University of Chicago Professional Education

MSCA 37010 Programming for Analytics
Week 5 Lecture Notes

Autumn 2020

❑ Introduction

Instructor: Mei Najim
Email: mnajim@uchicago.edu
Tel: 847-800-9979 (C)
Class Meeting Time: 6:00 - 9:00pm, Mondays (01 Section)
1:30 - 4:30pm, Saturdays (02 Section)
Office Hour: Mondays (9:00pm until last students)
Saturdays (4:30pm until last students)
Or, email with questions first and following by an appointment if needed

Notes: 1) First ten-minute quiz; 10-minute breakout group in zoom; Three 5-minute breaks
2) Set up a weekly discussion group on canvas, allow 24 hours to respond
3) Email questions with the **Section Number (01/02)** in the subject line
4) If it is urgent, feel free to text me directly (847-800-9979)

Week 5 Class Agenda

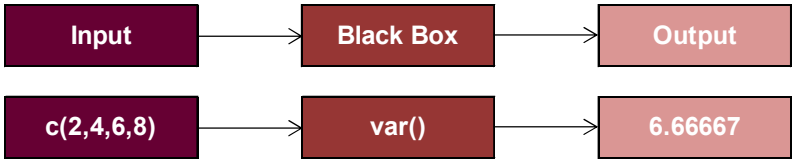
- Functions
- Data Visualization
- Memory Management
- R Shiny

Week 5 Class Agenda

- Functions
- Data Visualization
- Memory Management
- R Shiny

❑ Function – Built-in Functions

- A function, in a programming environment, is a set of instructions. **A developer or programmer** for analytics builds a function to **avoid repeating the same task or reduce complexity**.
- A function flow chart: an input > a box > generates output



- Many things in R are done using function calls.

  Example: ***log***(x) or ***plot***(height, weight)

❑ Function – Create Your Own Functions

- The real advantage of R is that users are able to write their own functions
- A Function is a useful device that groups together a set of statements so they can be run more than once. They can also allow us to specify parameters/arguments that can serve as inputs to the functions
- Functions in R are created by assignment; A general approach to a function is to use the argument part as inputs, feed the body part and finally return an output.
- The Syntax of a function is the following:

```
Function_Name <- function(argu1, argu2, argu3, argu4, argu5=default_value) {
  #  Function body / Code Execute
  result <- argu1 + argu2 - argu3 * argu4/argu5
  return(result)
  }
```

- *Function_Name* is the name of the function
- ***function*** is a keyword
- *function body* defines the body of the function that can be any R statement, which is usually a group of statements and enclosed with brackets

❑ Function – When to Create a Function

- We should consider writing a function whenever you've **copied and pasted a block of code more than twice**
- Example: Suppose we would like to detect the number of extreme values from a numeric vector. The extreme values are the ones that are greater than or less than two times standard deviation from mean; We can generate a sequence of number that follow the standard normal distribution to test this example

```
> set.seed(39424); vec1<-rnorm(100); sum((abs(vec1-mean(vec1)))>2*sd(vec1)) [1] 3

> set.seed(39381); vec2<-rnorm(100); sum((abs(vec2-mean(vec2)))>2*sd(vec1)) [1] 1

> set.seed(59323); vec3<-rnorm(100); sum((abs(vec3-mean(vec3)))>2*sd(vec3)) [1] 5
```

- Extracting repeated code out into a function to prevent you from making careless mistakes
- Another advantage is that if our requirements change, we only need to make the change in one place – more robust

Note: We often need to create random data. To ensure we all generate the same data when we rerun the codes, we use the ***set.seed***() function with arbitrary values of 123. **The *set.seed()* function is generated through the process of pseudorandom number generator that make every modern computers to have the same sequence of numbers**. If we don't use set.seed() function, we will all have different sequence of number

❑ Function – How to Create Your Own Functions

- Example Continued:

```
extreme <- function (vec){
        extreme_n <- sum(abs(vec-mean(vec))>2*sd(vec))
        extreme_n
  }
```

> extreme(vec1) Output: [1] 4

> extreme(vec2) Output: [1] 3

> extreme(vec3) Output: [1] 5

❑ Function – How to Create Your Own Functions

- Solve a particular, well-defined problem: input >> ***triple()*** >> output

Format: ***myfunctionname*** <- **function**(arg1) {
founction body
}

- Example: Create a function to triple the values with one argument

```
triple <- function(x) {
  3*x
}
> triple(6)
```

Output:

[1] 18

Exercise: Create a function to take a square root of values

❑ Function – How to Create Your Own Functions

- Create a calculation function with two arguments

```
calc <- function(x,y) {
     x/y+sqrt(x)*y
 }
```

Output:

```
> calc(3,4)
[1] 7.678203
```

Exercise: Create a function to calculate a*b+log(a)

❑ Function – How to Create Your Own Functions

- Create a calculation function with optional arguments /default values

```
calc_opt <-function(x,y=2) {
    x/y+sqrt(x)*y
}
```

```
> calc(3,4)
```

Output: [1] 7.678203

```
> calc_opt(3)
```

Output: [1] 4.964102

```
> calc_opt(3,0)
```

Output: [1] Inf

❑ Function – How to Create Your Own Functions

- Create a calculation function with ***return()*** function

```
calc_return <-function(x,y=2) {
  if (y==0) {
    return(0)        # return 0 and exit function
    }
   2*x+sqrt(x)*y   # only execute when y is not equal to 0
}
```

> calc_return(3,0)

Output: [1] 0

> calc_return(3,1)

Output: [1] 7.732051

❑ Function – How to Create Your Own Functions

- Function Scoping implies that variables that are defined inside a function are not accessible outside that function. y was defined inside the calc3() function and therefore it is not accessible outside of that function. This is also true for the function's arguments of course - x in this case.

```
calc3 <-function(x) {
    y <- log(x)+exp(x)
    return(y)
}
```

```
> calc3(4)
Output: [1] 55.98444
> y
Output: Error: object 'y' not found
> x
Output: Error: object 'x' not found
```

❑ Function – How to Create Your Own Functions

- Create in R, the environment is a collection of objects like functions, variables, data frame, etc.
- R opens an environment each time RStudio is prompted.
- The top-level environment available is the global environment, called **R_GlobalEnv.** And we have the **local environment.**
- We can list the content of the current environment. This will output the list of global variables and functions on the right pane of RStudio Environment section.
- Difference between global and local environments:

  - Local variable is declared/defined inside a function (the scope is limited to the inside of the function)

    whereas Global variable is declared/defined outside the function.

  - Local variables are created when the function has started execution and is lost when the function terminates, on the other hand, Global variable is created as execution starts and is lost when the program ends.

  - More to refer: http://adv-r.had.co.nz/Environments.html

❑ Function – How to Create Your Own Functions

- Create a calculation function with optional argument ***return()*** function

```
calc_opt <-function(x,y=2) {
     x/y+sqrt(x)*y
}
```

```
> Calc_opt(3)
```

Output:

```
[1] 7.678203
```

Week 5 Class Agenda

- Functions
- Data Visualization
- Memory Management

❑ Data Visualization

- Creating effective data visualizations is one of the most valuable skills a Data Scientist can posses.
- More than just making fancy charts, visualization is a way of communicating a dataset's information in a way that's easy for people to understand. With a good visualization, one can most clearly see the patterns and information that often lie hidden within the data.

  - ***In the early stages of a project***, you'll often be doing an Exploratory Data Analysis (EDA) to gain some insights into your data. Creating visualizations will help you along the way to speed up your analysis.

  - ***Towards the end of your project***, it's important to be able to present your final results in a clear, concise, and compelling manner that your audience, who are often non-technical stakeholders, can understand.

- There's no doubt: taking your visualizations to the next level will turbocharge your analysis — and help you knock your next presentation out of the park.

❑ Data Visualization

- We are going to introduce Data Visualization in the following four topics:
  - ✓ Introduction
  - ✓ Basic Plots
  - ✓ Visualization Libraries in R
  - ✓ Visualizing Geographical Data in R

❑ Data Visualization – Introduction

- Data Visualization - Selecting the Right Chart Type. There are four basic descriptive exploratory data analysis (EDA) presentation types:
    1. Comparison
    2. Composition
    3. Distribution
    4. Relationship

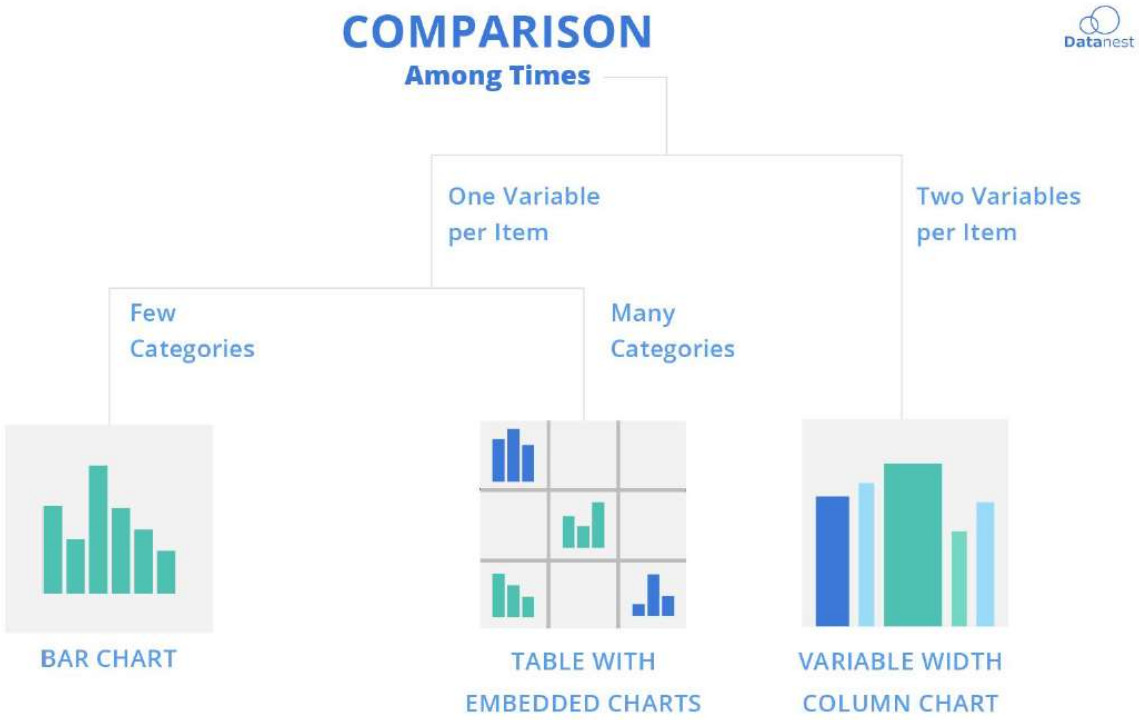
❑ Data Visualization – Introduction

Comparison

Composition

Distribution

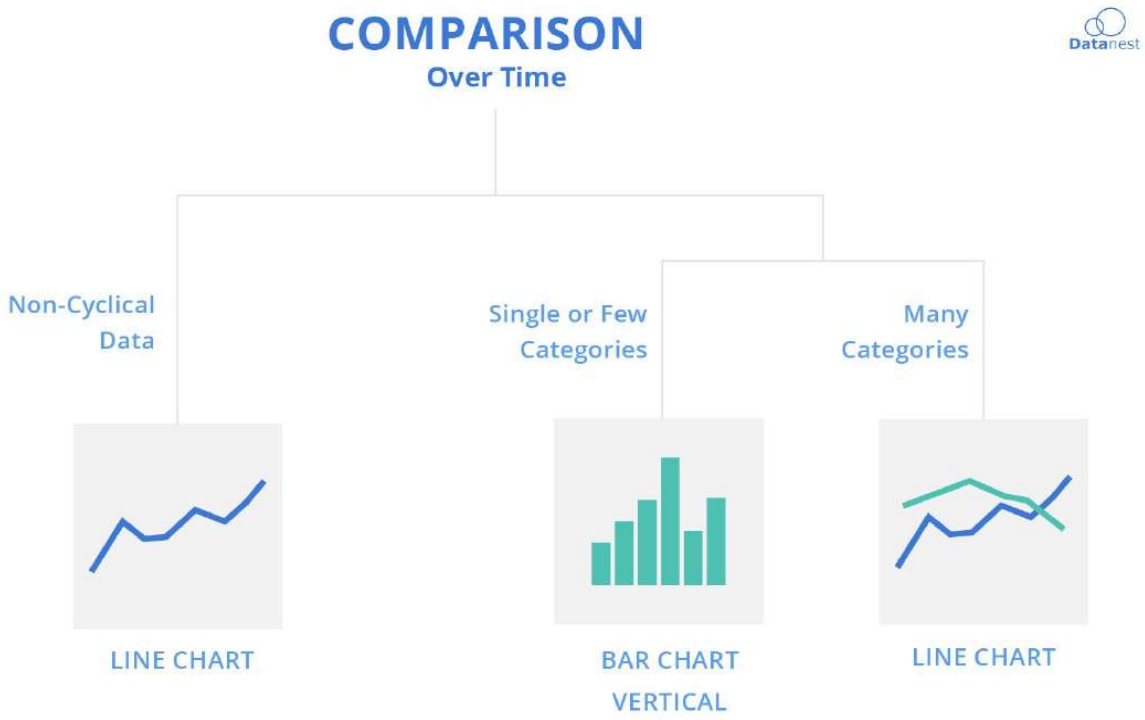
Relationship



Source: Datanest

❑ Data Visualization – Introduction

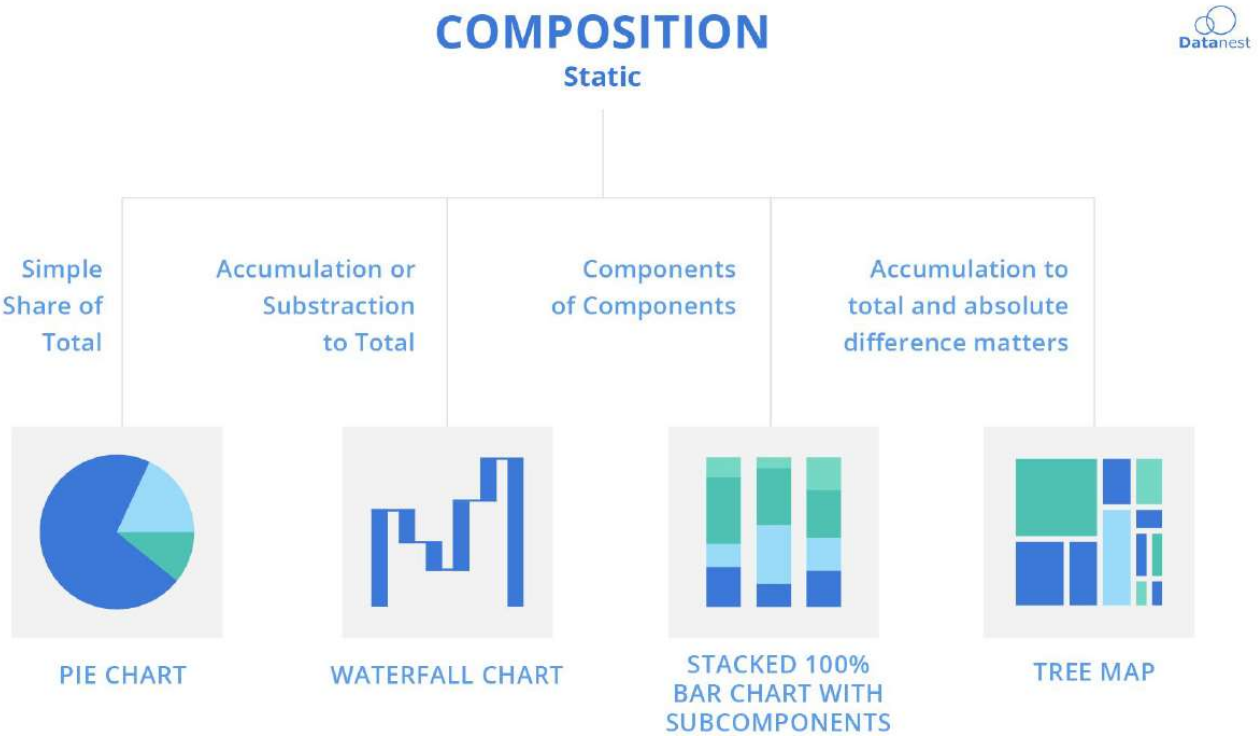
Comparison
Composition
Distribution
Relationship



Source: Datanest

❑ Data Visualization – Introduction

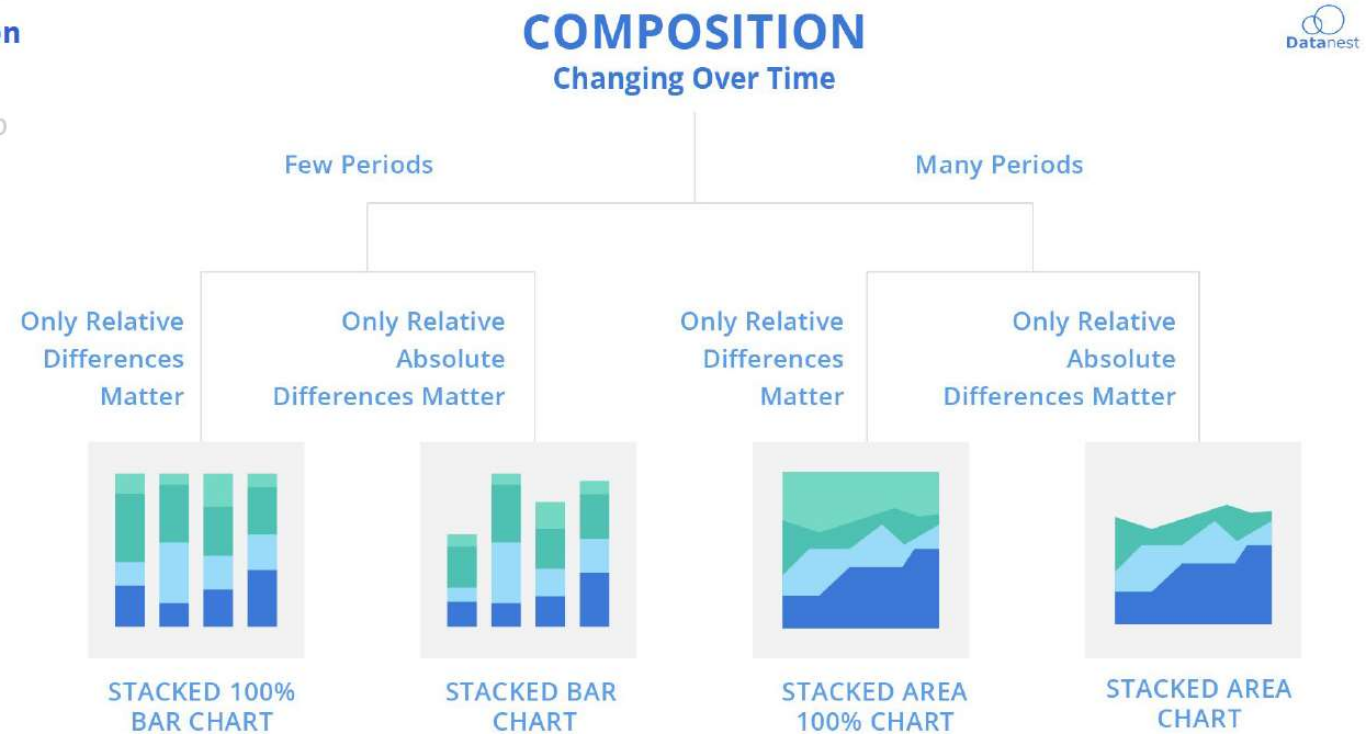
Comparison
Composition
Distribution
Relationship



Source: Datanest

❑ Data Visualization – Introduction

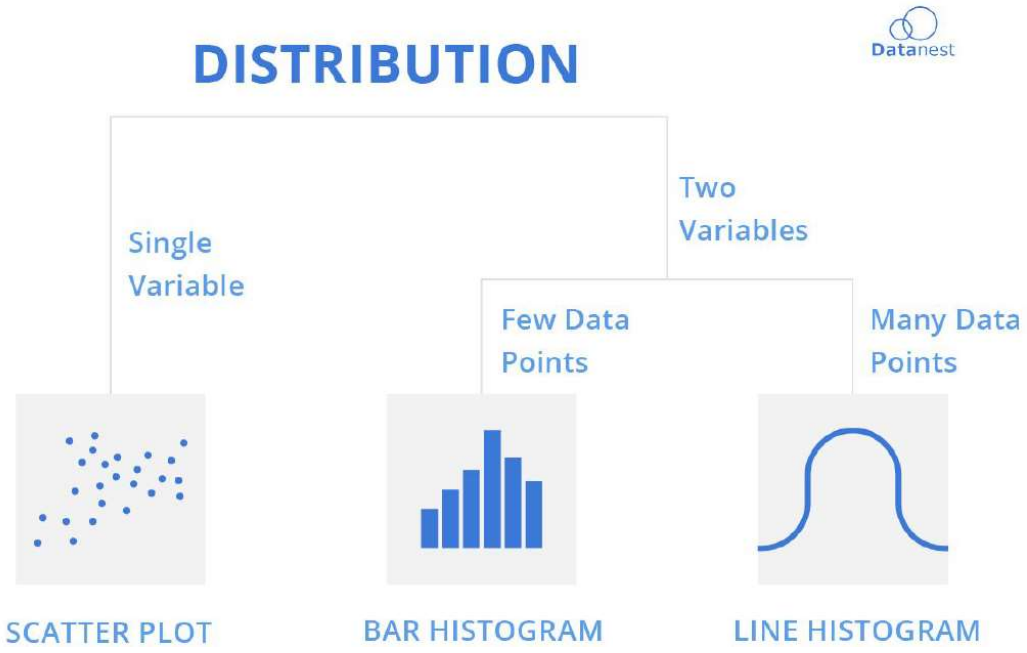
Comparison
Composition
Distribution
Relationship



Source: Datanest

❑ Data Visualization – Introduction

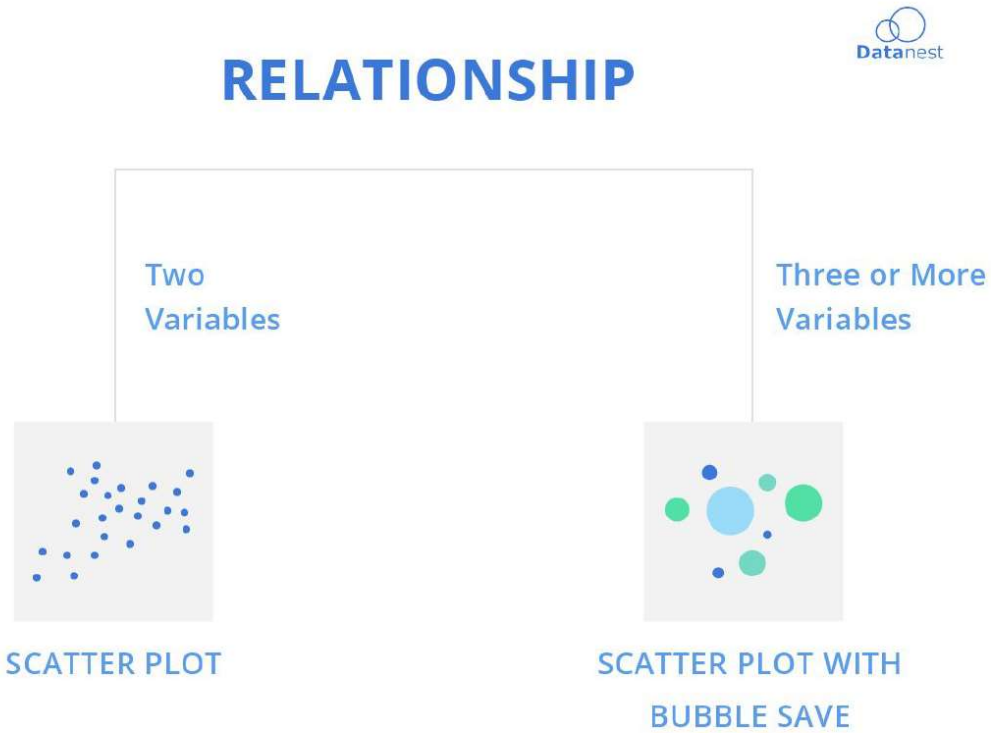
Comparison
Composition
Distribution
Relationship



Source: Datanest

❑ Data Visualization – Introduction

Comparison
Composition
Distribution
Relationship



Source: Datanest

❑ Data Visualization

- We are going to introduce Data Visualization in the following four topics:
  - ✓ Introduction
  - ✓ Basic Plots
  - ✓ Visualization Libraries in R
  - ✓ Visualizing Geographical data in R

❑ Data Visualization – Basic Plots

- To determine which amongst these is best suited for the data, a few questions:
  - How many variables do you want to show in a single chart?
  - How many data points will you display for each variable?
  - Will you display values over a period of time, or among items or groups?
- Basic Charts
  1. Scatter Plot
  2. Histogram
  3. Bar & Stack Bar Chart
  4. Box Plot
  5. Grid of Charts

❑ Data Visualization – Basic Plots

- We will use the dataset called the air quality dataset, which pertains to the daily air quality measurements in New York from May to September 1973.

  - This dataset consists of more than 100 observations on 6 variables:

  Ozone(mean parts per billion), Solar.R(Solar Radiation), Wind(Average wind speed),

  Temp(maximum daily temperature in Fahrenheit), Month(month of observation), Day(Day of the month)

- To load the built-in dataset into the R type the following command in the console:

  data(airquality)

❑ Data Visualization – Basic Plots

- Data Exploration - it is time to explore it to get an idea about its structure.

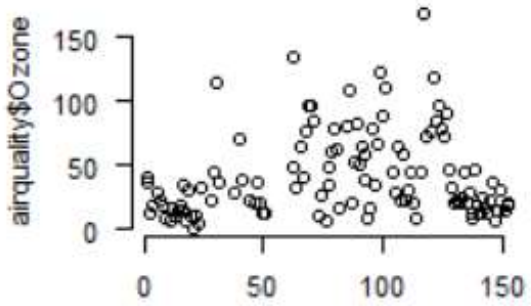
```
str(airquality)          # Displays the data structure and overview of the rows and columns of the dataset
head(airquality,5)
tail(airquality,5)
summary(airquality)    # The summary method displays descriptive statistics for every variable
na.omit(airquality)
```

- The graphics package is used for plotting base graphs like scatter plot, box plot, etc. A complete list of functions with help pages: library(help = "graphics")

❑ Data Visualization – Basic Plots

- Scatter/Dot Plot: Each dot represents the value of the Ozone in mean parts per billion
- The ***plot***() function is a kind of a generic function for plotting of R objects.

  plot(airquality$Ozone)

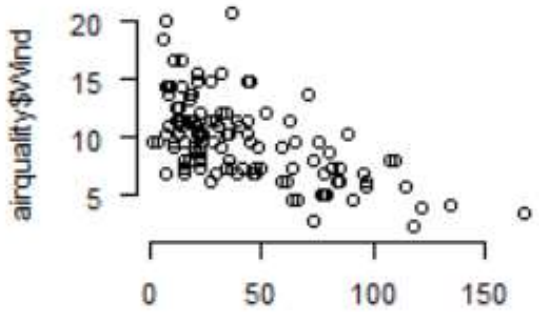


❑ Data Visualization – Basic Plots

- Let us now plot a graph between the Ozone and Wind values to study the relationship between the two

  plot(airquality$Ozone, airquality$Wind)

- The plot shows that Wind and Ozone values have a somewhat negative correlation.



❑ Data Visualization – Basic Plots

- What happens when we use plot command with the entire dataset without selecting any particular columns?

  plot(airquality)

- We get a matrix of scatterplots which is a correlation matrix of all the columns.

  The plot above instantly shows that:

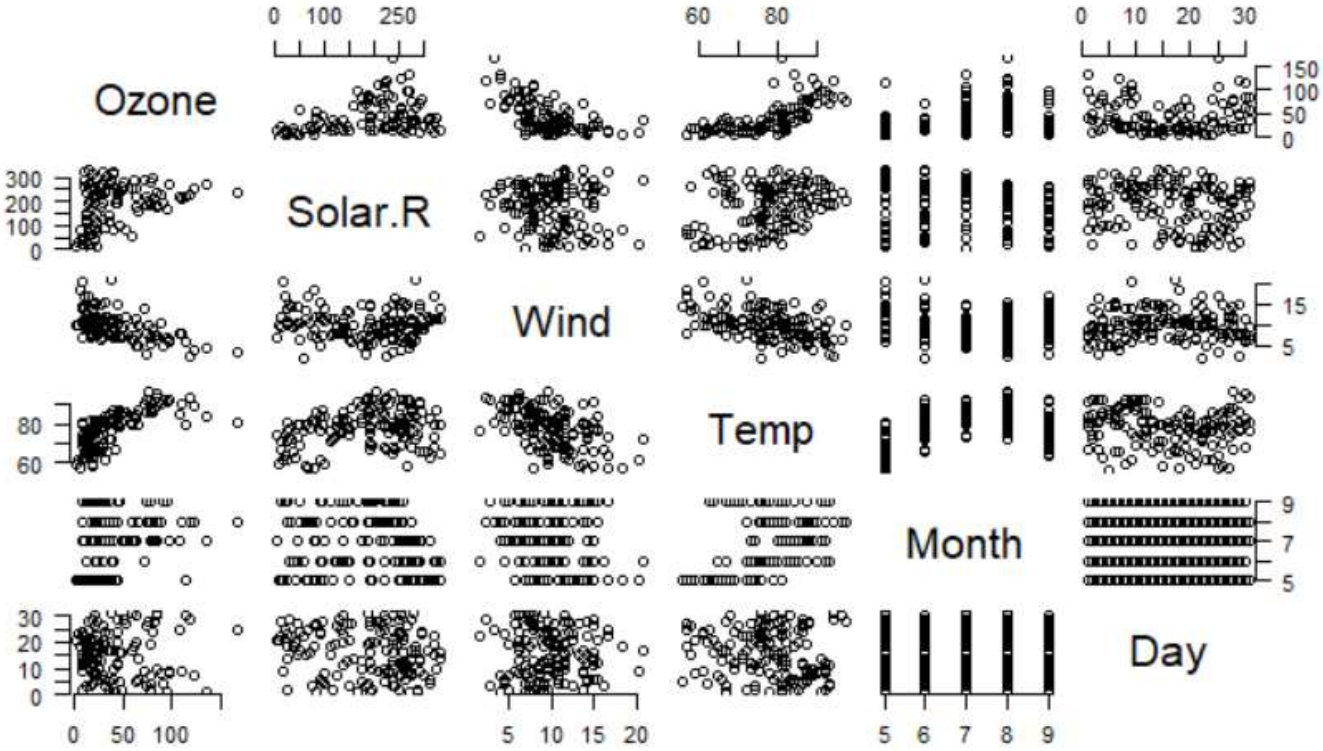
  - The level of Ozone and Temperature is correlated positively.

  - Wind speed is negatively correlated to both Temperature and Ozone level.

- We can quickly discover the relationship between variables by merely looking at the plots drawn between them

  - See the next page for the output of plot(airquality)

❑ Data Visualization – Basic Plots



❑ Data Visualization – Basic Plots

- ***plot(x, y, ...)*** Arguments:

x - the coordinates of points in the plot. Alternatively, a single plotting structure, function or any R object with a plot method can be provided.

y - the y coordinates of points in the plot, optional if x is an appropriate structure

Arguments to be passed to methods, such as graphical parameters (see par). Many methods will accept the following arguments:

Type - what type of plot should be drawn. Possible types are "p" for points, "l" for lines, "b" for both, "h" for 'histogram' like (or 'high-density') vertical lines, etc.

main - an overall title for the plot: see title

Sub - a subtitle for the plot: see title

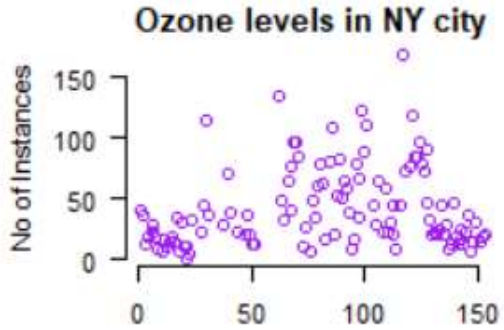
xlab - a title for the x axis: see title

ylab - a title for the y axis: see title

❑ Data Visualization – Basic Plots

- Read more about the ***plot()*** command by typing ***?plot()*** in the console
- Labels and Titles: We can also label the X and the Y axis and give a title to our plot.
- Additionally, we also have the option of giving color to the plot.

plot*(airquality$Ozone, ylab = 'No of Instances', main = 'Ozone levels in NY city', col = 'purple')*



❑ Data Visualization – Basic Plots

- **Barplot -** data is represented in the form of rectangular bars and the length of the bar is proportional to the value of the variable or column in the dataset.

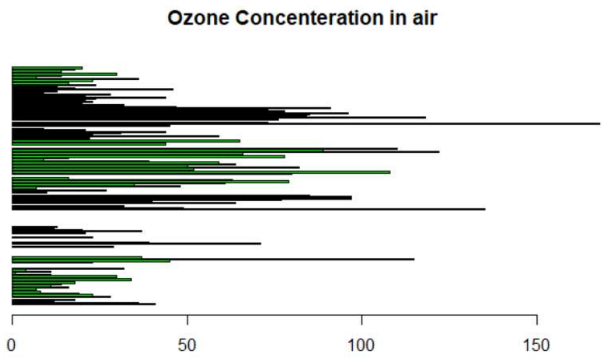
  - Both horizontal, as well as a vertical bar chart, can be generated by tweaking the horiz parameter.

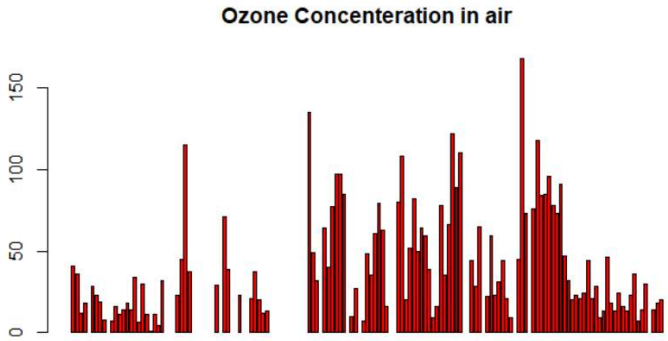
- Horizontal bar plot: barplot(airquality$Ozone, main = 'Ozone Concenteration in air',

  xlab = 'ozone levels', col= 'green',horiz = TRUE)

- Vertical bar plot: barplot(airquality$Ozone, main = 'Ozone Concenteration in air',

  xlab = 'ozone levels', col='red',horiz = FALSE)





❑ Data Visualization – Basic Plots

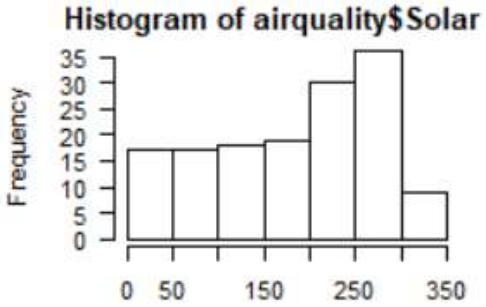
- **Histogram** is quite similar to a bar chart except that it groups values into continuous ranges. A histogram represents the frequencies of values of a variable bucketed into ranges.

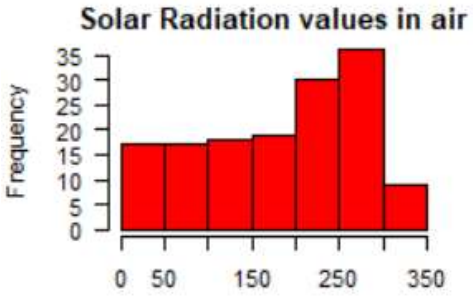
  hist(airquality$Solar.R)

- We get a histogram of the Solar.R values. By giving an appropriate value for the color argument, we can obtain a colored histogram as well.

  hist(airquality$Solar.R, main = 'Solar Radiation values in air',

  xlab = 'Solar rad.', col='red')





❑ Data Visualization – Basic Plots

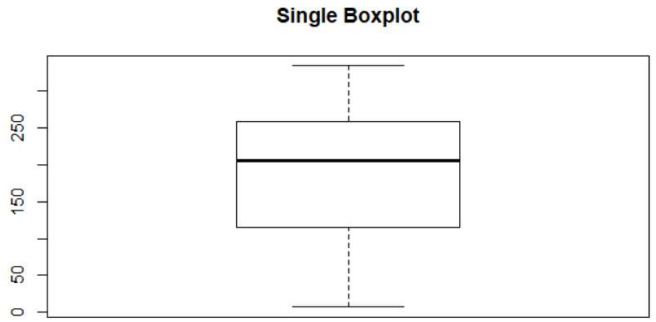
- **Boxplot:** We have seen how the ***summary()*** function in R can display the descriptive statistics for every variable in the dataset. Boxplot does the same albeit graphically in the form of quartiles. It is again very straightforward to plot a boxplot in R.

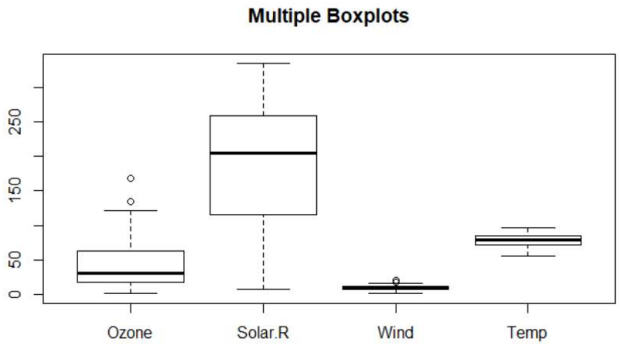
  - Single box plot

    boxplot(airquality$Solar.R, main='Single Boxplot')

  - Multiple box plots

    boxplot(airquality[,0:4], main='Multiple Boxplots')





❑ Data Visualization – Basic Plots

- **Grid of Charts**: There is a very interesting feature in R which enables us to plot multiple charts at once. This comes in very handy during the EDA since the need to plot multiple graphs one by one is eliminated.
- ***par()*** function: We can use it to put multiple graphs in a single plot by setting some graphical parameters. R programming has a lot of graphical parameters which control the way our graphs are displayed.

  You will see a long list of parameters and know what each does you can check the help section ***?par***.

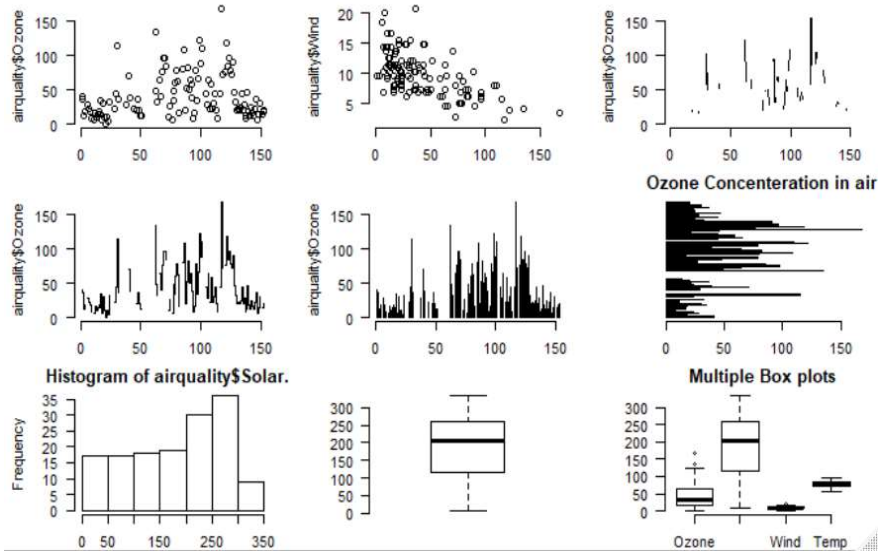
  It takes in a vector of form c(m, n) which divides the given plot into m*n array of subplots.

  For example, if we need to plot two graphs side by side, we would have m=1 and n=2. Following example illustrates this.

❑ Data Visualization – Basic Plots

- Example

```
par(mfrow=c(3,3), mar=c(2,5,2,1), las=1, bty="n")
plot(airquality$Ozone)
plot(airquality$Ozone, airquality$Wind)
plot(airquality$Ozone, type= "c")
plot(airquality$Ozone, type= "s")
plot(airquality$Ozone, type= "h")
barplot(airquality$Ozone, main = 'Ozone Concenteration in air',
  xlab = 'ozone levels', col='green',horiz = TRUE)
hist(airquality$Solar.R)
boxplot(airquality$Solar.R)
boxplot(airquality[,0:4], main='Multiple Box plots')
```



❑ Data Visualization

- We are going to introduce Data Visualization in the following four topics:
  - ✓ Introduction
  - ✓ Basic Plots
  - ✓ Visualization Libraries in R
  - ✓ Visualizing Geographical data

❑ Data Visualization – Visualization Libraries in R

- **Visualization Libraries in R**

  - R comes equipped with sophisticated visualization libraries having great capabilities. Let us have a closer look at some of the commonly used ones.
  - We will mainly introduce **Lattice Package** and **ggplot2 Package** in this class
  - We will use the built-in mtcars dataset to show the uses of the various libraries. This dataset has been extracted from the 1974 Motor Trend US magazine.

❑ Data Visualization – lattice Package in R

- The **Lattice Package** is essentially an improvement upon the R Graphics package and is used to visualize multivariate data. Lattice enables the use of trellis graphs. Trellis graphs exhibit the relationship between variables which are dependent on one or more variables. Let us start by installing and loading the package. It is very straightforward to use the lattice library. One simply needs to plug in the columns for which the plot is desired.

Installing & Loading the package: ***install.packages***("lattice")

Loading the dataset: ***library***(lattice); ***attach***(mtcars) – good option only when you work on one dataset

Exploring the dataset: ***head***(mtcars)

```
> 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
```

❑ Data Visualization – lattice Package in R

- Before proceeding with the working of the lattice package, let us do a little pre-processing of the data. There are two columns in our mtcars dataset namely gear and cyl which are categorical in nature. We need to factorize them to make them more meaningful.

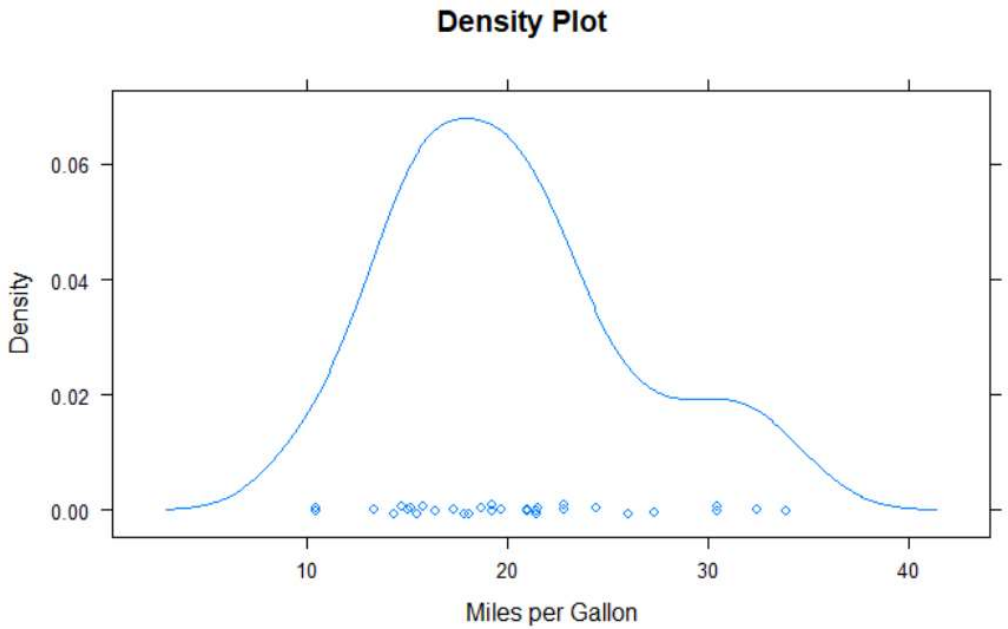
  gear_factor<-***factor***(gear,levels=c(3,4,5), labels=c("3gears","4gears","5gears"))

  cyl_factor <-***factor***(cyl,levels=c(4,6,8), labels=c("4cyl","6cyl","8cyl"))

  Now let us see how we can use the lattice package to create some basic plots in R in the next three slides.

❑ Data Visualization – lattice Package in R

- **Kernel Density Plots:** ***densityplot***(~mpg, main="Density Plot", xlab="Miles per Gallon")

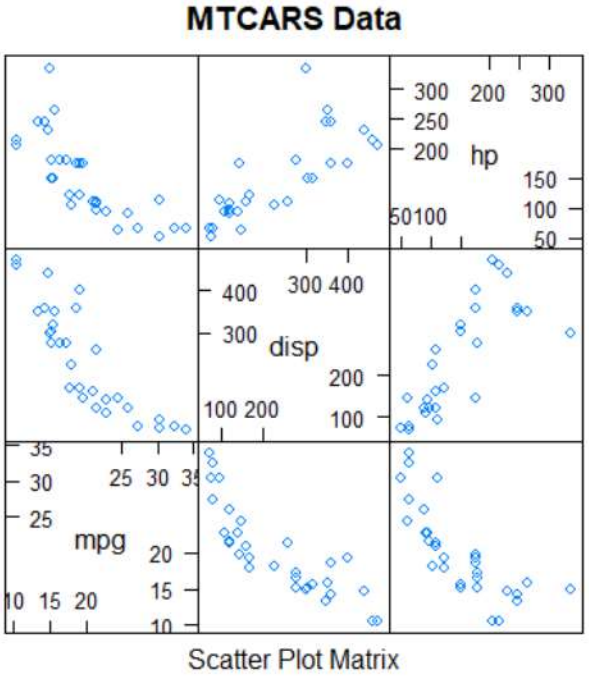


Note: A density plot is a representation of the distribution of a numeric variable. It uses a kernel density estimate to show the probability density function of the variable. It is a smoothed version of the histogram and it is used in the same concept

❑ Data Visualization – lattice Package in R

- **Scatterplot Matrix:** ***splom***(mtcars[c(1,3,4)], main="MTCARS Data")

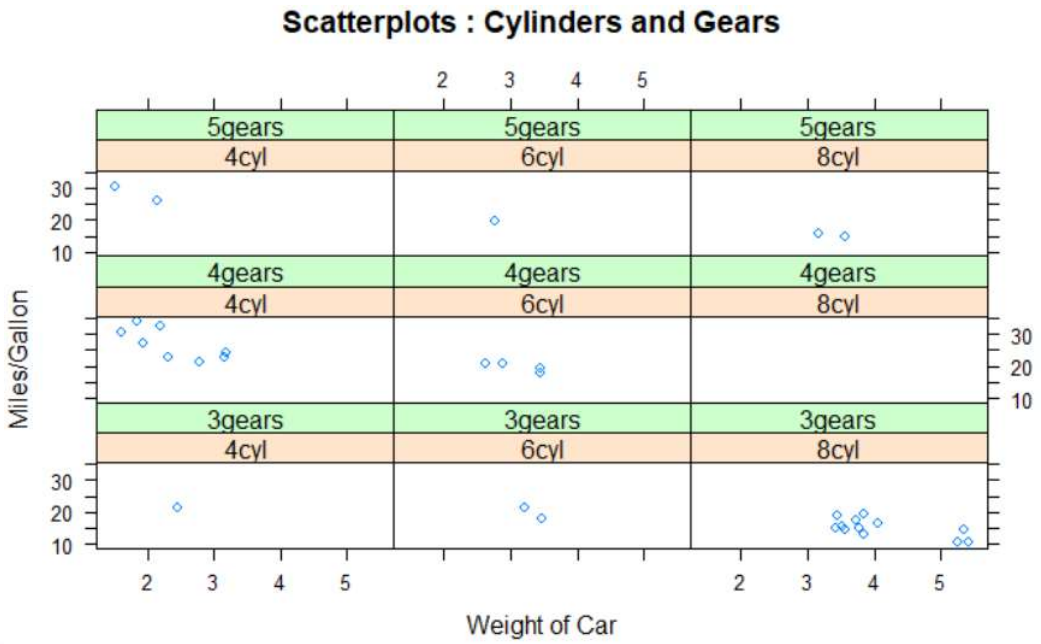
  Exercise: ***splom***(mtcars[c(1,3,5,6)], main="MTCARS Data")



❑ Data Visualization – lattice Package in R

- **The Scatterplots Depicting a Combination of Two Factors**

xyplot(mpg~wt | cyl_factor*gear_factor, main="Scatterplots : Cylinders and Gears",
ylab="Miles/Gallon", xlab="Weight of Car")



❑ Data Visualization – ggplot2 Package in R

- The ***ggplot2*** Package is one of **the most widely used visualization packages in R**. It enables the users to create sophisticated visualizations with little code using the Grammar of Graphics. The Grammar of Graphics is a general scheme for data visualization which breaks up graphs into semantic components such as scales and layers
- The popularity of ggplot2 has increased tremendously in recent years since it makes it possible to create graphs that contain both univariate and multivariate data in a very simple manner

  # Installing & Loading the package: ***install.packages***('ggplot2'); ***library(***ggplot2)

  # Loading the dataset: ***attach***(mtcars)

  # create factors with value labels

  ```
  mtcars$gear <- factor(mtcars$gear,levels=c(3,4,5), labels=c("3gears", "4gears", "5gears"))
  mtcars$am <- factor(mtcars$am,levels=c(0,1), labels=c("Automatic","Manual"))
  mtcars$cyl <- factor(mtcars$cyl,levels=c(4,6,8), labels=c("4cyl","6cyl","8cyl"))
  ```

- Let us create a few plots to understand the capability of ggplot2

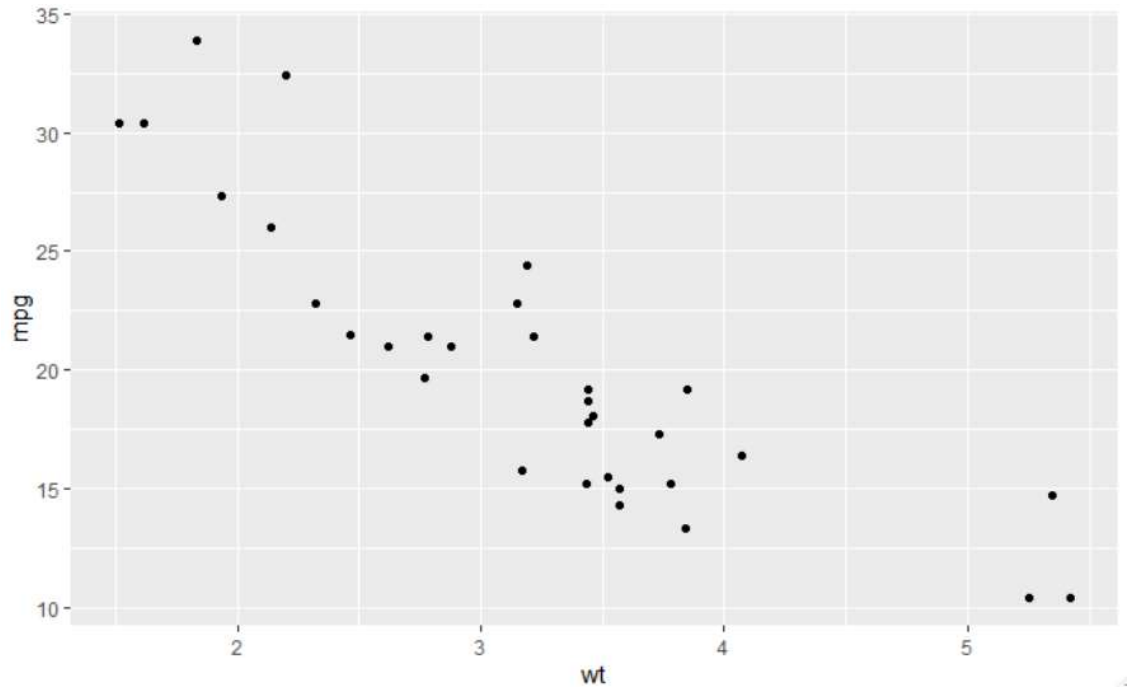
❑ Data Visualization – ggplot2 Package in R

- The ***ggplot2*** has a lot of options for plot types and the main idea is to add layers together:
  - Plot1 <- ggplot(data=mtcars, aes(x=mpg,y=hp) + geom_point() # Data, Aesthetics, and Geometries (Basics)
    - For geometries layer: adding binwidth, color, fill, transparency, xlab, ylab, ggtitle
    - For color, besides standard colors, hex color could be a nice option: https://www.color-hex.com/
  - Plot2 <- Plot1 + facet_grid(cyl~.) # Facets (adjust multiple plots in a single canvas)
  - Plot3 <- Plot2 + stat_smooth() # Statistics (adjust stats)
  - Plot4 <- Plot3 + coord_cartesian(xlim=c(15,30)) # Coordinates (adjust x and y limits)
  - Plot5 <- Plot4 + theme_bw() # Theme (adjust font size, color, grid, etc.)
    - Set theme for all the plots: theme_set(theme_minimal())
    - Add a theme to each individual plot: theme_dark() etc.
- Referring documents or ggplot2 cheat sheet for different types of plots and adding layers

❑ Data Visualization – ggplot2 Package in R

- **Scatter Plots:** ***ggplot***(*data* = mtcars, *mapping* = *aes*(x = wt, y = mpg)) + *geom_point*()

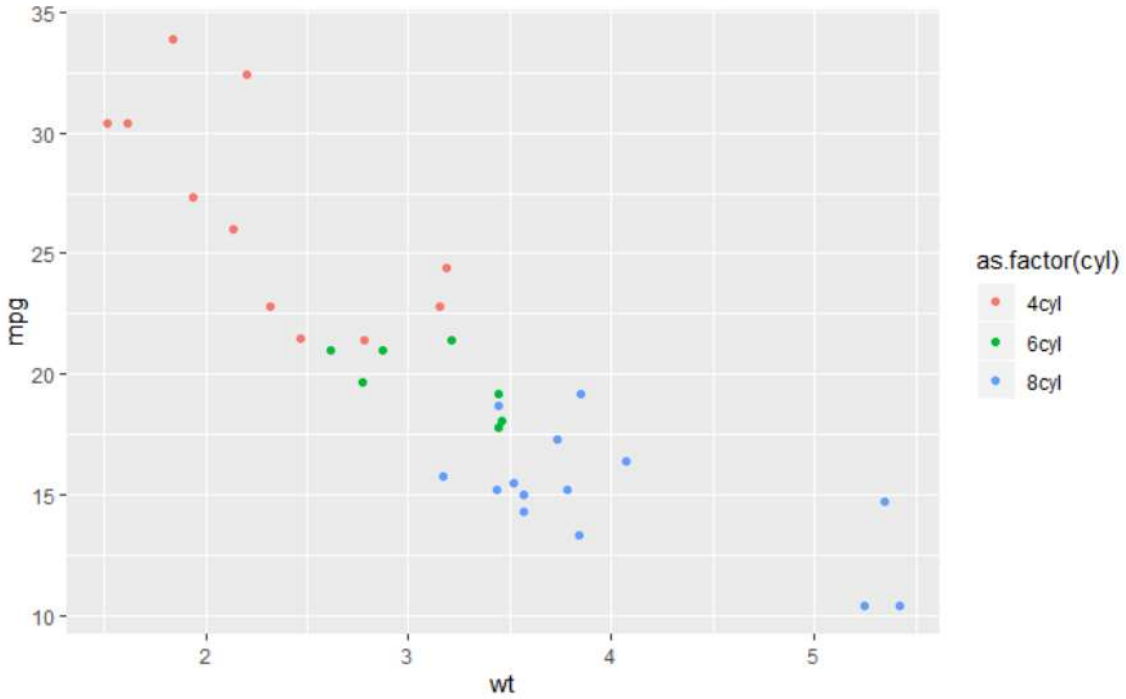
  - *geom_point*() is used to create scatterplots and geom can have many variations like geom_jitter(), geom_count(), etc.



❑ Data Visualization – ggplot2 Package in R

- **Styling scatter plots by factor:** We know that the dataset mtcars consists of certain variables which are in the form of factors. We can utilize this property to split our dataset

ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, color = as.factor(cyl))) + geom_point())

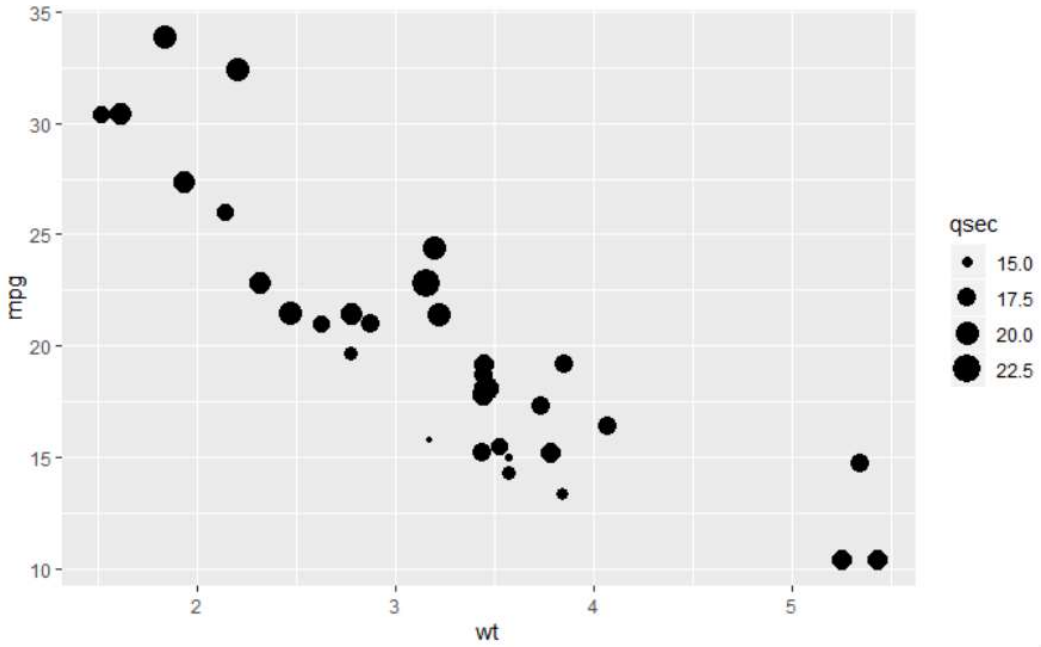


❑ Data Visualization – ggplot2 Package in R

- **Styling scatter plots by size:** Another useful feature of ggplot2 is that it can be styled according to the size of the attributes.

  ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, size = qsec)) + geom_point()

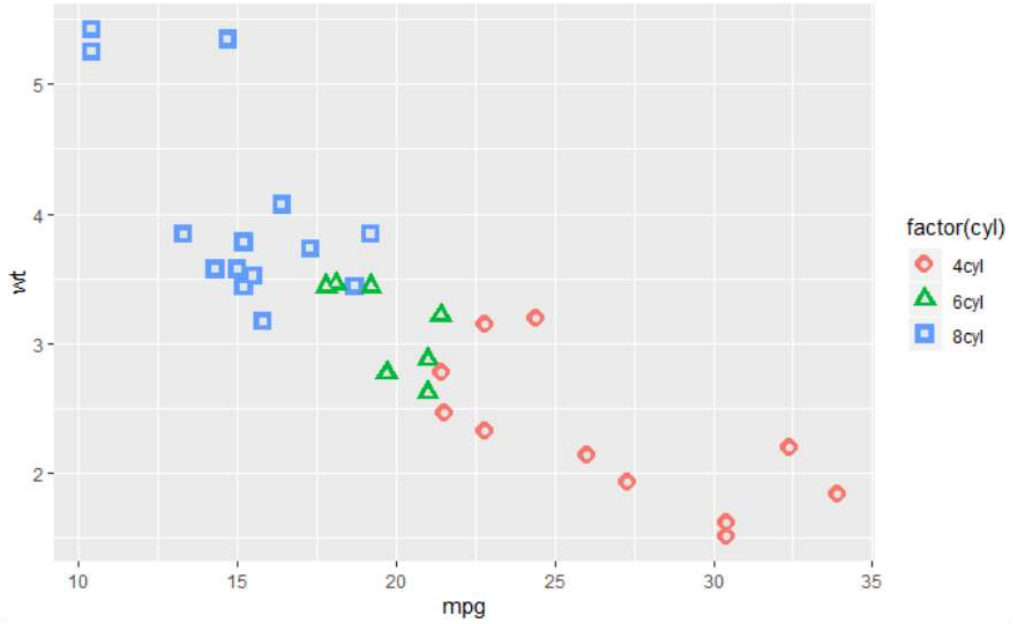
  - In the above example, the value of qsec indicates the acceleration which decides the size of the points



❑ Data Visualization – ggplot2 Package in R

- **Different symbols for different sizes:** With ggplot2, one can also create unique and interesting shapes by layering multiple points of different sizes

```
ggplot(mtcars,aes(mpg, wt, shape = factor(cyl))) + geom_point(aes(colour = factor(cyl)), size = 4) +
        geom_point(colour = "grey90", size = 1.5)
```



❑ Data Visualization

- We are going to introduce Data Visualization in the following four topics:
  - ✓ Introduction
  - ✓ Basic Plots
  - ✓ Visualization Libraries in R
  - ✓ Visualizing Geographical data in R

❑ Visualizing Geographical Data in R

- **Visualizing Geographical Data in R**: Geographic data (Geo data) relates to the location-based data. It primarily deals with describing objects with respect to their relationship in space. The data is usually stored in the form of coordinates. It makes more sense to be able to see a state or a country in the form of a map as it gives a more realistic overview.
- Geographical maps: We will be working with a sample superstore dataset of the ABC company. The dataset consists of locations of their stores in the US. Let's load in the data and check out its columns.

```
getwd()
setwd("C:/TeachingUChicago/Autumn2020/Data")
library(readr)
ABC <- read_csv('ABCLocationsNew.csv')
```

❑ Visualizing Geographical Data in R

- **Visualizing Geographical Data in R**: Geographic data (Geo data) relates to the location-based data. It primarily deals with describing objects with respect to their relationship in space. The data is usually stored in the form of coordinates. It makes more sense to be able to see a state or a country in the form of a map as it gives a more realistic overview.
- **Geographical Maps**: We will be working with a sample superstore dataset of the ABC company. The dataset consists of locations of their stores in the US. Let's load in the data and check out its columns.

  head(ABC)

	Address	City	State	`Zip Code`	Latitude	Longitude
	<chr>	<chr>	<chr>	<chr>	<dbl>	<dbl>
1	1205 N. Memorial Parkway	Huntsville	Alabama	35801-5930	34.7	-86.6
2	3650 Galleria Circle	Hoover	Alabama	35244-2346	33.4	-86.8
3	8251 Eastchase Parkway	Montgomery	Alabama	36117	32.4	-86.2
4	5225 Commercial Boulevard	Juneau	Alaska	99801-7210	58.4	-134.
5	330 West Dimond Blvd	Anchorage	Alaska	99515-1950	61.1	-150.
6	4125 DeBarr Road	Anchorage	Alaska	99508-3115	61.2	-150.

❑ Visualizing Geographical Data in R

- ***plot()*** function: We will create a crude map by simply the Latitude and the Longitude column.

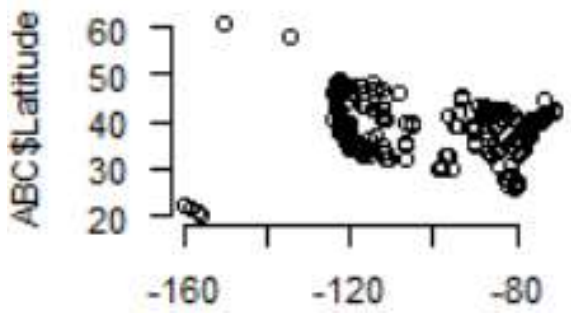
  plot(ABC$Longitude,ABC$Latitude)

- ***map()*** function: The output isn't an exact map but it does give a faint outline of the US boundary.

  maps package is very useful and pretty straightforward when it comes to plotting the geographical data.

  install.packages('maps', dependencies=TRUE)

  library(maps)



❑ Visualizing Geographical Data in R

- Using the map() function to plot a base map of the US

  map(database="state")

- Building a point map on top of the base map using symbols() function

  symbols(ABC$Longitude, ABC$Latitude, squares =rep(1, length(ABC$Longitude)), inches=0.03, add=TRUE)

- Giving the symbols a color

  symbols(ABC$Longitude, ABC$Latitude,bg = 'red', fg = 'red', squares =rep(1, length(ABC$Longitude)),
  inches=0.03, add=TRUE)

Week 5 Class Agenda

- Functions
- Data Visualization
- Memory Management

❑ Memory Management in R

- **Memory Management in R**

  - Objective: Learn how to manage memory usage by R

  - An Overview of Memory Management in R - Why is this sometimes a problem?

  - Simply put, R is not very efficient in its use of memory. Although this inefficiency occurs on both x32- and x64-bit CPUs, it is really only of concern in older x32-bit CPUs with <4GB RAM. Newer x64-bit with larger RAM rarely encounter processing that exceeds the amount of available RAM on a personal CPU.

  - Nonetheless, you will, on occasion, be running some sort of R code and suddenly the error message:

    Cannot allocate vector of size……

    appears, **indicating a possible memory shortage issue**. In short, R is telling you it cannot find sufficient memory to complete the analysis.

❑ Memory Management in R

- **Determining Your Memory Limits in R on Windows:**
- RAM is capped at ~3.5GB in x32 Windows systems, and at the RAM installed in x64 Windows (W7/W8/W10) / MAC OS / Linux-build CPUs. Two calls, memory.limit() and memory.size() return the amount of RAM in your CPU, and how much is being used by your current R session, respectively.
- ***memory.size()*** will grow as your R session progresses
- Memory management functions

```
> memory.limit()  # how much do you have?
[1] 1.759219e+13
>
> memory.size()  # how much is being used?
[1] 148.07
```

Notes: The first return (in kb) indicates how much available in your office CPU, while the second shows usage. Note that not all RAM is truly available for R; your CPU OS uses a substantial portion of RAM as well for basic background OS operations, as well as other programs you may be running

❑ Memory Management in R

- **Determining Your Memory Limits in R on Mac:**
- Monitoring Memory Usage: To check on the (approximate) size of a specific object, use the function ***object.size***:

```
> object.size(1) 3

2 bytes

> object.size("Hello world!")

72 bytes

> object.size(audioscrobbler)

39374504 bytes
```

- The function ***memory.profile*** displays information on memory usage by object type:

```
> memory.profile()
```

❑ Memory Management in R

- **Managing Memory Limits in R**
- Your options for dealing with RAM revolve around two basic calls that clear your workspace, rm() and gc(). rm(NameofObject) deletes objects - permanently. Once they are removed they cannot be recovered except by re-running the code that created the object in the first place. You forget this piece of advice at your own peril. Multiple objects can be removed by separating them with a "," (comma).

```
# cleaning up workspace; start by creating some simple objects
x <- 1    # object x; x is assigned value of 1
y <- 2    # object y; y is assigned value of 2
ls()      # what's in workspace?
rm(x)     # rm object x
ls()      # what's in workspace now?
rm(y)     # rm objects y
ls()      # what's in workspace now?
```

❑ Memory Management in R

- ***gc*() Garbage Collection**: releases memory, freeing it up to used once again. This is in contrast to ***rm***(), which merely removes the object but does not release the memory to which the object was attached. These commands are also applicable to CPUs with larger available RAM.
- ***gc***() # garbage clean-up - frees all memory

```
> memory.size()     # how much memory used now?
[1] 1291.65
> gc()              # clean up memory
         used (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells 1456778 77.9    2653274  141.8   1964512  105.0
Vcells 3763276 28.8  149287341 1139.0 153763801 1173.2
> memory.size()     # how much memory now?
[1] 147.15
```

- Systematic memory issues (i.e., x32 RAM) more difficult to resolve. Package ***bigmemory*** and those it references can be useful for memory management x32 CPUs

❑ Memory Management in R

- Exercise: Determine the memory size and limits on your personal CPU. Run the code below line by line - accept as is, not what it does or how it works - and see how memory usage changes

```
# observe how memory changes (R codes on Windows)
memory.size()          # current memory occupied
a1 <- c(1:1e+08)       # take lots of memory w/object
a2 <- a1 * 2           # take even more ...
memory.size()          # how much memory used now?
rm(a1, a2)             # rm objects a1,a2
memory.size()          # how much memory used now?
gc()                   # clean up memory
memory.size()          # how much memory now?
```

Q&A & Thank You

THE UNIVERSITY OF
CHICAGO

Contact Information:

Your comments and questions are valued and encouraged.

Mei Najim

E-mail: mnajim@uchicago.edu

LinkedIn: https://www.linkedin.com/in/meinajim/