DEPARTMENT FOR CONTINUING EDUCATION

UNIVERSITY OF OXFORD

Data Science Tools: Intro to R

3-March-2020

Sepi Chakaveh & Assel Altayeva

Overview Data Science

Plan

Python or R

R essentials

- Installing R
- Navigating packages
- Entering data
- Data types and data structures

Importing datasets and Data preparation in R

Data Visualization with GGPLOT2 package

Statistical applications – beyond summaries and basic graphs towards data exploration and modelling

- SPSS
- SAS
- jamovi

Data Oriented Programming languages –towards analyzing data to address the questions that matter to you

- Python
- Julia
- R

Python or R

R

R is one of the oldest programming language developed by academics and statisticians. R comes into existence in the year 1995. Now R is providing the richest ecosystem for data analysis.

Python

On the other hand Python can do the same tasks as R programming language does. The major features of python are data wrangling, engineering, web scraping and so on. Python is also having the tools that help in implementing the machine learning at large scale.

Python or R

R or Python Usage

Python has developed by Guido van Rossum in 1991. Python is the most popular programming language in the world. It has most powerful libraries for math, statistic, artificial intelligence and machine learning. But still python is not useful for econometrics and communication, and also for business analytics.

Python or R

R is more functional, Python is more object-oriented

R is more functional, it provides variety of functions to the data scientist i.e lm, predict and so on. Most of the work done by functions in R. On the other hand Python use classes to perform any task within the python.

R has more data analysis built-in, Python relies on packages.

R provides the build in data analysis for summary statistics, it is supported by summary built-in functions in R. But on the other hand we have to import the statsmodel packages in Python to use this function. In addition there is also a built in constructor in R i.e is dataframe. On the other hand we have to import it in Python.

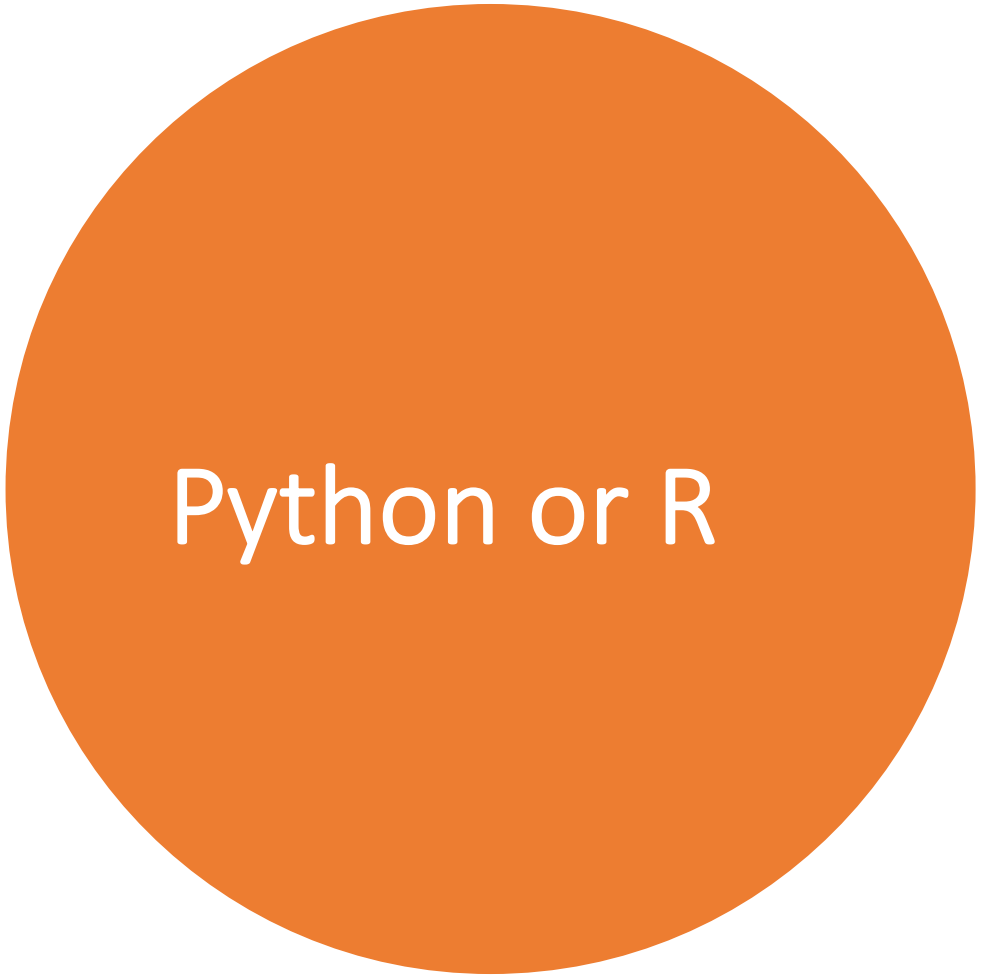
Python or R

R has more statistical support in general.

R was created as a statistical language, and it shows . statsmodels in Python and other packages provide decent coverage for statistical methods, but the R ecosystem is far more large.

It’s usually more straightforward to do non-statistical tasks in Python.

With well-placed libraries like beautifulsoup and request, web scraping in Python is much easier than R. This applies to other tasks that we don’t see closely, such as saving the database, deploying the Web server, or running a complex selfie.



There are many parallels between the data analysis workflow in both.

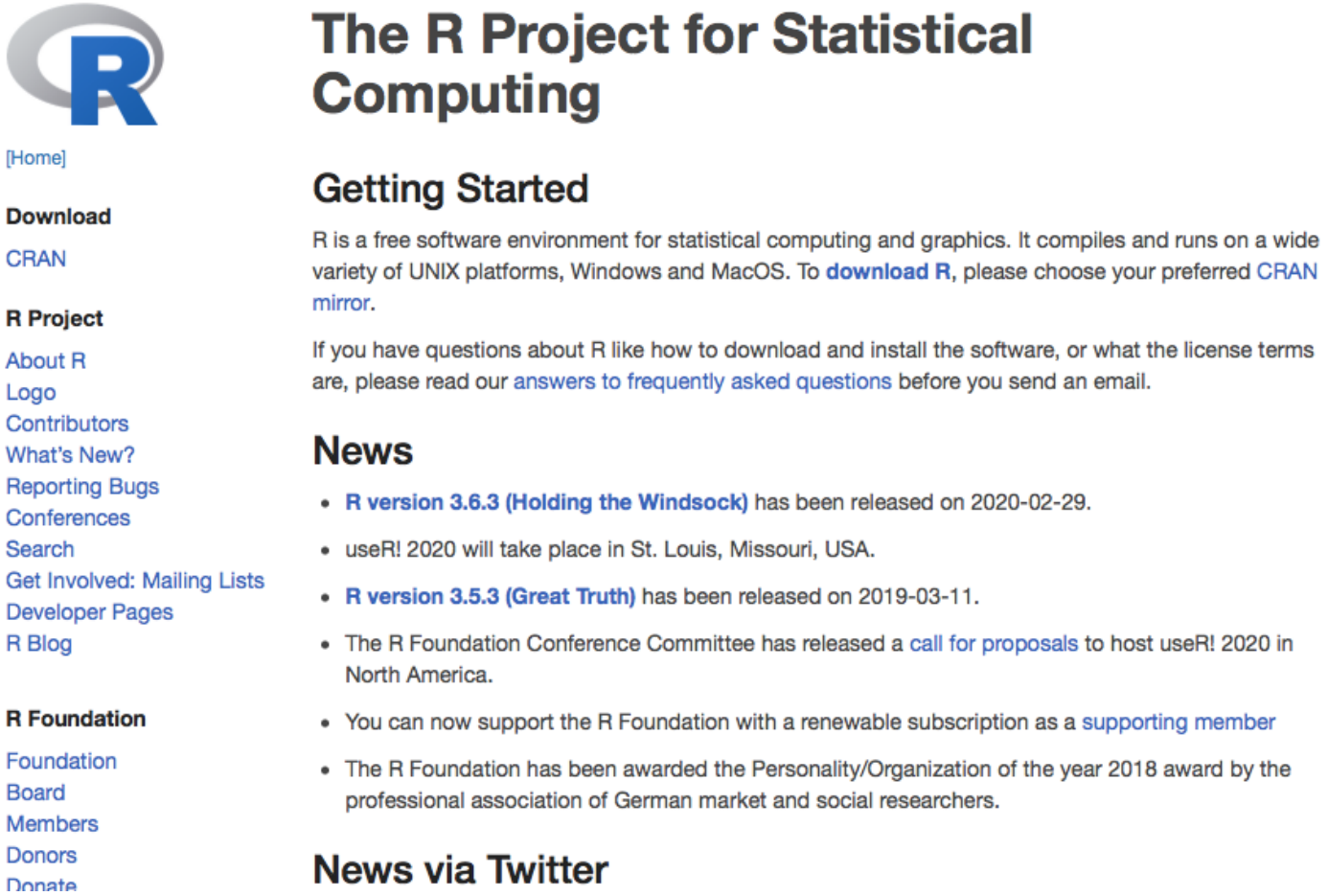
R and Python are the clearest points of inspiration between the two (pandas were inspired by the Dataframe R Dataframe, the rvest package was inspired by the Sundersaute), and the two ecosystems are getting stronger. It may be noted that the syntax and approach for many common tasks in both languages are the same.

Why R?

R is the most preferred programming tool for statisticians, data scientists, data analysts and data architects

R has over 10,000 packages (a lot of available algorithms) from multiple repositories.

Installing R



[Home]

Download

CRAN

R Project

About R
Logo
Contributors
What's New?
Reporting Bugs
Conferences
Search
Get Involved: Mailing Lists
Developer Pages
R Blog

R Foundation

Foundation
Board
Members
Donors
Donate

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

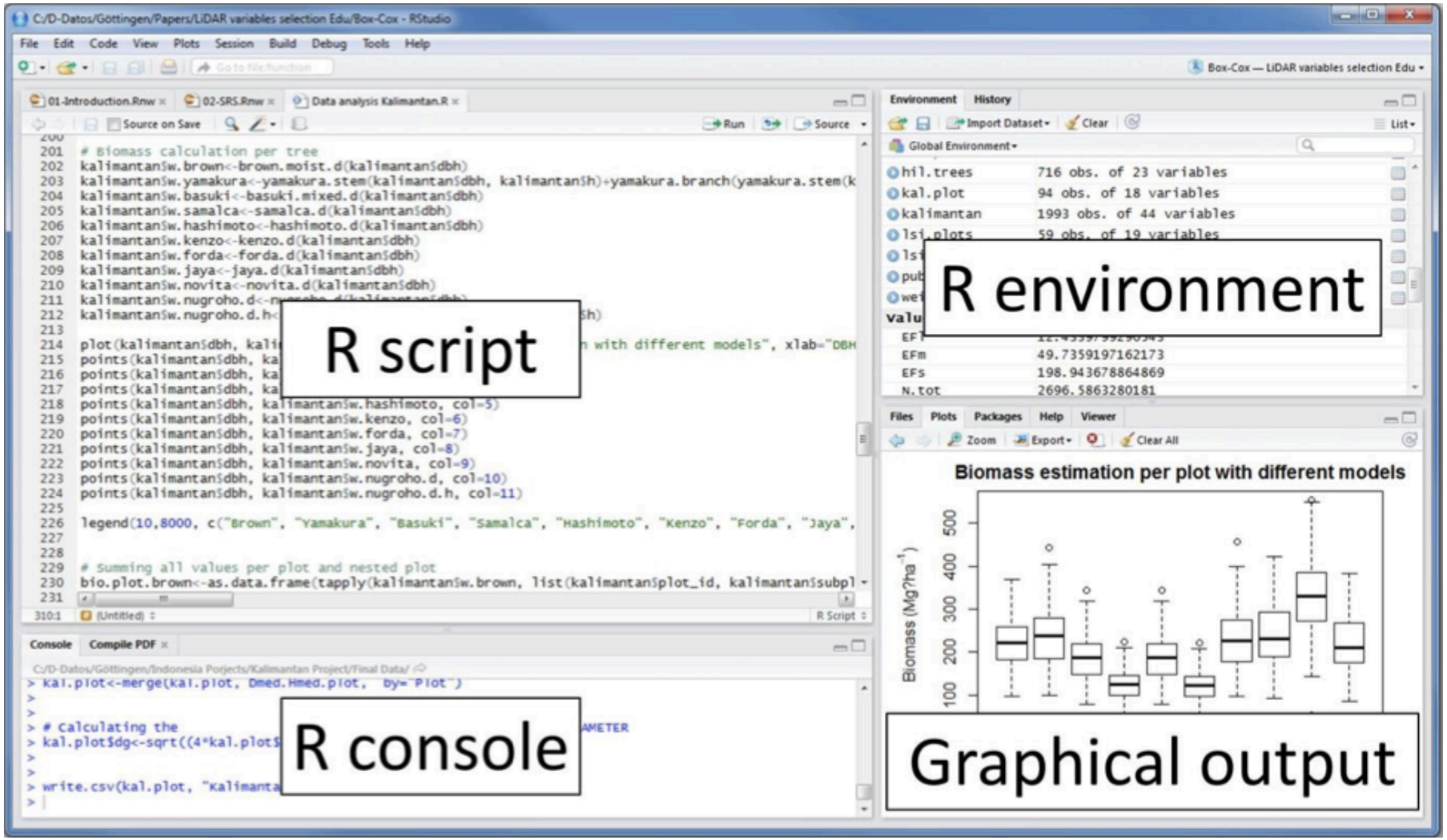
News

- **R version 3.6.3 (Holding the Windsock)** has been released on 2020-02-29.
- useR! 2020 will take place in St. Louis, Missouri, USA.
- **R version 3.5.3 (Great Truth)** has been released on 2019-03-11.
- The R Foundation Conference Committee has released a call for proposals to host useR! 2020 in North America.
- You can now support the R Foundation with a renewable subscription as a supporting member
- The R Foundation has been awarded the Personality/Organization of the year 2018 award by the professional association of German market and social researchers.

News via Twitter

- Go to -> https://www.r-project.org
- Click download R
- Environments:
- https://notebooks.azure.com
- https://www.anaconda.com/distribution/
- https://rstudio.cloud/ or https://rstudio.com/products/rstudio/download/#download

R &R studio IDE



Navigating R

 # INSTALL AND LOAD PACKAGES ################

Load base packages manually

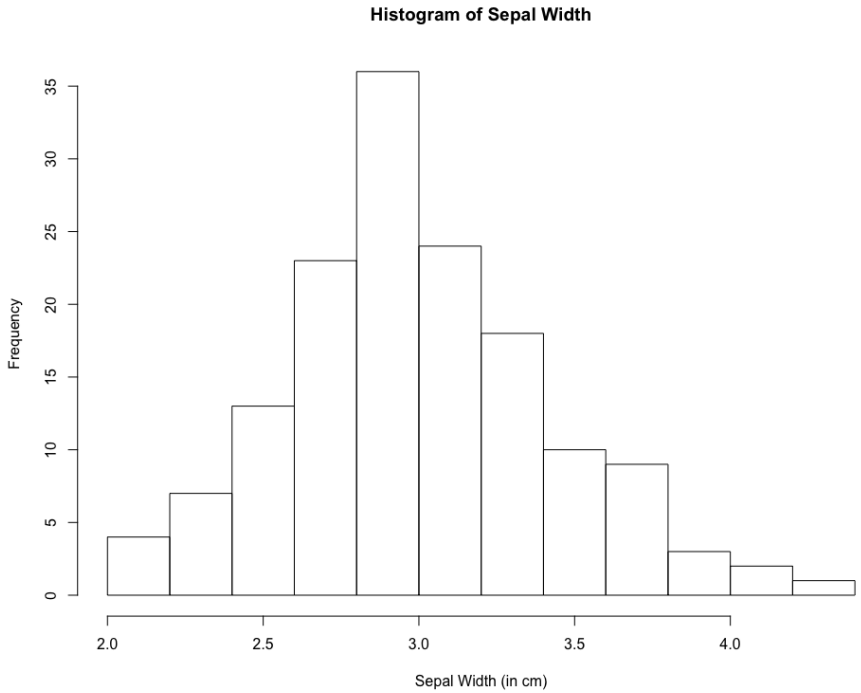
```
library(datasets)                    # For example datasets
```

 # LOAD AND PREPARE DATA ####################

```
?iris
df <- iris
head(df)
```

 # ANALYZE DATA ##########################

```
hist(df$Sepal.Width,
     main = "Histogram of Sepal Width",
     xlab = "Sepal Width (in cm)")
```



Navigating R

```
# CLEAN UP ##########################################

# Clear environment
   rm(list = ls())
# Clear packages
  detach("package:datasets", unload = TRUE)  # For base

# Clear plots
  dev.off()                                  # But only if there IS a plot
# Clear console
  cat("\014")                                # ctrl+L
```

Entering Data tutorial link

```
# BASIC COMMANDS #########################################

2+2                     # Basic math; press cmd/ctrl enter
1:100                   # Prints numbers 1 to 100 across several lines
print("Hello World!")   # Prints "Hello World" in console

# ASSIGNING VALUES #######################################
# Individual values
a <- 1                  # Use <- and not =
2 -> b                  # Can go other way, but silly
c <- d <- e <- 3                # Multiple assignments
# Multiple values
x <- c(1, 2, 5, 9)              # c = Combine/concatenate
x                               # Print contents of x in Console
```

Entering Data

```
# SEQUENCES ############################################
                                  # Create sequential data
0:10                               # 0 through 10
10:0                               # 10 through 0
seq(10)                            # 1 to 10
seq(30, 0, by = -3)                # Count down by 3

# MATH #################################################
                                  # Surround command with parentheses to also print
(y <- c(5, 1, 0, 10))
x + y                              # Adds corresponding elements in x and y
x * 2                              # Multiplies each element in x by 2
2^6                               # Powers/exponents
sqrt(64)                          # Squareroot
log(100)                 # Natural log: base e (2.71828...), base 10 would be log10
```

Data types tutorial link

 # Numeric (Real, Integers, Complex)##########

```
n1 <- 15                                    # Double precision by default
n1
typeof(n1)
```

#Character###################

```
c1 <- "c"
typeof(c1)
c2 <- "a string of text"
typeof(c2)
```

#Logical (True / False)################

```
l1 <- TRUE
l1
typeof(l1)
```

Data Structures tutorial link

Vector: a vector contains object of same class

List: a special type of vector which contain elements of different data types

Matrix: A matrix is represented by set of rows and columns.

Data frame: Every column of a data frame acts like a list

Data Structures

Vector ======

```
v1 <- c(1, 2, 3, 4, 5)                    #vector of numbers
v1
is.vector(v1)
```

Matrix =======

```
m1 <- matrix(c(T, T, F, F, T, F), nrow = 2)
m1
m2 <- matrix(c("a", "b",   "c", "d"),
             nrow = 2,
             byrow = T)
m2
```

Data Structures

Array ======================

```
# Give data, then dimensions (rows, columns, tables)
a1 <- array(c( 1:24), c(4, 3, 2))
a1
```

List ======================

```
o1 <- c(1, 2, 3)
o2 <- c("a", "b", "c", "d")
o3 <- c(T, F, T, T, F)
list1 <- list(o1, o2, o3)
list1
list2 <- list(o1, o2, o3, list1)  # Lists within lists!
list2
```

Data Structures

```
## Data Frame ==========================================

                                    # Can combine vectors of the same length
vNumeric   <- c(1, 2, 3)
vCharacter <- c("a", "b", "c")
vLogical   <- c(T, F, T)

df1 <- cbind(vNumeric, vCharacter, vLogical)
df1                                 # Coerces all values to most basic data type

df2 <- as.data.frame(cbind(vNumeric, vCharacter, vLogical))
df2                           # Makes a data frame with three different data types
```

Coercing types

Automatic Coercion ==========

Goes to "least restrictive" data type

```
(coerce1 <- c(1, "b", TRUE))
typeof(coerce1)
```

Coerce Numeric to Integer ==========

```
(coerce2 <- 5)
typeof(coerce2)

(coerce3 <- as.integer(5))
typeof(coerce3)
```

Coercing types

Coerce Character to Numeric ==========

```
(coerce4 <- c("1", "2", "3"))
        typeof(coerce4)

(coerce5 <- as.numeric(c("1", "2", "3")))
        typeof(coerce5)
```

Coerce Matrix to Data Frame ==========

as_tibble turns an existing object, such as a data frame, list, or matrix, into a so-called tibble, a data frame with class tbl_df.

```
(coerce6 <- matrix(1:9, nrow= 3))
        is.matrix(coerce6)
(coerce7 <- as.data.frame(matrix(1:9, nrow= 3)))
        is.data.frame(coerce7)
```

R essentials

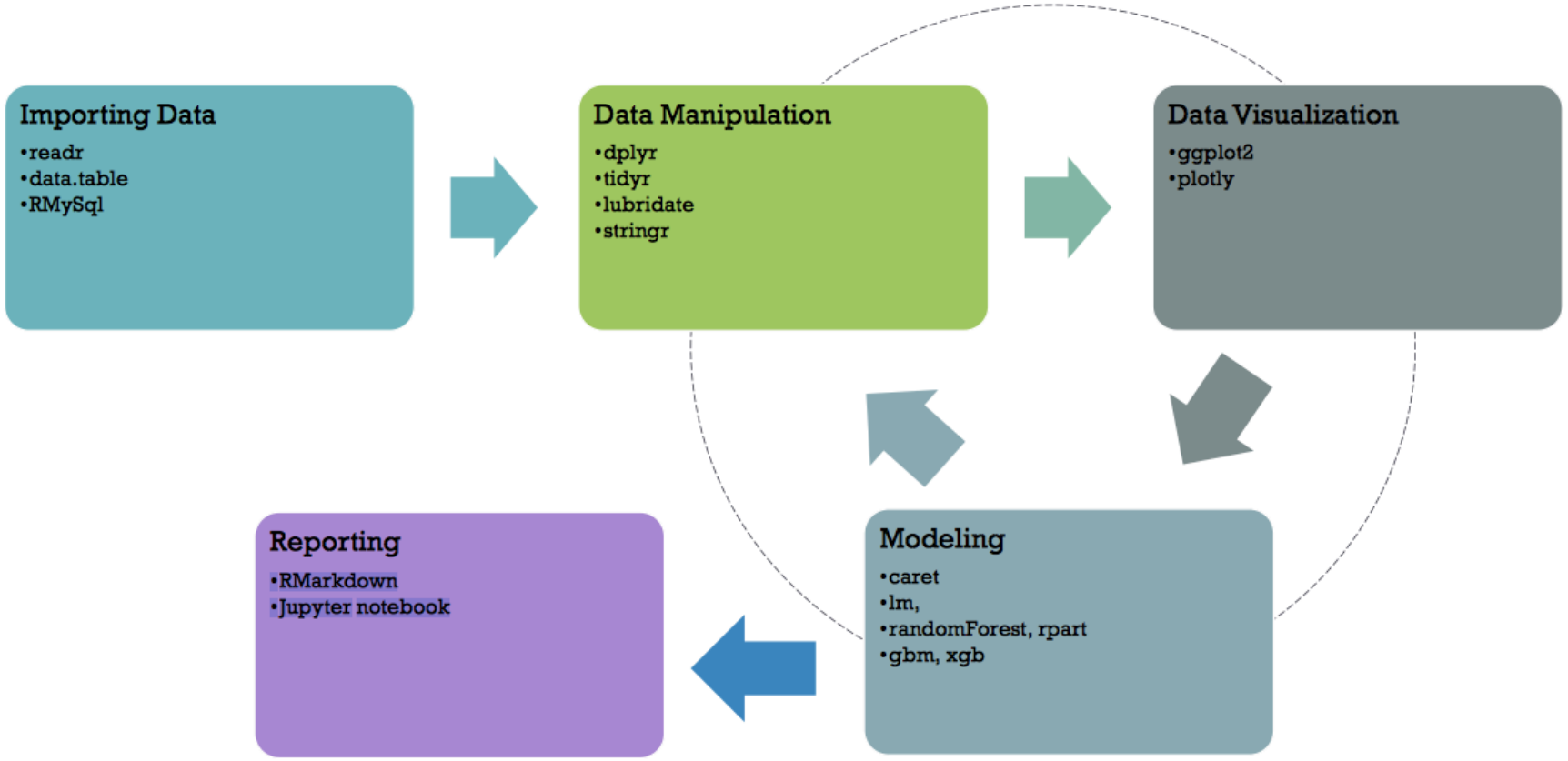
- **Control structures**
  - **If (condition){**
    Do something
    }else{
    Do something else
    }
- **Loop**
  - **For loop**
  - **While loop**
- **Function**
  - ***function.name* <- *function(arguments)* { computations on the arguments some other code }**

```
x <- runif(1, 0, 10)
if(x > 3) {
        y <- 10
} else {
        y <- 0
}
for(i in 1:10) {
        print(i)
}

mySquaredFunc<-function(n){
        # Compute the square of integer `n`
        n*n
}
mySquaredVal(5)
```

Useful R packages

Install packages: install.packages (“readr”, ”ggplot”, ”dplyr”, ”caret”)

Load packages: library(package_name)



Installing and loading packages

```
        # Install pacman ("package manager") if needed
if (!require("pacman")) install.packages("pacman")
        # Load contributed packages with pacman
pacman::p_load(pacman, party, psych, rio, tidyverse)
        # pacman: for loading/unloading packages
        # party: for decision trees
        # psych: for many statistical procedures
        # rio: for importing data
        # tidyverse: for so many reasons
                # Load base packages manually
library(datasets)  # For example datasets
```

Tidyverse

Tidyverse Packages Blog Learn Help Contribute

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

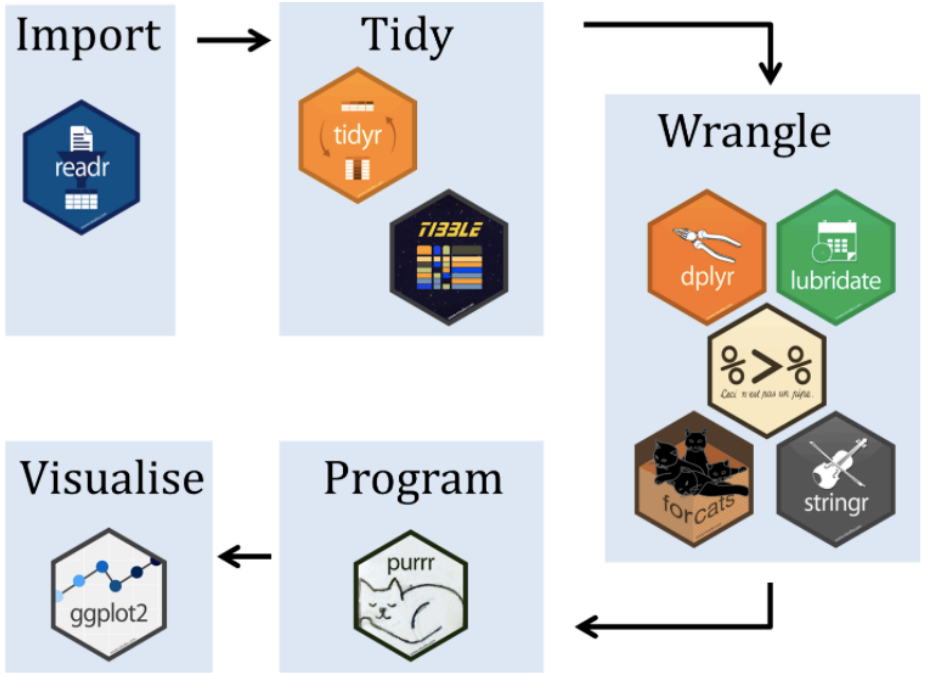
Install the complete tidyverse with:

```
install.packages("tidyverse")
```

https://www.tidyverse.org

Tidyverse

A collection of packages by Hadley Wickham for::



These packages are all designed to work nicely together

More readable by people than most R code

Journal of Statistical Software

August 2014, Volume 59, Issue 10. *http://www.jstatsoft.org/*

Tidy Data

Hadley Wickham
RStudio

Abstract

A huge amount of effort is spent cleaning data to get it ready for analysis, but there has been little research on how to make data cleaning as easy and effective as possible. This paper tackles a small, but important, component of data cleaning: data tidying. Tidy datasets are easy to manipulate, model and visualize, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table. This framework makes it easy to tidy messy datasets because only a small set of tools are needed to deal with a wide range of un-tidy datasets. This structure also makes it easier to develop tidy tools for data analysis, tools that both input and output tidy datasets. The advantages of a consistent data structure and matching tools are demonstrated with a case study free from mundane data manipulation chores.

Importing Data

- **CSV file**

```
mydata <- read.csv("mydata.csv")  # read csv file
library(readr)
mydata <- read_csv("mydata.csv")  # 10x faster
```

- **Tab-delimited text file**

```
mydata <- read.table("mydata.txt")  # read text file
 mydata <- read_table("mydata.txt")
```

- **Excel file:**

```
library(XLConnect)
wk <- loadWorkbook("mydata.xls")
df <- readWorksheet(wk, sheet="Sheet1")
```

- **SAS file**

```
library(sas7bdat)

mySASData <- read.sas7bdat("example.sas7bdat")
```

- **Other files:**
  - Minitab, SPSS(foreign),
  - MySQL (RMySQL)

Importing Data (tutorial link)

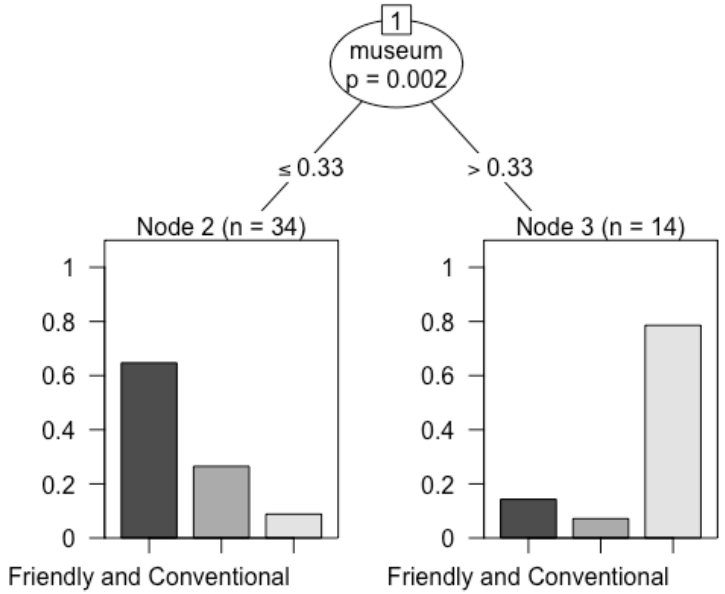
```
# Import CSV files with readr::read_csv() from tidyverse

(df <- read_csv("data/StateData.csv"))

# Import other formats with rio::import() from rio

(df <- import("data/StateData.xlsx") %>% as_tibble())
```

StateData dataset

State	state_code	region	governor	psychRegions	extraversion	agreeableness	conscientiousness	neurot
Alabama	AL	South	Republican	Friendly and Conventional	55.5	52.7	55.5	
Arizona	AZ	West	Republican	Relaxed and Creative	50.6	46.6	58.4	
Arkansas	AR	South	Republican	Friendly and Conventional	49.9	52.7	41	
California	CA	West	Democrat	Relaxed and Creative	51.4	49	43.2	
Colorado	CO	West	Democrat	Friendly and Conventional	45.3	47.5	58.8	
Connecticut	CT	Northeast	Democrat	Temperamental and Uninhibited	57.6	38.6	34.2	
Delaware	DE	South	Democrat	Temperamental and Uninhibited	47	38.8	36.5	
Florida	FL	South	Republican	Friendly and Conventional	60.9	50.7	62.7	
Georgia	GA	South	Republican	Friendly and Conventional	63.2	60	68.8	
Idaho	ID	West	Republican	Relaxed and Creative	40.7	52.9	44.5	
Illinois	IL	Midwest	Republican	Friendly and Conventional	62.5	48.3	50.9	
Indiana	IN	Midwest	Republican	Friendly and Conventional	48.9	50.2	56.2	
Iowa	IA	Midwest	Republican	Friendly and Conventional	62.8	56.6	52.2	

Data Analysis example using party::ctree

```
# df[, -1]) excludes the state_code
fit <- ctree(y ~ ., data = df[, -1])      # Create tree
fit %>% plot()                            # Plot tree
fit %>%                                   # Predicted vs true
  predict() %>%
  table(df$y)
```



Data manipulation with 'DPLYR'

- Some of the key "verbs":
  - **select**: return a subset of the columns of a data frame, using a flexible notation
  - **filter**: extract a subset of rows from a data frame based on logical conditions
  - **arrange**: reorder rows of a data frame
  - **rename**: rename variables in a data frame

Code example is from "R for Data Science" online book

Code link

```
library(nycflights13)
flights
select(flights, year, month, day)
select(flights, year:day)
select(flights, -(year:day))

jan1 <- filter(flights, month == 1, day == 1)
nov_dec <- filter(flights, month %in% c(11, 12))
filter(flights, !(arr_delay > 120 | dep_delay > 120))
filter(flights, arr_delay <= 120, dep_delay <= 120)

arrange(flights, year, month, day)
arrange(flights, desc(arr_delay))
rename(flights, tail_num = tailnum)
```

Data manipulation with 'DPLYR'

- **Some of the key "verbs":**
  - **mutate**: add new variables/columns or transform existing variables
  - **summarize**: generate summary statistics of different variables in the data frame
  - **%>%**: the "pipe" operator is used to connect multiple verb actions together into a pipeline

Code example is from "R for Data Science" online book

```
flights_sml <- select(flights,   year:day, ends_with("delay"),
                distance,   air_time )
mutate(flights_sml,   gain = arr_delay - dep_delay,
                speed = distance / air_time * 60 )
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
 count = n(),
 dist = mean(distance, na.rm = TRUE),
 delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delays, count > 20, dest != "HNL")
ggplot(data = delay, mapping = aes(x = dist, y = delay)) +
 geom_point(aes(size = count), alpha = 1/3) +
 geom_smooth(se = FALSE)
```

Piping Commands with %>%

- The pipe operator is key to why the tidyverse packages are so usable and readable
- It passes the thing on its left as the first argument to a function on its right
- x %>% f() is equivalent to f(x)

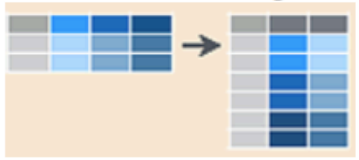
```
function(data)
data %>% function()
```

This is amazing for chaining together the various steps some data goes through without needing to create intermediary objects

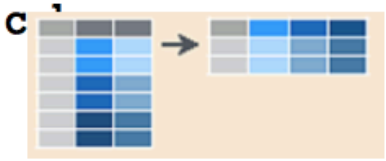
```
three(two(one(data, a), b), c)
data %>%
  one(a) %>%
  two(b) %>%
  three(c)
```

Data Manipulation with 'TIDYR'

- **Some of the key "verbs":**
  - **gather**: takes multiple columns, and gathers them into key-value pairs



  - **spread**: takes two columns (key & value) and spreads in to multiple c



  - **separate**: splits a single column into multiple columns

  - **unite:** combines multiple columns into a single column

```
library(tidyr)
tidy4a <- table4a %>%
  gather(`1999`, `2000`, key = "year", value = "cases")
tidy4b <- table4b %>%
  gather(`1999`, `2000`, key = "year", value = "population")
left_join(tidy4a, tidy4b)
```

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

```
spread(table2, key = type, value = count)
```

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504896
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

table2

```
separate(table3, year, into = c("century", "year"), sep = 2)
separate(table3, rate, into = c("cases", "population"))
unite(table5, "new", century, year, sep = "")
```

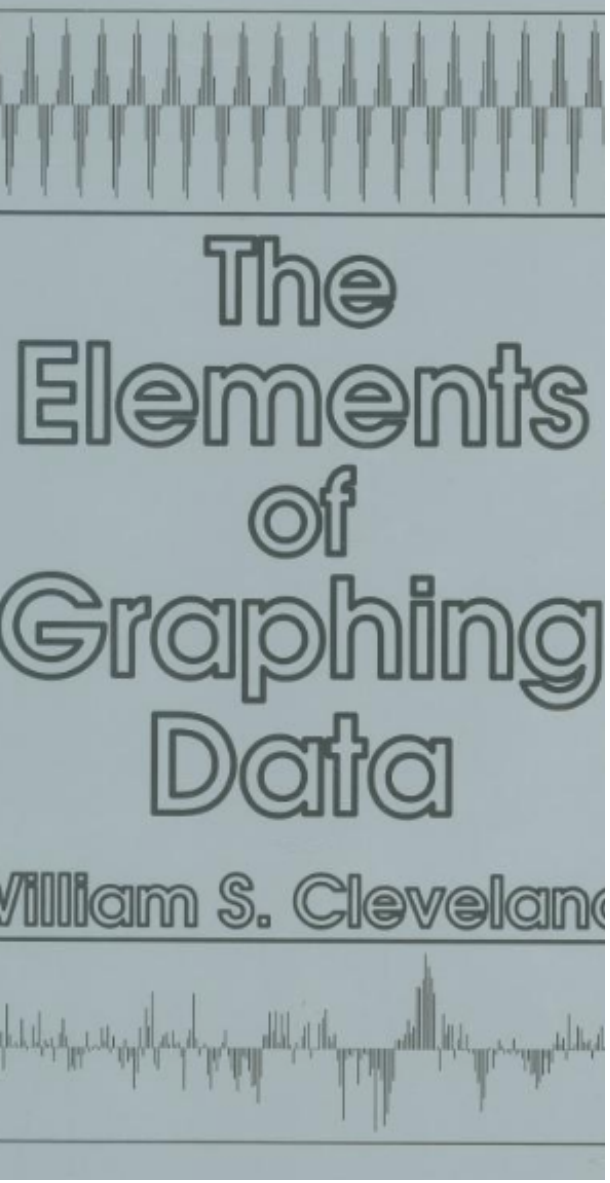
Data Visualization with ‘GGPLOT2’

Philosophy

- «ggplot2 is designed to work in a layered fashion, starting with a layer showing the raw data then adding layers of annotation and statistical summaries. [..]»
- «ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and latticegraphics and none of the bad parts.»
- «It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics.»

How to use ggplot

- In contrast to the regular plot() function, ggplot is based on the grid-package for drawing all plots (as is lattice)
- Therefore, ggplot is not «really» compatible with the default plot-functions
- Plots are drawn in layers that are stacked on top of each other
- To create a plot we either use the **qplot()** or **ggplot()** function



Cleveland's Principles

"Clear vision"

- Data should stand out—nothing superfluous
- Use visual elements to highlight the data
- Declutter
- Include reference lines

"Clear understanding"

- Conclusions should be in graphical form
- Legends should inform (comprehensively!)

Scales

- consider limits, logarithmic, dual, etc.

Source: Cleveland, *The Elements of Graphing Data,* 1985

Albert Cairo, *The Truthful Art* [2016]

Five qualities of “great visualizations” (p. 45)

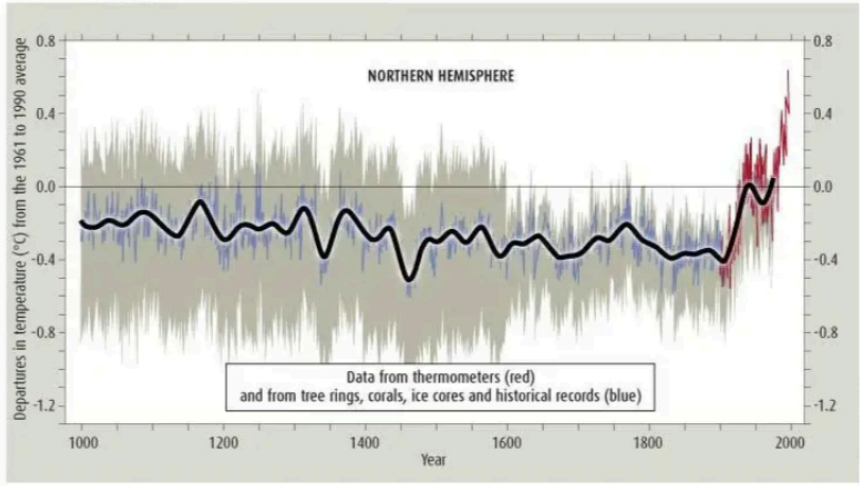
– Truthful

– Functional

– Beautiful
– Insightful
– Enlightening

Source: https://www.newscientist.com/article/dn11646-climate-myths-the-hockey-stick-graph-has-been-proven-wrong/

“Variations of the Earth’s surface temperature over the past 1000 years”



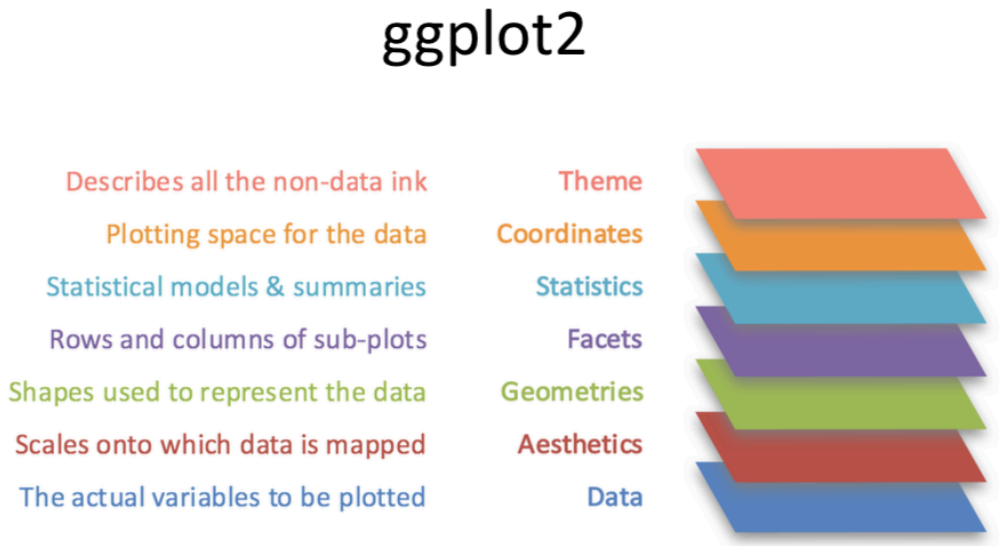
GGPLOT2 (layered) grammar of graphics

- See “A Layered Grammar of Graphics” [Wickham 2010]

https://www.tandfonline.com/doi/abs/10.1198/jcgs.2009.07098

- Builds on “The Grammar of Graphics” by Wilkinson, Anand, and Grossman (2005)

https://towardsdatascience.com/murdering-a-legendary-data-story-what-can-we-learn-from-a-grammar-of-graphics-ad6ca42f5e30



GGPLOT2 layered graphics tutorial (tutorial link)

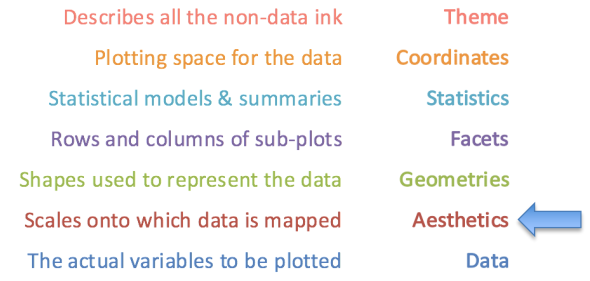
```
library(car); data(SLID)
```

data

```
g <- ggplot(data=SLID)
```

aesthetic

```
g <- ggplot(data=SLID,
        aes(x=education,y=wages))
```

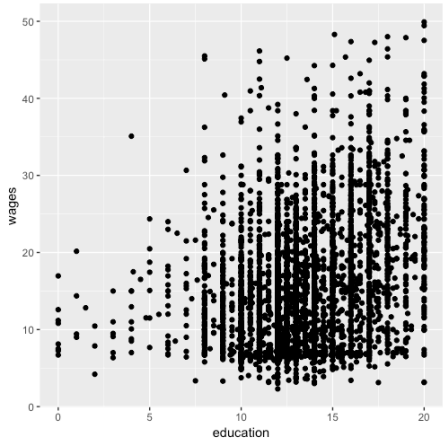


GGPLOT2 layered graphics tutorial (tutorial link)

geometry

```
g <- ggplot(data=SLID,
        aes(x=education,y=wages)) +
     geom_point()
g
```

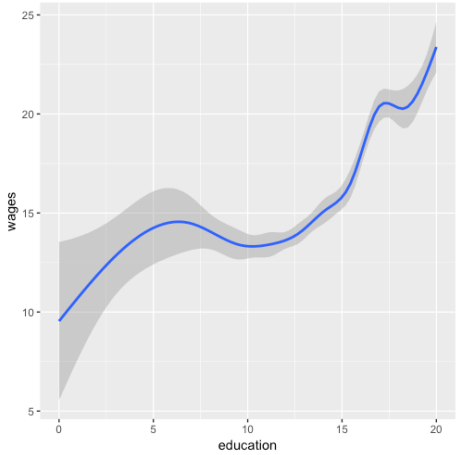
Describes all the non-data ink — Theme
Plotting space for the data — Coordinates
Statistical models & summaries — Statistics
Rows and columns of sub-plots — Facets
Shapes used to represent the data — Geometries
Scales onto which data is mapped — Aesthetics
The actual variables to be plotted — Data



geometry

```
g <- ggplot(data=SLID,
        aes(x=education,y=wages)) +
     geom_smooth()
g
```

Describes all the non-data ink — Theme
Plotting space for the data — Coordinates
Statistical models & summaries — Statistics
Rows and columns of sub-plots — Facets
Shapes used to represent the data — Geometries
Scales onto which data is mapped — Aesthetics
The actual variables to be plotted — Data

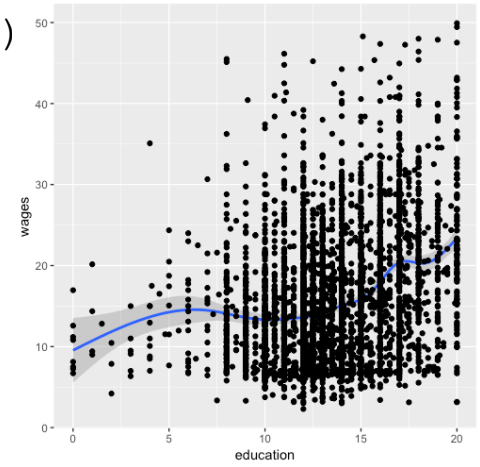


GGPLOT2 layered graphics tutorial (tutorial link)

geometry

```
g <- ggplot(data=SLID,
        aes(x=education,y=wages)) +
     geom_smooth()
g <- g + geom_point()
g
```

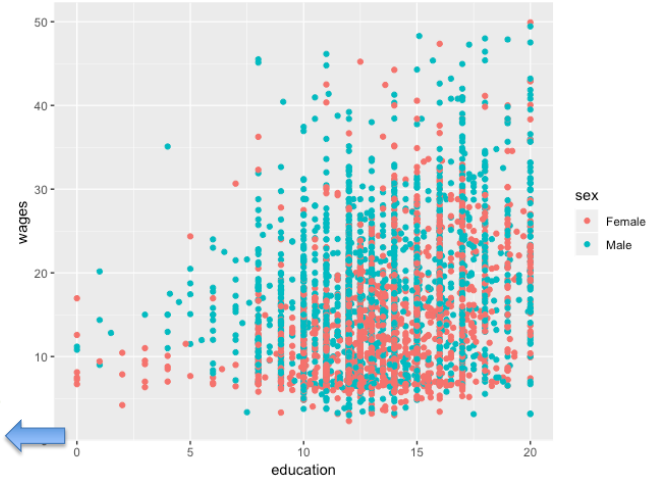
Describes all the non-data ink — Theme
Plotting space for the data — Coordinates
Statistical models & summaries — Statistics
Rows and columns of sub-plots — Facets
Shapes used to represent the data — Geometries
Scales onto which data is mapped — Aesthetics
The actual variables to be plotted — Data



aesthetic (again)

```
g <- ggplot(data=SLID)
g <- g + geom_point(
            aes(x=education,y=wages,colour=sex))
g
```

Describes all the non-data ink — Theme
Plotting space for the data — Coordinates
Statistical models & summaries — Statistics
Rows and columns of sub-plots — Facets
Shapes used to represent the data — Geometries
Scales onto which data is mapped — Aesthetics
The actual variables to be plotted — Data

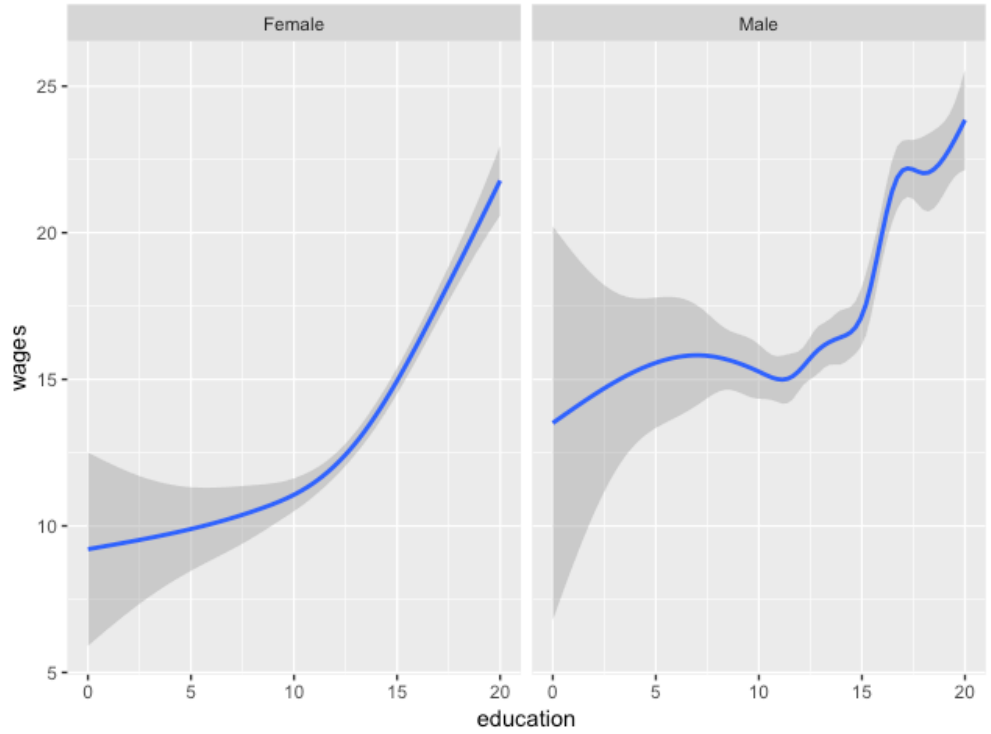


GGPLOT2 layered graphics tutorial (tutorial link)

facet

```
g <- ggplot(data=SLID,
        aes(x=education,y=wages)) +
    geom_smooth()
g <- g + facet_wrap(~sex)
g
```

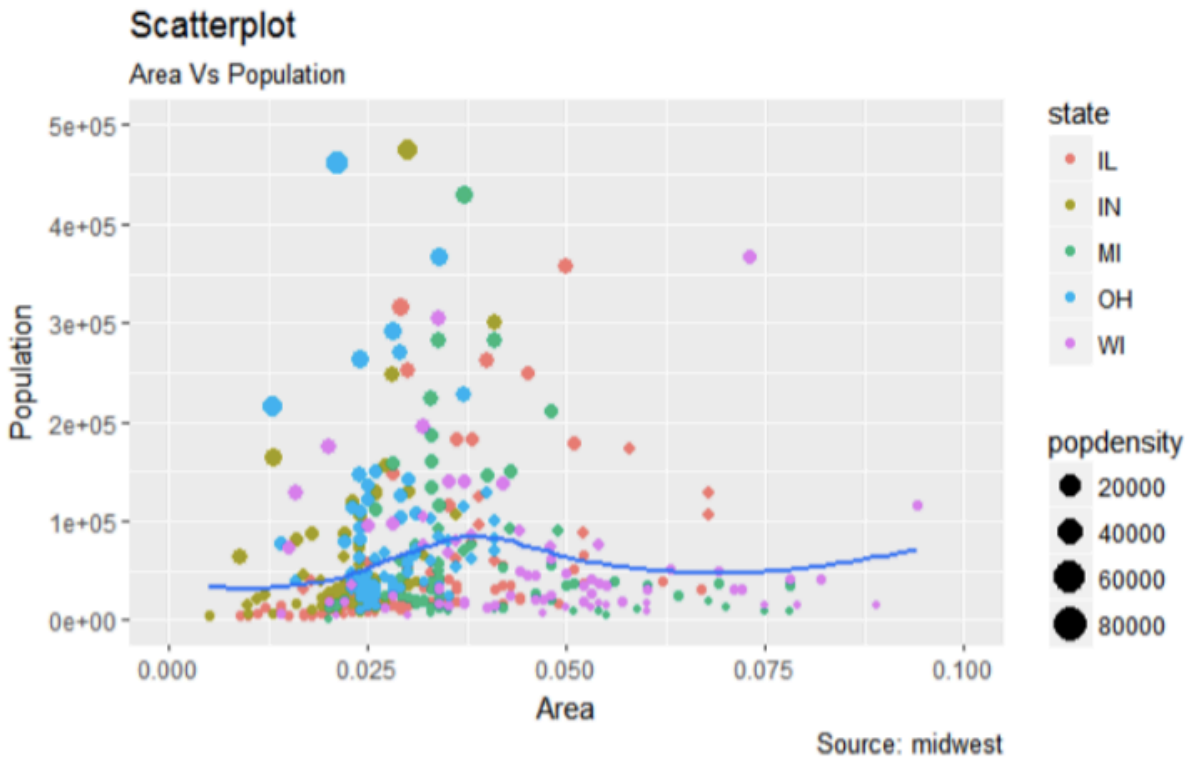
Describes all the non-data ink	Theme
Plotting space for the data	Coordinates
Statistical models & summaries	Statistics
Rows and columns of sub-plots	Facets
Shapes used to represent the data	Geometries
Scales onto which data is mapped	Aesthetics
The actual variables to be plotted	Data



Data Visualization with ‘GGPLOT2’ (examples)

Code example is from [“R for Data Science” online book]

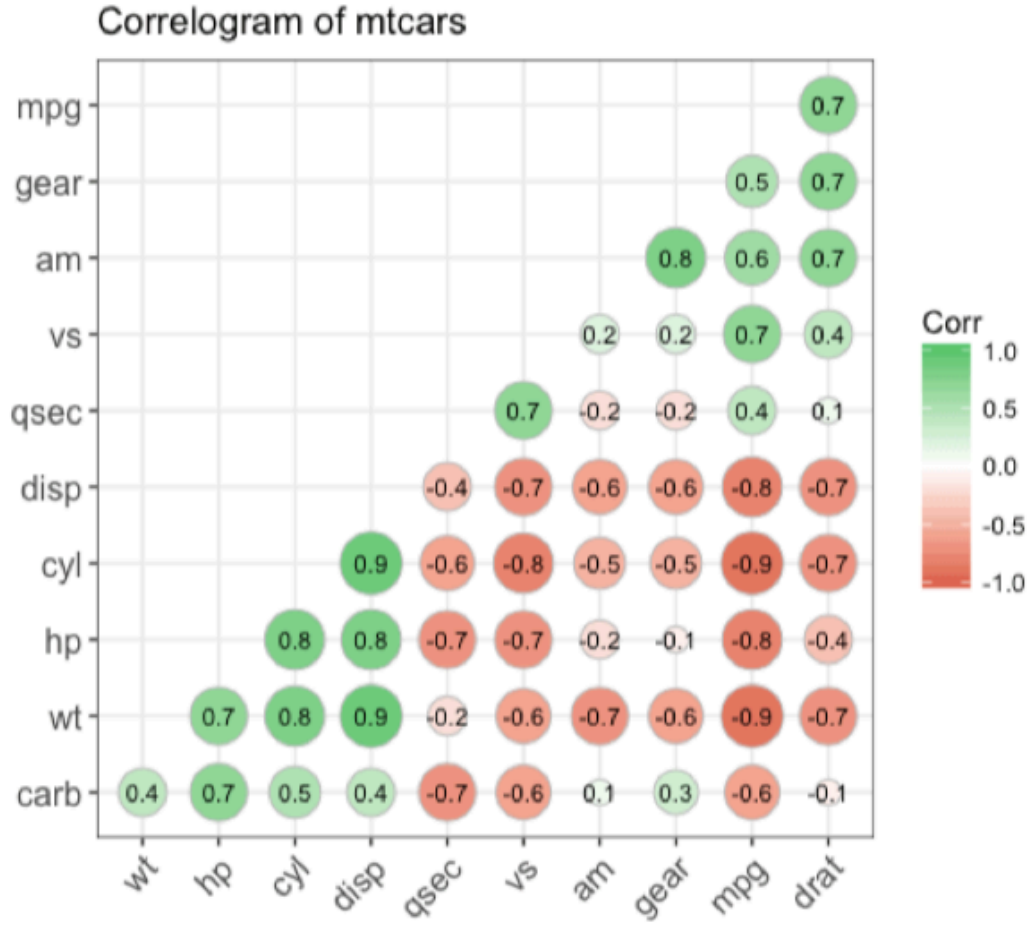
- **Scatter plot**




```
library(ggplot2)

ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() +
geom_smooth(method="lm")


ggplot(midwest, aes(x=area, y=poptotal)) +
 geom_point(aes(col=state, size=popdensity)) +
 geom_smooth(method="loess", se=F) +
 xlim(c(0, 0.1)) +
 ylim(c(0, 500000)) +
 labs(subtitle="Area Vs Population",
      y="Population",
      x="Area",
      title="Scatterplot",
      caption = "Source: midwest")
```

Data Visualization with ‘GGPLOT2’ (mtcars)

- **Correlogram**

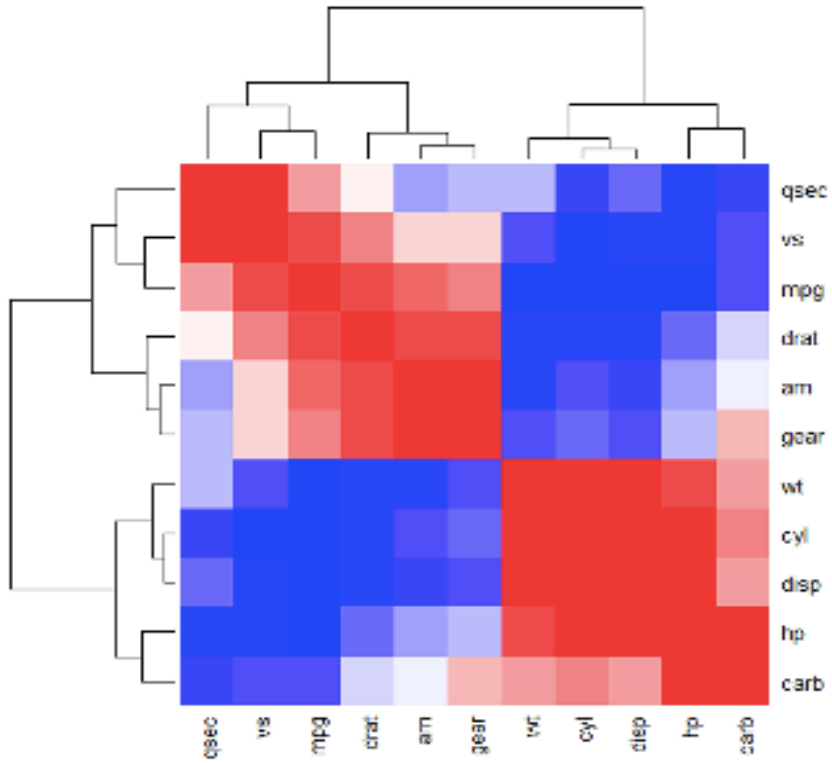



```
library(ggplot2)
library(ggcorrplot)
# Correlation matrix
data(mtcars)
corr <- round(cor(mtcars), 1)
# Plot
ggcorrplot(corr, hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 3,
           method="circle",
           colors = c("tomato2", "white", "springgreen3"),
           title="Correlogram of mtcars",
           ggtheme=theme_bw)
```

Does it correlate? Tutorial link

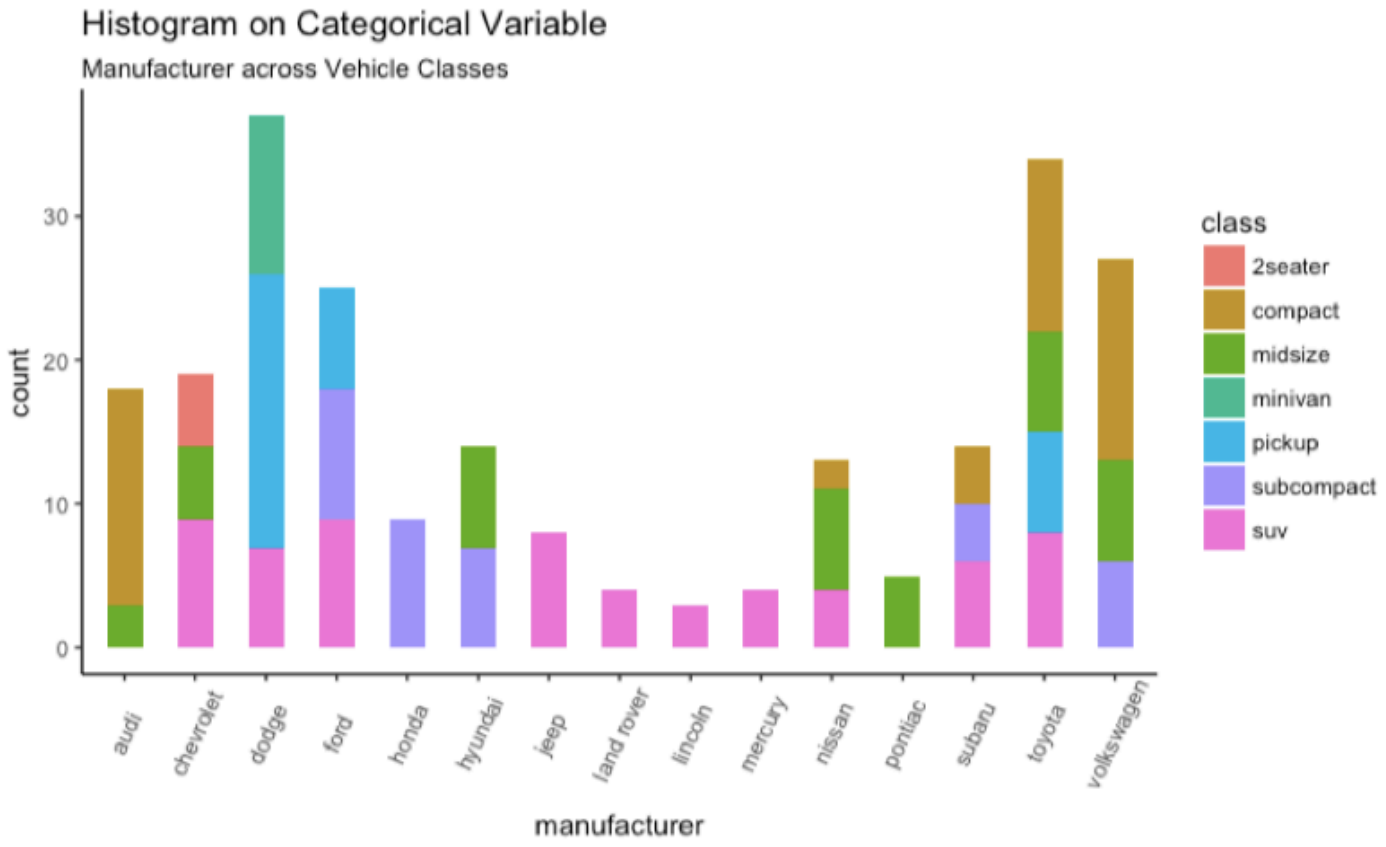
```
>> install.packages("corrplot")

>> library(corrplot)

>> #cor(x, method = "pearson", use =

"complete.obs")

>> cor(mtcars)

>> library(corrplot)

>> col<- colorRampPalette(c("blue",

"white", "red"))(20)

>> heatmap(x = res, col = col, symm =

TRUE)
```

Output:



Data Visualization with ‘GGPLOT2’

- **Histogram on Categorical Variables**



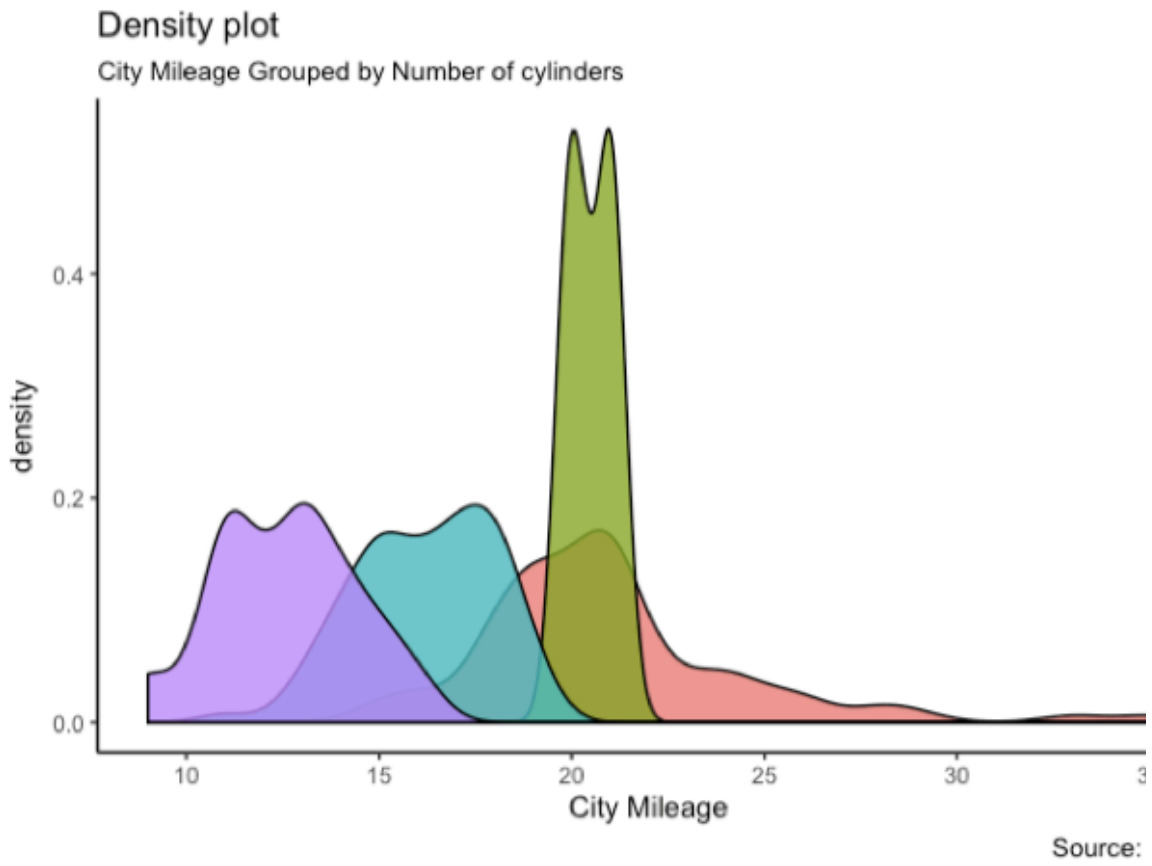

```
ggplot(mpg, aes(manufacturer)) +
geom_bar(aes(fill=class), width = 0.5) +

theme(axis.text.x = element_text(angle=65,
                  vjust=0.6)) +

labs(title="Histogram on Categorical Variable",
                  subtitle= "Manufacturer across Vehicle
                  Classes")
```

Data Visualization with ‘GGPLOT2’

Export

ggsave(“filename.pdf”, g, width=12, height=4)

- **Density plot**

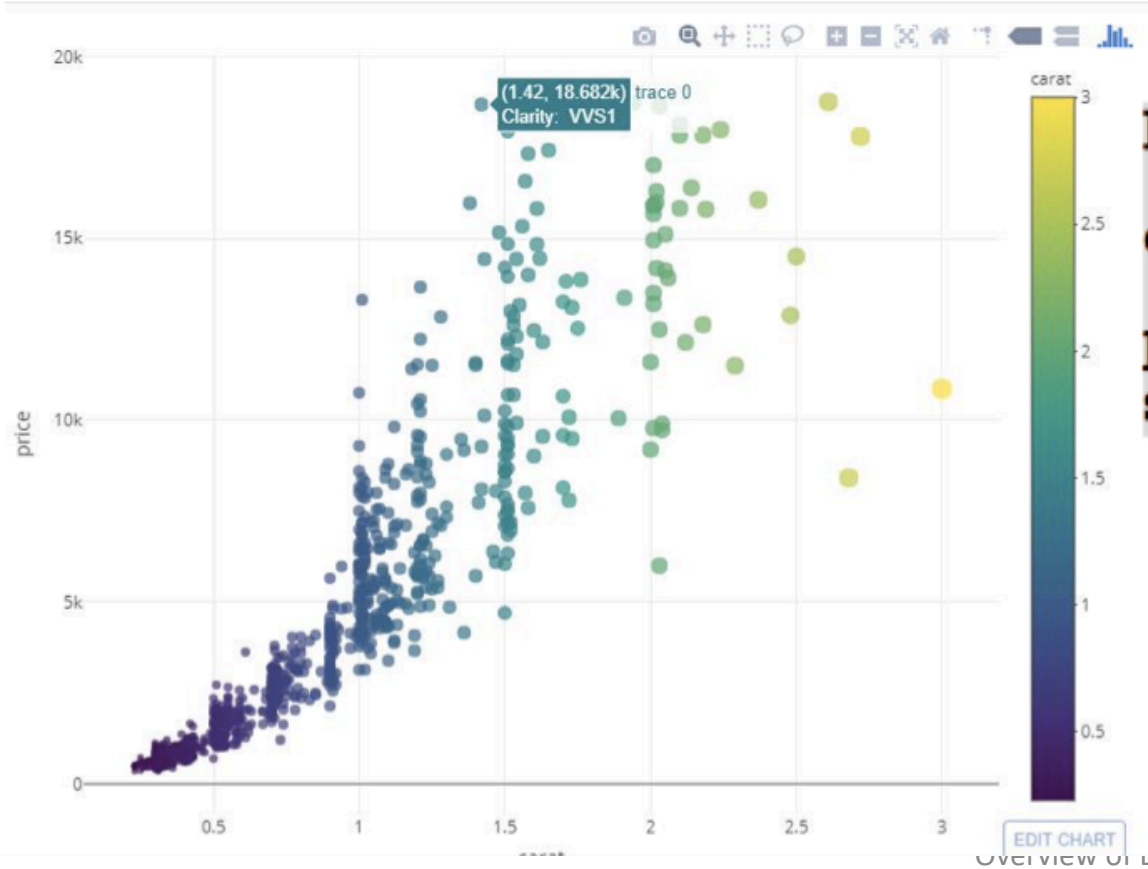


Where to look for help?

- Where to look for help?
- https://ggplot2.tidyverse.org/reference/
- http://sape.inf.usi.ch/quick-reference/ggplot2/
- Built-in help (?function)
- StackOverflow / Google: always include “ggplot2” as a search term
-

Interactive visualization with 'PLOTLY'

Plotly library makes interactive, publication-quality graphs online. It supports line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heat maps, subplots, multiple-axes, and 3D charts.




```
library(plotly)

d <- diamonds[sample(nrow(diamonds), 1000), ]

plot_ly(d, x = ~carat, y = ~price, color = ~carat,
size = ~carat, text = ~paste("Clarity: ", clarity))
```

References

Tutorial links:

https://github.com/asel-datascience/ODS2020/tree/master/Week_6_Intro_R/IntroR_ODS/code

R for Data Science online book and H. Wickham's site (author of the tidyverse package)

https://r4ds.had.co.nz

http://hadley.nz