> x
[1] 1 3 4 5
> x<- append(x, c(6,7))
> x
[1] 1 3 4 5 6 7
> x<- append(x, c(2), after=1)
> x
[1] 1 2 3 4 5 6 7
```

* after = <<position>>

R - VECTORS

- A lot of statistical programming in R relies on mathematical operations applied to a vector a matrix
- Basic calculator-like functions may apply to all elements in a given vector

```
> v1 = c(1, 5.5, 1e2)
> v1
[1]   1.0   5.5 100.0
> v1 + 2
[1]   3.0   7.5 102.0
> |
```

- Operations between two vectors

```
> v1 = c(1, 2, 3)
> v2 = c(4, 5, 6)
> v1 + v2
[1] 5 7 9
```

```
> v1 * v2
[1]  4 10 18
```

```
> v1 %*% v2
     [,1]
[1,]   32
```

Inner product
Vectors must have the same length

R - VECTORS

- We can use the function CLASS( ) to check the class of an element

```
> v3
[1] "42" "12"
> class(v3)
[1] "character"
> class(v3[2])
[1] "character"
```

- We can populate a vector using SEQ( ) function

```
> v4 <- seq(1,10)
> v4
 [1]  1  2  3  4  5  6  7  8  9 10
> v5 <- seq(1, 10, by=2)
> v5
[1] 1 3 5 7 9
> v6 <- 1:10
> v6
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
> v7 <- seq(1, 2, len=20)
> v7
 [1] 1.000000 1.052632 1.105263 1.157895 1.210526
 [6] 1.263158 1.315789 1.368421 1.421053 1.473684
[11] 1.526316 1.578947 1.631579 1.684211 1.736842
[16] 1.789474 1.842105 1.894737 1.947368 2.000000
```

```
> x <- rnorm(10, mean=8, sd=2)
> x
 [1]  6.242771  9.036264 10.057671  8.258305  7.889897
 [6]  7.727140 10.152282  7.601948  9.113974  6.273276
>
```

random generation for the normal distribution

R - VECTORS

- We can use relational and logical operator for selecting elements in a vector

```
> v4[ v4 > 5]
[1]  6  7  8  9 10
> v4 > 6
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
 [9]  TRUE  TRUE
```

- REP( ) function

```
> x <- rep(1,5)
> x
[1] 1 1 1 1 1
> x <- rep(c(1,2), c(3,5))
> x
[1] 1 1 1 2 2 2 2 2
```

rep {base} R Documentation

Replicate Elements of Vectors and Lists

Description

`rep` replicates the values in `x`. It is a generic function, and the (internal) default method is described here.

`rep.int` and `rep_len` are faster simplified versions for two common cases. They are not generic.

Usage

```
rep(x, ...)

rep.int(x, times)

rep_len(x, length.out)
```

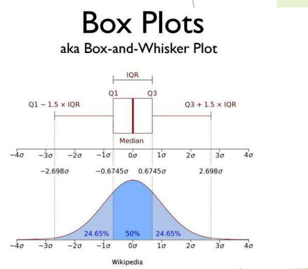
R - VECTORS

- Summary Statistics of Vectors

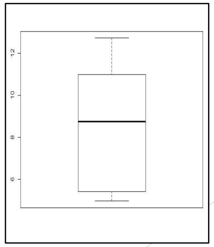
```
> x
 [1] 12.719618 10.986686  5.161110  6.783574 12.282892
 [6]  8.487787  4.963919  5.412336  9.026972 10.868430
> #maximum
> max(x)
[1] 12.71962
> #minimum
> min(x)
[1] 4.963919
> #sum
> sum(x)
[1] 86.69333
> #mean
> mean(x)
[1] 8.669333
> #median
> median(x)
[1] 8.75738
> #variance
> var(x)
[1] 8.892385
> #standard deviation
> sd(x)
[1] 2.98201
```

Box Plots

aka Box-and-Whisker Plot




```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.964   5.755   8.757   8.669  10.960  12.720
> boxplot(x)
```



Generated boxplot for x

R - VECTORS

- Names
  - Elements in a vector have names!
  - And we can access them using the function NAMES( )

```
> weight_kg <- c(40.5, 68.7, 90)
> names(weight_kg)
NULL
```

  - NULL implies that the elements in the vector currently do not have names. We assign names using NAMES( ) and '=' operator

```
> names(weight_kg) = c("Susy", "Maria", "Kevin")
```

  - Now we have

```
> names(weight_kg)
[1] "Susy"  "Maria" "Kevin"
> weight_kg
 Susy Maria Kevin
 40.5  68.7  90.0
```

R - MATRICES

- Matrices are like two-dimensional vectors, organizing values into rows and columns
- The easiest way to create a matrix is using MATRIX( )

```
> ma = matrix(1:6, nrow = 3, ncol = 2)
> ma
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> mb = matrix(7:9, nrow=3, ncol=1)
> mb
     [,1]
[1,]    7
[2,]    8
[3,]    9
```

- A matrix cannot contain multiple data types
- Here, both MA and MB contain only numeric values

R - MATRICES

- Combining
  - Sometimes we want to combine different matrices and vectors
  - We can use CBIND( ) and RBIND( ) functions
    - As long as their lengths and dimensions are comparable. *Example of error:*

```
> #binding rows of MA with a new vector
> ma <- rbind(ma, c(1,2,3))
Warning message:
In rbind(ma, c(1, 2, 3)) :
  number of columns of result is not a multiple of vector length (arg 2)
>
```

- Combining MA and MB into a new matrix M

```
> ma
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

\+

```
> mb
     [,1]
[1,]    7
[2,]    8
[3,]    9
```

=

```
> m <- cbind(ma,mb)
> m
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

R - MATRICES

- Extracting values from matrices is straightforward

```
> m
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> m[1,3]
[1] 7
> m[2, 1:3]
[1] 2 5 8
> m[,3]
[1] 7 8 9
```

- Obtaining info about a matrix

```
> nrow(m)
[1] 3
> ncol(m)
[1] 3
> dim(m)
[1] 3 3
```

- Setting ROWNAME and COLNAME

```
> rownames(m) = c("1st", "2nd", "3rd")
> m
    [,1] [,2] [,3]
1st    1    4    7
2nd    2    5    8
3rd    3    6    9
> colnames(m) = c("Men", "Women", "Children")
> m
    Men Women Children
1st   1     4        7
2nd   2     5        8
3rd   3     6        9
>
```

R - ARRAYS

- An array in R can have one, two or more dimensions
- It is simply a vector which is stored with additional atributes giving the dimensions and optionally names for those dimensions

```
> ar1 <- array(1:24, dim=c(3,4,2))
> ar1
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

dim=c(3,4,2) means TWO dimensions having a matrix with FOUR columns and THREE rows each

Now, try this:

```
ar1 <- array(1:24, dim=c(3,4,2)) ar1[,2:3,]
ar1[2,,1]
sum(ar1[,,1])
sum(ar1[1:2,,1])
```

R – LISTS and DATA FRAMES

- Lists and Data frames
  - Matrices are extremely useful for processing and storing large datasets
    - But have several limitations that may not suit our needs (one datatype only, for example)
- List
  - It is a vector containing other objects which may be of different data types or different lengths

```
> v1 <- c(1:10)
> char<-'oi'
> ma = matrix(1:9, nrow=3, ncol=3)
> lista <- list(v1,char, ma)
> lista
[[1]]
 [1]  1  2  3  4  5  6  7  8  9
[10] 10

[[2]]
[1] "oi"

[[3]]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
> lista[1]
[[1]]
 [1]  1  2  3  4  5  6  7  8  9
[10] 10

> lista[[1]]
 [1]  1  2  3  4  5  6  7  8  9
[10] 10
> |
```

R – LISTS and DATA FRAMES

- Data Frames
  - Data frames are lists with a set of restrictions
  - It is a list of vectors which are conveniently arranged as columns
  - All vectors or columns in a data frame must have the same length
  - Data frames mimic matrices when needed and appropriate
- MTCARS
  - R comes with built-in datasets. MTCARS contains statistics about 32 cars in 1974

```
> data(mtcars)
> class(mtcars)
[1] "data.frame"
```

  - Use the command **View(mtcars)** to display the data in a spreadsheet

R – LISTS and DATA FRAMES

If you want to see only the first 6 rows, you can use the **head( )** function

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
>
```

One of the first steps when we have a data frame or a dataset is try to understand about its statistics

```
> summary(mtcars)
      mpg             cyl             disp             hp             drat             wt
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760   Min.   :1.51
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080   1st Qu.:2.58
 Median :19.20   Median :6.000   Median :196.3   Median :123.0   Median :3.695   Median :3.32
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597   Mean   :3.21
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920   3rd Qu.:3.61
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930   Max.   :5.42
      qsec             vs               am              gear             carb
 Min.   :14.50   Min.   :0.0000   Min.   :0.0000   Min.   :3.000   Min.   :1.000
 1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
 Median :17.71   Median :0.0000   Median :0.0000   Median :4.000   Median :2.000
 Mean   :17.85   Mean   :0.4375   Mean   :0.4062   Mean   :3.688   Mean   :2.812
 3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :22.90   Max.   :1.0000   Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

DATA FRAMES

- We can retrieve a specific column by name, using **$columnname**
  - For example, let's look just at miles per gallon (mpg)

```
> mtcars$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4 14.7
[18] 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
```

  - Or you can use **mtcars[, "mpg"]** or still **mtcars[, 1]**
  - We can also obtain multiple rows at once as well: **mtcars[1:3,]**
- How to create a new data frame?
  - Using **data.frame** function

```
> n <- c(2,3,5)
> s <- c("aa", "bb", "cc")
> b <- c(TRUE, FALSE, TRUE)
> df = data.frame(n,s,b)
> df
  n  s     b
1 2 aa  TRUE
2 3 bb FALSE
3 5 cc  TRUE
```

MISSING VALUES

- In R missing values are represented by the symbol (NA – not available)
  - Impossible values (e.g., dividing by zero) are represented by NaN
- We have functions to deal with NA values, as follows:

```
> y <- c(1, 2, 3, NA)
> is.na(y)
[1] FALSE FALSE FALSE  TRUE
> mean(y)
[1] NA
> mean(x, na.rm = TRUE)
[1] 1.625
```

GUIDED EXERCISE

- Here we will learn by practicing with an example
- We will learn
  - How to load files into R (e.g., CSV files)
  - How to deal with NA values
  - How to apply functions into a data frame
  - How to plot basic graphics

- Firstly, you need to download the **grades.csv** from https://www.dropbox.com/s/5ry1kfbx6d05kn3/grades.csv?dl=0
- Save the file into R workspace

This exercise is based on http://www.utsc.utoronto.ca/~sdamouras/summer/Rworkshop1.pdf

Exercise – Part I

- Firstly, we need to load Grades.csv into a new data frame

```
> grade = read.csv("grades.csv", header = TRUE, sep = "\t")
```

- Let's discover some info about our file

```
> dim(grade)
[1] 117  16
>
> head(grade)
  Student.ID First.Name Last.Name Tutorial Quiz.1 Quiz.2 Quiz.3 Quiz.4 Quiz.5 Quiz.6 Quiz.7 Quiz.8 Quiz.9 Midterm.1 Midterm.2 Final.Exam
1  998000001        Fae  Grijalva      101    8.0    6.5    7.5    7.5   10.0    9.5    9.5    9.0     NA        42      31.0       58.0
2  998000002     Cheree   Sumrell      201    8.5    8.5    6.0    9.0    5.0     NA     NA    6.5     NA        31      24.5       76.0
3  998000003     Judson   Stephan      201    9.0   10.0    0.0    9.0   10.0    6.5    8.0   10.0     NA        42      35.5       96.0
4  998000004     Janina     Kuzma      101    8.5    9.0    9.5    7.5   10.0    9.0    9.5    7.0     NA        37      34.0       73.5
5  998000005     Jarrod     Bring      301    9.0    7.5    5.5    6.0    0.0    8.0    7.0    6.0     NA        33      25.5       42.5
6  998000006     Isreal       Fey      301    9.0   10.0    9.0    8.0    9.5   10.0   10.0    9.0     NA        49      38.0       89.0
```

- We have NA values in our data frame. For example, Quiz.9 is a NA column. We can create a new grade data frame without column 13 (quiz 9) **grade[, -13]**

```
> grade2 <- grade[, -13]
> head(grade2)
  Student.ID First.Name Last.Name Tutorial Quiz.1 Quiz.2 Quiz.3 Quiz.4 Quiz.5 Quiz.6 Quiz.7 Quiz.8 Midterm.1 Midterm.2 Final.Exam
1  998000001        Fae  Grijalva      101    8.0    6.5    7.5    7.5   10.0    9.5    9.5    9.0        42      31.0       58.0
2  998000002     Cheree   Sumrell      201    8.5    8.5    6.0    9.0    5.0     NA     NA    6.5        31      24.5       76.0
3  998000003     Judson   Stephan      201    9.0   10.0    0.0    9.0   10.0    6.5    8.0   10.0        42      35.5       96.0
```

Exercise – Part II

- The next step is another approach for dealing with NA values. Here we will replace all NA values for zero

```
> grade2[is.na(grade2)] = 0
> head(grade2)
  Student.ID First.Name Last.Name Tutorial Quiz.1 Quiz.2 Quiz.3 Quiz.4 Quiz.5 Quiz.6 Quiz.7 Quiz.8 Midterm.1 Midterm.2 Final.Exam
1  998000001        Fae  Grijalva      101    8.0    6.5    7.5    7.5   10.0    9.5    9.5    9.0        42      31.0       58.0
2  998000002     Cheree   Sumrell      201    8.5    8.5    6.0    9.0    5.0    0.0    0.0    6.5        31      24.5       76.0
3  998000003     Judson   Stephan      201    9.0   10.0    0.0    9.0   10.0    6.5    8.0   10.0        42      35.5       96.0
```

- How we can get the sum of all quizzes for each student?
  - We can use the APPLY( ) function

Apply Functions Over Array Margins

Description

Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

Usage

```
apply(X, MARGIN, FUN, ...)
```

Arguments

`X`	an array, including a matrix.
`MARGIN`	a vector giving the subscripts which the function will be applied over. E.g., for a matrix `1` indicates rows, `2` indicates columns, `c(1, 2)` indicates rows and columns. Where `X` has named dimnames, it can be a character vector selecting dimension names.
`FUN`	the function to be applied: see 'Details'. In the case of functions like `+`, `%*%`, etc., the function name must be backquoted or quoted.
`...`	optional arguments to `FUN`.

Exercise – Part III

```
apply(X, MARGIN, FUN, ...)
```

- So, if we want to apply a sum, we will use FUN = sum and this function must be applied to all rows, so MARGIN = 1

quiz.sum = apply(X=grade2[, 5:12], MARGIN = 1, FUN = sum)

```
> dim(grade2)
[1] 117  15
> quiz.sum = apply(X = grade2[, 5:12], MARGIN = 1, FUN = sum)
>
> quiz.sum
  [1] 67.5 43.5 62.5 70.0 49.0 74.5 37.0 68.5 61.5 75.0 52.5 60.0 59.0 66.0 45.0 64.0 77.5 64.0 32.0 70.0 68.5 70.5 65.5 39.0 66.0 63.5 57.5 73.5
 [29] 71.5 51.0 33.5 44.5 69.5 71.5 55.5 69.5 60.0 66.5 70.0 48.0 58.0 56.5 57.5 55.5 54.5 57.0 69.5 70.0 68.5 65.5 75.0 35.0 56.0 53.0 51.5 57.5
 [57] 30.0 46.5 28.0 47.5 49.0 65.0 24.5 63.0 61.5 55.5 23.5 47.5 71.5 72.5 27.0 54.5 74.0 59.0 34.5 64.5 60.0 68.0 68.5  8.5 69.5 32.5 69.5 70.5
 [85] 63.5 58.0 51.0 66.5 72.0 42.5 58.0 71.5 65.0 33.5 51.5 20.5 45.0  0.0 43.0 36.5 60.5 40.5 66.0 51.0 63.0 58.5 77.0 58.5 71.0 35.0 61.0 70.5
[113] 49.0 10.0  0.0 51.0 49.5
>
```

- Now we have the sum of all quizzes for each student!

Exercise – Part IV

- Now, we can calculate the final grade

```
Final.grade = quiz.sum/80*20 + grade2$Midterm.1/50*15
+ grade2$Midterm.2/50*15 + grade$Final.Exam/100*50
```

```
> Final.grade
  [1] 67.775 65.525 86.875 75.550 51.050 89.225 45.900 88.125 73.475 89.500 69.725 72.800 75.450 76.650 44.950 76.100 96.775 71.650 48.250 88.900
 [21] 74.925 88.525 44.825 52.350 85.500 86.875 60.925 90.975 92.825 72.450 80.675 52.925 82.275 91.375 77.325 81.275 88.750 87.925 88.100 76.850
 [41] 63.350 75.025 70.275 77.975 59.625 67.100 84.025 82.000 86.125 92.675 64.300 44.650 77.550 70.400 58.875 85.725 29.050 70.725 48.900 47.125
 [61] 47.750 75.250 38.825 83.400 67.825 73.225 53.225 31.475 83.375 87.825 54.450 63.625 89.300 74.250 54.825 70.175 75.100 62.100 89.825 52.525
 [81] 85.725 66.625 86.175 85.325 87.425 80.600 69.700 87.425 87.550 56.275 70.550 95.275 82.500 50.225 32.775 57.675 45.600  9.250 64.750 63.675
[101] 82.675 48.075 73.850 64.050 71.950 56.875 89.600 65.475 85.150 65.850 63.800 79.625 55.650 15.800 15.500 71.350 70.725
```

- Let's round Final.grade

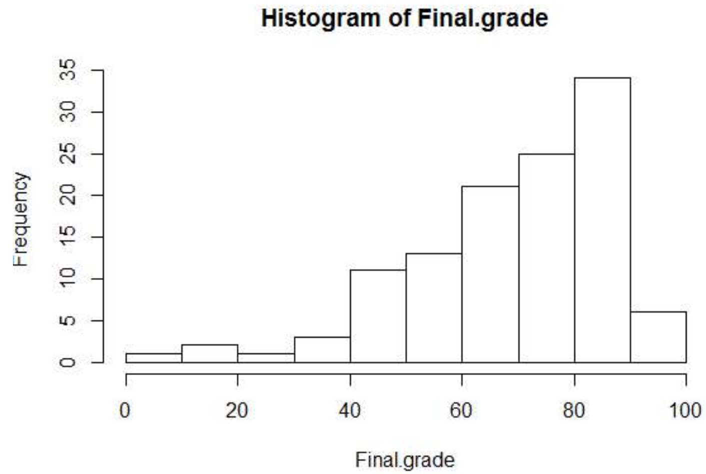
```
Final.grade <- round(Final.grade, 0)
```

- What about to discover how good were the student final grade?
  - We can generate a histogram for this!

Exercise – Part V

- Histogram

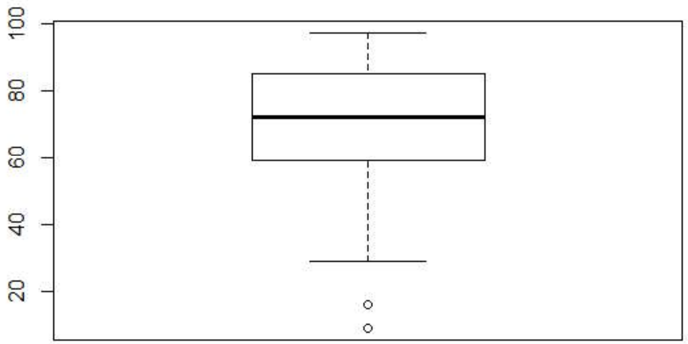
```
hist(Final.grade)
```



Exercise – Part VI

- BoxPlot

```
boxplot(Final.grade)
```



Exercise – Part VII

- We can now assign concepts for our students! For example:

FinalGrade < 50 → "F"

50 <= FinalGrade < 60 → "D"

60 <= FinalGrade < 70 → "C"

70 <= FinalGrade < 80 → "B"

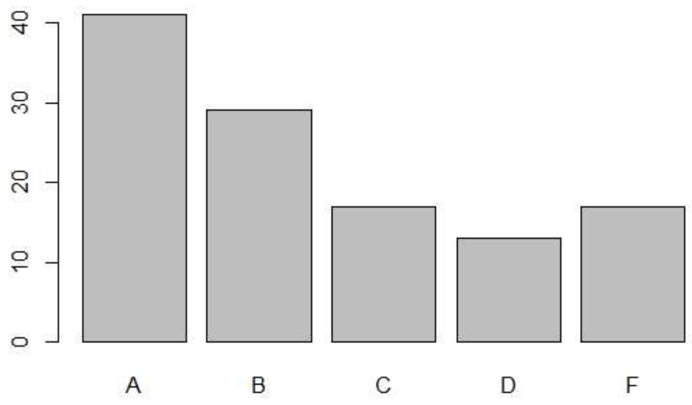
FinalGrade >= 80 → "A"

```
> grade[Final.grade < 50] = "F"
> grade[(Final.grade >= 50) & (Final.grade < 60)] = "D"
> grade[(Final.grade >= 60) & (Final.grade < 70)] = "C"
> grade[(Final.grade >= 70) & (Final.grade < 80)] = "B"
> grade[Final.grade >= 80] = "A"
```

Exercises - VIII

- Now we will generate a **barplot**

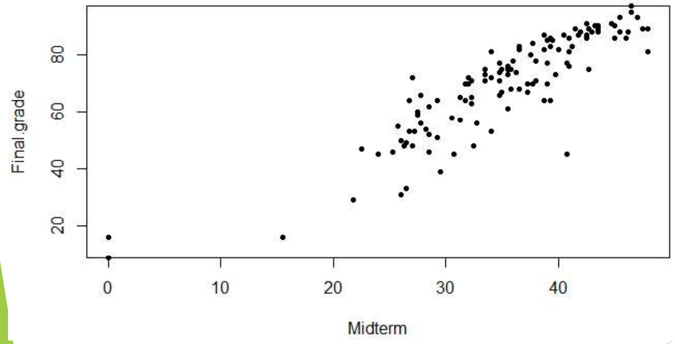
```
> table(grade)
grade
 A  B  C  D  F
41 29 17 13 17
> count <- table(grade)
> barplot(count)
```

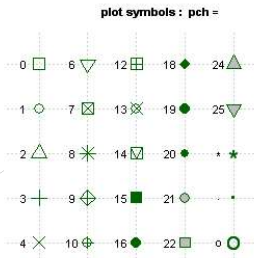


Exercise - IX

- Let's calculate the Midterm for each student and see the relationship between Midterm and Final.Grade

```
Midterm = (grade2$Midterm.1 + grade2$Midterm.2) /2
plot(Midterm, Final.grade, pch=20)
```





Exercise - X

- Lately we will export final grades to a new CSV using **write.csv** function

```
write.csv(Final.grade, file="finalgrade.csv")
```

Demonstração Adicional

http://andrefmb.sdf.org/cursoR/graficosBasicos.html

Part II

GGPLOT2

Ggplot2 and R

- A Picture really is worth a thousand words
- **Visual Analysis** let us understand the basic nature of the data
- We will use ggplot2 – a powerful R package that produces data visualizations easily and intuitively
- ggplot2 is a third package
  - We have to install it
  - *install.packages("ggplot2")*
- Each time we reopen R, we need to load this library using
  - *library("ggplot2")*

Diamonds

- ggplot2 comes with some data available to use as demonstration
- We will use the **Diamonds** dataset
  - It contains information about several attributes of 54000 diamonds
  - We can access it with
  - **data("diamonds")**

  - Try **?diamonds**
    - **View(diamonds)**

> ?diamonds

R: Prices of 50,000 round cut diamonds ▾ Find in Topic

Description

A dataset containing the prices and other attributes of almost 54,000 diamonds. The variables are as follows:

Usage

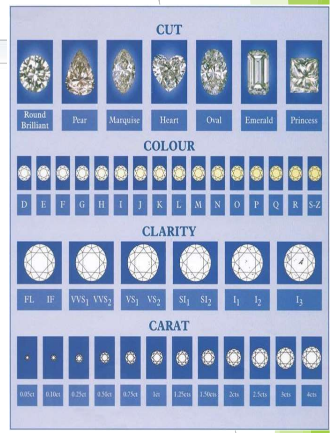
```
data(diamonds)
```

Format

A data frame with 53940 rows and 10 variables

Details

- price. price in US dollars (\$326–\$18,823)
- carat. weight of the diamond (0.2–5.01)
- cut. quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- colour. diamond colour, from J (worst) to D (best)
- clarity. a measurement of how clear the diamond is (I1 (worst), SI1, SI2, VS1, VS2, VVS1, VVS2, IF (best))
- x. length in mm (0–10.74)
- y. width in mm (0–58.9)
- z. depth in mm (0–31.8)
- depth. total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
- table. width of top of diamond relative to widest point (43–95)



http://www.bluediamondtexas.com/images/diamond-chart.jpg

Scatterplots and Bar Graph

Interesting Questions - Diamonds

- How does weight, in carats, affect the price?
- How does the quality of color, or the diamond's clarity, affect the price?
- How can we determine the relationship between attributes??
  - We can use, for example, a **scatter plot**
    - Scatter plot is a type of mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data [Wikipedia]
  - Aesthetics
    - A dimension of a graph that we can perceive visually
      - Color, size, shape of the points, etc.

Our first visualization

- Aesthetics attributes let us communicate some dimension of the data and understand complex relationship between them
- For our first example, we use ggplot2 to create a scatterplot where we put carat (weight) on the X axis and price, in dollars, on the Y axis

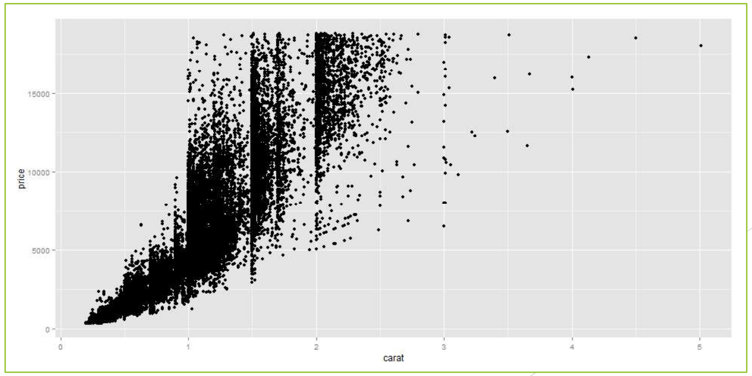
```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point()
```

Our first visualization

- Aesthetics attributes let us communicate some dimension of the data and understand complex relationship between them
- For our first example, we use ggplot2 to create a scatterplot where we put carat (weight) on the X axis and price, in dollars, on the Y axis

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


- And we obtain



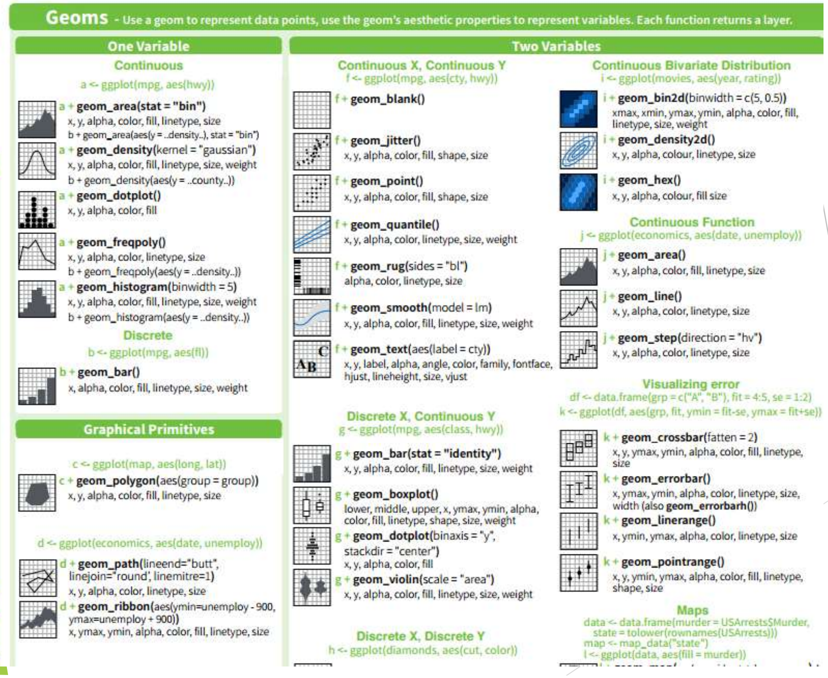
Scatterplot with ggplot2

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

- There are three parts to a ggplot2 graph
  - **1.** data we will be graphing → in this case we a plotting the diamonds data frame
  - **2.** Mapping the aesthetics to attributes we will be ploting → in this case we use aes( ) and set that X axis will be carat and Y axis will be price
  - **3.** Layer: what type of graph it is → In this case we make a scatter plot: the name for that layer is geom_point
    - "geom" is a typical start for each of these layers

Ggplot2 – Geom Types

https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf


Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous
a <- ggplot(mpg, aes(hwy))
a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")
a + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))
a + geom_dotplot()
x, y, alpha, color, fill
a + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))
a + geom_histogram(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete
b <- ggplot(mpg, aes(fl))
b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))
c + geom_polygon(aes(group = group))
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))
d + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, linetype, size
d + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900))
x, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y
f <- ggplot(mpg, aes(cty, hwy))
f + geom_blank()
f + geom_jitter()
x, y, alpha, color, fill, shape, size
f + geom_point()
x, y, alpha, color, fill, shape, size
f + geom_quantile()
x, y, alpha, color, linetype, size, weight
f + geom_rug(sides = "bl")
alpha, color, linetype, size
f + geom_smooth(model = lm)
x, y, alpha, color, fill, linetype, size, weight
f + geom_text(aes(label = cty))
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y
g <- ggplot(mpg, aes(class, hwy))
g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight
g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight
g + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill
g + geom_violin(scale = "area")
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y
h <- ggplot(diamonds, aes(cut, color))

Continuous Bivariate Distribution
i <- ggplot(movies, aes(year, rating))
i + geom_bin2d(binwidth = c(5, 0.5))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight
i + geom_density2d()
x, y, alpha, colour, linetype, size
i + geom_hex()
x, y, alpha, colour, fill size

Continuous Function
j <- ggplot(economics, aes(date, unemploy))
j + geom_area()
x, y, alpha, color, fill, linetype, size
j + geom_line()
x, y, alpha, color, linetype, size
j + geom_step(direction = "hv")
x, y, alpha, color, linetype, size

Visualizing error
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
k + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, linetype, size
k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also geom_errorbarh())
k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size
k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

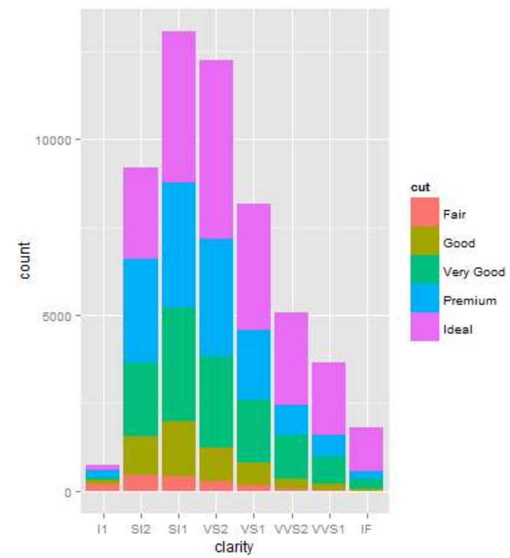
Maps
data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

Bar Graph

```
ggplot(diamonds, aes(x=clarity, fill=cut)) + geom_bar()
```


Bar Graph

```
ggplot(diamonds, aes(x=clarity, fill=cut)) + geom_bar()
```

Our second visualization with ggplot2

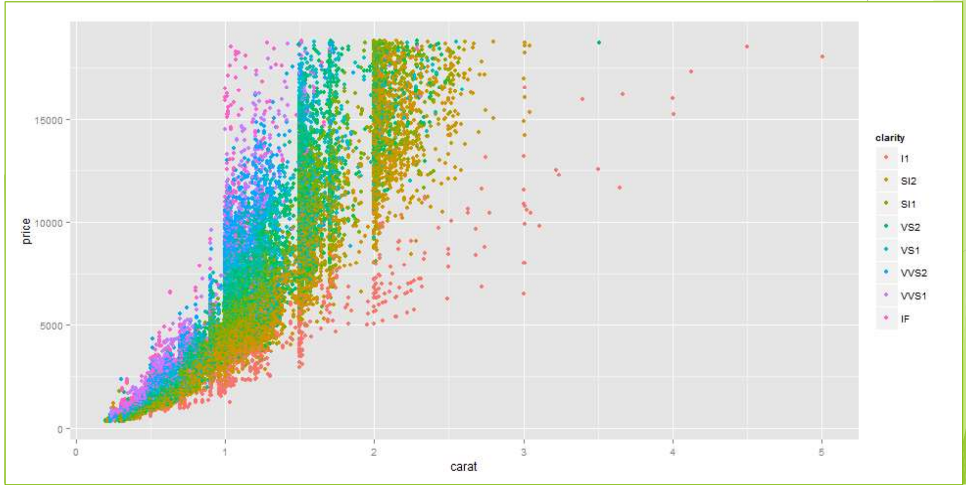
- There are many attributes of the data we can communicate
- Let's put one of diamonds attributes as the color of points

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

Our second visualization with ggplot2

- There are many attributes of the data we can communicate
- Let's put one of diamonds attributes as the color of points

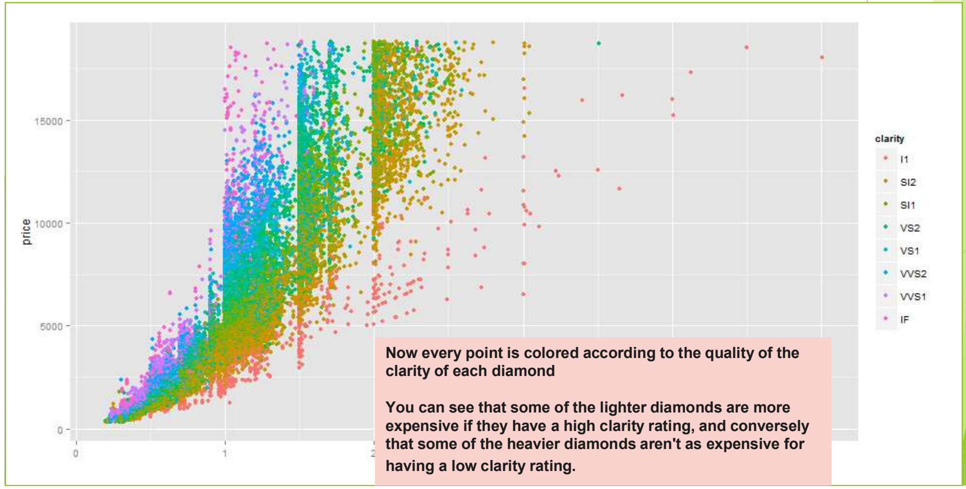
```
ggplot(diamonds, aes(x=carat, y=price, color=clarity)) + geom_point()
```



Our second visualization with ggplot2

- There are many attributes of the data we can communicate
- Let's put one of diamonds attributes as the color of points

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

Now every point is colored according to the quality of the clarity of each diamond

You can see that some of the lighter diamonds are more expensive if they have a high clarity rating, and conversely that some of the heavier diamonds aren't as expensive for having a low clarity rating.

Our third visualization with ggplot2

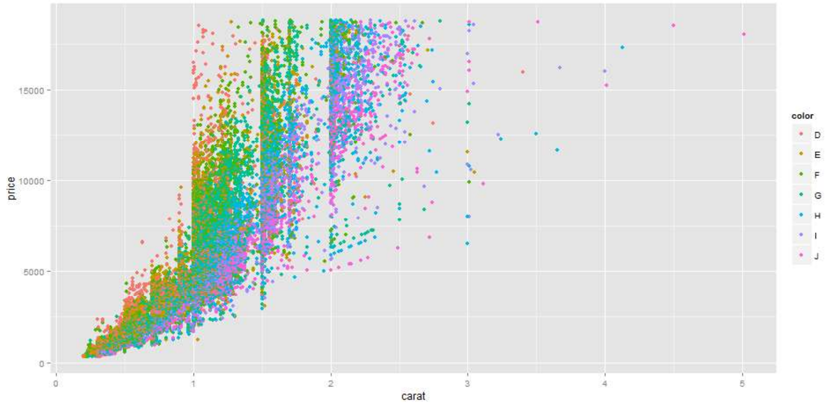
- If we would rather see how the quality of the color or cut of the diamond affects the price?
  - We can change the aesthetic

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

Our third visualization with ggplot2

- If we would rather see how the quality of the color or cut of the diamond affects the price?
  - We can change the aesthetic

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



Add more aesthetic attribute

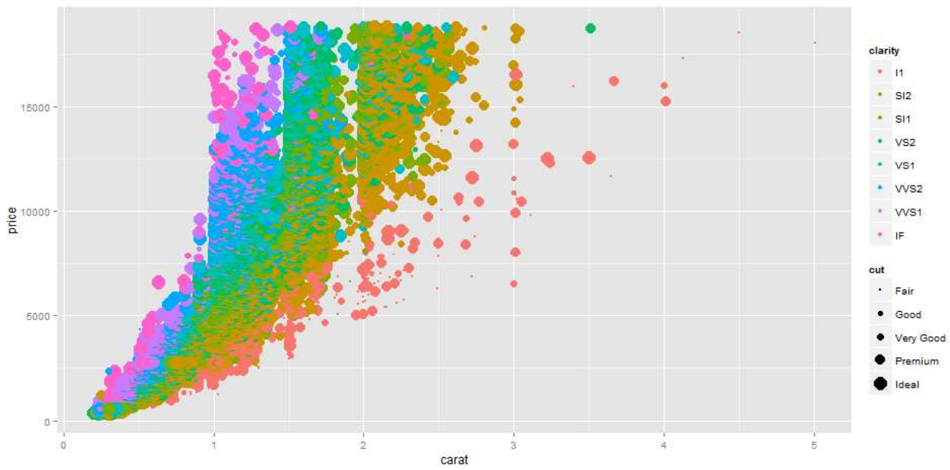
- Now, try this:

```
ggplot(diamonds, aes(x=carat, y=price, color=clarity, size=cut)) + geom_point()
```

Add more aesthetic attribute

- Now, try this:

```
ggplot(diamonds, aes(x=carat, y=price, color=clarity, size=cut)) + geom_point()
```



Adding Layers

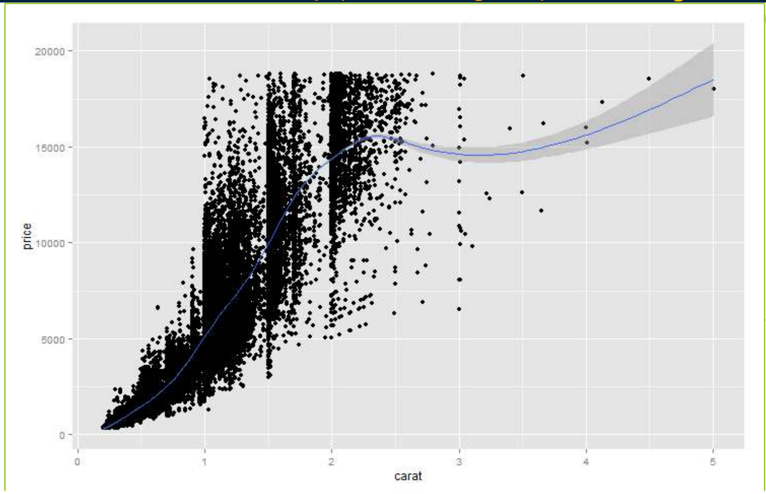
- Scatter plot is only one layer of our graph
- We can add additional layers besides the scatter plot using the "+" sign
- Try this:

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

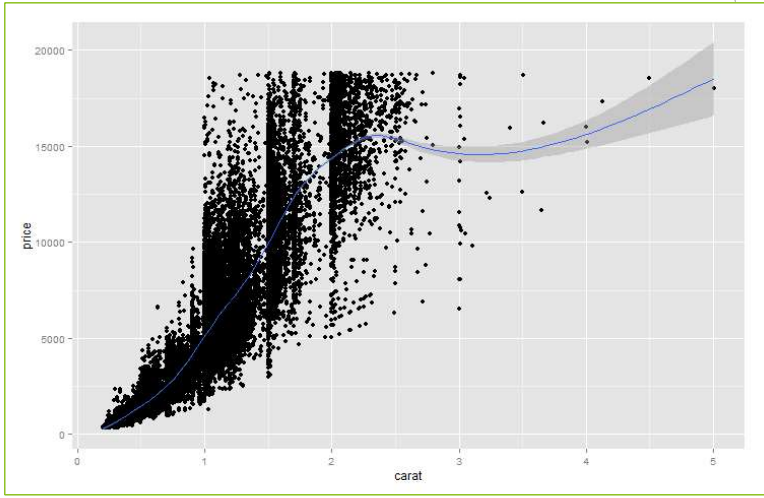
Adding Layers

- Scatter plot is only one layer of our graph
- We can add additional layers besides the scatter plot using the "+" sign
- Try this:

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



geom_smooth()



- Gray around the curve → confidence interval
- Suggesting how much uncertainty there is in this smoothing curve

Linear Method

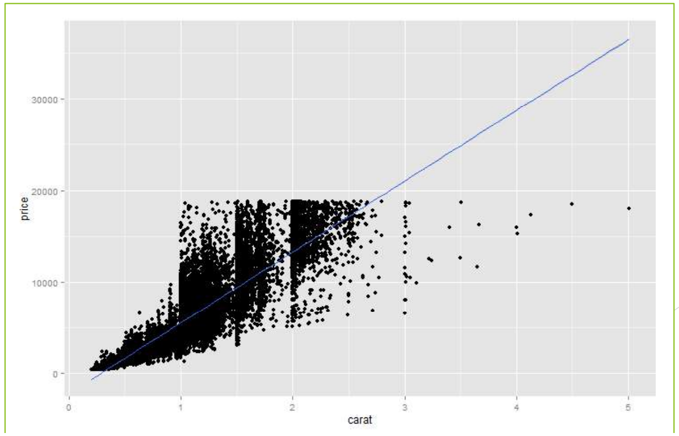
- Similarly, if we would rather show a best fit straight line rather than a curve, we can change the "method" option in the geom_smooth layer. In this case it's method="lm", where "lm" stands for "Linear model".

```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point() + geom_smooth(method="lm")
```

Linear Method

- Similarly, if we would rather show a best fit straight line rather than a curve, we can change the "method" option in the geom_smooth layer. In this case it's method="lm", where "lm" stands for "Linear model".

```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point() + geom_smooth(method="lm")
```



Faceting

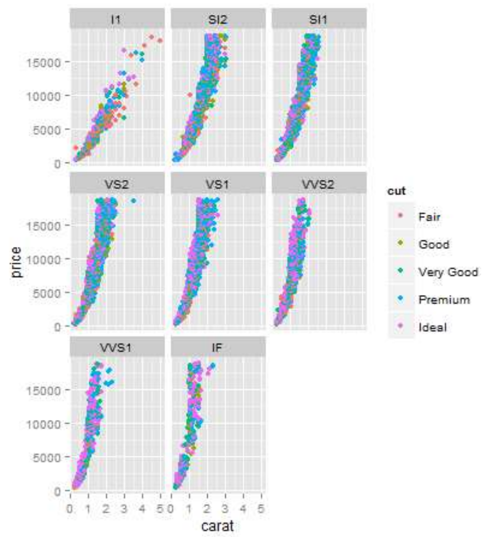
- Another way we can communicate information about an attribute is to divide our plot up into multiple plot
  - This is called "faceting"
  - Ggplot2 makes it very easy with the "facet_wrap" function

```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + facet_wrap(~ clarity)
```

  - We put a tilde (~) and then the attribute we would like to divide the plots by, here "clarity"

Faceting

```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + facet_wrap(~ clarity)
```



Faceting

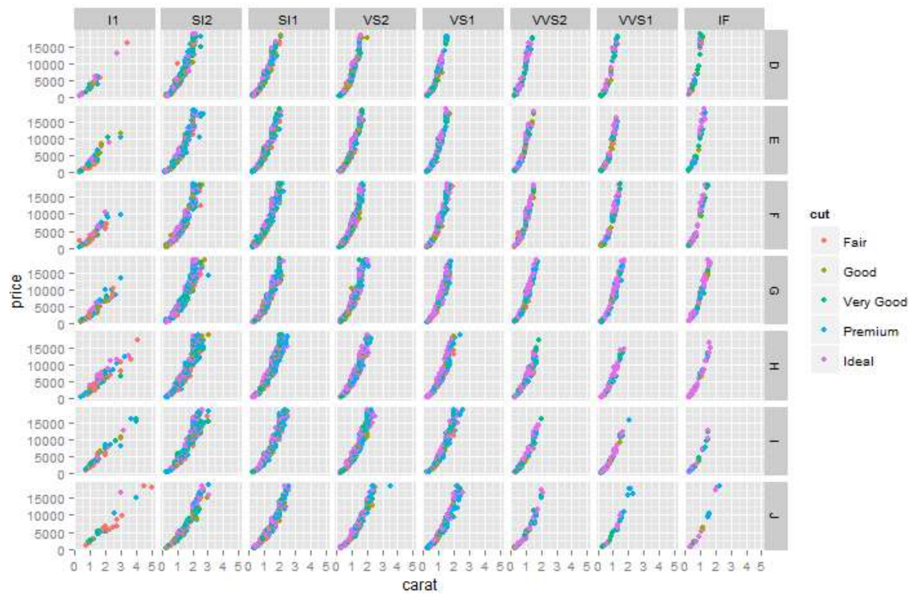
- Let's zoom in on this
- We have divided it into eight subplots, each of which has a different clarity value
- We can even divide our graph based on two different attributes, such as both color and clarity, using *facet_grid*
- For example

```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + facet_grid(color ~ clarity)
```

- In this case we have: color ~ clarity
  - It means: color explained by clarity
  - Color will be on X axis (row)
  - Clarity on Y axis (column)

Fatecing - Grid

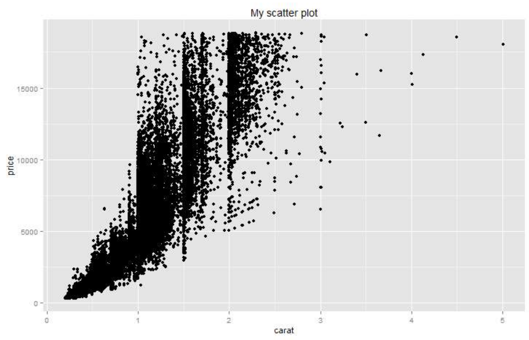
```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + facet_grid(color ~ clarity)
```



Ggplot2: Title and Labels

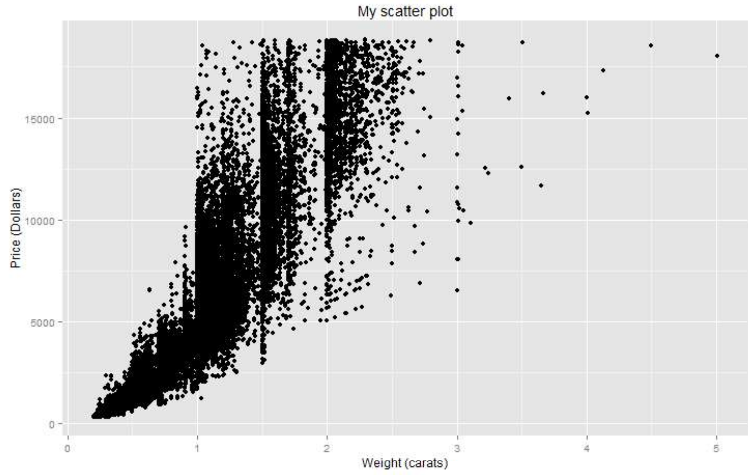
- There are many other ways to customize a plot
- Firstly, we want to set a title or set the x or y axis labels manually
- We change these options adding to the end of the line of code

```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point() + ggtitle("My scatter plot")
```



Ggplot2: Title and Labels

```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point()
+ ggtitle("My scatter plot")
+ xlab("Weight (carats)")
+ ylab("Price (Dollars)")
```

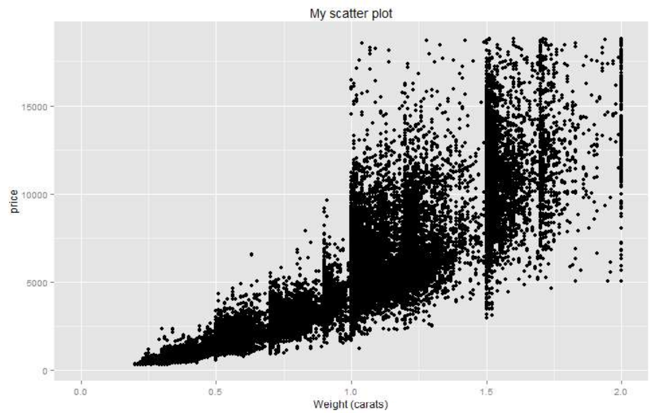


Limiting ranges

- We can also limit the range of the x or the y axes

```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point()
+ ggtitle("My scatter plot")
+ xlab("Weight (carats)")
+ xlim(0, 2)
```

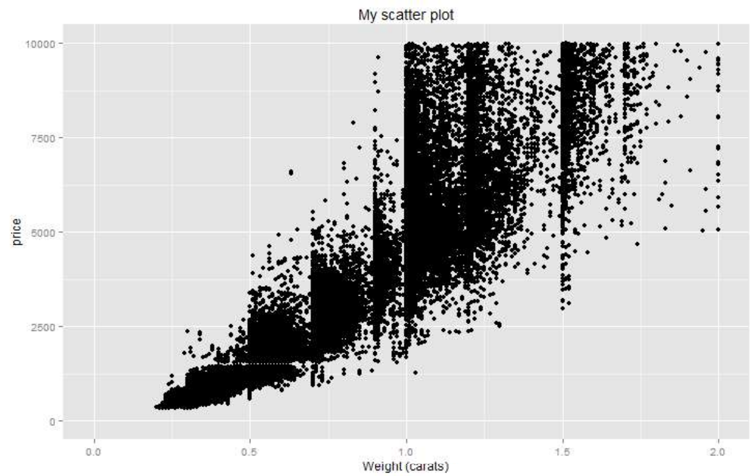
```
Warning message: Removed 1889 rows containing missing values (geom_point).
```



Limiting ranges

- Similarly, if we wanted to show only the y-axis from 0 to 10000

```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point()
+ ggtitle("My scatter plot")
+ xlab("Weight (carats)")
+ ylim(0,10000) + xlim(0,2)
```



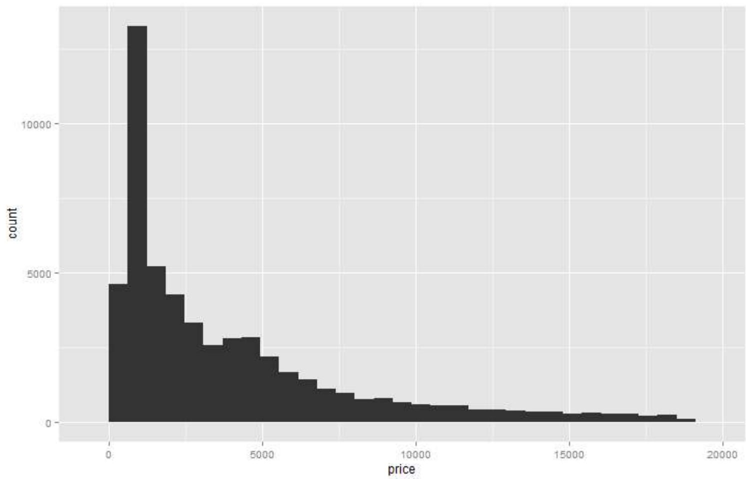
Histograms and Density Curves

Histograms

- Scatter plots are just one kind of graph!
- Sometimes we want to look at just one dimension of our data and observe its distribution: for that, we'll use a histogram
- It is very easy: all you need to do to make a histogram is to change your layer from **geom_point( )** to **geom_histogram( )**

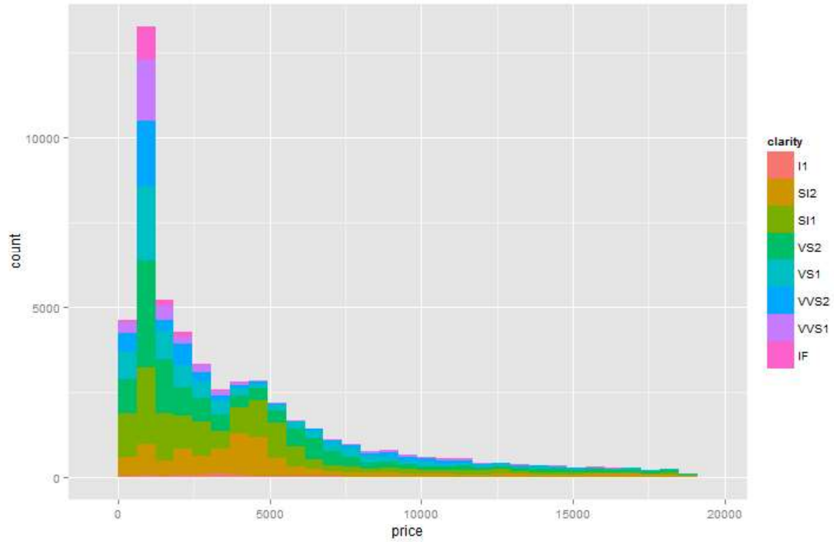
Histograms

```
ggplot(diamonds, aes(x=price)) + geom_histogram()
```



Another example

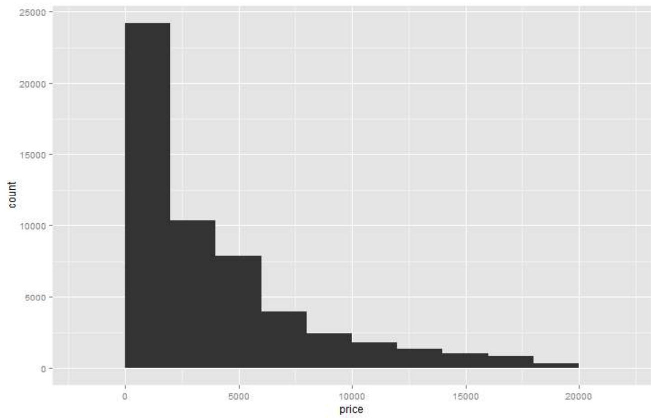
```
ggplot(diamonds, aes(x=price, fill=clarity)) + geom_histogram()
```



Histograms: Aesthetic

- We can change the width of each bin as an options to geom_histogram layer

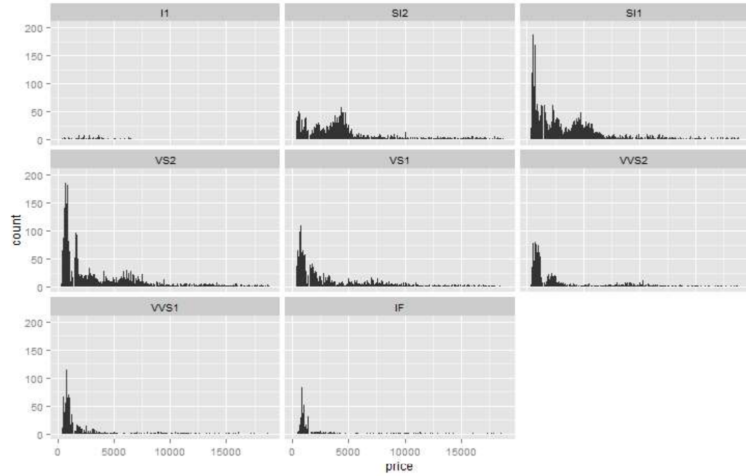
```
ggplot(diamonds, aes(x=price)) + geom_histogram(binwidth=2000)
```



Histograms and Facet_wrap

- Let's divide our histogram by clarity

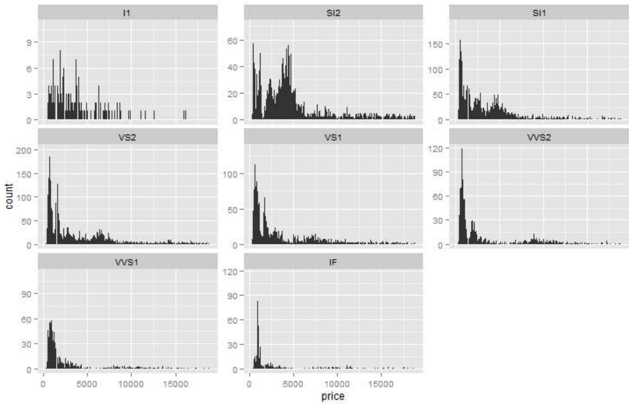
```
ggplot(diamonds, aes(x=price)) + geom_histogram(binwidth=20) + facet_wrap(~clarity)
```



Histograms and Facet_wrap

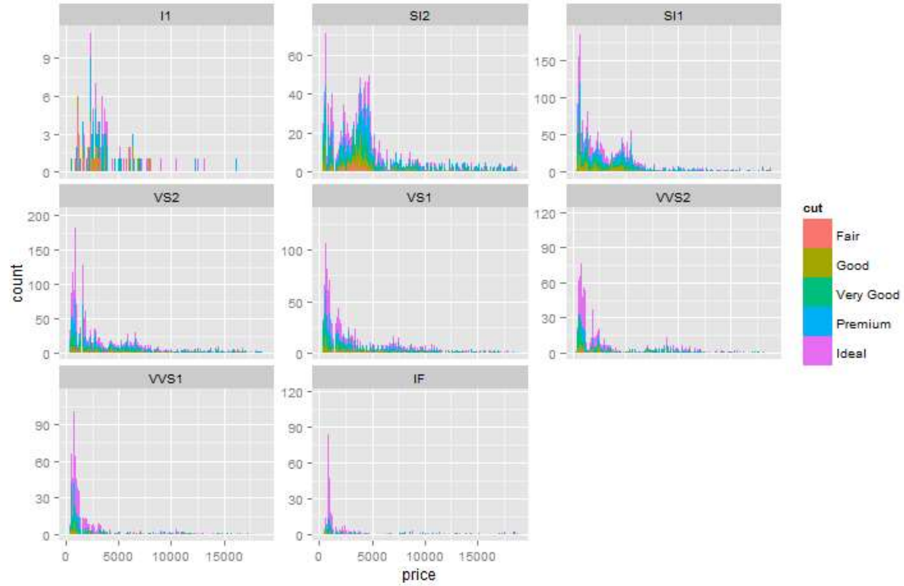
- Each subplot shares the same Y axis, which might make it hard to interpret the frequencies
- We can add **scale=free_y**

```
ggplot(diamonds, aes(x=price)) + geom_histogram(binwidth=20)
+ facet_wrap(~clarity, scale="free_y")
```



Add more information

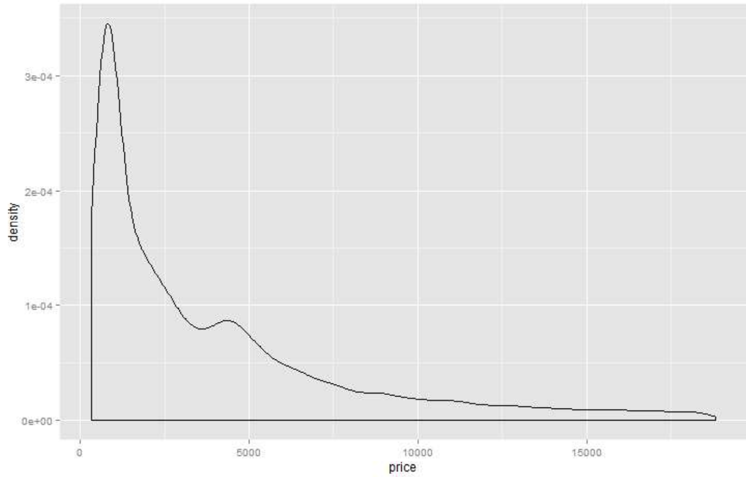
```
ggplot(diamonds, aes(x=price, fill=cut))
+ geom_histogram(binwidth=20)
+ facet_wrap(~clarity, scale="free_y")
```



Density

- Another way to view the distribution is as a **density curve**

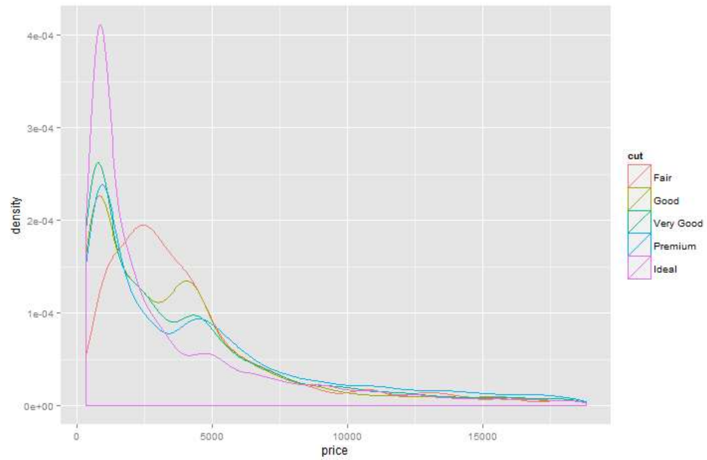
```
ggplot(diamonds, aes(x=price)) + geom_density()
```



Density

- We can want to divide this density curve up based on one of attributes

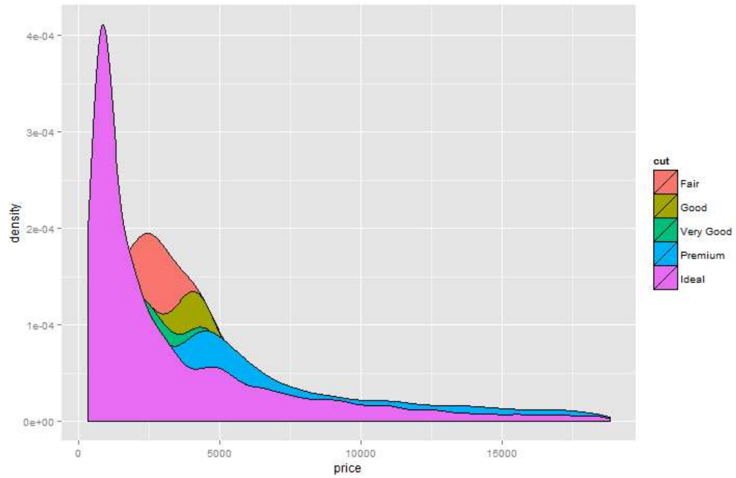
```
ggplot(diamonds, aes(x=price, color=cut)) + geom_density()
```



Density

- We can want to divide this density curve up based on one of attributes

```
ggplot(diamonds, aes(x=price, fill=cut)) + geom_density()
```



Boxplots and Violin Plots

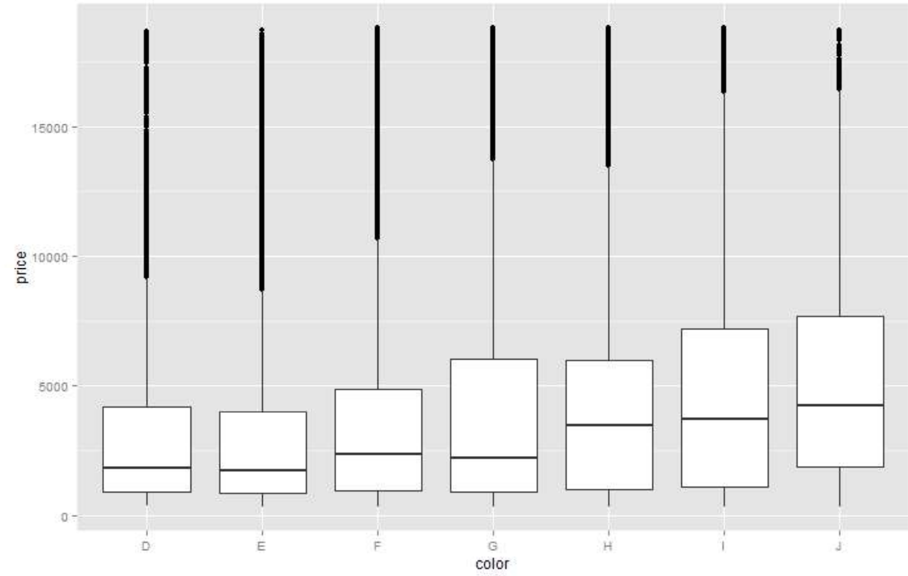
Boxplots

- One common method in statistics for comparing multiple densities is to use a **boxplot**
- A boxplot has two attributes: an x – which is usually a classification into categories, and y, the actual variable that we're comparing
- Let's say we want to compare the distribution of the price within each color

```
ggplot(diamonds, aes(x=color, y=price)) + geom_boxplot()
```

Boxplots

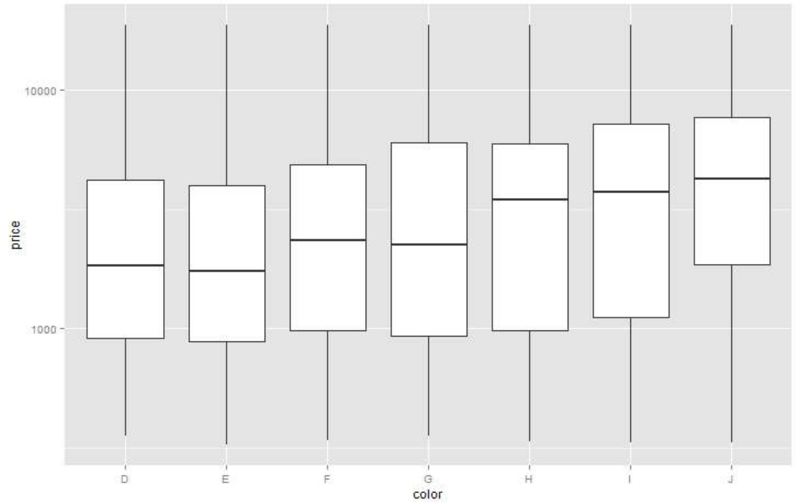
```
ggplot(diamonds, aes(x=color, y=price)) + geom_boxplot()
```



Boxplots

- Let's see the boxplots using LOG(10) on Y axis

```
ggplot(diamonds, aes(x=color, y=price)) + geom_boxplot() + scale_y_log10()
```

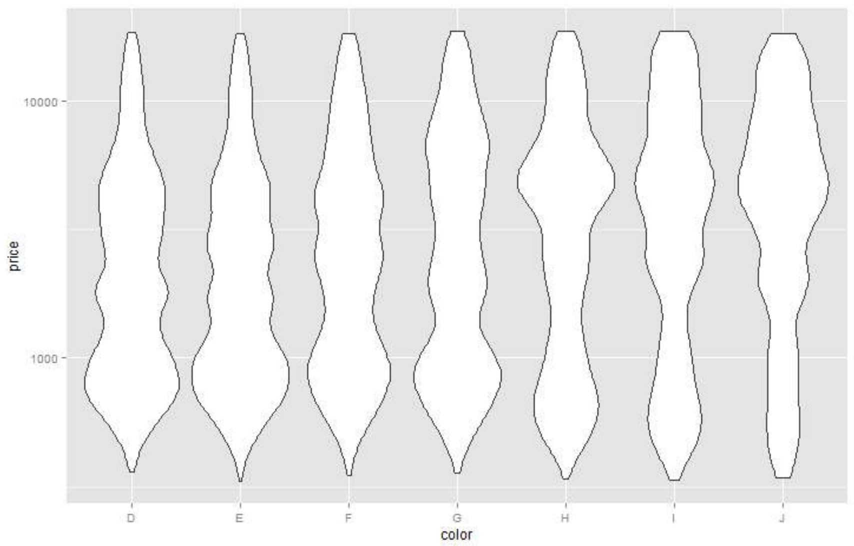


Violin Plot

- Boxplots does not show details of the distribution besides the quantiles
  - It works well when the data follows a Normal distribution
  - But it might not work well for stranger distributions
- We can instead view the distribution as a density using what's called a **violin plot**
- It's very straightforward, we only need to change **geom_boxplot** to **geom_violin**

Violin Plot

```
ggplot(diamonds, aes(x=color, y=price)) + geom_violin() + scale_y_log10()
```



qplot

qplot

- So far all of our analysis have started with a data frame
  - One row per observation
  - One column for each attribute
- BUT … let's say you have just one vector of numbers and you want to create a histogram
  - Or you have two vectors and want to make a scatterplot
- We don't need to construct a dataframe
- Ggplot2 provides a simple way to plot one or two vectors, which is the **qplot** function

Qplot - Example

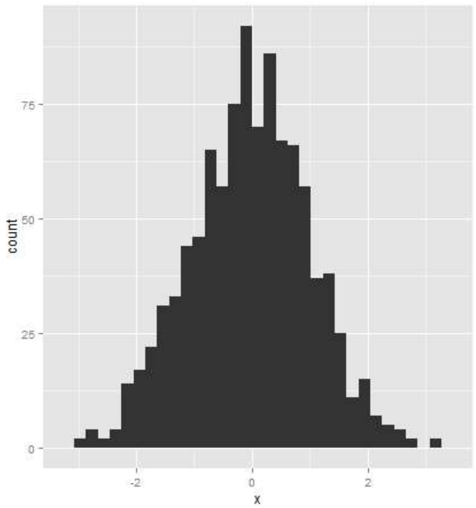
- Try this

```
x = rnorm(1000)
qplot(x)
```

Qplot - Example

- Try this

```
x = rnorm(1000)
qplot(x)
```



Qplot - Example

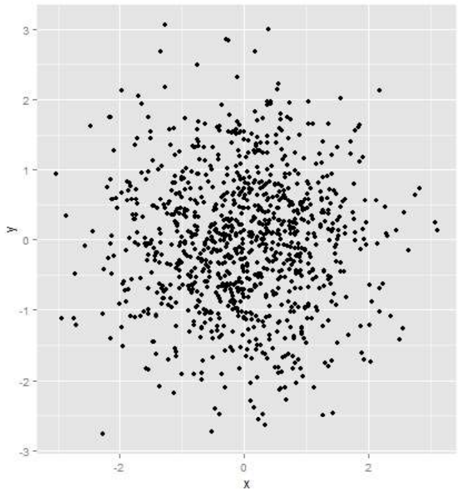
- Try this

```
x = rnorm(1000)
y = rnorm(1000)
qplot(x,y)
```

Qplot - Example

- Try this

```
x = rnorm(1000)
y = rnorm(1000)
qplot(x,y)
```



Qplot - Example

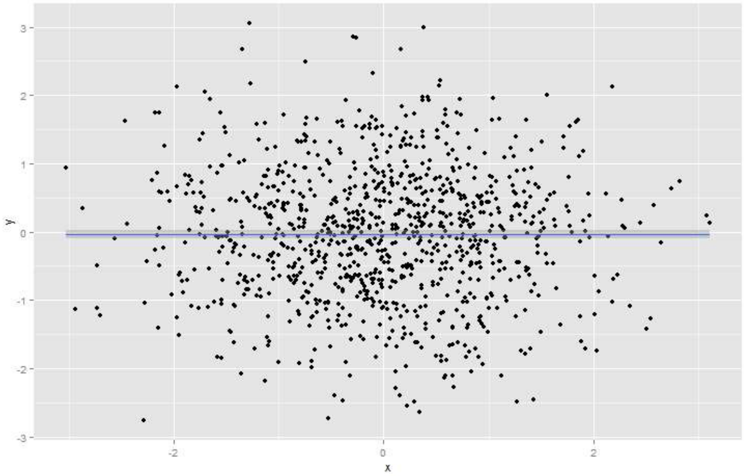
- Try this

```
x = rnorm(1000)
y = rnorm(1000)
qplot(x,y)
+ geom_smooth()
```

Qplot - Example

- Try this

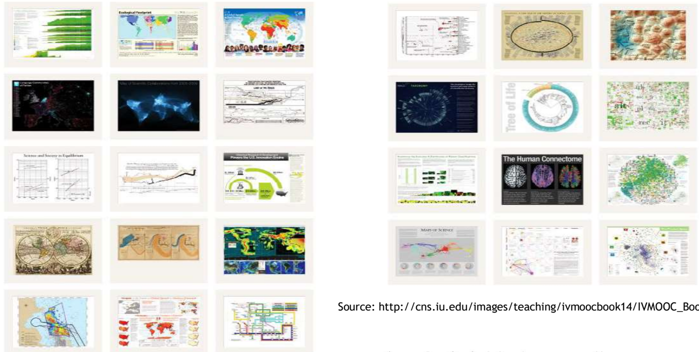
```
x = rnorm(1000)
y = rnorm(1000)
qplot(x,y)
+ geom_smooth()
```



Additional References for GGPLOT2

- GGPLOT2 CHEAT SHEET
  - https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Data Analysis and Visualization with R



Source: http://cns.iu.edu/images/teaching/ivmoocbook14/IVMOOC_Book_Preview.html

ESCOLA POLITÉCNICA DA USP

Bio Comp

André Batista, Ph.D. Student

andrefmb@usp.br

2016