Title:
Visualization using RStudio and Protein dataset

Dr. Alon Friedman, Associate professor

==========

In this presentation, I will focus on *R* and visualization.

> Log on to RSudio Cloud https://rstudio.cloud/

As a platform, *R* is allocated towards statistical analysis and data visualization, but this core development consists of the extensible through packages—many of theose packages are specialized—that are created and maintained by a developer community.

R allows us to generate visualization to extend data analysis.

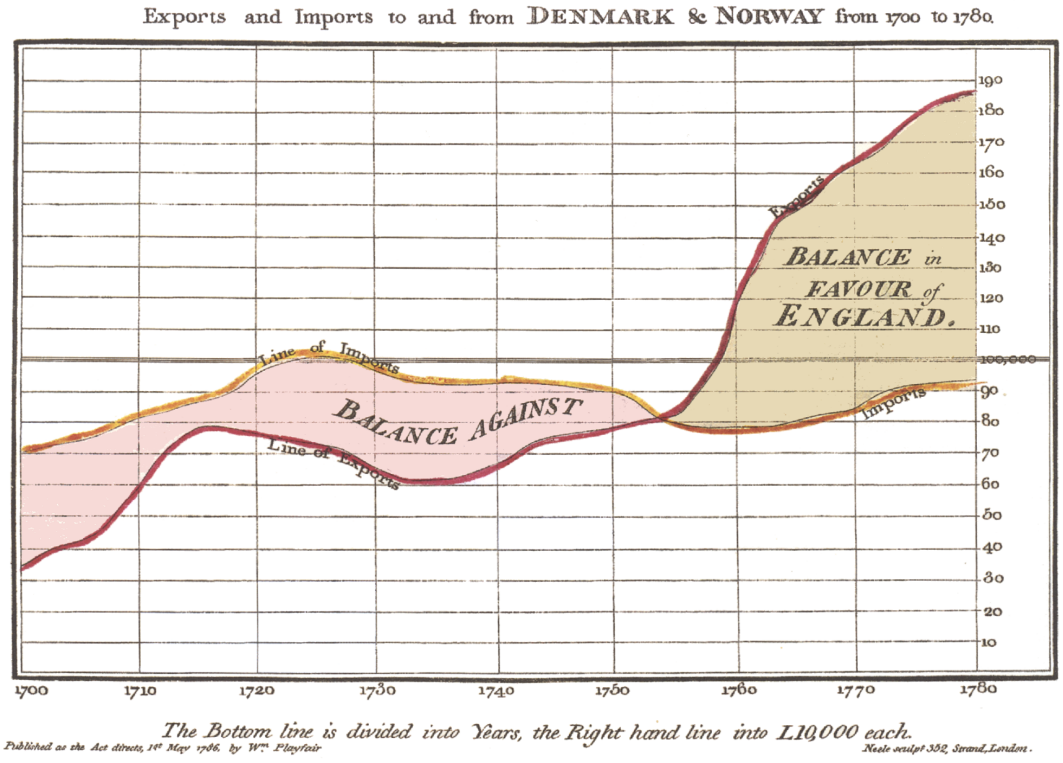
I will also discuss the *Three common strategies* for working with Big Data in *R*:
1. Sample
2. Chunk and Pull
3. Push Compute to Data

Python vs. R

The basis of much of data science is based on statistics, data mining, and visualization. Many of the standard statistical techniques are adequately covered by Python but if you try unconventional techniques and methodologies, you will get lost. *R* is designated as a "language and environment for statistical computing and graphics" (What is R?, n.d.). There are now more than 35,000 R packages on the official repository CRAN alone! I will demonstrate how you can use GitHub to pull R packages and code to help us analyze JSONL files using the knowledge already created in *R*.

Why visualization and Big Data/Data Science?

- We as humans react to visual data faster than text. (Mani, M., & Fei, S. 2017).
- The first benefit of visualization with Big Data/Data science is easier to understand the data.
- Visualization lets us distinguish the data spread over time.
- Visualization helps us recognize similarities and random connections.
- Playfair (1821) is widely considered the father of statistical graphics including the most popular graphs we use today (line, bar, circle, and pie charts).

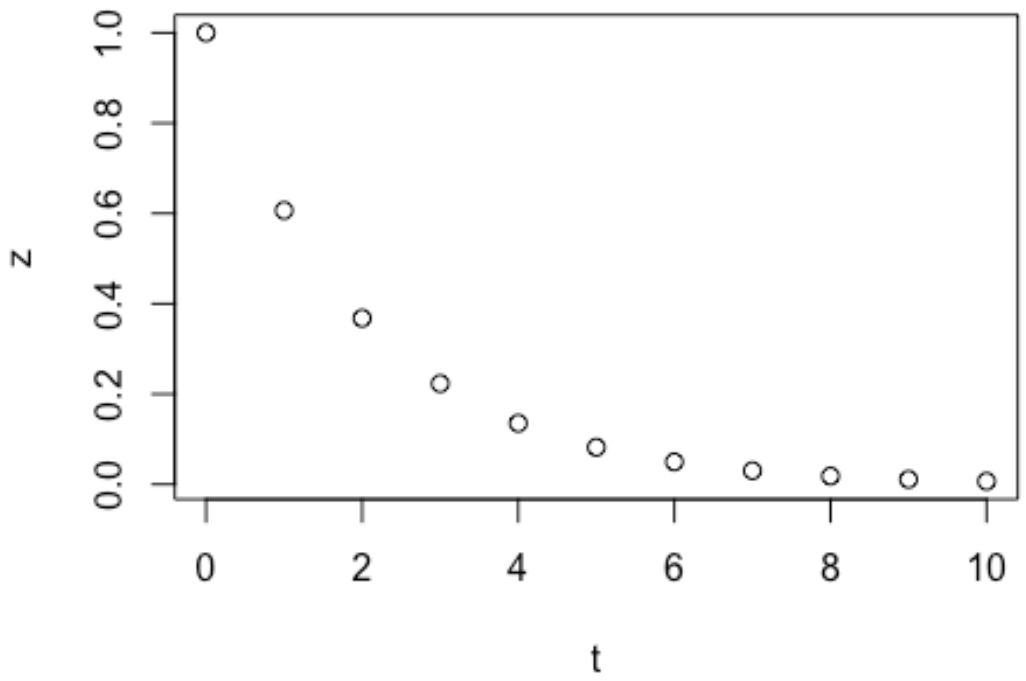


Basic Visualization in *R*

R provides us basic visualization functions without any additional packages:
Those functions includes: plot: generic x-y plotting * barplot: bar plots. * hist: histograms.

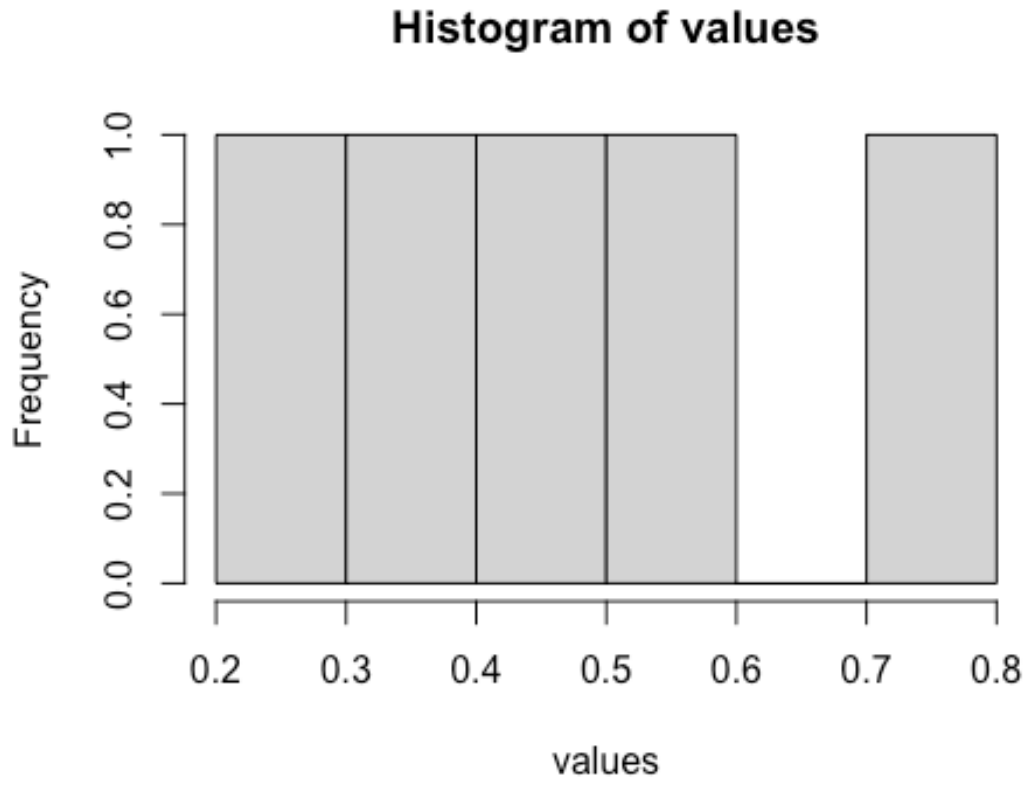
* pie: pie charts. R code for plot
==========

```
t=0:10
z= exp(-t/2)
plot(t,z)
```



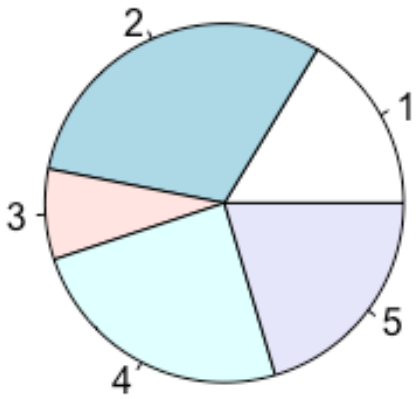
R code for Histogram

```
values <- c(0.4, 0.75, 0.2, 0.6, 0.5)
hist(values)
```



R code for pie

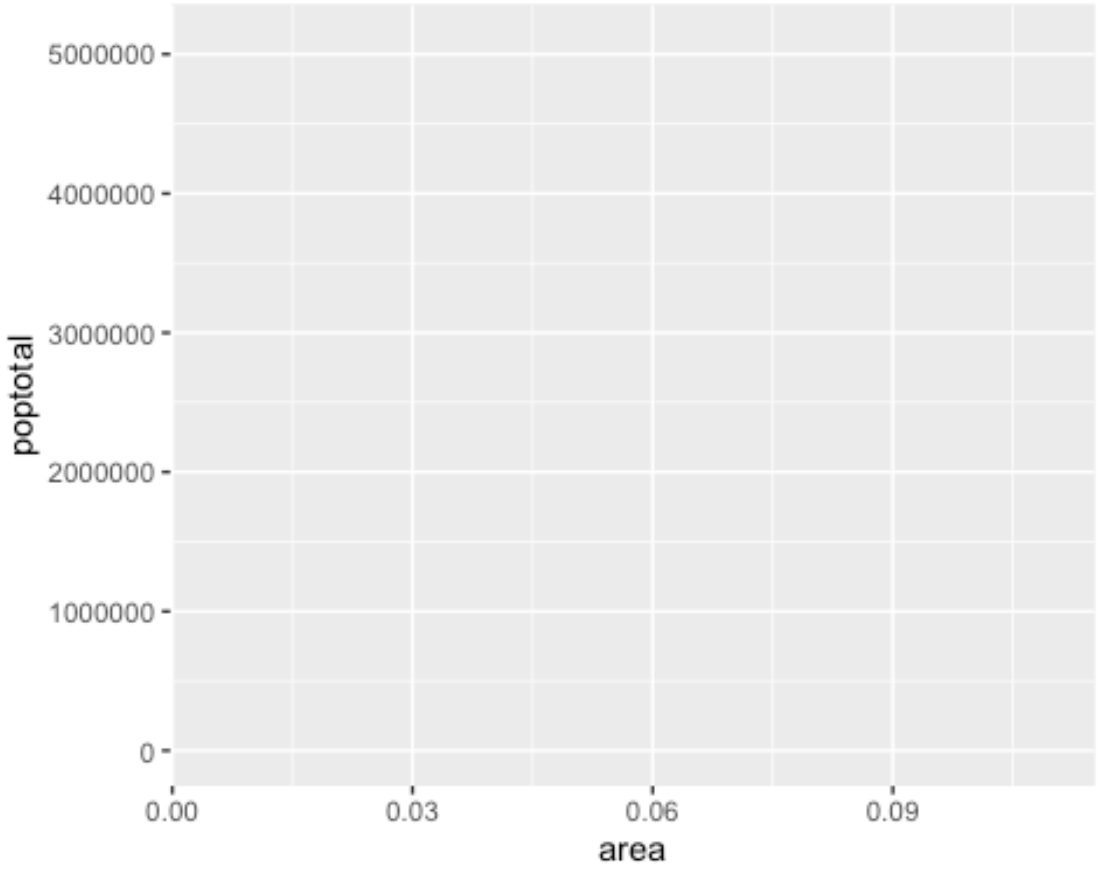
```
values <- c(0.4, 0.75, 0.2, 0.6, 0.5)
pie(values)
```



ggplot2 breaks away

ggplot2 ggplot2 is an R package for creating graphics, based on The Grammar of Graphics. One of the things it allows us to have a Canvas background.

```
# Init Ggplot
ggplot(midwest, aes(x=area, y=poptotal))  # area and poptotal are columns in
'midwest'
```

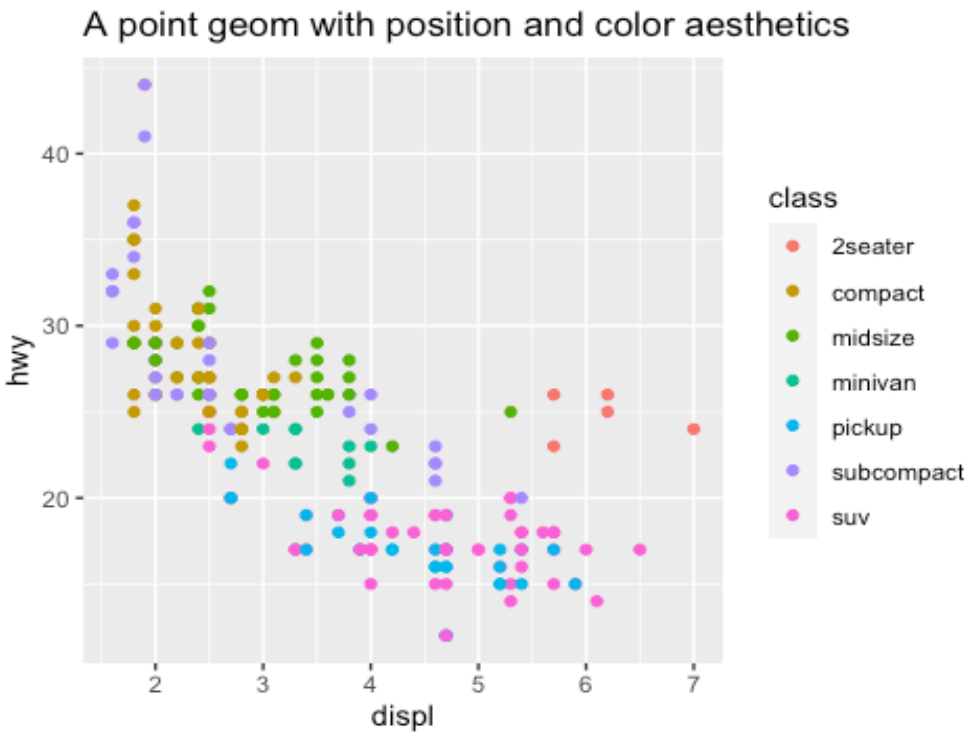


ggplot2 and Visual Grammar

Core components of the layered grammar of graphics using ggplot2.

- Data.
- Mapping.
- Statistical transformation (stat).
- Geometric object (geom).
- Position adjustment (position).

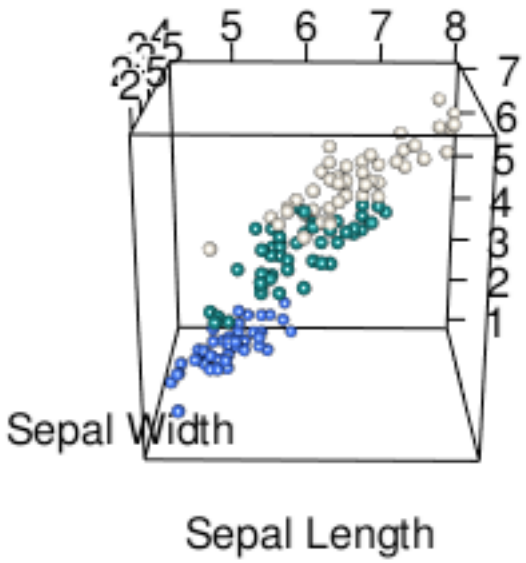
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = class)) +
  geom_point() +
  ggtitle("A point geom with position and color aesthetics")
```



3D scatter plot

To build a 3D plot, you need to have **3 numeric variables**, each being used on an axis. In this example, I employed *iris* dataset. Plus I used *rgl* packages that come with the function called *plot3d()*

```
library(rgl)
data <- iris

# Add a new column with color
mycolors <- c('royalblue1', 'darkcyan', 'oldlace')
data$color <- mycolors[ as.numeric(data$Species) ]

# Plot
par(mar=c(0,0,0,0))
plot3d(
  x=data$`Sepal.Length`, y=data$`Sepal.Width`, z=data$`Petal.Length`,
  col = data$color,
  type = 's',
  radius = .1,
  xlab="Sepal Length", ylab="Sepal Width", zlab="Petal Length")
```



Next

- Using R packages
- Importing data
- Extracting data
- Working with GitHub
- Large data files in R
- Using Visualization in this work

Using R package

R is designated as a “language and environment for statistical computing and graphics” (What is R?, n.d.). This multilayered software core development consists of the extensible through packages. The packages are created and maintained by a developer community.

A package is a way to organize your work in R and help share it with others.

Each of the R packages contains sets of code (called *functions*) that accomplish specific tasks. In addition, every package also includes documentation, the data score, and some tests to check everything works as it should, and data sets.

Here are some of the most popular R packages

-**Tidyverse**- provides a set of functions that help you get to tidy data. For more details on dplyr https://www.tidyverse.org

-**dplyr**- is primarily a set of functions designed to enable dataframe manipulation of the data. For more details on dplyr https://github.com/tidyverse/dplyr

-**ggplot2**- is a package that helps to enhance the data visualization from your work in R. For more details on ggplot2 https://ggplot2.tidyverse.org.

R packages repositories

-*CRAN* - The Comprehensive R Archive Network

-*BioConductor*- is a set of repositories containing R packages used in the analysis of scientific data.

-*GitHub* - is the largest code repository that offers version control using Git. It offers the distributed version control and source code.

GitHub

GitHub is the most popular version control system for developers of R packages.

Using *devtools* package, allow us to install the package directly from GitHub.

1. install.packages("devtools")
2. library("devtools")
3. install_github("name of the creator of the repository/name of the package")

Importing data

The new Rstudio interface provides easy features to import data from *csv*, *xls*, *xlsx*, *sav*, *dta*, *por*, *sas* and *stata* files. For full step by step, Rstudio provides For more details on importing data see https://support.rstudio.com/hc/en-us/articles/218611977-Importing-Data-with-RStudio.

JSON and R.

JSON stands for JavaScript Object Notation. These files contain the data in human-readable format, i.e. as text. Like any other file format(s), one can read as well as write into the JSON files. To work with JSON files in R, one needs to install the *"rjson"* package.

install.packages("rjson") library("rjson")

Extracting Data

There are many techniques for extracting data in R.

Here are the most common practices:

-**Data.frame** is a list of variables of the same number of rows with unique row names, given class "data.frame"

-The most general way to subset a data frame by rows and/or columns is the base R

-**Extract function** d[rows, columns] here *d* is the data frame.

For example:

```
data <- data.frame(x1 = 1:10,
                   x2 = letters[1:10],
                   x3 = "x")
data
```

```
##    x1 x2 x3
## 1   1  a  x
## 2   2  b  x
## 3   3  c  x
## 4   4  d  x
## 5   5  e  x
## 6   6  f  x
## 7   7  g  x
## 8   8  h  x
## 9   9  i  x
## 10 10  j  x
```

```
head(data)
```

```
##   x1 x2 x3
## 1  1  a  x
## 2  2  b  x
## 3  3  c  x
## 4  4  d  x
## 5  5  e  x
## 6  6  f  x
```

-Extract or Replace Parts of a Data Frame. x[i, j, drop =] x[i, j] <- value x[[…, exact = TRUE]] x[[i, j]] <- value x$name <- value

Cleaning Data

What exactly is clean data? Clean data is accurate, complete, and in a format that is ready to analyze.

Characteristics of clean data include data that are:

-Free of duplicate rows/values -Error-free (e.g. free of misspellings) -Relevant (e.g. free of special characters) -The appropriate data type for analysis -Free of outliers
-and using “tidy data” structure

For more information about how to clean data, I highly recommend you to visit *Brief Introduction to the 12 Steps of Data Cleaning* [Morrow, 2013] (https://www.slideshare.net/jamorrow/brief-introduction-to-the-12-steps-of-evaluagio)

Large data files in R

R is known to have difficulties handling large data files.

-In this presentation, we will work with the entire file in the computer memory.

Some common recommendations for a faster process:

-Use *data.frame* -Use *data.table*'s package.

-Use *sqldf* function for *csv* files.

-Use a *SQLite database* and employ *dplyr*'s package.

-Convert your *csv* file to a *sqlite* database to the query.

- Limit the number of lines you read using *fred* function under.

For example

*size_t fread(void * buffer, size_t size, size_t count, FILE * stream)*

In this code **buffer**: it specifies the pointer to the block of memory with a size of at least (size*count) bytes to store the objects.

size: it specifies the size of each object in bytes. size_t is an unsigned integral type.

count: it specifies the number of elements, each one with a size of size bytes.

stream: it specifies the file stream to read the data from.

Using Visualization to help to read a large data file

We will show how to create visualization from **three different datasets** and different R packages.

The first example is using **esquisse** package. The purpose of this add-in package is to let you explore your data quickly to extract the information they hold.

The second example is with a small dataset without the add on of esquisse. The third and last example is with a larger data set that holds 31 megabits and 23 columns. Both data sets were taken from the *National Oceanic and Atmospheric'* database on the subject of *protein*.

We will use four R packages:

-**ggplot2**- we discussed before

-**igraph** - Visualization Network analysis package. -ggalluvial- is a ggplot2 extension for producing alluvial plots

-**viridis** - provides the base functions for generating the color maps in base R. -esquisse - is add-in is to let you explore your data quickly.

You work

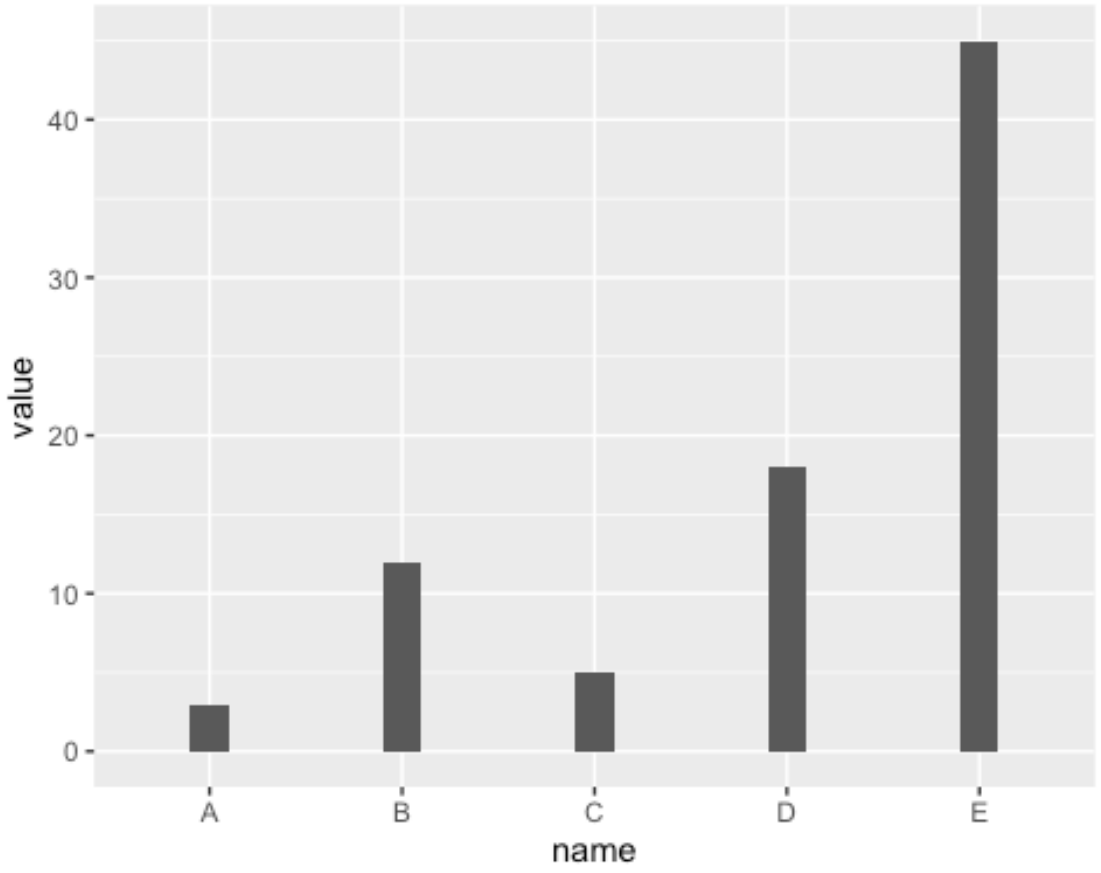
1. Log on to National Centers for Environmental Information https://www.ncdc.noaa.gov and search for your area of interst.
2. Convert the data you selected and insert to R.
3. Then, try to select 7 common charts including:
4. Scatter Plot
5. Histogram
6. Bar Chart
7. Box Plot
8. Area Chart
9. Heat Map and
10. Correlogram

Here few examples of code

```
library(ggplot2)
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +
    geom_point()
```

Histogram is often used for continuous variables. It captures the data into bins and frequency.

```
library(ggplot2)

# Create data
data <- data.frame(
  name=c("A","B","C","D","E") ,
  value=c(3,12,5,18,45)
  )

# Barplot
ggplot(data, aes(x=name, y=value)) +
  geom_bar(stat = "identity", width=0.2)
```



Box Plot is also well recognized visual format. It allows us to combination categorical and continuous variables. R allows us to point the outliers in the data.

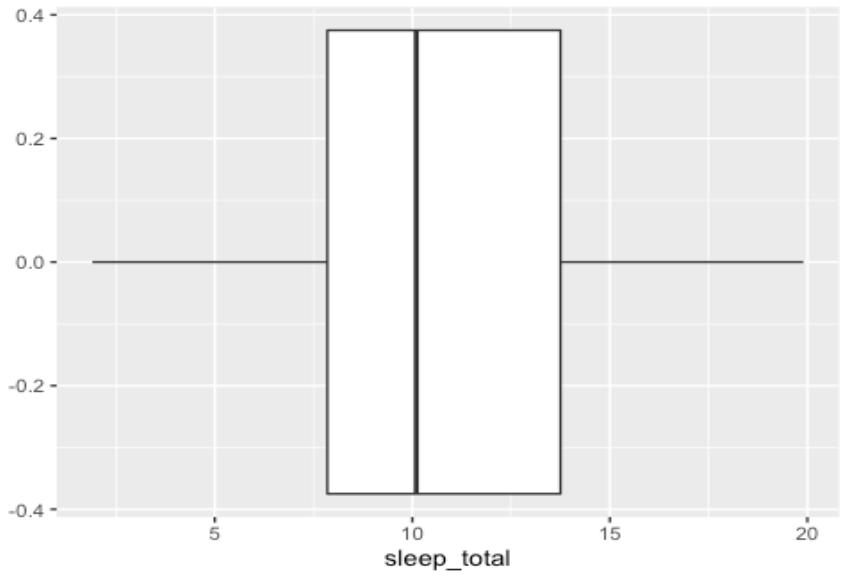
```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.
3.1 ──

## ✓ tibble  3.1.0     ✓ dplyr   1.0.5
## ✓ tidyr   1.1.3     ✓ stringr 1.4.0
## ✓ readr   1.4.0     ✓ forcats 0.5.1
## ✓ purrr   0.3.4

## ── Conflicts ────────────────────────────────────────── tidyverse_conflict
s() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
msleep %>% glimpse()
```

```
## Rows: 83
## Columns: 11
## $ name         <chr> "Cheetah", "Owl monkey", "Mountain beaver", "Greater
shor…
## $ genus        <chr> "Acinonyx", "Aotus", "Aplodontia", "Blarina", "Bos",
"Bra…
## $ vore         <chr> "carni", "omni", "herbi", "omni", "herbi", "herbi", "
carn…
## $ order        <chr> "Carnivora", "Primates", "Rodentia", "Soricomorpha",
"Art…
## $ conservation <chr> "lc", NA, "nt", "lc", "domesticated", NA, "vu", NA, "
dome…
## $ sleep_total  <dbl> 12.1, 17.0, 14.4, 14.9, 4.0, 14.4, 8.7, 7.0, 10.1, 3.
0, 5…
## $ sleep_rem    <dbl> NA, 1.8, 2.4, 2.3, 0.7, 2.2, 1.4, NA, 2.9, NA, 0.6, 0
.8, …
## $ sleep_cycle  <dbl> NA, NA, NA, 0.1333333, 0.6666667, 0.7666667, 0.383333
3, N…
## $ awake        <dbl> 11.9, 7.0, 9.6, 9.1, 20.0, 9.6, 15.3, 17.0, 13.9, 21.
0, 1…
## $ brainwt      <dbl> NA, 0.01550, NA, 0.00029, 0.42300, NA, NA, NA, 0.0700
0, 0…
## $ bodywt       <dbl> 50.000, 0.480, 1.350, 0.019, 600.000, 3.850, 20.490,
0.04…
```

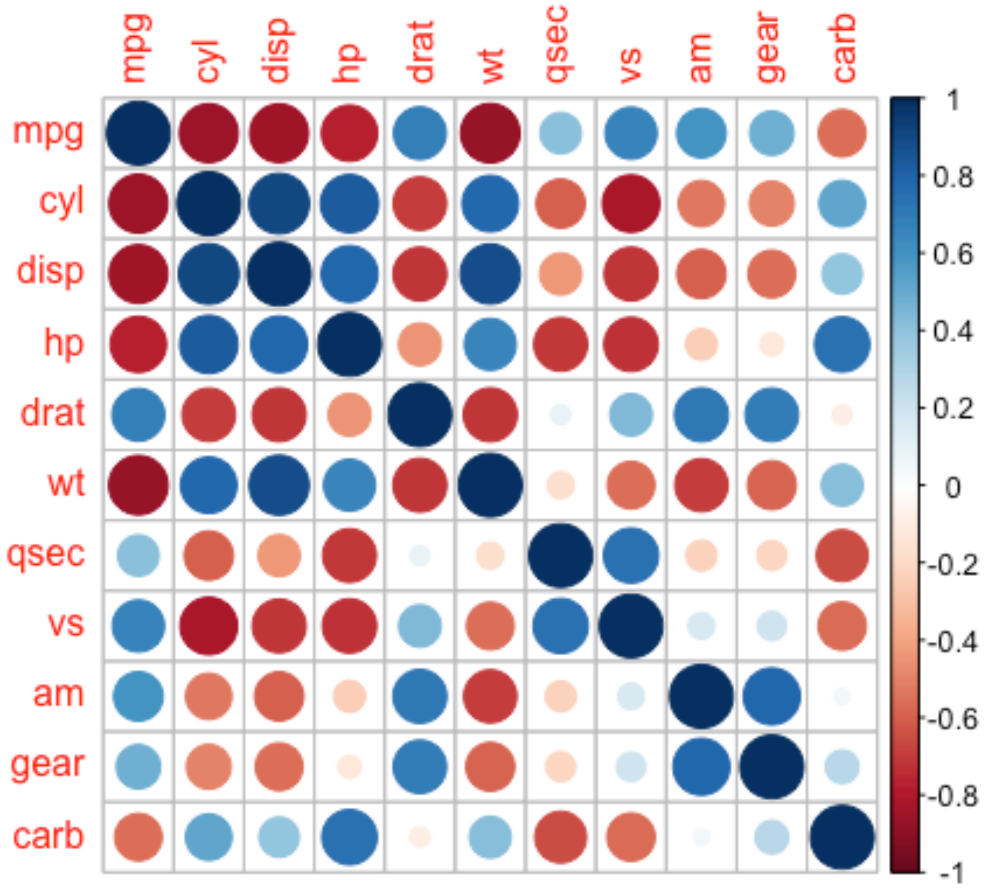
```
ggplot(data = msleep, aes(x = sleep_total)) +
  geom_boxplot()
```



Last one, **Correlgram** In this example, the darker the color, higher the co-relation between variables. I am using a new R package called *Corrgram*. It allows us to build this type of visualization.

```
library(corrplot)

## corrplot 0.88 loaded

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

#correlation matrix
M<-cor(mtcars)
head(round(M,2))

##        mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
## mpg   1.00 -0.85 -0.85 -0.78  0.68 -0.87  0.42  0.66  0.60  0.48 -0.55
## cyl  -0.85  1.00  0.90  0.83 -0.70  0.78 -0.59 -0.81 -0.52 -0.49  0.53
## disp -0.85  0.90  1.00  0.79 -0.71  0.89 -0.43 -0.71 -0.59 -0.56  0.39
## hp   -0.78  0.83  0.79  1.00 -0.45  0.66 -0.71 -0.72 -0.24 -0.13  0.75
## drat  0.68 -0.70 -0.71 -0.45  1.00 -0.71  0.09  0.44  0.71  0.70 -0.09
## wt   -0.87  0.78  0.89  0.66 -0.71  1.00 -0.17 -0.55 -0.69 -0.58  0.43
```

```
#visualizing correlogram
#as circle
corrplot(M, method="circle")
```



Enjoy