Sane's STATS Academy of Statistics

R and CRAN

An Introduction

December 18, 2021

Sane's STATS Academy of Statistics

What is R ?

- R is a programming language and software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing.
- It has been widely used by statisticians, academicians and students for many decades.

Sane's STATS Academy of Statistics

Why R ?

- R is the leading tool for statistics, data analysis, and machine learning.
- Not only is R free, but it's also open-source.
- R allows you to integrate with other languages (C/C++, Java, Python) and enables you to interact with many data sources
- It's platform-independent, so you can use it on any operating system.

Evolution of R

- R is an implementation of the S programming language combined with lexical scoping semantics inspired by Scheme.
- Firstly, language S was created by John Chambers while at Bell Labs.
- R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team, of which John Chambers is a member.
- R is named partly after the first names of the first two R authors and partly as a play on the name of S.

- Courtesy: Wikipedia

R as a Project

- R is a GNU project.
- The source code for the R software environment is written primarily in C, Fortran, and R.
- R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems.
- While R has a command line interface, there are several graphical front-ends available.
    - Courtesy: Wikipedia

Sane's STATS Academy of Statistics

Extensibility of R

- A rich collection of libraries are available for R functionality called packages.
- There are some packages provided by default for the basic installation of R
- If you want to add more functionality it can be extended by installing packages to your software environment.
- We can think of packages as add-ins of R.

Programming Features of R

- R being interpreted language users access it using command line interpreter.
- R supports procedural programming like calling functions as well as object oriented programming like handling objects.
- R provides a matrix approach towards handling the data unlike the data step approach of SAS.

Sane's STATS Academy of Statistics

CRAN

- The “Comprehensive R Archive Network” (CRAN) is a collection of sites which carry identical material, consisting of the R distribution(s), the contributed extensions, documentation for R, and binaries.
- There are several mirror sites to it.
- From CRAN, you can obtain the latest official release of R, daily snapshots of R, documentation to R etc.

Sane's STATS Academy of Statistics

CRAN Website: https://cran.rstudio.com

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2015-12-10, Wooden Christmas-Tree) R-3.2.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

- 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.

Sane's STATS Academy of Statistics

Installing R

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

R-3.2.3 for Windows (32/64 bit)

Download R 3.2.3 for Windows (62 megabytes, 32/64 bit)

Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded exactly matches the package distributed by R, you can compare the md5sum of the .exe to the true fingerprint. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

- How do I install R when using Windows Vista?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is <CRAN MIRROR>/bin/windows/base/release.htm.

Last change: 2015-12-10, by Duncan Murdoch

Sane's STATS Academy of Statistics

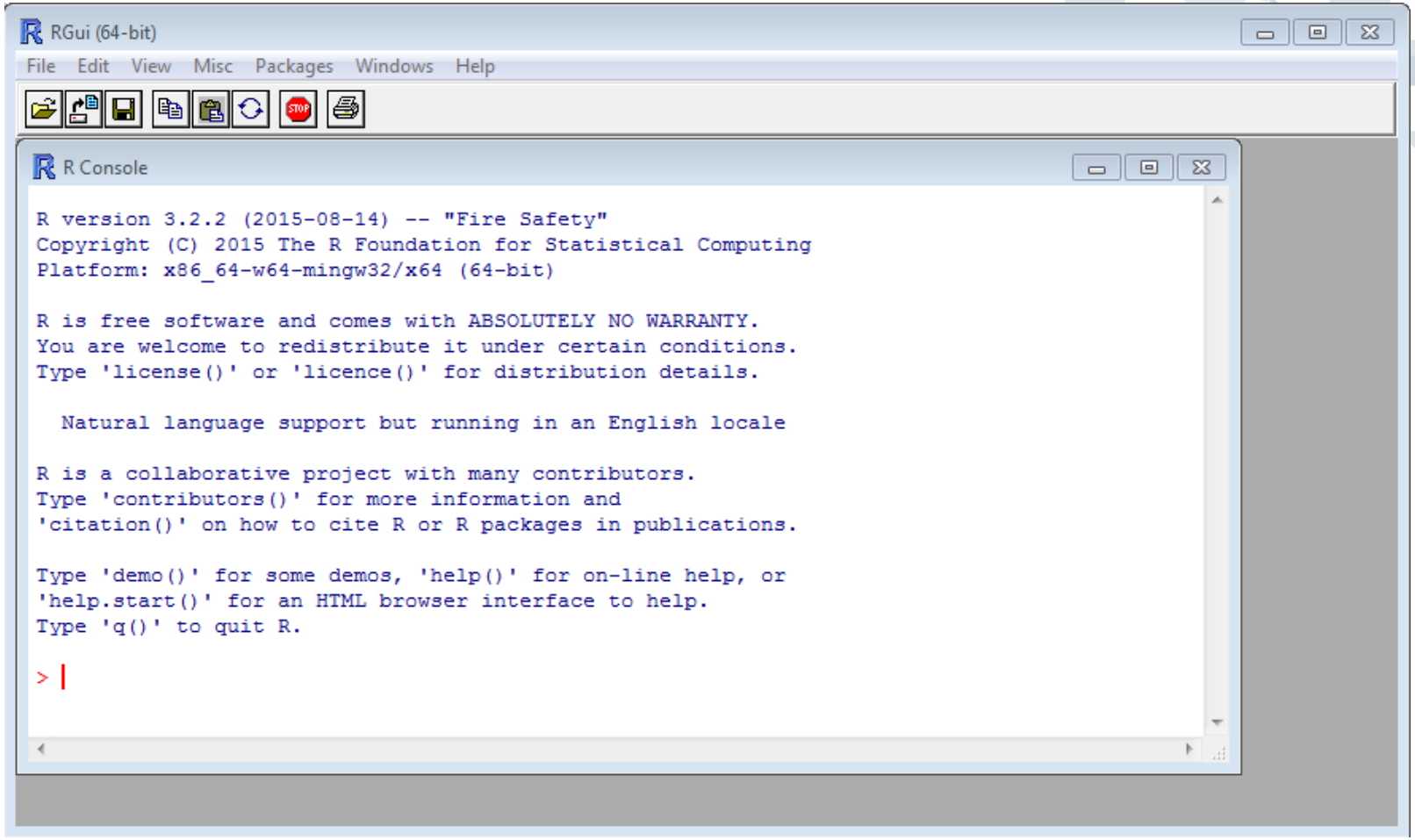
R Command Line Interface

Line Interface of R

December 18, 2021

Sane's STATS Academy of Statistics

R GUI



Warming-Up with CLI

Assigning Values to Variables

```
> a <- 9
> a
[1] 9
> b <- 7
> a+b
[1] 16
> c <- a * b
> c
[1] 63
> s <- "Analytics"
> s
[1] "Analytics"
```

Up and **down arrow keys** scroll through your command history.

Assigning combination of values to Variables

```
> d <- c(1,3,56,17)
> d
[1]  1  3 56 17
```

```
> f <- c("z","r","p","UIJ")
> f
[1] "z"   "r"   "p"   "UIJ"
```

For clearing the screen :
Ctrl + L

Sane's STATS Academy of Statistics

R Workspace

- R workspace is your current **R** working environment and includes all the objects and functions you have created.
- The user can anytime save an image of the current workspace that is automatically reloaded the next time **R** is started.

Workspace Objects

Viewing list of objects in workspace

```
> ls()
[1] "a" "b" "c" "d" "f" "s"
```

Saving workspace with name “first”

```
> save.image("D:\\first")
```

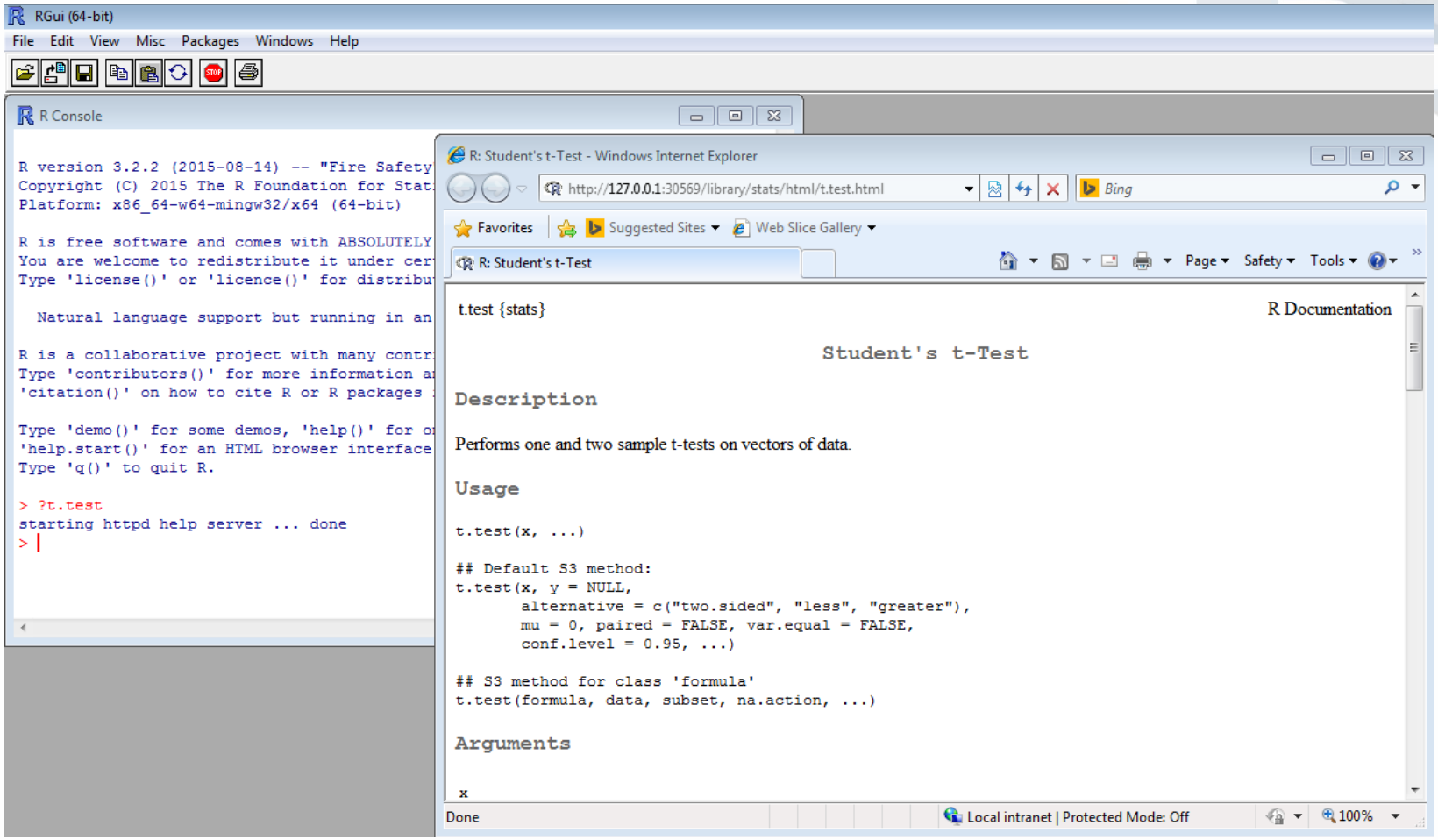
Loading workspace with name “first”

```
> load("D:\\first")
```

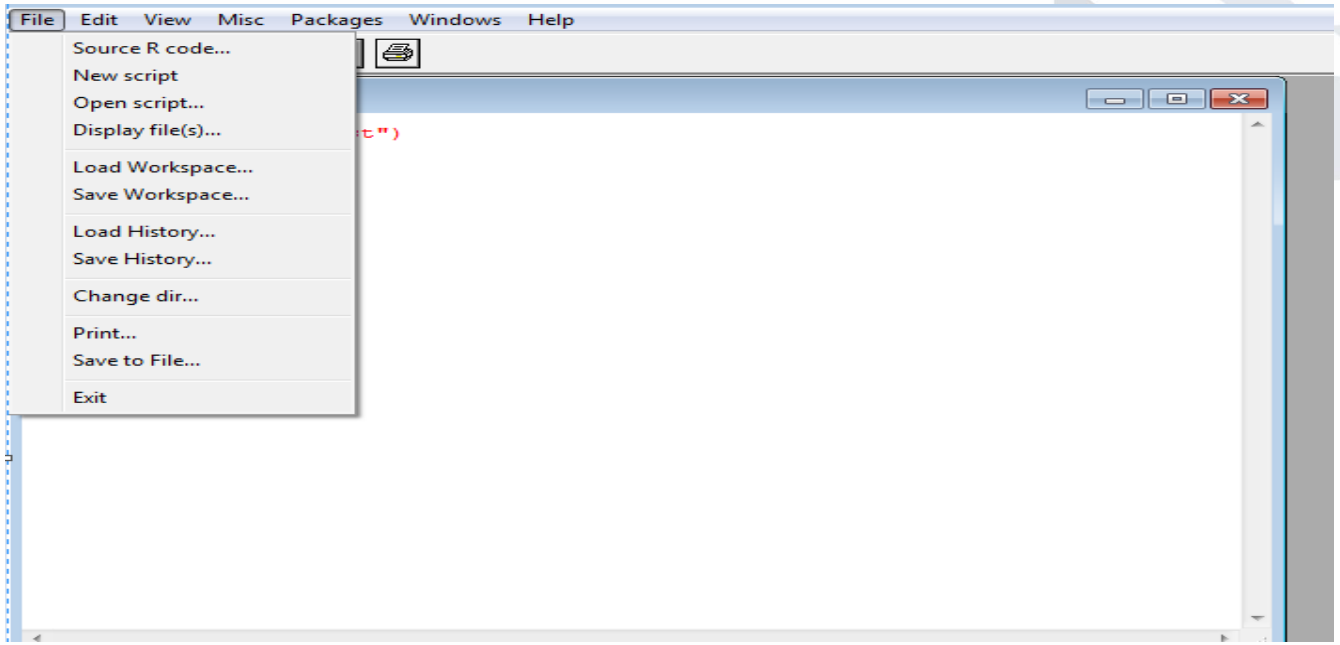
Sane's STATS Academy of Statistics

Getting Help

For getting help for any function you need to type the name of the function after "?"



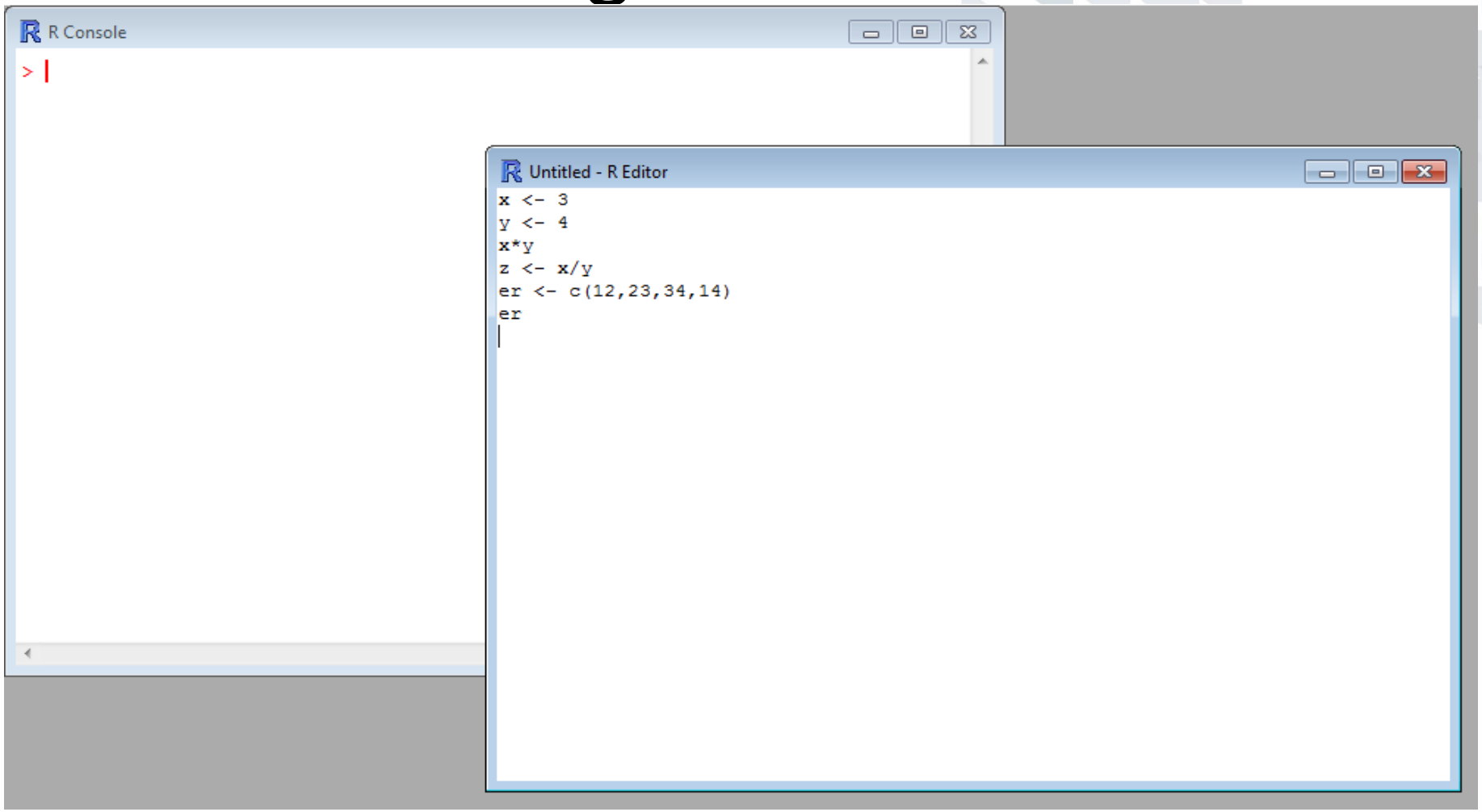
Working with Program File



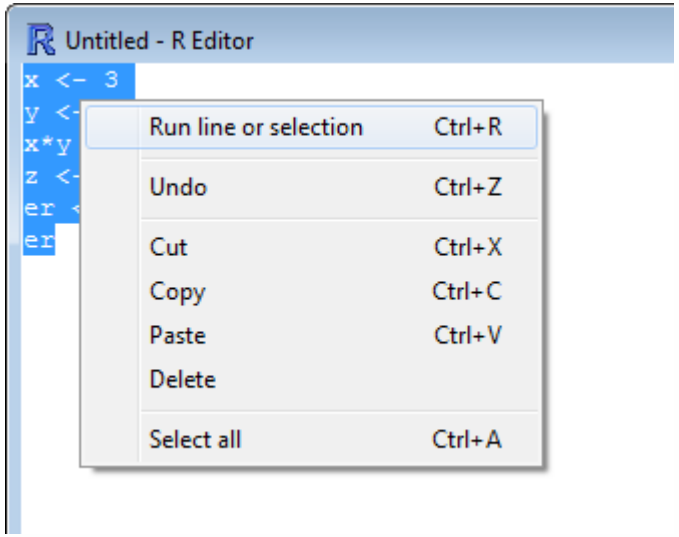
- Choose the option New Script to open the program editor

Sane's STATS Academy of Statistics

Program Editor

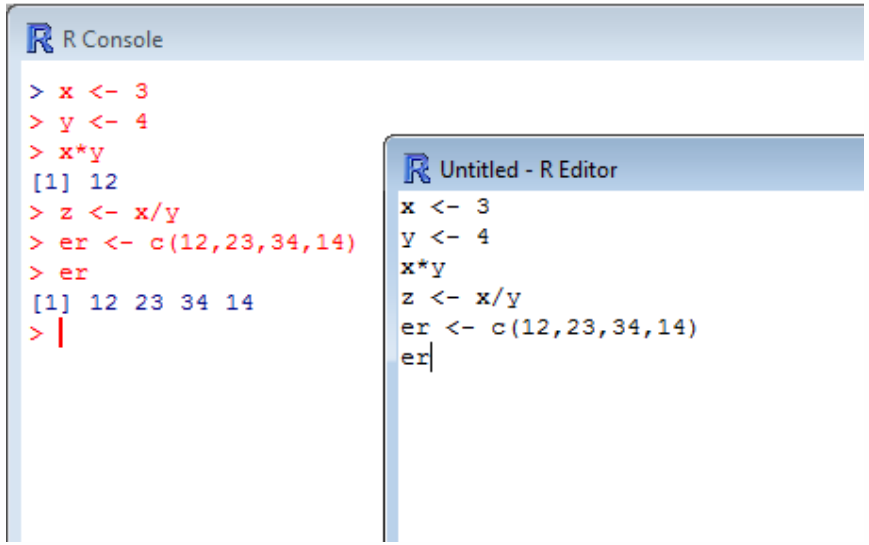


Running the code



Alternatively, we can press **Ctrl + R** for running the code

The code executed is seen on the command line as line by line command execution.



Sane's STATS Academy of Statistics

IDEs of R

- **R Studio**
- R Commander
- Tinn-R
- Eclipse with plug-in StatET
- VS Editor

Sane's STATS Academy of Statistics

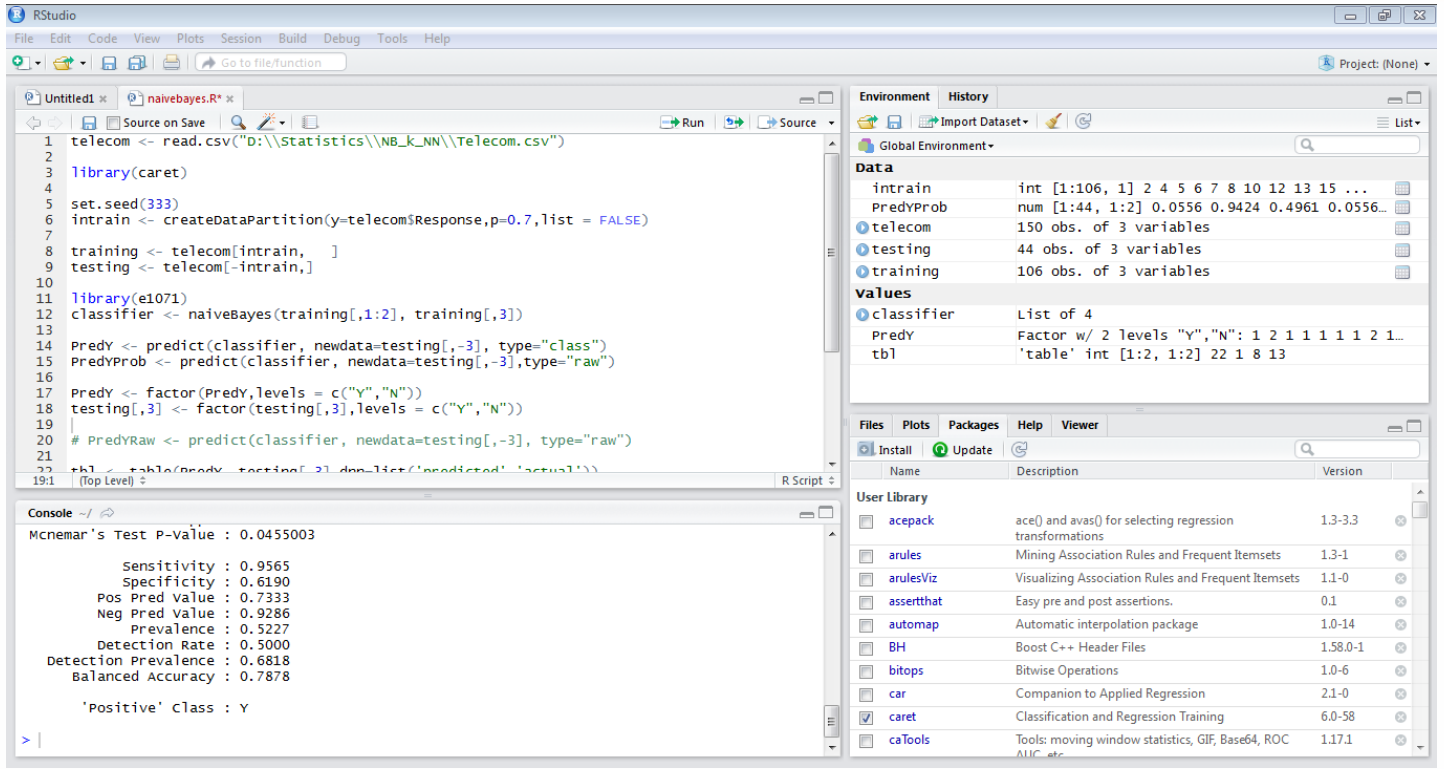
R Studio

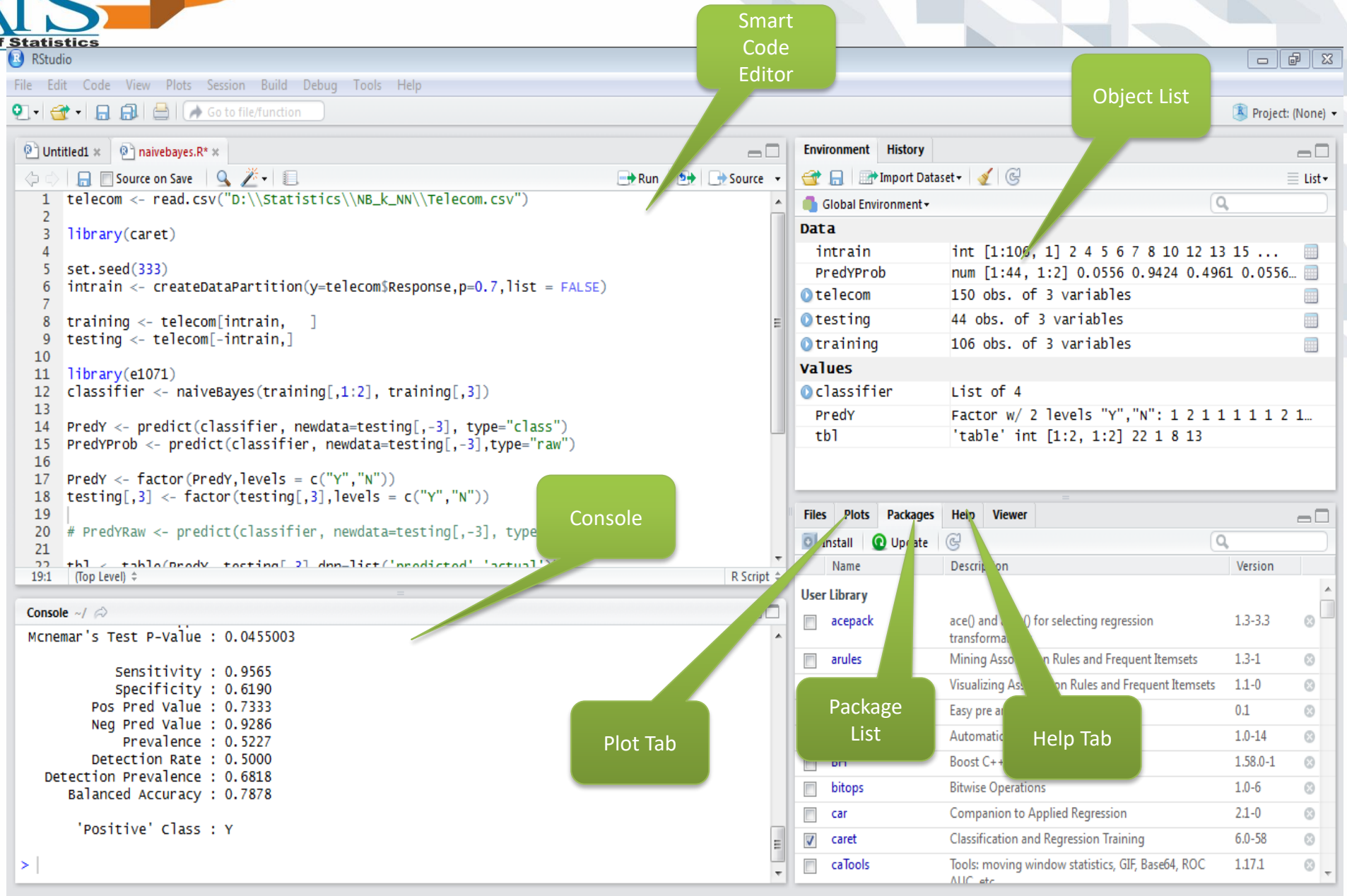
Introduction

December 18, 2021

R Studio

- It is a free and open source integrated development environment (IDE) for R




Smart Code Editor
Object List
Console
Plot Tab
Package List
Help Tab
RStudio
File Edit Code View Plots Session Build Debug Tools Help
Untitled1
naivebayes.R*
Source on Save
Run
Source
1 telecom <- read.csv("D:\\Statistics\\NB_k_NN\\Telecom.csv")
2
3 library(caret)
4
5 set.seed(333)
6 intrain <- createDataPartition(y=telecom$Response,p=0.7,list = FALSE)
7
8 training <- telecom[intrain,]
9 testing <- telecom[-intrain,]
10
11 library(e1071)
12 classifier <- naiveBayes(training[,1:2], training[,3])
13
14 PredY <- predict(classifier, newdata=testing[,-3], type="class")
15 PredYProb <- predict(classifier, newdata=testing[,-3],type="raw")
16
17 PredY <- factor(PredY,levels = c("Y","N"))
18 testing[,3] <- factor(testing[,3],levels = c("Y","N"))
19
20 # PredYRaw <- predict(classifier, newdata=testing[,-3], type
21
19:1 (Top Level) R Script
Console ~/
Mcnemar's Test P-Value : 0.0455003
Sensitivity : 0.9565
Specificity : 0.6190
Pos Pred Value : 0.7333
Neg Pred Value : 0.9286
Prevalence : 0.5227
Detection Rate : 0.5000
Detection Prevalence : 0.6818
Balanced Accuracy : 0.7878
'Positive' Class : Y
>
Environment History
Import Dataset
List
Global Environment
Data
intrain int [1:106, 1] 2 4 5 6 7 8 10 12 13 15 ...
PredYProb num [1:44, 1:2] 0.0556 0.9424 0.4961 0.0556...
telecom 150 obs. of 3 variables
testing 44 obs. of 3 variables
training 106 obs. of 3 variables
Values
classifier List of 4
PredY Factor w/ 2 levels "Y","N": 1 2 1 1 1 1 1 2 1...
tbl 'table' int [1:2, 1:2] 22 1 8 13
Files Plots Packages Help Viewer
Install Update
Name Description Version
User Library
acepack ace() and ... for selecting regression transforma... 1.3-3.3
arules Mining Asso... Rules and Frequent Itemsets 1.3-1
Visualizing As... Rules and Frequent Itemsets 1.1-0
Easy pre ar... 0.1
Automatic... 1.0-14
Boost C++... 1.58.0-1
bitops Bitwise Operations 1.0-6
car Companion to Applied Regression 2.1-0
caret Classification and Regression Training 6.0-58
caTools Tools: moving window statistics, GIF, Base64, ROC AUC, etc... 1.17.1

Sane's STATS Academy of Statistics

Basics

Sane's STATS Academy of Statistics

Features of R

- R is an interpreted language
- R is case sensitive
- R has powerful graphics with package ggplot2
- R can perform complex statistical calculations

Sane's STATS Academy of Statistics

Data Types in R

- atomic classes
- vectors, lists
- factors
- missing values
- matrices
- arrays
- data frames

Sane's STATS Academy of Statistics

Atomic Classes

- Character
  - “a”, “Good”, “Bad”
- numeric (real numbers)
  - 12.3, 0, -0.92, -1200.76
- Integer
  - 12, -13, 90, 0
- Complex
  - 2 + 4i, 5 – 5i
- logical
  - TRUE, FALSE

Sane's STATS Academy of Statistics

Vector

- A vector can only contain objects of the same atomic class
- It can be created with the help of c() function

```
> d <- c(12,23,15,19,90)
> d
[1] 12 23 15 19 90
```

```
> s <- c("ABC","L","FRE","LLL")
> s
[1] "ABC" "L"   "FRE" "LLL"
```

List

- List can contain objects of different types
- Many times we observe that the outputs of many algorithmic functions are expressed as list
- List can be created with the help of the function list()

```
> suchi <- list(1,"Amit",TRUE,3,FALSE,"Swapnil","Neha")
> suchi
[[1]]
[1] 1

[[2]]
[1] "Amit"

[[3]]
[1] TRUE

[[4]]
[1] 3

[[5]]
[1] FALSE

[[6]]
[1] "Swapnil"

[[7]]
[1] "Neha"
```

Sane's STATS Academy of Statistics

Accessing the contents in List

- Accessing members in the list is little bit different

```
> suchi <- list(1,"Amit",TRUE,3,FALSE,"Swapnil","Neha")
> suchi[1]
[[1]]
[1] 1

> suchi[5]
[[1]]
[1] FALSE
```

```
> vis <- list("DR",c(12,13,34),TRUE,13)
> vis[2]
[[1]]
[1] 12 13 34

> vis[1]
[[1]]
[1] "DR"

> vis[[2]][1]
[1] 12
```

Sane's STATS Academy of Statistics

Factor

- Factors are used to represent categorical data.
- Factors can be unordered or ordered.
- One can think of a factor as an integer vector where each integer has a *label.*
- *E.g.*
  - *Categorical variables like*
    - *Yes / No*
    - *Sold / Not Sold / On hold*

Sane's STATS Academy of Statistics

Missing Values

- Missing values are presented as NA or NaN.
- is.na() is used to test objects if they are NA
- is.nan() is used to test for Not a Number
- Also there are functions indicating finite / infinite values
- A NaN value is also NA but the NA is not always NaN.
- Best example of NaN is result of zero by zero

```
> f <- NA
> is.na(f)
[1] TRUE
> d <- c(NA,13,17,NA,13,NA)
> is.na(d)
[1]  TRUE FALSE FALSE  TRUE FALSE  TRUE
```

```
> f <- 0
> p <- 0
> g <- f/p
> is.nan(g)
[1] TRUE
> is.na(g)
[1] TRUE
```

```
> r <- 23
> p <- 0
> e <- r/p
> is.finite(e)
[1] FALSE
> is.infinite(e)
[1] TRUE
```

Sane's STATS Academy of Statistics

Matrix

- Matrix can hold the data in matrix form
- The matrix can be created with the help of R function
- Commonly used syntax arguments are
  - matrix(data , nrow , ncol)

```
> mat <- matrix(10,3,2)
> mat
     [,1] [,2]
[1,]   10   10
[2,]   10   10
[3,]   10   10
```

```
> mat2 <- matrix(c(2,3,1,4,2,4),3,2)
> mat2
     [,1] [,2]
[1,]    2    4
[2,]    3    2
[3,]    1    4
```

rbind( ) and cbind( )

- Matrices can also be created by *using* functions cbind() and rbind().
- rbind() binds the rows and cbind() binds the columns

```
> a <- 1 : 5
> a
[1] 1 2 3 4 5
> b <- 46 : 50
> b
[1] 46 47 48 49 50
> cb <- cbind(a,b)
> cb
     a  b
[1,] 1 46
[2,] 2 47
[3,] 3 48
[4,] 4 49
[5,] 5 50
```

```
> rb <- rbind(a,b)
> rb
  [,1] [,2] [,3] [,4] [,5]
a    1    2    3    4    5
b   46   47   48   49   50
```

When values specified ≠ ncols / nrows

- While using any binding function in case if no. of rows / columns are not equal to no. of values specified then the values get repeated in that order with a warning.

```
> a <- 1:3
> b <- 20:30
> rbind(a,b)
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
a    1    2    3    1    2    3    1    2    3     1     2
b   20   21   22   23   24   25   26   27   28    29    30
Warning message:
In rbind(a, b) :
  number of columns of result is not a multiple of vector length (arg 1)
> cbind(a,b)
      a  b
 [1,] 1 20
 [2,] 2 21
 [3,] 3 22
 [4,] 1 23
 [5,] 2 24
 [6,] 3 25
 [7,] 1 26
 [8,] 2 27
 [9,] 3 28
[10,] 1 29
[11,] 2 30
Warning message:
In cbind(a, b) :
  number of rows of result is not a multiple of vector length (arg 1)
```

Sane's STATS Academy of Statistics

Array

- Array can be treated as a vector with multiple dimensions
- Array can be created with the function array specifying the dimension using dim as argument
- By default the values are NA

Syntax : array(dim=c(r,c,…))

```
> k <- array(dim=c(8))
> k
[1] NA NA NA NA NA NA NA NA
```

```
> d <- array(dim=c(3,4))
> d
     [,1] [,2] [,3] [,4]
[1,]   NA   NA   NA   NA
[2,]   NA   NA   NA   NA
[3,]   NA   NA   NA   NA
```

Sane's STATS Academy of Statistics

Data Frames

- Data frame are used hold tabular data
- We can combine several vectors of same length into a data frame
- Data frames provide a simpler way for handling the data than lists
- Data frame gets usually created by calling functions like read.csv(), read.xlsx etc.

```
> a <- c(12,23,14,15)
> b <- c("X","Y","Z","W")
> df <- data.frame(a,b)
> df
   a b
1 12 X
2 23 Y
3 14 Z
4 15 W
```

```
> bollywood <- read.csv("D:/Data Science Training/Datasets/Bollywood_2015.csv")
> bollywood
                  Movie.Name BO_Collection Budget   Box_Office_Verdict
1        Pyaar Ka Punchnama 2          53.25   25.0                  Hit
2                   Shandaar          38.28   68.0                 Flop
3            Singh is Bliing          74.87   92.0                 Flop
4                     Jazbaa          24.30   30.0                 Flop
5                      Talvar          24.00   22.0                 Plus
6     Kis Kisko Pyaar Karoon          44.80   20.0                  Hit
7             Calendar Girls           7.00   12.0                 Flop
8                Katti Batti          22.96   36.0                 Flop
```

Sane's STATS Academy of Statistics

colnames

- Colnames are set of row or column names of a matrix-like object.

	Order.ID	Order.Date	Place.of.Shipment	Payment.Terms	Item.ID	Qty
1	32 90 001	31-Dec-10	Pune	Cheque	121 021	7
2	32 90 001	31-Dec-10	Pune	Cheque	121 003	49
3	32 90 001	31-Dec-10	Pune	Cheque	121 023	1
4	32 90 001	31-Dec-10	Pune	Cheque	121 018	9
5	32 90 001	31-Dec-10	Pune	Cheque	121 015	29
6	32 90 001	31-Dec-10	Pune	Cheque	121 014	44

```
> colnames(ords)
[1] "Order.ID"         "Order.Date"       "Place.of.Shipment" "Payment.Terms"     "Item.ID"
[6] "Qty"
```

names

- For knowing the names of the elements of any object names() function can be used

```
> names(df)
[1] "a" "b"
```

```
> names(bollywood)
[1] "Movie.Name"          "BO_Collection"       "Budget"
[4] "Box_Office_Verdict"
```

```
> names(ords)
[1] "Order.ID"          "Order.Date"        "Place.of.Shipment" "Payment.Terms"     "Item.ID"
[6] "Qty"
```

```
> fit <- lm(Qty ~ Payment.Terms , data = ords)
> names(fit)
 [1] "coefficients"  "residuals"     "effects"       "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "contrasts"     "xlevels"       "call"          "terms"
[13] "model"
> colnames(fit)
NULL
```

Setting the current working directory

- Instead of calling the same path multiple no. of times we can set the working directory to the frequently used file path with the help of the function setwd( )

```
> setwd("D:/Data Science Training/Datasets")
```

Hence instead of typing as…

```
> read.csv("D:/Data Science Training/Datasets/Bollywood_2015.csv")
```

We can simply type as…

```
> read.csv("Bollywood_2015.csv")
```

Sane's STATS Academy of Statistics

FILE INPUT AND OUTPUT

Sane's STATS Academy of Statistics

Reading Data

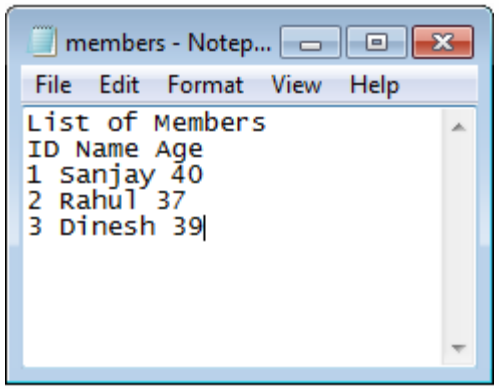
- The data from files read gets directly stored into data frame objects
- For reading data from flat (notepad) files we can use any of the functions:
  - read.csv()
  - read.table()
- For reading data from Excel format (.xlsx) we can use read.xlsx from xlsx package

read.table( ) - Some Important Arguments

```
read.table(file, header = FALSE, sep = "",
row.names, col.names, colClasses = NA, skip = 0,…)
```

- This function is a general form of read.csv( ) and read.csv2()
- The default values have been shown above
- file : the name of the file which the data are to be read from
- header : (optional) a logical value indicating whether the file contains the names of the variables as its first line. Here default is FALSE
- row.names : a vector of row names
- col.names : a vector of column names
- colClasses : a character vector indicating the class of each column in the dataset
- skip : no. of lines to skip from the top of the file
- sep : separator (optional). Default is space
- stringsAsFactors : (optional) logical indicating should the character vectors be converted to factors?

Example of read.table( )

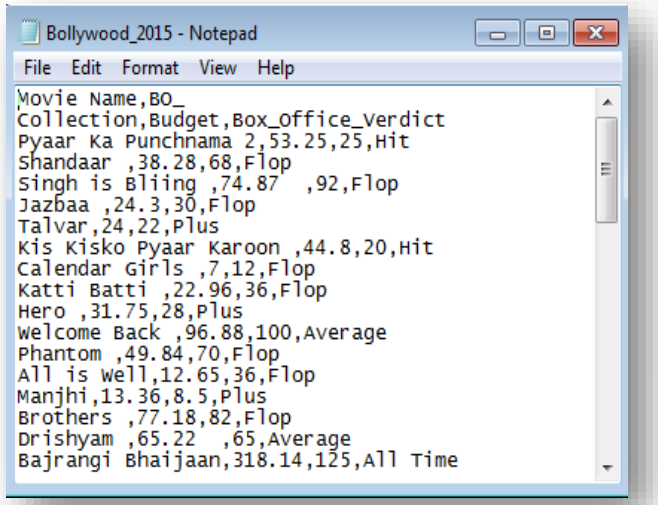



```
mem <- read.table("D:/Data Science Training/Datasets/members.txt",header=TRUE,
                  colclasses = c("character","character","integer"),skip=1,
                  sep=" ")
```

ID	Name	Age
1	Sanjay	40
2	Rahul	37
3	Dinesh	39

read.csv( )

- This function assumes comma as a delimiter by default while we read a file
- It assumes that file already has a header i.e. a line indicating the names of the variables separated by a delimiters




```
> bollywood <- read.csv("D:/Data Science Training/Datasets/Bollywood_2015.csv")
> bollywood
                  Movie.Name BO_Collection Budget   Box_Office_Verdict
1       Pyaar Ka Punchnama 2         53.25   25.0                  Hit
2                   Shandaar         38.28   68.0                 Flop
3             Singh is Bliing         74.87   92.0                 Flop
4                     Jazbaa         24.30   30.0                 Flop
5                      Talvar         24.00   22.0                 Plus
6     Kis Kisko Pyaar Karoon          44.80   20.0                  Hit
7              Calendar Girls          7.00   12.0                 Flop
8                 Katti Batti         22.96   36.0                 Flop
```

read.csv( ) arguments

```
read.csv(file, header = TRUE, sep = ",", stringsAsFactors ...)
```

- The default values have been shown above
  - file : the name of the file which the data are to be read from
  - header : (optional) a logical value indicating whether the file contains the names of the variables as its first line
  - sep : separator (optional)
  - stringsAsFactors : (optional) logical indicating should the character vectors be converted to factors?

read.csv2( ) arguments

```
read.csv2(file, header = TRUE, sep = ";", stringsAsFactors...)
```

- The default values have been shown above
  - file : the name of the file which the data are to be read from
  - header : (optional) a logical value indicating whether the file contains the names of the variables as its first line
  - sep : separator (optional). Default is semi-colon
  - stringsAsFactors : (optional) logical indicating should the character vectors be converted to factors?

Reading from MS Excel Sheet

- There are various alternatives for reading Excel sheet. One of them is provided by package xlsx
- We can use read.xlsx( ) to read the Excel sheet

Syntax: read.xlsx(file,sheetNo)

file : file path

sheetNo : sheet number of the sheet to be read

```
install.packages("xlsx")
library(xlsx)
bank <- read.xlsx("D:\\Data Science Training\\Datasets\\bankruptcy.xlsx",3)
```

Writing to a file

- For writing to file the functions write.table( ) , write.csv() etc. can be used

Syntax Usage : write.table(data frame , file path)

write.csv(data frame , file path)

```
write.table(Pens3,"Pen.txt")

write.csv(Pens3,"Pen.csv")
```

Sane's STATS Academy of Statistics

SUBSETTING THE DATA

Sane's STATS Academy of Statistics

Subsetting a vector

- A vector can be subsetted by typing various options in between the brackets [ ]

```
> x <- c(12,23,52,78,90,10,28,93,95,92,95,79)
> x[1:5]
[1] 12 23 52 78 90
> x>50
 [1] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> x[x>50]
[1] 52 78 90 93 95 92 95 79
```

Subsetting a list

- A list can be subsetted by typing various options in between the brackets [ ]

```
> f <- list(a=1:7, b="XYZ",c=FALSE,d=c(23.4,14,6))
> f[1]
$a
[1] 1 2 3 4 5 6 7

> f[2]
$b
[1] "XYZ"

> f$c
[1] FALSE
> f[["c"]]
[1] FALSE
> f["c"]
$c
[1] FALSE
```

Sane's STATS Academy of Statistics

Subsetting a matrix

- Matrix can be subsetted by specifying the row or column numbers

```
> m <- matrix(c(1:12),4,3)
> m
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> m[3,2]
[1] 7
> m[2,]
[1]  2  6 10
> m[,3]
[1]  9 10 11 12
> m[,3,drop=FALSE]
     [,1]
[1,]    9
[2,]   10
[3,]   11
[4,]   12
```

Sane's STATS Academy of Statistics

Specifying the column / row indices

- Subsetting can be done to a data frame by specifying rows / columns

```
> items[2,] # only 2nd row
  Item.ID                                          Item.Name Item.Type Brand Price  UOM
2 121 002 Pilot V7 Liquid Ink Roller Ball Pen (2 Blue + 1 Black)       Pen Pilot   135 Pack
> items[c(1,3,5,6),] # only rows 1,3,5,6
  Item.ID                                        Item.Name Item.Type    Brand Price   UOM
1 121 001          Parker Quink Roller Ball Pen Refill, Blue       Pen   Parker    69 Piece
3 121 003                   Parker Beta Premium Gold Ball Pen       Pen   Parker   125 Piece
5 121 005 Reynolds 045 Fine Carbure Blue Ballpen, Pack of 10       Pen Reynolds    60  Pack
6 121 006                 Pilot Frixion Roller Ball Pen, Blue       Pen   Parker    92 Piece
> items[,4] # only 4th column
 [1] Parker        Pilot         Parker        Pilot         Reynolds      Parker        Staedtler     Parker
 [9] Puro          Cello         Staedtler     Staedtler     Sheaffer      Lamy          Pierre Cardin Pierre Cardin
[17] Reynolds      Camlin        Camlin        Camlin        Artline       Luxor         Pilot         Camlin
[25] Artline
Levels: Artline Camlin Cello Lamy Luxor Parker Pierre Cardin Pilot Puro Reynolds Sheaffer Staedtler
> items[,c(2,4)] # only 2nd and 4th column
                                                                  Item.Name         Brand
1                                       Parker Quink Roller Ball Pen Refill, Blue        Parker
2                                 Pilot V7 Liquid Ink Roller Ball Pen (2 Blue + 1 Black)         Pilot
3                                              Parker Beta Premium Gold Ball Pen        Parker
4                           Pilot V5 Liquid Ink Roller Ball Pen - 1 Blue + 1 Black + 1 Red         Pilot
5                                 Reynolds 045 Fine Carbure Blue Ballpen, Pack of 10      Reynolds
6                                            Pilot Frixion Roller Ball Pen, Blue         Parker
7                                  Staedtler Triangular Ball Pen (Set of 10 Colours)     Staedtler
8                           Parker Vector Mettalix CT Roller Ball Pen (Blue) + Swiss Knife        Parker
```

```
> items[,-c(2,4)] # exclude 2nd and 4th column
   Item.ID   Item.Type Price   UOM
1  121 001         Pen    69 Piece
2  121 002         Pen   135  Pack
3  121 003         Pen   125 Piece
4  121 004         Pen   135  Pack
```

Subsetting the Data frame

- We can subset the data frame with the following functions:
  - which( ) function
  - subset ( ) function

which( )

- which() gives us the indices for those observations for which the condition specified is TRUE.
- Inside which() we need to specify a condition.
- It can be used as the row argument in the data frame or matrix

Syntax: which(condition)

```
> Pens <- items[which(items$Item.Type=="Pen"),]
```

	Item.ID	Item.Name	Item.Type	Brand	Price	UOM
1	121 001	Parker Quink Roller Ball Pen Refill, Blue	Pen	Parker	69	Piece
2	121 002	Pilot V7 Liquid Ink Roller Ball Pen (2 Blue + 1 Black)	Pen	Pilot	135	Pack
3	121 003	Parker Beta Premium Gold Ball Pen	Pen	Parker	125	Piece
4	121 004	Pilot V5 Liquid Ink Roller Ball Pen - 1 Blue + 1 Black + ...	Pen	Pilot	135	Pack
5	121 005	Reynolds 045 Fine Carbure Blue Ballpen, Pack of 10	Pen	Reynolds	60	Pack
6	121 006	Pilot Frixion Roller Ball Pen, Blue	Pen	Parker	92	Piece
7	121 007	Staedtler Triangular Ball Pen (Set of 10 Colours)	Pen	Staedtler	160	Pack

subset ( )

- subset() function gives the data frame object

Syntax: subset(data frame , condition, select, …)

```
> Pens2 <- subset(items,Item.Type=="Pen")
```

	Item.ID	Item.Name	Item.Type	Brand	Price	UOM
1	121 001	Parker Quink Roller Ball Pen Refill, Blue	Pen	Parker	69	Piece
2	121 002	Pilot V7 Liquid Ink Roller Ball Pen (2 Blue + 1 Black)	Pen	Pilot	135	Pack
3	121 003	Parker Beta Premium Gold Ball Pen	Pen	Parker	125	Piece
4	121 004	Pilot V5 Liquid Ink Roller Ball Pen - 1 Blue + 1 Black + …	Pen	Pilot	135	Pack
5	121 005	Reynolds 045 Fine Carbure Blue Ballpen, Pack of 10	Pen	Reynolds	60	Pack
6	121 006	Pilot Frixion Roller Ball Pen, Blue	Pen	Parker	92	Piece
7	121 007	Staedtler Triangular Ball Pen (Set of 10 Colours)	Pen	Staedtler	160	Pack

subset() Contd…

```
ExpPens <- subset(items,Item.Type=="Pen" & Price>200)
```

	Item.ID	Item.Name	Item.Type	Brand	Price	UOM
8	121 008	Parker Vector Mettalix CT Roller Ball Pen (Blue) + Swis...	Pen	Parker	316	Pack
11	121 011	Staedtler 334 SB4 Triplus Fineliner Pen - Multicolor B...	Pen	Staedtler	320	Pack
12	121 012	Staedtler Luna RiteClic Ball Pen - Transparent Body, B...	Pen	Staedtler	300	Pack
16	121 016	Pierre Cardin Kriss Satin Nickle Roller Pen and Ball Pe...	Pen	Pierre Cardin	300	Pack

```
HighRef <- subset(items,Item.Type=="Highlighter" | Item.Type=="Refill" )
```

	Item.ID	Item.Name	Item.Type	Brand	Price	UOM
14	121 014	Lamy M63 Blue Rollerball Refill	Refill	Lamy	310	Piece
18	121 018	Camlin Office Highlighter - Pack of 5 Assorted Colors	Highlighter	Camlin	100	Pack
19	121 019	Camlin Office Highlighter Pen, Yellow	Highlighter	Camlin	190	Pack
23	121 023	Pilot Frixion Colour Highlighter (Pack of 6)	Highlighter	Pilot	465	Pack

subset() Contd…

```
sItem <- subset(items , select = c(Item.ID, Brand, Price))
```

	Item.ID	Brand	Price
1	121 001	Parker	69
2	121 002	Pilot	135
3	121 003	Parker	125
4	121 004	Pilot	135
5	121 005	Reynolds	60
6	121 006	Parker	92
7	121 007	Staedtler	160
8	121 008	Parker	316
9	121 009	Puro	179
10	121 010	Cello	90
11	121 011	Staedtler	320
12	121 012	Staedtler	300

Sane's STATS Academy of Statistics

Control Structures

Sane's STATS Academy of Statistics

Control Structures

- if, else
- for loop
- while loop
- break – breaking an execution of a loop
- next – skipping an iteration

if , else Structure

- Conditional processing: If the condition given in the if() is true then the corresponding code block gets executed otherwise else code block is executed

if Syntax : if(condition) {

statements

}

If-else syntax : if(condition) {

statements

} else {

statements

}

```
a <- 34000
b <- 50000
if(a+b>10000) {
  paste("Total greater than 10000")
} else {
  paste("Total not greater than 10000")
}
```

for loop

- Loop can be created with for() using following syntax :

for(var in seq) expr

```
> for(i in 1:4) {
+   print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
```

while( ) loop

- Loop can be generated with while() using following syntax :

while(cond) expr

- So long as the condition remains true the body of loop continues to execute

```
> cnt <- 1
> while(cnt < 5) {
+   print(cnt)
+   cnt <- cnt + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
```

Sane's STATS Academy of Statistics

Breaking an execution of loop

- Breaking a loop can be possible with break statement

```
> for(i in 1:4) {
+   if(i==3) break
+   print(i)
+ }
[1] 1
[1] 2
```

Skipping an iteration of a loop

- For skipping the iteration of the loop, next statement is used

```
> for(i in 1:4) {
+    if(i==3) next
+    print(i)
+ }
[1] 1
[1] 2
[1] 4
```

Sane's STATS Academy of Statistics

Functions

Sane's STATS Academy of Statistics

Some commonly used functions…

- str
- seq
- table
- log
- exp
- ifelse
- attach

Sane's STATS Academy of Statistics

str()

- str function displays the internal structure of an R object
- It can be called as a diagnostic function, we often use to know about the object before we work on it

```
> str(items)
'data.frame':   25 obs. of  6 variables:
 $ Item.ID  : Factor w/ 25 levels "121 001","121 002",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Item.Name: Factor w/ 25 levels "Artline EK-999XF Metallic Ink Marker - Silver",..: 11 18 10 17 20 16 25
 12 19 7 ...
 $ Item.Type: Factor w/ 4 levels "Highlighter",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ Brand    : Factor w/ 12 levels "Artline","Camlin",..: 6 8 6 8 10 6 12 6 9 3 ...
 $ Price    : int  69 135 125 135 60 92 160 316 179 90 ...
 $ UOM      : Factor w/ 2 levels "Pack","Piece": 2 1 2 1 1 2 1 1 2 1 ...
```

Sane's STATS Academy of Statistics

seq()

- For sequence generation, seq() is used

Syntax :

seq(from = 1, to = 1, by = incr/decr...)

```
> seq(1,20)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
> seq(1,20,by=4)
[1]  1  5  9 13 17
```

table()

- Frequency table and Cross-tabulation can be generated with table()

Syntax: table(var1,var2,…)

```
> table(survey$Exer)

Freq None Some
 115   24   98
```

```
> table(survey$Sex)

Female   Male
   118    118
> table(survey$Sex, useNA = "ifany")

Female   Male   <NA>
   118    118      1
```

table()

- Multivariate Frequencies

```
> table(survey$Sex,survey$Exer, useNA = "ifany")

        Freq None Some
Female    49   11   58
Male      65   13   40
<NA>       1    0    0
```

log()

- For calculating the logarithm, we can use log() function

Syntax:

log(x)

log10(x)

log2(x)

log1p(x)

log() Contd...

- log computes logarithms, by default natural logarithms,
- log10 computes common (i.e., base 10) logarithms,
- log2 computes binary (i.e., base 2) logarithms.
- The general form log(x, base) computes logarithms with base base.
- log1p(x) computes *log(1+x)* accurately also for $|x| << 1$

log() and exp()

```
> log(2)
[1] 0.6931472
> log(2,10)
[1] 0.30103
> log1p(2)
[1] 1.098612
> log10(2)
[1] 0.30103
> log2(2)
[1] 1
```

```
> exp(0.6931472)
[1] 2
> 10^0.30103
[1] 2
> expm1(1.098612)
[1] 1.999999
> 10^0.30103
[1] 2
> 2^1
[1] 2
```

exp()

- exp computes the exponential function.
- expm1(x) computes exp(x) - 1 accurately also for |x| << 1.

Syntax : exp(x)
expm1(x)

```
> exp(0.6931472)
[1] 2
> expm1(0.6931472)
[1] 1
```

Sane's STATS Academy of Statistics

ifelse()

- In a simple way, we can treat ifelse() as a functional version of the if-else structure

Syntax : ifelse(condition, *true-value*, *false-value*)

- If the condition is TRUE then the function returns *true-value* otherwise it returns *false-value*

```
> v <- c(23,13,9,24,09,3,14,8,18,20)
> result <- ifelse(v > 10, "Pass","Fail")
> result
 [1] "Pass" "Pass" "Fail" "Pass" "Fail" "Fail" "Pass" "Fail"
 [9] "Pass" "Pass"
```

Mean and Variance Functions

- mean()
- sd()
- var()
- Each of the functions above have the syntax usage in the following way: function-name(variable-name,na.rm)
  - na.rm by default is FALSE. It should be set to TRUE if we want to ignore the NA values while computing

```
> mean(items$Price,na.rm = TRUE)
[1] 180.4
> sd(items$Price,na.rm = TRUE)
[1] 105.7107
> var(items$Price,na.rm = TRUE)
[1] 11174.75
```

summary()

- For numerical variables, summary function outputs the Minimum, Maximum, 1st Quartile, Median, 3rd Quartile and Mean

```
> summary(items$Price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   50.0   100.0   135.0   180.4   270.0   465.0
```

- For categorical variables, summary function outputs the frequency counts

```
> summary(items$Item.Type)
Highlighter      Marker         Pen      Refill
          3           5          16           1
```

summary()

```
> summary(items)
    Item.ID                                                    Item.Name
 121 001: 1   Artline EK-999XF Metallic Ink Marker - Silver          : 1
 121 002: 1   Artline EK157R Whiteboard Marker - Black, Pack of 10 : 1
 121 003: 1   Camlin CD - DVD Marker Pen, Blue - Pack of 10         : 1
 121 004: 1   Camlin Office Highlighter - Pack of 5 Assorted Colors: 1
 121 005: 1   Camlin Office Highlighter Pen, Yellow                 : 1
 121 006: 1   Camlin PB White Board Marker Pen, Blue                : 1
 (Other):19   (Other)                                               :19
      Item.Type             Brand       Price           UOM
 Highlighter: 3   Camlin        :4   Min.   : 50.0   Pack :15
 Marker     : 5   Parker        :4   1st Qu.:100.0   Piece:10
 Pen        :16   Pilot         :3   Median :135.0
 Refill     : 1   Staedtler     :3   Mean   :180.4
                  Artline       :2   3rd Qu.:270.0
                  Pierre Cardin:2   Max.   :465.0
                  (Other)       :7
```

- We have some few more functions in R like predict(), plot() which behave according to the class of the argument

attach()

- The data is attached to the **R** search path with attach().
- Data is searched by **R** when evaluating a variable, so objects in the data can be accessed by simply giving their names.

Syntax : attach(data)

Hence, instead of typing…

```
> table(items$Item.Type)

Highlighter      Marker         Pen      Refill
          3           5          16           1
```

```
> mean(items$Price,na.rm = TRUE)
[1] 180.4
```

It can be simply typed as…

```
> attach(items)
> table(Item.Type)
Item.Type
Highlighter      Marker         Pen      Refill
          3           5          16           1
> mean(Price)
[1] 180.4
```

Sane's STATS Academy of Statistics

Creating Functions

- Some tasks which may be repeated in different situations can be coded as a function
- A function has inputs and outputs
- Functions play a very important role in interactive graphics technologies like Tibco Spotfire and Shiny
- We create user defined functions by the following syntax:

```
Function-Name <- function(argument-list) {
                                    statements
                                }
```

Function Examples

```
add <- function(a,b,c){
  a+b+c
}

# OR

add <- function(a,b,c){
  return(a+b+c)
}
```

```
> descriptive <- function(input) {
+   df <- data.frame(Mean = mean(input,na.rm = TRUE),SD = sd(input,na.rm = TRUE))
+   df
+ }
```

Calling the function:

```
> add(23,24,12)
[1] 59
```

```
> descriptive(items$Price)
    Mean        SD
1 180.4 105.7107
```

Lab Exercises

1. Consider the dataset *survey.csv* from datasets folder. Using the appropriate functions in **dplyr** package create the following data objects:

Name	Object description	Columns
MaleNonSmokers	Males who are have never smoked	All
PulseGT80	Students whose pulse rate is greater than 80	Sex, Exer, Smoke, Pulse
RtHand	Create a variable of ratio of variables Wr.Hnd / NW.Hnd as Ratio_Hnd	Ratio_Hnd, Clap, Age
DescStats	Calculate the mean and standard deviation for the variable Age	
DescGrp	Calculate the mean and standard deviation for the variable Age grouped by Sex	

2. Given the files Items.csv, Orders.csv and Ord_Details.csv in the folder datasets, merge them with appropriate keys to form a combined data.
3. Combine the data in the files ***Courses.csv*** and ***CourseSchedule.csv*** with appropriate keys

4. Consider a dataset comb1 in the datasets folder. Reshape the dataset to the following:

Data should be like this:

	District	ItemType	qty
1	Ahmednagar	Highlighter	225
2	Aurangabad	Highlighter	162
3	Buldhana	Highlighter	30
4	Dhule	Highlighter	193
5	Goa	Highlighter	81
6	Kolhapur	Highlighter	108
7	Mumbai	Highlighter	119
8	Nadurbar	Highlighter	162
9	Nagpur	Highlighter	107
10	Nanded	Highlighter	111
11	Nasik	Highlighter	152
12	Pune	Highlighter	154
13	Raigad	Highlighter	174
14	Ratnagiri	Highlighter	326

15	Sangli	Highlighter	121
16	Satara	Highlighter	72
17	Sindhudurga	Highlighter	75
18	Solapur	Highlighter	164
19	Thane	Highlighter	224
20	Ahmednagar	Marker	51
21	Aurangabad	Marker	159
22	Buldhana	Marker	32
23	Dhule	Marker	390
24	Goa	Marker	243
25	Kolhapur	Marker	118
26	Mumbai	Marker	160
27	Nadurbar	Marker	281
28	Nagpur	Marker	121
29	Nanded	Marker	240
30	Nasik	Marker	239
31	Pune	Marker	309
32	Raigad	Marker	436
33	Ratnagiri	Marker	398
34	Sangli	Marker	110
35	Satara	Marker	156
36	Sindhudurga	Marker	217
37	Solapur	Marker	160
38	Thane	Marker	246
39	Ahmednagar	Pen	674
40	Aurangabad	Pen	755
41	Buldhana	Pen	616
42	Dhule	Pen	553
43	Goa	Pen	518
44	Kolhapur	Pen	352
45	Mumbai	Pen	391
46	Nadurbar	Pen	386
47	Nagpur	Pen	280
48	Nanded	Pen	761
49	Nasik	Pen	619
50	Pune	Pen	768
51	Raigad	Pen	545
52	Ratnagiri	Pen	1504
53	Sangli	Pen	221
54	Satara	Pen	373
55	Sindhudurga	Pen	1253
56	Solapur	Pen	391
57	Thane	Pen	733
58	Ahmednagar	Refill	69
59	Aurangabad	Refill	50
60	Buldhana	Refill	38
61	Dhule	Refill	40
62	Goa	Refill	NA
63	Kolhapur	Refill	NA
64	Mumbai	Refill	29
65	Nadurbar	Refill	156
66	Nagpur	Refill	33
67	Nanded	Refill	28
68	Nasik	Refill	65
69	Pune	Refill	122
70	Raigad	Refill	40
71	Ratnagiri	Refill	74
72	Sangli	Refill	47
73	Satara	Refill	NA
74	Sindhudurga	Refill	77
75	Solapur	Refill	15
76	Thane	Refill	58

5. Consider the dataset comb2 in the datasets folder. PatientID in the data is to be broken up into the format projectID-SiteID/PatientNumber. Create the dataset with the following:

Data View Should be:

	projectID	SiteID	PatientNumber	Lab_test	Result
1	123	984	45	A	Present
2	123	345	34	A	Absent
3	134	093	43	A	Present
4	190	032	34	B	Present
5	190	309	32	B	Absent
6	234	092	23	B	Present
7	231	902	21	B	Absent
8	231	309	21	A	Absent
9	134	092	34	B	Present
10	234	984	43	A	Present

Sane's STATS Academy of Statistics

Tidyverse

Collection of R packages

What is *tidyverse* ?

- The tidyverse is a collection of R packages designed for data science.
- All packages share an underlying design philosophy, grammar, and data structures.
- Developed by Hadley Wickham

Sane's STATS Academy of Statistics

Components of *tidyverse*

- **ggplot2:** ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics.
- **dplyr:** dplyr provides a grammar of data manipulation, providing a consistent set of verbs that solve the most common data manipulation challenges
- **tidyr:** tidyr provides a set of functions that help you get to tidy data.
- readr: readr provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf).
- purrr: purrr enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors.
- **tibble:** tibble is a modern re-imagining of the data frame, keeping what time has proven to be effective, and throwing out what it has not.
- stringr: stringr provides a cohesive set of functions designed to make working with strings as easy as possible
- forcats: forcats provides a suite of useful tools that solve common problems with factors

Loading *tidyverse*

- All packages in tidyverse can be installed and loaded at one go

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

```
library(tidyverse)
```

Sane's STATS Academy of Statistics

Package *dplyr*

Handling the Data Efficiently

class tbl_df

- We can create an object of class tbl_df. It can be created by function as_tibble

Syntax : as_tibble(objDF)

where

objDF: An object of class data.frame, or a list with each element with same length

tibble Object

- A tibble is a modern class of data frame within R
- It has a convenient print method, will not convert strings to factors, and does not use row names

```
> dd = as_tibble(mtcars)
> class(dd)
[1] "tbl_df"     "tbl"        "data.frame"
> dd
# A tibble: 32 x 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs
 * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  21        6  160    110  3.9   2.62  16.5     0
 2  21        6  160    110  3.9   2.88  17.0     0
 3  22.8      4  108     93  3.85  2.32  18.6     1
 4  21.4      6  258    110  3.08  3.22  19.4     1
 5  18.7      8  360    175  3.15  3.44  17.0     0
 6  18.1      6  225    105  2.76  3.46  20.2     1
 7  14.3      8  360    245  3.21  3.57  15.8     0
 8  24.4      4  147.    62  3.69  3.19  20       1
 9  22.8      4  141.    95  3.92  3.15  22.9     1
10  19.2      6  168.   123  3.92  3.44  18.3     1
# ... with 22 more rows, and 3 more variables:
#   am <dbl>, gear <dbl>, carb <dbl>
```

Functions in dplyr

- **arrange**: reordering rows in the data frame
- **select**: selecting columns / variables
- **filter**: selecting rows / observations
- **rename**: renaming variables
- **mutate**: adding new columns to the data frame
- **summarize** / **summarise**: generating summary statistics of the data frames

Arranging the rows

- The rows in tbl_df object can be arranged using function arrange

Syntax : arrange(Obj_tbl_df, col1,col2,…)

Where

Obj_tbl_df : tbl_df object

col1, col2,… : Columns for sorting the data

Arrange Example

```
> tbl_Cars
# A tibble: 93 x 27
   Manufacturer      Model    Type Min.Price Price Max.Price MPG.city MPG.highway
*         <fctr>     <fctr>  <fctr>     <dbl> <dbl>     <dbl>    <int>       <int>
1         Acura    Integra   Small      12.9  15.9      18.8       25          31
2         Acura     Legend Midsize      29.2  33.9      38.7       18          25
3          Audi         90 Compact      25.9  29.1      32.3       20          26
4          Audi        100 Midsize      30.8  37.7      44.6       19          26
5           BMW       535i Midsize      23.7  30.0      36.2       22          30
6         Buick    Century Midsize      14.2  15.7      17.3       22          31
7         Buick    LeSabre   Large      19.9  20.8      21.7       19          28
8         Buick Roadmaster   Large      22.6  23.7      24.9       16          25
9         Buick    Riviera Midsize      26.3  26.3      26.3       19          27
10     Cadillac    DeVille   Large      33.0  34.7      36.3       16          25
# ... with 83 more rows, and 19 more variables: AirBags <fctr>, DriveTrain <fctr>,
#   Cylinders <fctr>, EngineSize <dbl>, Horsepower <int>, RPM <int>, Rev.per.mile <int>,
#   Man.trans.avail <fctr>, Fuel.tank.capacity <dbl>, Passengers <int>, Length <int>,
#   Wheelbase <int>, Width <int>, Turn.circle <int>, Rear.seat.room <dbl>,
#   Luggage.room <int>, Weight <int>, Origin <fctr>, Make <fctr>
```

```
> ord_Model <- arrange(tbl_Cars , Model)
> ord_Model
# A tibble: 93 x 27
    Manufacturer  Model    Type Min.Price Price Max.Price MPG.city MPG.highway
          <fctr> <fctr>  <fctr>     <dbl> <dbl>     <dbl>    <int>       <int>
1           Audi    100 Midsize      30.8  37.7      44.6       19          26
2  Mercedes-Benz   190E Compact      29.0  31.9      34.9       20          29
3          Volvo    240 Compact      21.8  22.7      23.5       21          28
4  Mercedes-Benz   300E Midsize      43.8  61.9      80.0       19          25
5          Mazda    323   Small       7.4   8.3       9.1       29          37
6            BMW   535i Midsize      23.7  30.0      36.2       22          30
7          Mazda    626 Compact      14.3  16.5      18.7       26          34
8          Volvo    850 Midsize      24.8  26.7      28.5       20          28
9           Audi     90 Compact      25.9  29.1      32.3       20          26
10          Saab    900 Compact      20.3  28.7      37.1       20          26
# ... with 83 more rows, and 19 more variables: AirBags <fctr>, DriveTrain <fctr>,
#   Cylinders <fctr>, EngineSize <dbl>, Horsepower <int>, RPM <int>, Rev.per.mile <int>,
#   Man.trans.avail <fctr>, Fuel.tank.capacity <dbl>, Passengers <int>, Length <int>,
#   Wheelbase <int>, Width <int>, Turn.circle <int>, Rear.seat.room <dbl>,
#   Luggage.room <int>, Weight <int>, Origin <fctr>, Make <fctr>
```

Arranging Multiple Columns Example

```
> ord_mnf_mdl <- arrange(tbl_Cars,Manufacturer,Model)
> ord_mnf_mdl
# A tibble: 93 x 27
   Manufacturer      Model    Type Min.Price Price Max.Price MPG.city MPG.highway
          <fctr>     <fctr>  <fctr>     <dbl> <dbl>     <dbl>    <int>       <int>
1         Acura    Integra   Small      12.9  15.9      18.8       25          31
2         Acura     Legend Midsize      29.2  33.9      38.7       18          25
3          Audi        100 Midsize      30.8  37.7      44.6       19          26
4          Audi         90 Compact      25.9  29.1      32.3       20          26
5           BMW       535i Midsize      23.7  30.0      36.2       22          30
6         Buick    Century Midsize      14.2  15.7      17.3       22          31
7         Buick    LeSabre   Large      19.9  20.8      21.7       19          28
8         Buick    Riviera Midsize      26.3  26.3      26.3       19          27
9         Buick Roadmaster   Large      22.6  23.7      24.9       16          25
10     Cadillac    DeVille   Large      33.0  34.7      36.3       16          25
# ... with 83 more rows, and 19 more variables: AirBags <fctr>, DriveTrain <fctr>,
#   Cylinders <fctr>, EngineSize <dbl>, Horsepower <int>, RPM <int>, Rev.per.mile <int>,
#   Man.trans.avail <fctr>, Fuel.tank.capacity <dbl>, Passengers <int>, Length <int>,
#   Wheelbase <int>, Width <int>, Turn.circle <int>, Rear.seat.room <dbl>,
#   Luggage.room <int>, Weight <int>, Origin <fctr>, Make <fctr>
```

Arranging Multiple Columns Example

```
> ord_mnf_mdl <- arrange(tbl_Cars,Manufacturer,desc(Model))
> ord_mnf_mdl
# A tibble: 93 x 27
   Manufacturer      Model    Type Min.Price Price Max.Price MPG.city MPG.highway
          <fctr>     <fctr>  <fctr>     <dbl> <dbl>     <dbl>    <int>       <int>
1         Acura     Legend Midsize      29.2  33.9      38.7       18          25
2         Acura    Integra   Small      12.9  15.9      18.8       25          31
3          Audi         90 Compact      25.9  29.1      32.3       20          26
4          Audi        100 Midsize      30.8  37.7      44.6       19          26
5           BMW       535i Midsize      23.7  30.0      36.2       22          30
6         Buick Roadmaster   Large      22.6  23.7      24.9       16          25
7         Buick    Riviera Midsize      26.3  26.3      26.3       19          27
8         Buick    LeSabre   Large      19.9  20.8      21.7       19          28
9         Buick    Century Midsize      14.2  15.7      17.3       22          31
10     Cadillac    Seville Midsize      37.5  40.1      42.7       16          25
# ... with 83 more rows, and 19 more variables: AirBags <fctr>, DriveTrain <fctr>,
#   Cylinders <fctr>, EngineSize <dbl>, Horsepower <int>, RPM <int>, Rev.per.mile <int>,
#   Man.trans.avail <fctr>, Fuel.tank.capacity <dbl>, Passengers <int>, Length <int>,
#   Wheelbase <int>, Width <int>, Turn.circle <int>, Rear.seat.room <dbl>,
#   Luggage.room <int>, Weight <int>, Origin <fctr>, Make <fctr>
```

Selecting Columns

- For selecting specific columns, we can use select function

Syntax : select(objtbl , col 1, col 2, ...)

OR

select(objtbl , col 1:col n)

where
objtbl: Object of class tbl_df

Select Examples

```
> select(tbl_Cars,1:3)
# A tibble: 93 x 3
   Manufacturer       Model    Type
*         <fctr>      <fctr>  <fctr>
1          Acura     Integra   Small
2          Acura      Legend Midsize
3           Audi          90 Compact
4           Audi         100 Midsize
5            BMW        535i Midsize
6          Buick     Century Midsize
7          Buick     LeSabre   Large
8          Buick Roadmaster   Large
9          Buick     Riviera Midsize
10      Cadillac     DeVille   Large
# ... with 83 more rows
```

```
> select(tbl_Cars,Model:Max.Price)
# A tibble: 93 x 5
        Model    Type Min.Price Price Max.Price
*      <fctr>  <fctr>     <dbl> <dbl>     <dbl>
1     Integra   Small      12.9  15.9      18.8
2      Legend Midsize      29.2  33.9      38.7
3          90 Compact      25.9  29.1      32.3
4         100 Midsize      30.8  37.7      44.6
5        535i Midsize      23.7  30.0      36.2
6     Century Midsize      14.2  15.7      17.3
7     LeSabre   Large      19.9  20.8      21.7
8  Roadmaster   Large      22.6  23.7      24.9
9     Riviera Midsize      26.3  26.3      26.3
10    DeVille   Large      33.0  34.7      36.3
# ... with 83 more rows
```

```
> select(tbl_Cars, ends_with("Price"))
# A tibble: 93 x 3
   Min.Price Price Max.Price
*      <dbl> <dbl>     <dbl>
1       12.9  15.9      18.8
2       29.2  33.9      38.7
3       25.9  29.1      32.3
4       30.8  37.7      44.6
5       23.7  30.0      36.2
6       14.2  15.7      17.3
7       19.9  20.8      21.7
8       22.6  23.7      24.9
9       26.3  26.3      26.3
10      33.0  34.7      36.3
# ... with 83 more rows
```

```
> select(tbl_Cars, starts_with("MPG"))
# A tibble: 93 x 2
   MPG.city MPG.highway
*     <int>       <int>
1        25          31
2        18          25
3        20          26
4        19          26
5        22          30
6        22          31
7        19          28
8        16          25
9        19          27
10       16          25
# ... with 83 more rows
```

Sane's STATS Academy of Statistics

Subsetting the data

- The data can be subsetted with function filter

Syntax : filter(objtbl , criteria)

where

objtbl: Object of class tbl_df

criteria: Condition of filtering

filter examples

```
> filter(tbl_Cars, Type=="Small")
# A tibble: 21 x 27
   Manufacturer   Model   Type Min.Price Price Max.Price MPG.city MPG.highway     AirBags DriveTrain
          <fctr>  <fctr> <fctr>     <dbl> <dbl>     <dbl>    <int>       <int>      <fctr>     <fctr>
1         Acura Integra  Small      12.9  15.9      18.8       25          31        None      Front
2         Dodge    Colt  Small       7.9   9.2      10.6       29          33        None      Front
3         Dodge  Shadow  Small       8.4  11.3      14.2       23          29 Driver only      Front
4         Eagle  Summit  Small       7.9  12.2      16.5       29          33        None      Front
5          Ford Festiva  Small       6.9   7.4       7.9       31          33        None      Front
6          Ford  Escort  Small       8.4  10.1      11.9       23          30        None      Front
7           Geo   Metro  Small       6.7   8.4      10.0       46          50        None      Front
8         Honda   Civic  Small       8.4  12.1      15.8       42          46 Driver only      Front
9       Hyundai   Excel  Small       6.8   8.0       9.2       29          33        None      Front
10      Hyundai Elantra  Small       9.0  10.0      11.0       22          29        None      Front
# ... with 11 more rows, and 17 more variables: Cylinders <fctr>, EngineSize <dbl>, Horsepower <int>,
#   RPM <int>, Rev.per.mile <int>, Man.trans.avail <fctr>, Fuel.tank.capacity <dbl>,
#   Passengers <int>, Length <int>, Wheelbase <int>, Width <int>, Turn.circle <int>,
#   Rear.seat.room <dbl>, Luggage.room <int>, Weight <int>, Origin <fctr>, Make <fctr>
```

```
> filter(tbl_Cars, Type=="Small" & Max.Price<10)
# A tibble: 6 x 27
  Manufacturer   Model   Type Min.Price Price Max.Price MPG.city MPG.highway AirBags DriveTrain
         <fctr>  <fctr> <fctr>     <dbl> <dbl>     <dbl>    <int>       <int>  <fctr>     <fctr>
1         Ford Festiva  Small       6.9   7.4       7.9       31          33    None      Front
2      Hyundai   Excel  Small       6.8   8.0       9.2       29          33    None      Front
3        Mazda     323  Small       7.4   8.3       9.1       29          37    None      Front
4      Pontiac  LeMans  Small       8.2   9.0       9.9       31          41    None      Front
5       Subaru   Justy  Small       7.3   8.4       9.5       33          37    None        4WD
6   Volkswagen     Fox  Small       8.7   9.1       9.5       25          33    None      Front
# ... with 17 more variables: Cylinders <fctr>, EngineSize <dbl>, Horsepower <int>, RPM <int>,
#   Rev.per.mile <int>, Man.trans.avail <fctr>, Fuel.tank.capacity <dbl>, Passengers <int>,
#   Length <int>, Wheelbase <int>, Width <int>, Turn.circle <int>, Rear.seat.room <dbl>,
#   Luggage.room <int>, Weight <int>, Origin <fctr>, Make <fctr>
```

filter examples

```
> filter(tbl_cars, Manufacturer %in% c("Acura","Audi"))
Source: local data frame [4 x 27]

  Manufacturer   Model    Type Min.Price Price Max.Price MPG.city MPG.highway
        (fctr)  (fctr)  (fctr)     (dbl) (dbl)     (dbl)    (int)       (int)
1        Acura Integra   Small      12.9  15.9      18.8       25          31
2        Acura  Legend Midsize      29.2  33.9      38.7       18          25
3         Audi      90 Compact      25.9  29.1      32.3       20          26
4         Audi     100 Midsize      30.8  37.7      44.6       19          26
Variables not shown: AirBags (fctr), DriveTrain (fctr), Cylinders (fctr),
  EngineSize (dbl), Horsepower (int), RPM (int), Rev.per.mile (int),
  Man.trans.avail (fctr), Fuel.tank.capacity (dbl), Passengers (int), Length
  (int), Wheelbase (int), Width (int), Turn.circle (int), Rear.seat.room (dbl),
  Luggage.room (int), Weight (int), Origin (fctr), Make (fctr)
```

Renaming Columns

- The columns can renamed with function rename()

Syntax : rename(objtbl, newname1=oldname1, newname2=oldname2,…)

where

objtbl: Object of class tbl_df

rename example

```
> rename(tbl_Cars,Minimum=Min.Price, Maximum=Max.Price)
# A tibble: 93 x 27
   Manufacturer      Model    Type Minimum Price Maximum MPG.city MPG.highway
*         <fctr>     <fctr>  <fctr>   <dbl> <dbl>   <dbl>    <int>       <int>
1         Acura    Integra   Small    12.9  15.9    18.8       25          31
2         Acura     Legend Midsize    29.2  33.9    38.7       18          25
3          Audi         90 Compact    25.9  29.1    32.3       20          26
4          Audi        100 Midsize    30.8  37.7    44.6       19          26
5           BMW       535i Midsize    23.7  30.0    36.2       22          30
6         Buick    Century Midsize    14.2  15.7    17.3       22          31
7         Buick    LeSabre   Large    19.9  20.8    21.7       19          28
8         Buick Roadmaster   Large    22.6  23.7    24.9       16          25
9         Buick    Riviera Midsize    26.3  26.3    26.3       19          27
10     Cadillac    DeVille   Large    33.0  34.7    36.3       16          25
# ... with 83 more rows, and 19 more variables: AirBags <fctr>, DriveTrain <fctr>,
#   Cylinders <fctr>, EngineSize <dbl>, Horsepower <int>, RPM <int>, Rev.per.mile <int>,
#   Man.trans.avail <fctr>, Fuel.tank.capacity <dbl>, Passengers <int>, Length <int>,
#   Wheelbase <int>, Width <int>, Turn.circle <int>, Rear.seat.room <dbl>,
#   Luggage.room <int>, Weight <int>, Origin <fctr>, Make <fctr>
```

Adding new column

- We can create one or more new columns / variables in the data with function mutate

Syntax : mutate(objtbl , assign)

where

objtbl: Object of class tbl_df

assign: Specification of assignment for new column

mutate example

```
tbl_Cars_rng <- mutate(tbl_Cars , Price_Range = Max.Price - Min.Price ,
                               ratio = Weight/Passengers)
```

```
> select(tbl_Cars_rng,Model,Price_Range,ratio)
# A tibble: 93 x 3
         Model Price_Range    ratio
        <fctr>       <dbl>    <dbl>
1     Integra         5.9 541.0000
2      Legend         9.5 712.0000
3          90         6.4 675.0000
4         100        13.8 567.5000
5        535i        12.5 910.0000
6     Century         3.1 480.0000
7     LeSabre         1.8 578.3333
8  Roadmaster         2.3 684.1667
9     Riviera         0.0 699.0000
10    DeVille         3.3 603.3333
# ... with 83 more rows
```

Summarizing the data

- The data can be summarized with the function summarize/summarise

Syntax : summarize(objtbl, assign)

where

objtbl: Object of class tbl_df

assign: Specification of assignment for new column

summarize example

```
> summarize(tbl_Cars, avg_Price = mean(Price,na.rm = TRUE),
+           sd_engSize = sd(EngineSize,na.rm = TRUE))
Source: local data frame [1 x 2]

  avg_Price sd_engSize
      (dbl)      (dbl)
1  19.50968   1.037363
```

Grouping

- The group_by function takes an existing tbl and converts it into a grouped tbl where operations are performed "by group".

Syntax : group_by(objtbl)

where

objtbl: Object of class tbl_df

Group by example

```
> by_Air_Origin <- group_by(tbl_Cars, Origin,AirBags)
> summarise(by_Air_Origin, avg_Price = mean(Price,na.rm = TRUE),
+           sd_engSize = sd(EngineSize,na.rm = TRUE))
Source: local data frame [6 x 4]
Groups: Origin [?]

   Origin            AirBags avg_Price sd_engSize
   (fctr)             (fctr)     (dbl)      (dbl)
1     USA Driver & Passenger  24.57778  0.5600099
2     USA        Driver only  19.86957  1.2303796
3     USA               None  13.33125  0.9949874
4 non-USA Driver & Passenger  33.24286  0.4270608
5 non-USA        Driver only  22.78000  0.7680974
6 non-USA               None  13.03333  0.5883676
```

Chaining / Pipelining

- We can pipeline the operations which are consecutive to one tbl object using %>% operator.

Syntax :

objtbl %>% operations

where

objtbl: Object of class tbl_df

Pipelining the data operations

```
##Instead of
filter(select(tbl_Cars, Model,Price, Type) , Type=="Small")

## We can type
tbl_Cars %>%
  select(Model,Price, Type) %>%
  filter(Type=="Small")
```

```
#Considering
x1 <- 1:5; x2 <- 2:6

##Instead of
sqrt(sum((x1-x2)^2))

## We can type
(x1-x2)^2 %>% sum() %>% sqrt()
```

Sane's STATS Academy of Statistics

Joins

- In package ***dplyr***, we have all the types of joins like
  - Inner join
  - Left Join
  - Right Join
  - Full Join

Inner Join

```
> A
  IdNum   A
1     1 234
2     2 134
3     3 145
4     4 653
5     5 246
```

```
> B
  IdNum    B
1     1  200
2     2  100
3     3 1444
4     6  400
5     7  160
```

```
> inner_join(A,B,by="IdNum")
  IdNum   A    B
1     1 234  200
2     2 134  100
3     3 145 1444
```

Left Outer Join

```
> A
  IdNum   A
1     1 234
2     2 134
3     3 145
4     4 653
5     5 246
```

```
> B
  IdNum    B
1     1  200
2     2  100
3     3 1444
4     6  400
5     7  160
```

```
> left_join(A,B,by="IdNum")
  IdNum   A    B
1     1 234  200
2     2 134  100
3     3 145 1444
4     4 653   NA
5     5 246   NA
```

Right Outer Join

```
> A
  IdNum   A
1     1 234
2     2 134
3     3 145
4     4 653
5     5 246
```

```
> B
  IdNum    B
1     1  200
2     2  100
3     3 1444
4     6  400
5     7  160
```

```
> right_join(A,B,by="IdNum")
  IdNum   A    B
1     1 234  200
2     2 134  100
3     3 145 1444
4     6  NA  400
5     7  NA  160
```

Sane's STATS Academy of Statistics

Full Outer Join

```
> A
  IdNum   A
1     1 234
2     2 134
3     3 145
4     4 653
5     5 246
```

```
> B
  IdNum    B
1     1  200
2     2  100
3     3 1444
4     6  400
5     7  160
```

```
> full_join(A,B,by="IdNum")
  IdNum   A    B
1     1 234  200
2     2 134  100
3     3 145 1444
4     4 653   NA
5     5 246   NA
6     6  NA  400
7     7  NA  160
```

Sane's STATS Academy of Statistics

Sane's STATS Academy of Statistics

Package *tidyr* in R

Introduction

A same data can be represented or captured in multiple ways as shown here. Also all are not equally easy to use

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

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

country	year	Type	Count
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	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

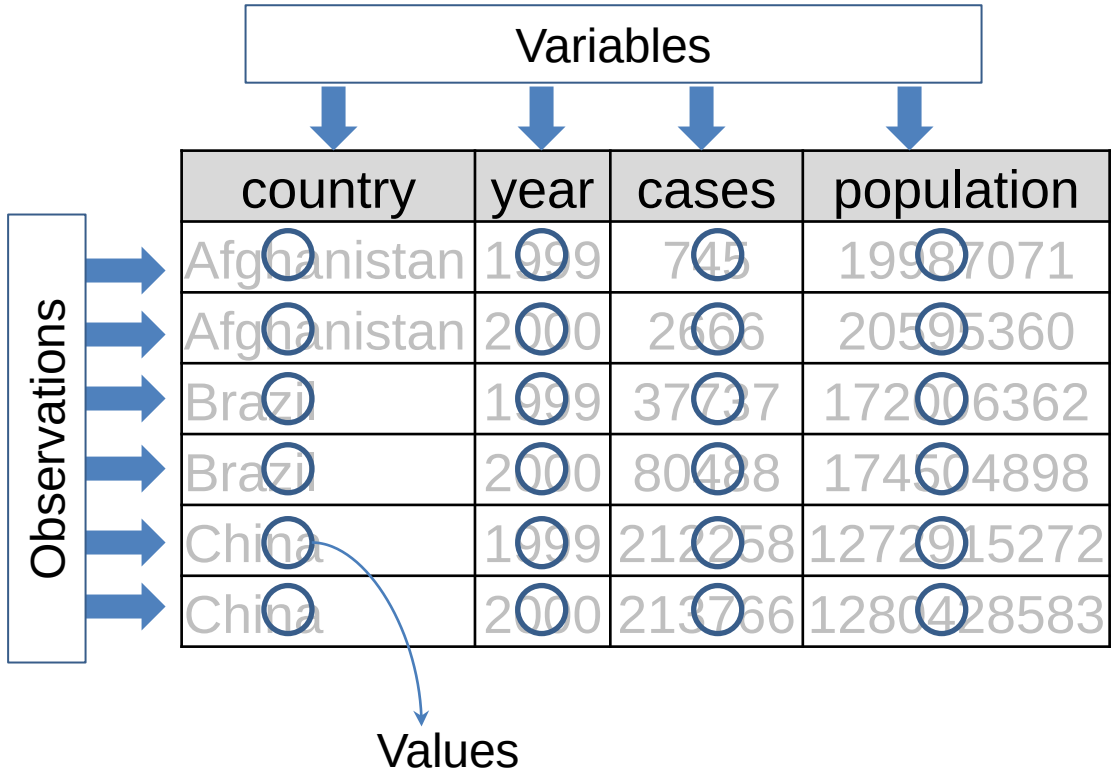
country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

country	1999	2000
Afghanistan	19987071	20595360
Brazil	172006362	174504898
China	1272915272	1280428583

Tidy data

Following are the three interrelated rules which makes a data set tidy

- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell



That interrelationship leads to an even simpler set of practical instructions

1. Put each dataset in a tibble
2. Put each variable in a column

Prerequisites

tidyr is a member of the core tidyverse

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

Spreading and gathering

We need to resolve two common problems in the data

1. One variable might be spread across multiple columns
2. One observation might be scattered across multiple rows

1. the column names 1999 and 2000 represent values of the year variable

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

country	year	Type	Count
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	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

2. Observation is scattered across multiple rows. an observation is a country in a year, but each observation is spread across two rows

Gathering

We need to gather where

1. some of the column names are not names of variables, but values of a variable i.e. one variable spread across multiple columns
2. Each row represents two observations, not one

```
> table4a
# A tibble: 3 x 3
      country `1999` `2000`
*       <chr>  <int>  <int>
1 Afghanistan    745   2666
2      Brazil  37737  80488
3       China 212258 213766
```

column names 1999 and 2000 represent values of the year variable

To **gather** those columns into a new pair of variables. we need three parameters:

1. The set of columns that represent values, not variables.
2. The name of the variable whose values form the column names. It is called the **Key**
3. The name of the variable whose values are spread over the cells. It is called **Value**

Syntax: gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

Gathering

Syntax: gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

```
> table4a
# A tibble: 3 x 3
      country `1999` `2000`
*       <chr>  <int>  <int>
1 Afghanistan    745   2666
2      Brazil  37737  80488
3       China 212258 213766
```

```
> table4a %>% gather(`1999`, `2000`, key= "year", value= "cases")
# A tibble: 6 x 3
      country  year  cases
        <chr> <chr>  <int>
1 Afghanistan  1999    745
2      Brazil  1999  37737
3       China  1999 212258
4 Afghanistan  2000   2666
5      Brazil  2000  80488
6       China  2000 213766
```

```
> table4b
# A tibble: 3 x 3
      country     `1999`     `2000`
*       <chr>      <int>      <int>
1 Afghanistan   19987071   20595360
2      Brazil  172006362  174504898
3       China 1272915272 1280428583
```

```
> table4b %>% gather(`1999`, `2000`, key= "year", value= "population")
# A tibble: 6 x 3
      country  year population
        <chr> <chr>      <int>
1 Afghanistan  1999   19987071
2      Brazil  1999  172006362
3       China  1999 1272915272
4 Afghanistan  2000   20595360
5      Brazil  2000  174504898
6       China  2000 1280428583
```

Note that “1999” and “2000” are non-syntactic names so need to surround them in backticks

Spreading

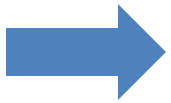
Spreading is the opposite of gathering. We use it when an observation is scattered across multiple rows. we need two parameters:

1. The column that contains variable names, the **key** column
2. The column that contains values forms multiple variables, the **value** column

Syntax: spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

an observation is a country in a year, but each observation is spread across two rows

```
> table2
# A tibble: 12 × 4
       country  year       type      count
         <chr> <int>      <chr>      <int>
1  Afghanistan  1999      cases        745
2  Afghanistan  1999 population   19987071
3  Afghanistan  2000      cases       2666
4  Afghanistan  2000 population   20595360
5       Brazil  1999      cases      37737
6       Brazil  1999 population  172006362
7       Brazil  2000      cases      80488
8       Brazil  2000 population  174504898
9        China  1999      cases     212258
10       China  1999 population 1272915272
11       China  2000      cases     213766
12       China  2000 population 1280428583
```



```
> spread(table2, key = "type", value = "count")
# A tibble: 6 × 4
      country  year  cases population
*       <chr> <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3      Brazil  1999  37737  172006362
4      Brazil  2000  80488  174504898
5       China  1999 212258 1272915272
6       China  2000 213766 1280428583
```

gather() makes wide tables narrower and longer;
spread() makes long tables shorter and wider

Separate

separate() pulls apart one column into multiple columns, by splitting wherever a separator character appears. We can use this when one column contains two variables

Syntax: separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, ...)

```
> table3
# A tibble: 6 × 3
      country  year              rate
*       <chr> <int>             <chr>
1 Afghanistan  1999      745/19987071
2 Afghanistan  2000     2666/20595360
3      Brazil  1999   37737/172006362
4      Brazil  2000   80488/174504898
5       China  1999 212258/1272915272
6       China  2000 213766/1280428583
```

By default, separate() will split values wherever it sees a non-alphanumeric character. We can convert to better types using convert = TRUE

```
> table3 %>% separate(rate, into = c("cases", "population"), sep ="/", convert = TRUE)
# A tibble: 6 × 4
      country  year  cases population
*       <chr> <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3      Brazil  1999  37737  172006362
4      Brazil  2000  80488  174504898
5       China  1999 212258 1272915272
6       China  2000 213766 1280428583
```

Separate cntd…

•You can also pass a vector of integers to **sep**.
•separate() will interpret the integers as positions to split at. Positive values start at 1 on the far-left of the strings; negative value start at -1 on the far-right of the strings.
•When using integers to separate strings, the length of **sep** should be one less than the number of names in the **into** option

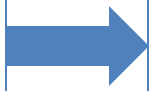
```
> table3 %>% separate(year, into = c("century", "year"), sep =2)
# A tibble: 6 x 4
      country century  year              rate
*       <chr>   <chr> <chr>             <chr>
1 Afghanistan      19    99      745/19987071
2 Afghanistan      20    00     2666/20595360
3      Brazil      19    99   37737/172006362
4      Brazil      20    00   80488/174504898
5       China      19    99 212258/1272915272
6       China      20    00 213766/1280428583
```

Unite

unite() is the inverse of separate(): it combines multiple columns into a single column

Syntax: unite(data, col, ..., sep = "_", remove = TRUE)

```
> table5
# A tibble: 6 × 4
      country century  year              rate
*       <chr>   <chr> <chr>             <chr>
1 Afghanistan      19    99      745/19987071
2 Afghanistan      20    00     2666/20595360
3      Brazil      19    99   37737/172006362
4      Brazil      20    00   80488/174504898
5       China      19    99 212258/1272915272
6       China      20    00 213766/1280428583
```



```
> table5 %>% unite(new,century, year, sep = "" )
# A tibble: 6 × 3
      country   new              rate
*       <chr> <chr>             <chr>
1 Afghanistan  1999      745/19987071
2 Afghanistan  2000     2666/20595360
3      Brazil  1999   37737/172006362
4      Brazil  2000   80488/174504898
5       China  1999 212258/1272915272
6       China  2000 213766/1280428583
```

The default will place an underscore (_) between the values from different columns. Here we don't want any separator so we use ""

Missing values

Value can be missing in one of two possible ways

- Explicitly, i.e. flagged with NA
- Implicitly, i.e. simply not present in the data

	year	qtr	return
1	2015	1	1.88
2	2015	2	0.59
3	2015	3	0.35
4	2015	4	*NA*
5	2016	2	0.92
6	2016	3	0.17
7	2016	4	2.66

There are two missing values in this dataset

- The return for the fourth quarter of 2015 is explicitly missing. As the cell contains *NA*
- The return for the first quarter of 2016 is implicitly missing. Does not appear in the dataset.

Missing values cntd..

We can make the implicit missing value explicit by putting years in the columns

```
> stocks %>%  spread(key = year, value = return)
# A tibble: 4 × 3
    qtr `2015` `2016`
* <dbl>  <dbl>  <dbl>
1     1   1.88     NA
2     2   0.59   0.92
3     3   0.35   0.17
4     4     NA   2.66
```

We can also use drop_na() function

```
> stocks %>%  drop_na()
# A tibble: 6 × 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2016     2   0.92
5  2016     3   0.17
6  2016     4   2.66
```

Complete function for Missing values

This is another important tool to turns implicit missing values into explicit missing values. complete() takes a set of columns, and finds all unique combinations. It then ensures the original dataset contains all those values, filling in explicit NAs where necessary

Syntax: complete(data, ..., fill = list())

```
> stocks %>% complete(year,qtr)
# A tibble: 8 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4     NA
5  2016     1     NA
6  2016     2   0.92
7  2016     3   0.17
8  2016     4   2.66
```

```
> # Imputing by Mean
> mu_return <- mean(stocks$return,na.rm = T)
> stocks %>% complete(year,qtr,
+                     fill = list(return=mu_return))
# A tibble: 8 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4   1.10
5  2016     1   1.10
6  2016     2   0.92
7  2016     3   0.17
8  2016     4   2.66
```

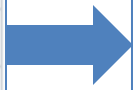
fill function for Missing values

Sometimes missing values indicate that the previous value should be carried forward.

We can fill in these missing values with fill(). Fills missing values in using the previous entry

Syntax: fill(data, ..., .direction = c("down", "up"))

	Channel	Program	Adrate
1	SAB TV	Tarak Mehta	600
2	NA	Chidiya Ghar	450
3	NA	FIR	250
4	Star Plus	Chandra	750
5	NA	Namkaran	550



```
> TVrate %>% fill(Channel)
# A tibble: 5 x 3
    Channel       Program Adrate
      <chr>         <chr>  <dbl>
1    SAB TV  Tarak Mehta    600
2    SAB TV Chidiya Ghar    450
3    SAB TV           FIR    250
4 Star Plus       Chandra    750
5 Star Plus      Namkaran    550
```

Filling values

```
> stocks
# A tibble: 7 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4     NA
5  2016     2   0.92
6  2016     3   0.17
7  2016     4   2.66
```

```
> stocks %>%
+   complete(year,qtr) %>%
+   fill(return)
# A tibble: 8 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4   0.35
5  2016     1   0.35
6  2016     2   0.92
7  2016     3   0.17
8  2016     4   2.66
```

```
> stocks %>%
+   complete(year,qtr) %>%
+   fill(return,.direction = "up")
# A tibble: 8 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4   0.92
5  2016     1   0.92
6  2016     2   0.92
7  2016     3   0.17
8  2016     4   2.66
```

Reading JSON

What is JSON?

- Javascript Object Notation
- Format of Data used for many APIs
- Data can be stored as
  - Numbers
  - Strings
  - Arrays
  - Objects

JSON Data

- JSON data is written as name/value pairs.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

```
{ “name”:”C-DAC” }
```

- Valid data types:
  - a string
  - a number
  - an object (JSON object)
  - an array
  - a boolean
  - null

Resemblance of XML & JSON by syntax

XML

```
<employees>
  <employee>
    <firstName>John</firstName>
          <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName>
          <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName>
          <lastName>Jones</lastName>
  </employee>
</employees>
```

JSON

```
{"employees":[
  { "firstName":"John",
    "lastName":"Doe" },
  { "firstName":"Anna",
    "lastName":"Smith" },
  { "firstName":"Peter",
    "lastName":"Jones" }
]}
```

How is JSON better than XML?

- JSON is shorter
- JSON can use arrays
- XML has to be parsed by XML parser whereas JSON can be parsed using a standard Javascript function

Reading JSON

- There can be two functions namely read_json and fromJSON to read the JSON data into R

Syntax : fromJSON(txt, simplifyVector = TRUE,…)

Where

txt : a JSON string, URL or file

simplifyVector : simplifies nested lists into vectors

Example

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address":
    {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber":
    [
        {
          "type": "home",
          "number": "212 555-1234"
        },
        {
          "type": "fax",
          "number": "646 555-4567"
        }
    ]
}
```

```
> library(jsonlite)
> jsonData <- fromJSON("D:/Data Science Training - Level III/Datasets/contacts.txt")
> class(jsonData)
[1] "list"
> names(jsonData)
[1] "firstName"   "lastName"    "age"         "address"     "phoneNumber"
>
> #nested objects
> jsonData$phoneNumber
  type       number
1 home 212 555-1234
2  fax 646 555-4567
> jsonData$phoneNumber$number
[1] "212 555-1234" "646 555-4567"
```

Reading JSON

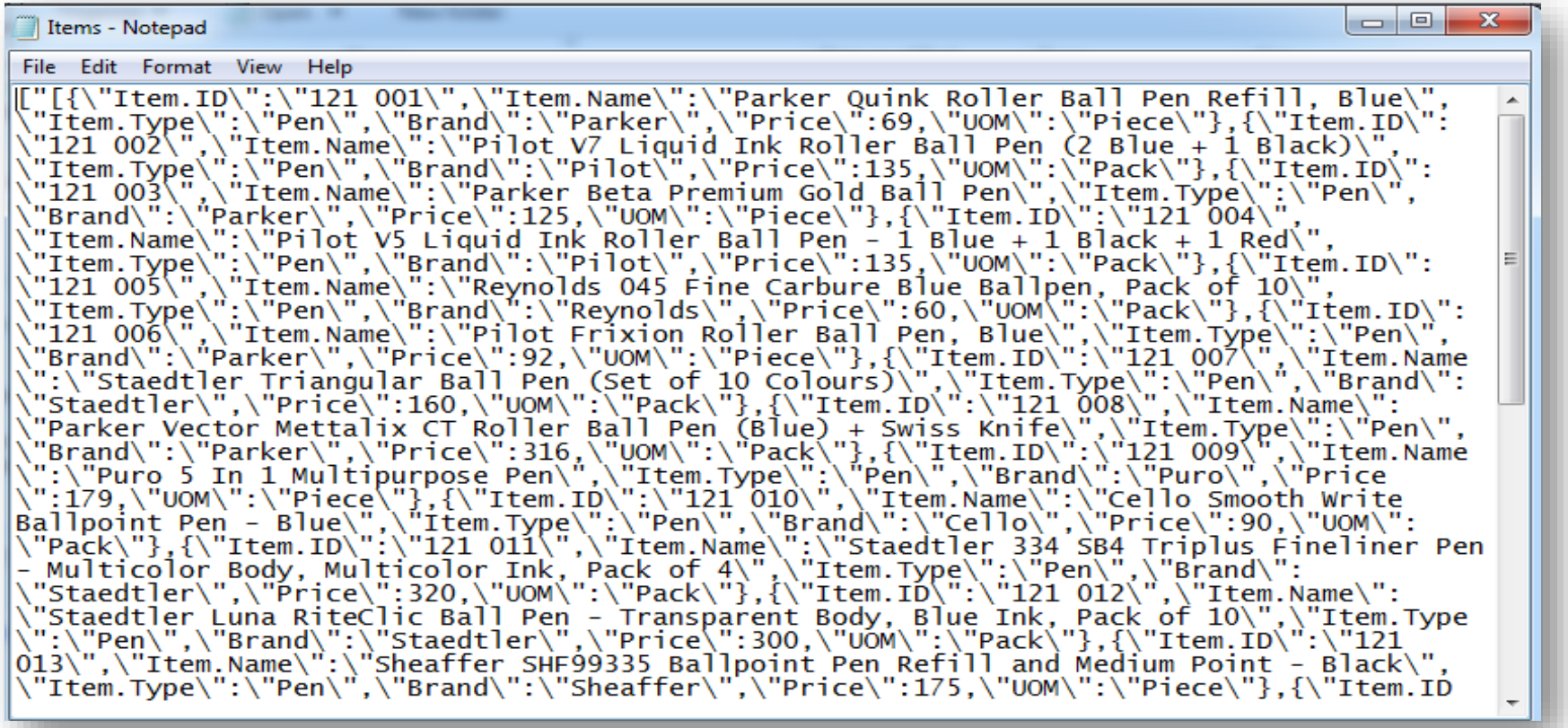
Syntax : read_json(path, simplifyVector, …)

Where

path : file on disk

simplifyVector : simplifies nested lists into vectors and data frames

Example




```
> items_json <- read_json("D:/Data Science Training - Level III/Datasets/Items.json",
+                         simplifyVector = T)
> class(items_json)
[1] "character"
> items <- fromJSON(items_json)
> class(items)
[1] "data.frame"
```

Reading XML

What is XML?

- Extensible Mark-up language is a mark-up language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- This language is widely used for the representation of arbitrary data structures such as those used in web services.

Why people use XML?

- XML stores data in plain text format which enables a software and hardware independent way of storing, transporting and sharing data.
- XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without loss of data.
- XML is a W3C (World Wide Web Consortium) Recommendation.
- Language resembles that of HTML which is made up of tags.

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<customermaster>
   <customer>
      <custid>1001</custid>
      <name>Ramesh Kumar</name>
      <age>28</age>
      <company>TCS</company>
   </customer>

   <customer>
      <custid>1002</custid>
      <name>Dilip Deo</name>
      <age>38</age>
      <company>Infosys</company>
   </customer>
</customermaster>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <customermaster>
  - <customer>
      <custid>1001</custid>
      <name>Ramesh Kumar</name>
      <age>28</age>
      <company>TCS</company>
    </customer>
  - <customer>
      <custid>1002</custid>
      <name>Dilip Deo</name>
      <age>38</age>
      <company>Infosys</company>
    </customer>
  </customermaster>
```

Sane's STATS Academy of Statistics

Statistical Graphics

Sane's STATS Academy of Statistics

Graphic Representation of Data

- Bar Plot
- Histogram
- Scatter Plot
- Density Plot
- Box Plot

Sane's STATS Academy of Statistics

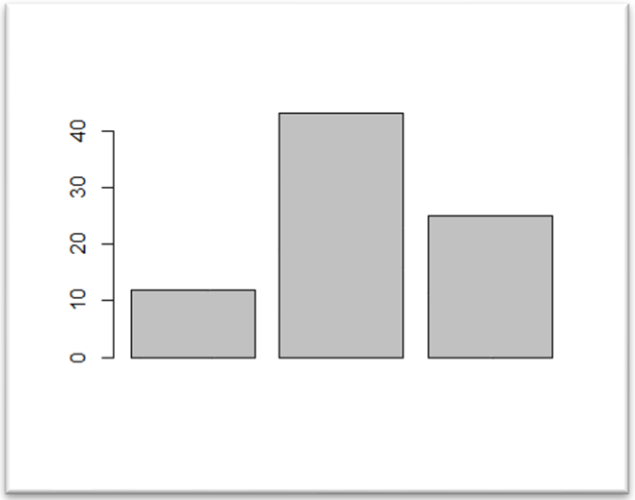
Bar Plot

- Bar chart can be drawn with the help of function **barplot**()

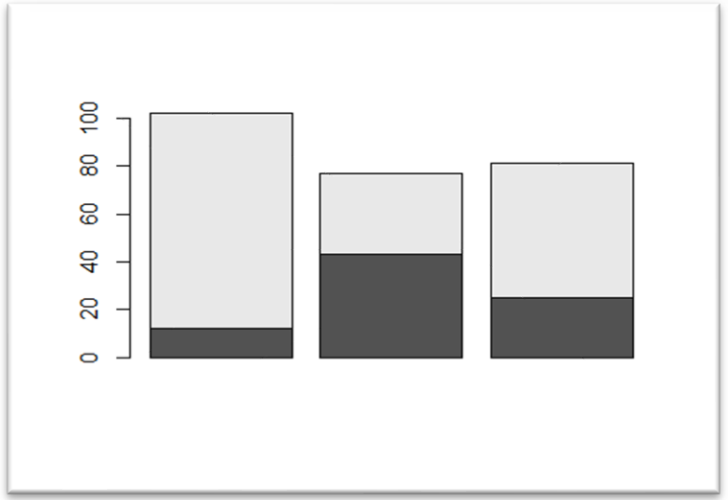
Syntax : barplot(height, horiz = FALSE, main = NULL, sub = NULL, xlab = NULL, ylab = NULL,...)
Where height : A vector / matrix
horiz : logical; If TRUE then graph results in horizontal bar graph
main : Title of Graph
xlab : X-axis label
ylab : Y-axis label

Bar Plot Examples

```
> ProductA <- c(12,43,25)
> barplot(ProductA)
```




```
> ProductA <- c(12,43,25)
> ProductB <- c(90,34,56)
> rb <- rbind(ProductA,ProductB)
> barplot(rb)
```



Bar Plot Examples

```
> barplot(table(Gender),col = "brown",main = "Gender-Wise Distribution")
```

Histogram

- Bar Plot on binned data can be said to be a histogram
- Difference between histogram and bar plot is that in bar plot we are free to rearrange the bars whereas in histogram we are not
- We can know the distribution of any variable with histogram

Syntax : hist(x, breaks = "", include.lowest = TRUE, main = paste("Histogram of" , xname), xlim = range(breaks), xlab = xname, ylab, ...)

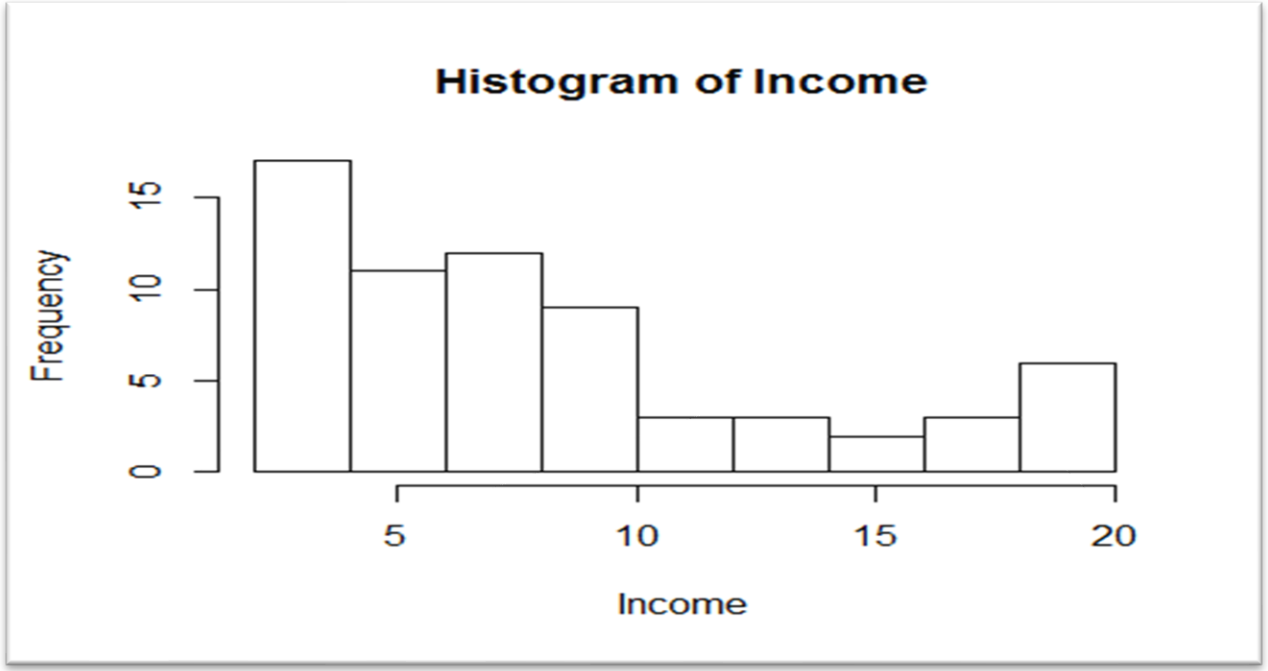
Where x : numeric vector or a function which can generate a numeric vector

breaks : specifying the number of breaks or a numeric vector of breaks

xlim : Scale of limits on X axis

Histogram Examples

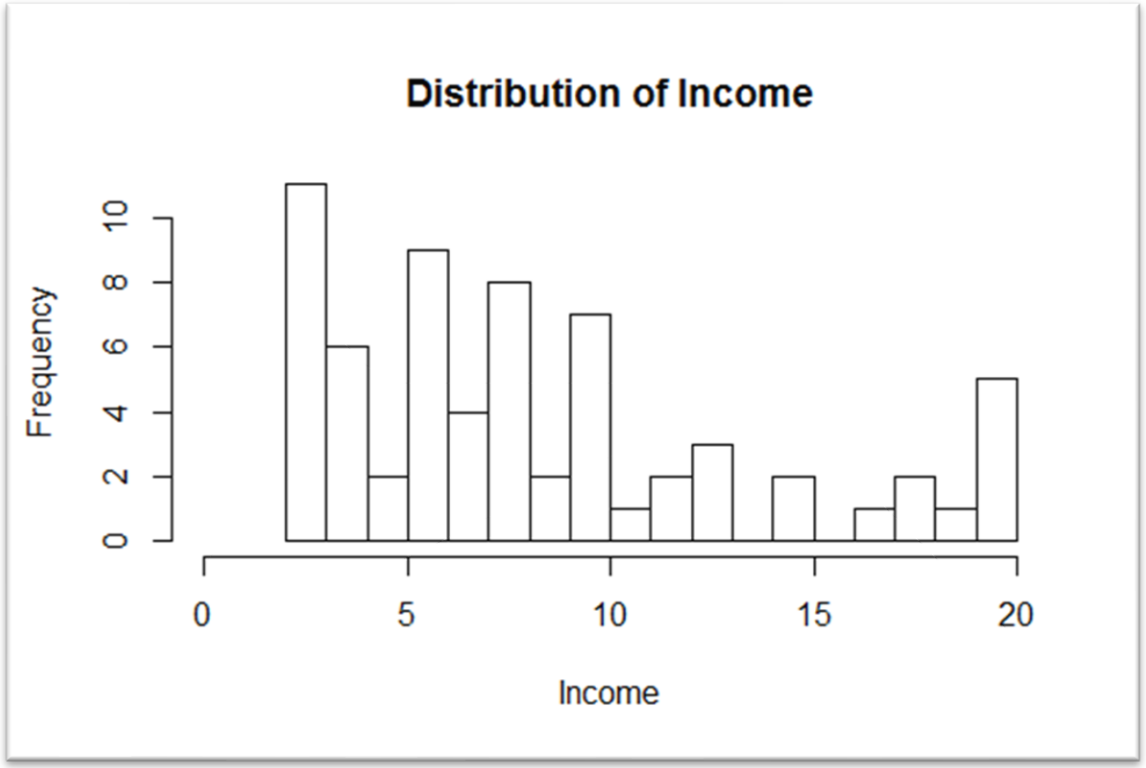
```
> hist(Income)
```



Sane's STATS Academy of Statistics

Histogram Examples

```
> hist(Income,breaks = 20,xlab="Income",
+      main="Distribution of Income", xlim = c(0,20))
```



Scatter Plot

- Scatter Plot can be understood as plot of points on 2-dimensional space of XY-plane
- Scatter plot gives us the information about correlation between the two variables plotted

Syntax : plot(x, y, type, col, pch...)

Where x , y : numeric vector or a function generating numerical vector

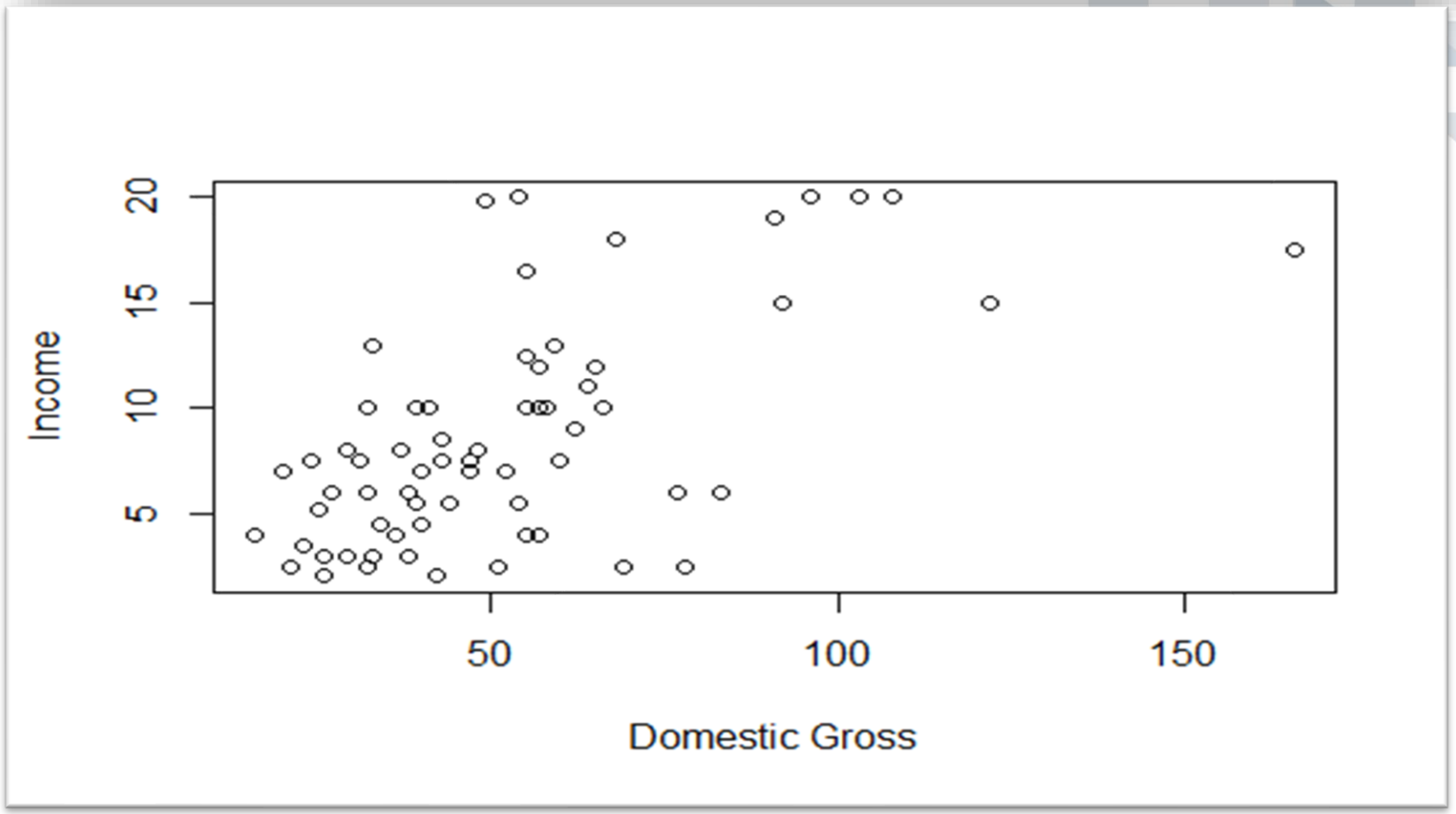
type : p for points, l for lines, b for both, s for stair steps etc.

col : Colour of plotting character

pch : Plotting character

Scatter Plot Examples

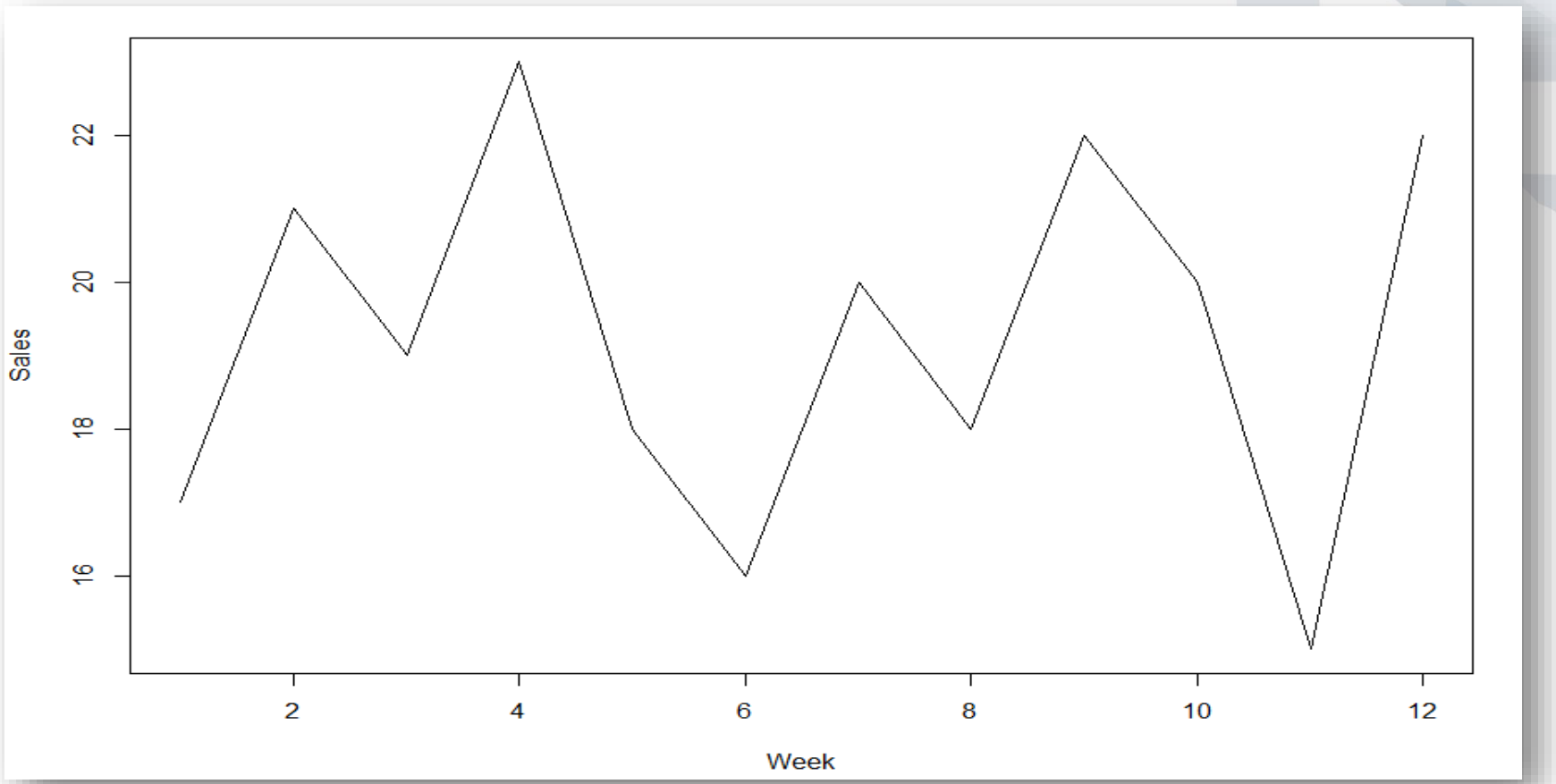
```
> plot(Domestic.Gross,Income,xlab="Domestic Gross",ylab="Income")
```



Sane's STATS Academy of Statistics

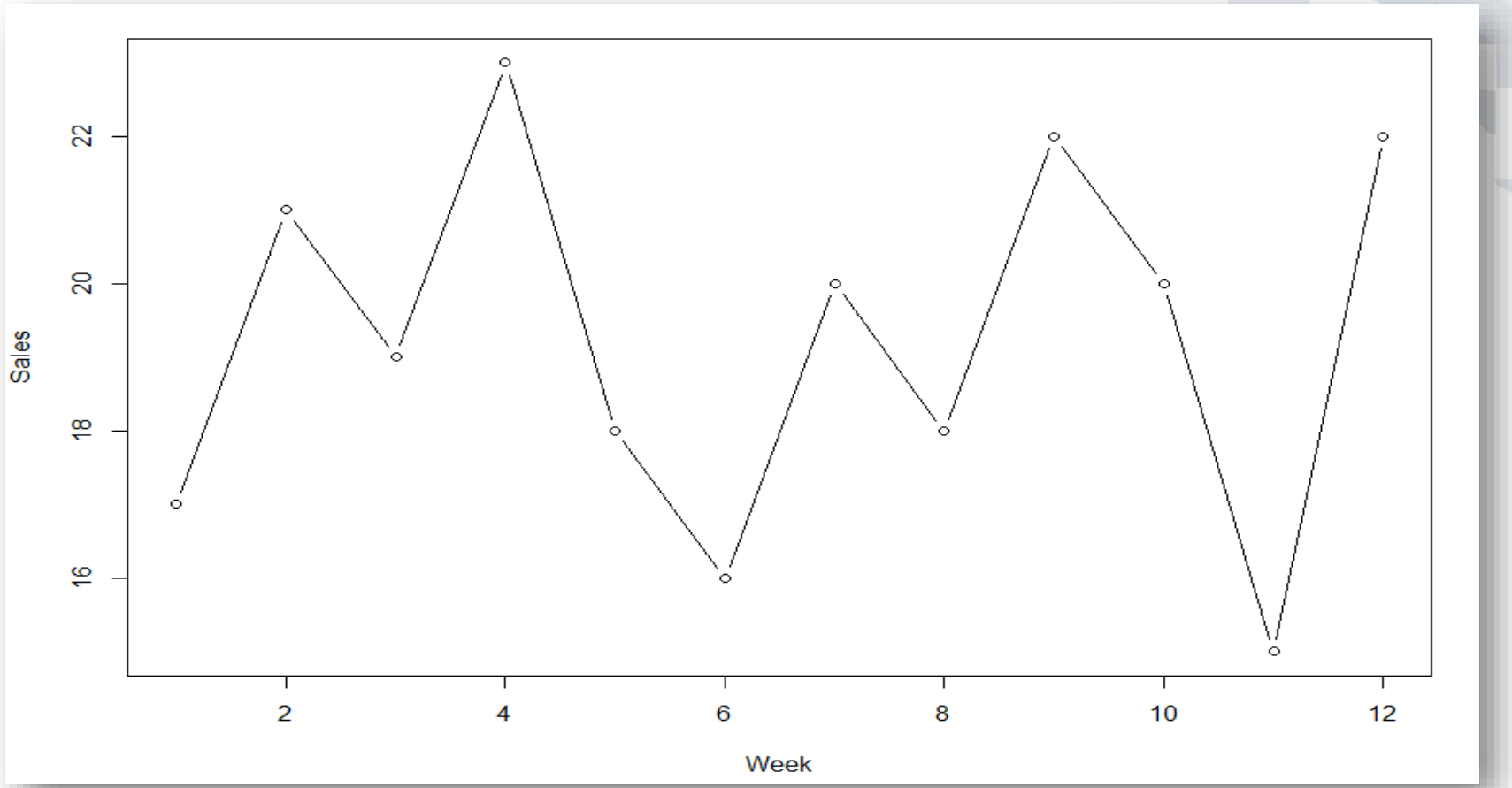
Line Plot using plot()

```
plot(gasoline$Week,gasoline$Sales,xlab = "Week",ylab = "Sales",type = 'l')
```



Line graph using plot()

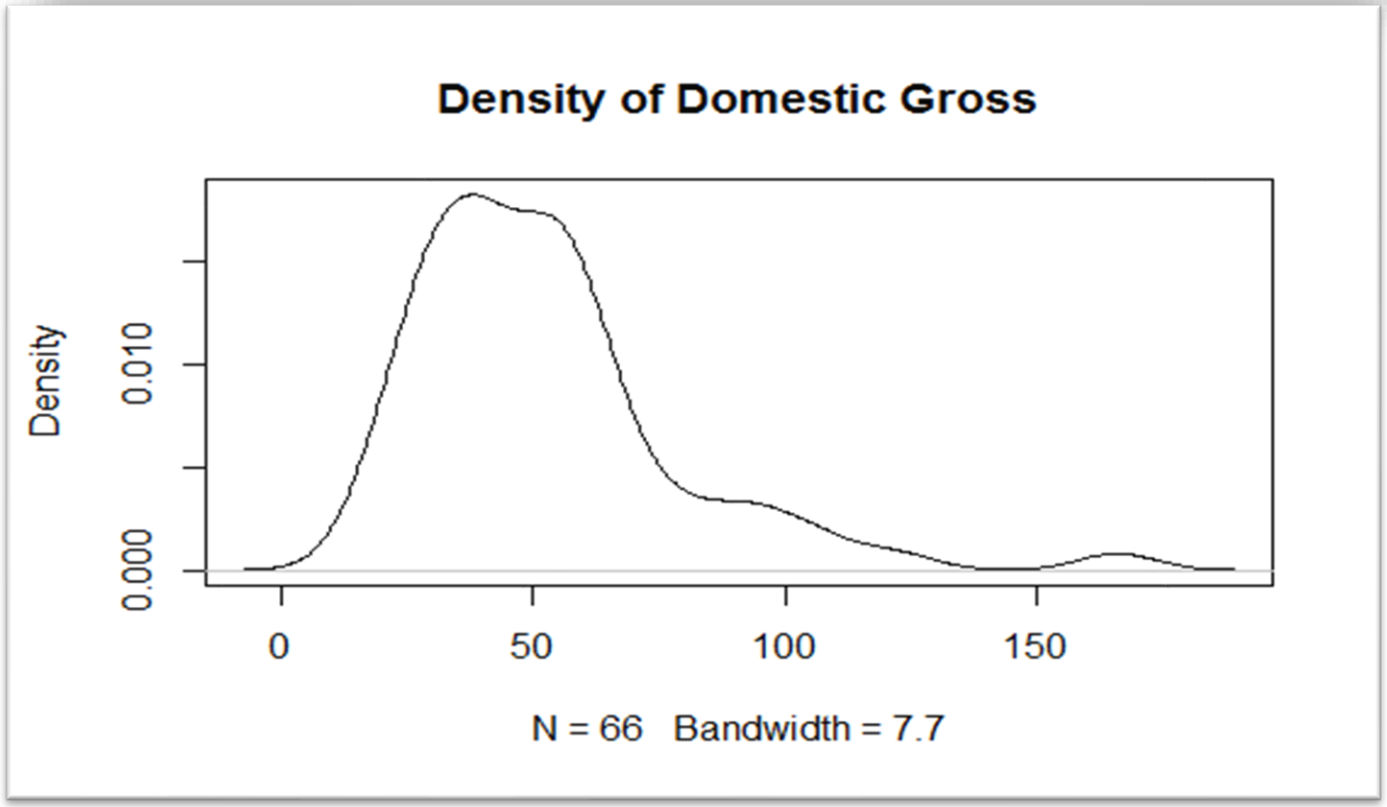
```
plot(gasoline$Week,gasoline$Sales,xlab = "Week",ylab = "Sales",type = 'b')
```



Density Plot

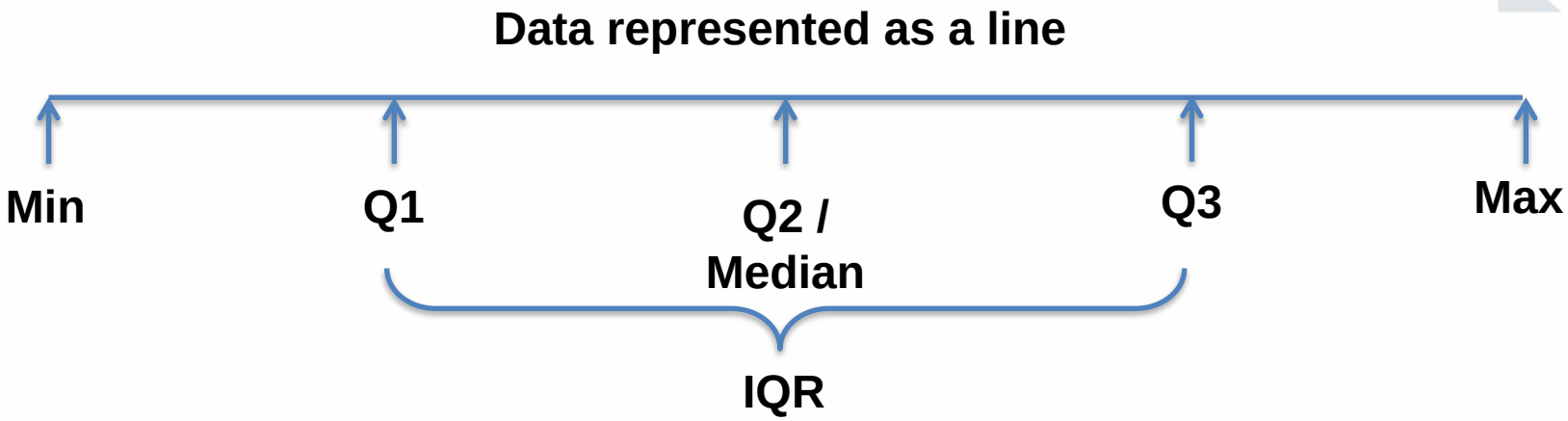
- Density plot is smoothened form of histogram
- Density plot can be plotted with function plot() by calling density function

```
> plot(density(Domestic.Gross),main="Density of Domestic Gross")
```



Boxplot

- Before we understand box plot we need to understand the quartiles
- Quartiles divide the given data into four equal parts.

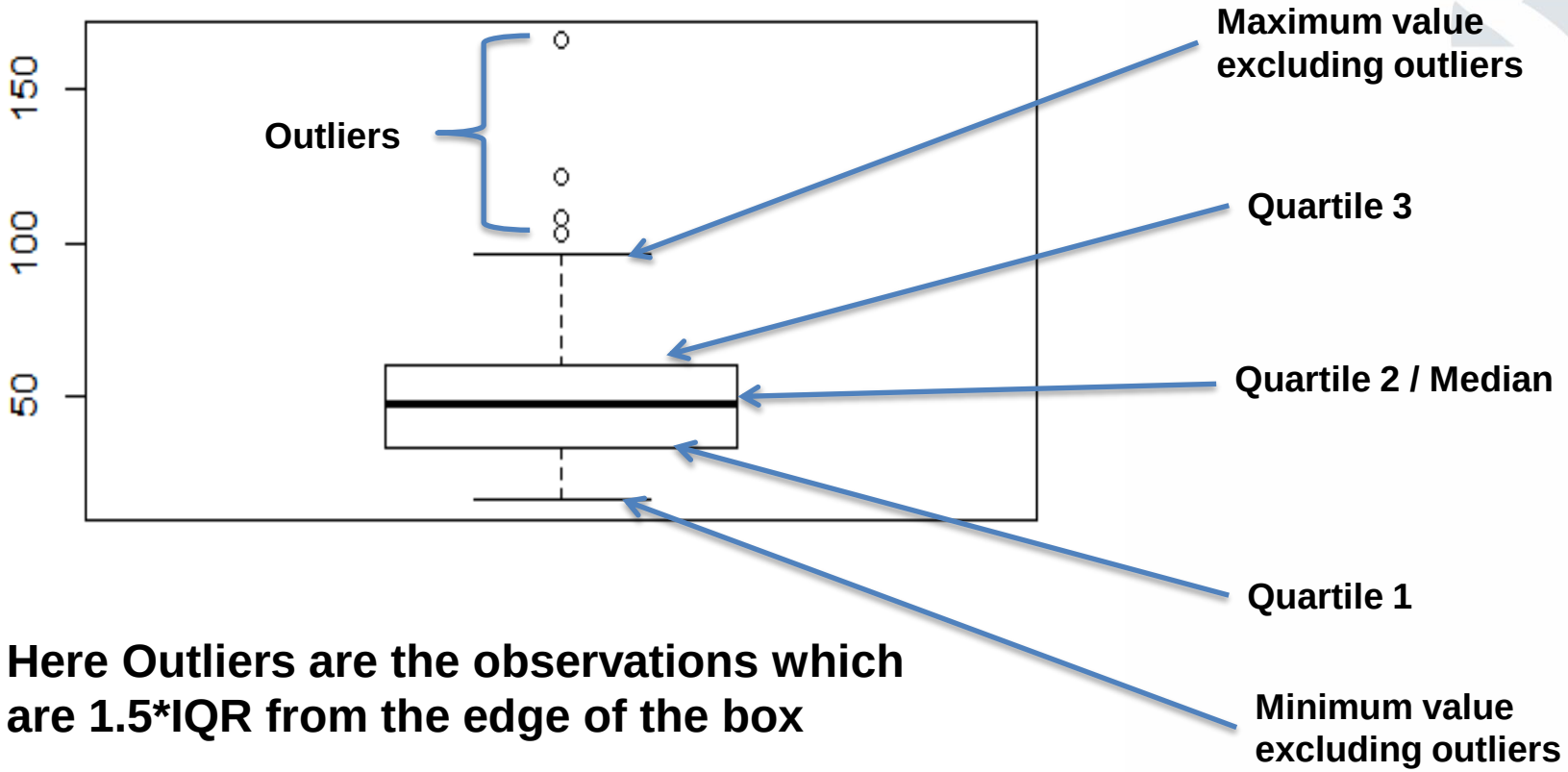


- Inter-quartile range (IQR) is given by the formula:

$$IQR = Q3 - Q1$$

Sane's STATS Academy of Statistics

Describing with Box Plots



Here Outliers are the observations which are 1.5*IQR from the edge of the box

Sane's STATS Academy of Statistics

Boxplot

- Boxplot gives us the spread of the data

Syntax : boxplot(x)

Where x : numeric vector

OR

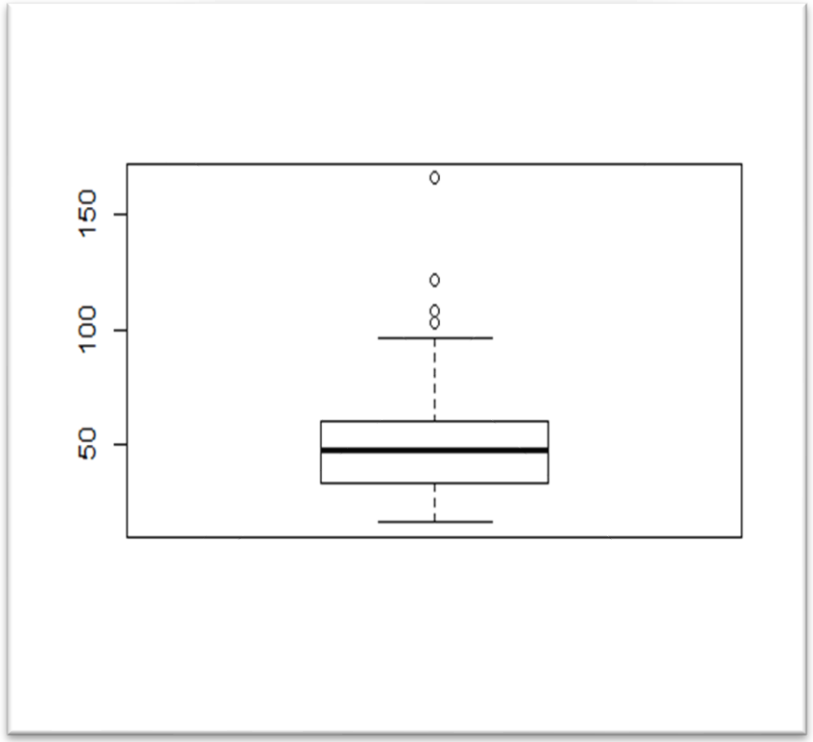
boxplot(y ~ x)

Where y : numeric vector

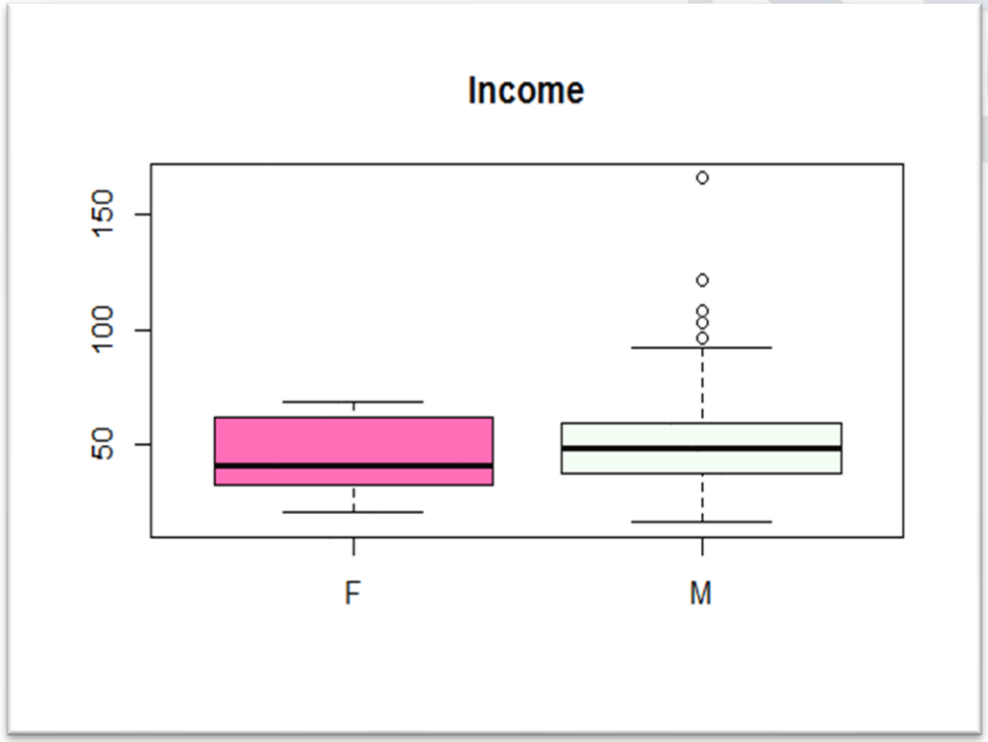
x : factor

Boxplot Examples

```
> boxplot(Domestic.Gross)
```

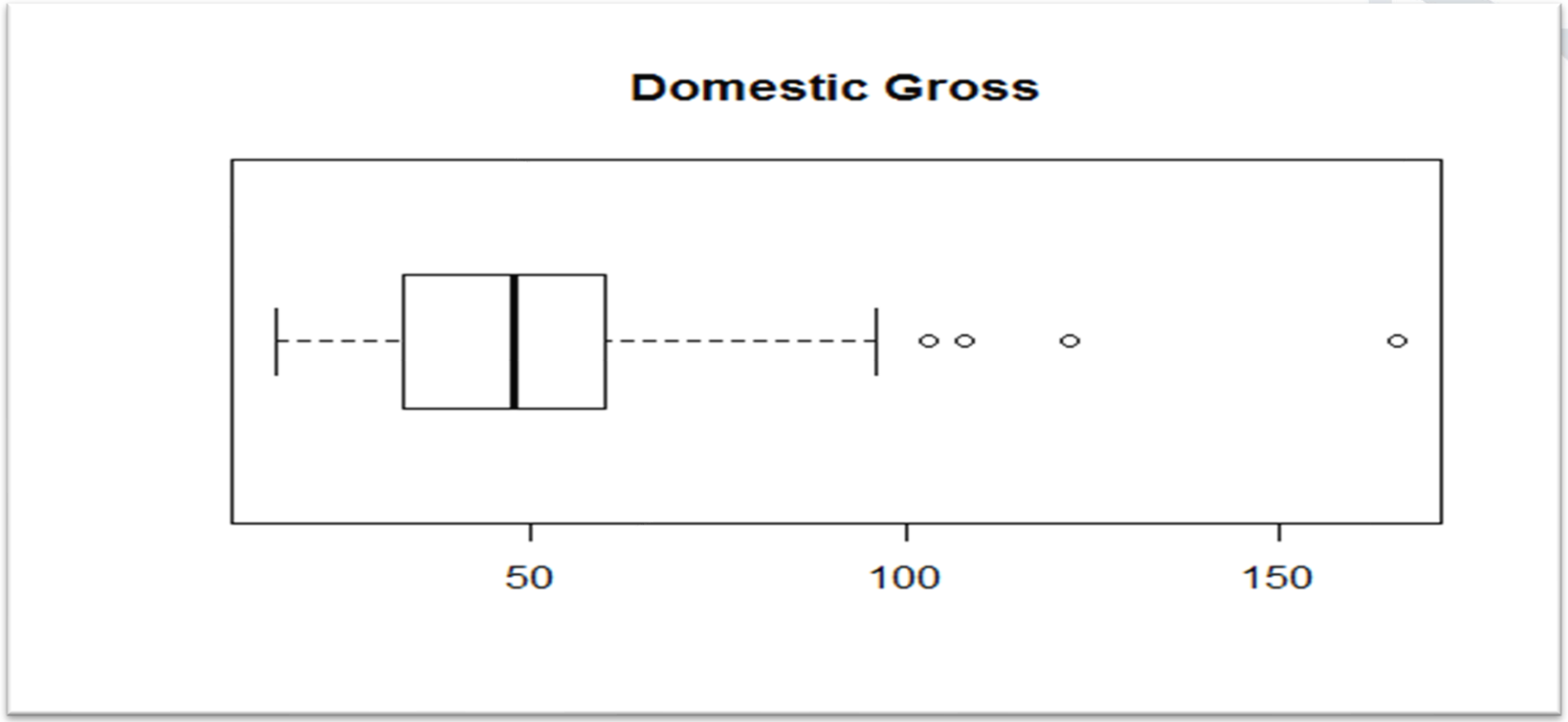



```
> boxplot(Domestic.Gross ~ Gender,
+         col = c("hotpink","honeydew1"),main="Income")
```



Boxplot Examples

```
> boxplot(Domestic.Gross,horizontal = TRUE,main="Domestic Gross")
```



Sane's STATS Academy of Statistics

ggplot2

Enhanced Graphics Package

About ggplot2

- Package ggplot2 provides a method of creating innovative graphs based on graphical grammar
- There are four graphic systems in R currently.

Sane's STATS Academy of Statistics

Four Graphics Systems in R

1. The `base` Graphics System written by the Ross Ihaka included by default in every R installation
2. The `grid` graphics system written by Paul Murrell (2011)
3. The `lattice` graphics system written by Deepayan Sarkar (2008)
4. The `ggplot2` graphics system written by Hadley Wickham (2009)

Sane's STATS Academy of Statistics

Base Graphics

- We have already covered it in the previous sessions
- Composed of functions like plot(), boxplot(), barplot() etc.

Grid Graphics

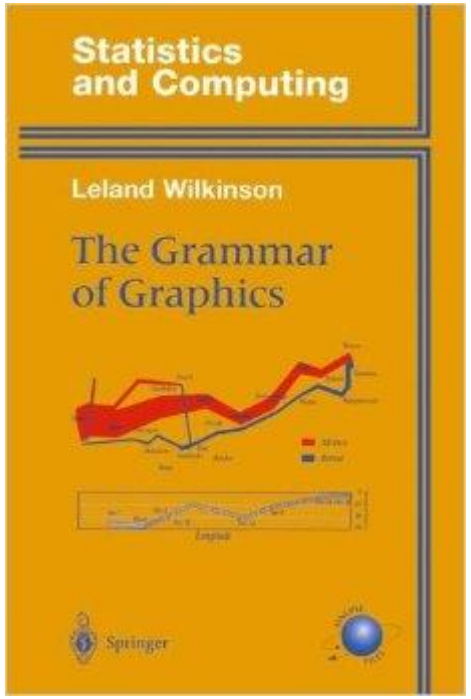
- Implemented by package `grid`
- Offers a low level alternative to the standard graphics system
- But the `grid` package doesn't provide functions for producing statistical graphics or complete plots

Lattice Graphics

- Implements *`trellis`* graphs with package `lattice`
- Provides a comprehensive system for creating statistical graphics
- Built using package `grid`

Ggplot2 Graphics

- The package *ggplot2* has been written by Hadley Wickham
- Provides a system for creating graphs based on the grammar of graphics described by Wilkinson and expanded by Wickham



Function qplot

- Before we look into the ggplot function, let us first have a look at the function qplot() (Quick Plot), which is a basic plotting function in package ggplot2.
- The function qplot() hides what goes on underneath (inside)

Syntax : qplot(x , y , data , …)

Where

x : variable to be considered on X-axis

y : variable to be considered on Y-axis (If not specified scatter plot won't be drawn)

data : data frame object

Sane's STATS Academy of Statistics

Function *ggplot*

- With ggplot function, the plots are created by putting together functions in a chain-like manner using plus (+) sign

Syntax :

ggplot(data,aes(x=,y=,…)) + geom function(s)

Where

data : data frame object

aes() : a function for specifying the role of variables

Sane's STATS Academy of Statistics

Basic Components of ggplot()

- **Data Frame**
- **Aesthetic mappings**: data mapping by color, shape etc.
- **Geoms**: Geometric object like points, shapes etc.
- **Facets** : Trellis plotting
- **Stats**: Statistical Transforms
- **Scales** : scale used by aesthetic map
- **Coordinate System**

Sane's STATS Academy of Statistics

Building Plots in ggplot()

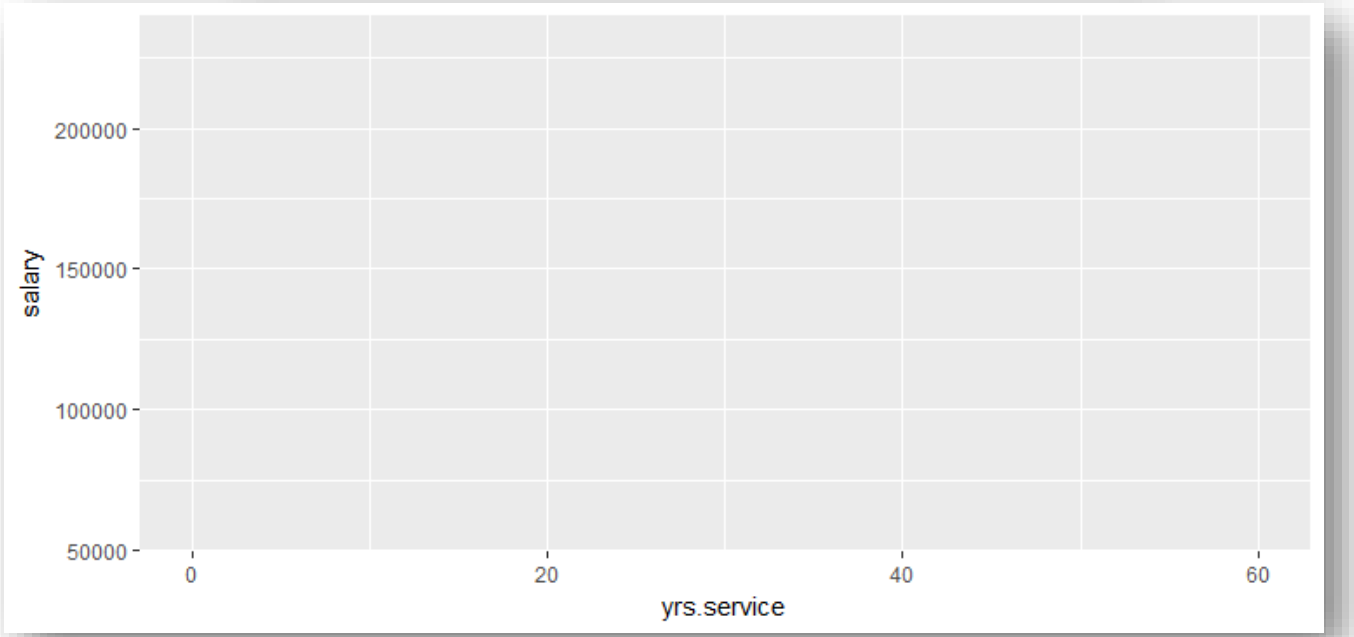
- Plots are built up in layers
  - Plotting the data
  - Overlaying the summary
  - Annotating the graph
- Let us have a simple example of displaying the scatter plot with *yrs.service* as X-axis and *salary* as Y-axis

Sane's STATS Academy of Statistics

Plotting by ggplot()

- We will find here that just specifying the dataset and the axes is not sufficient for generating graph.

```
p <- ggplot(Salaries,aes(yrs.service,salary))
print(p)
```



Sane's STATS Academy of Statistics

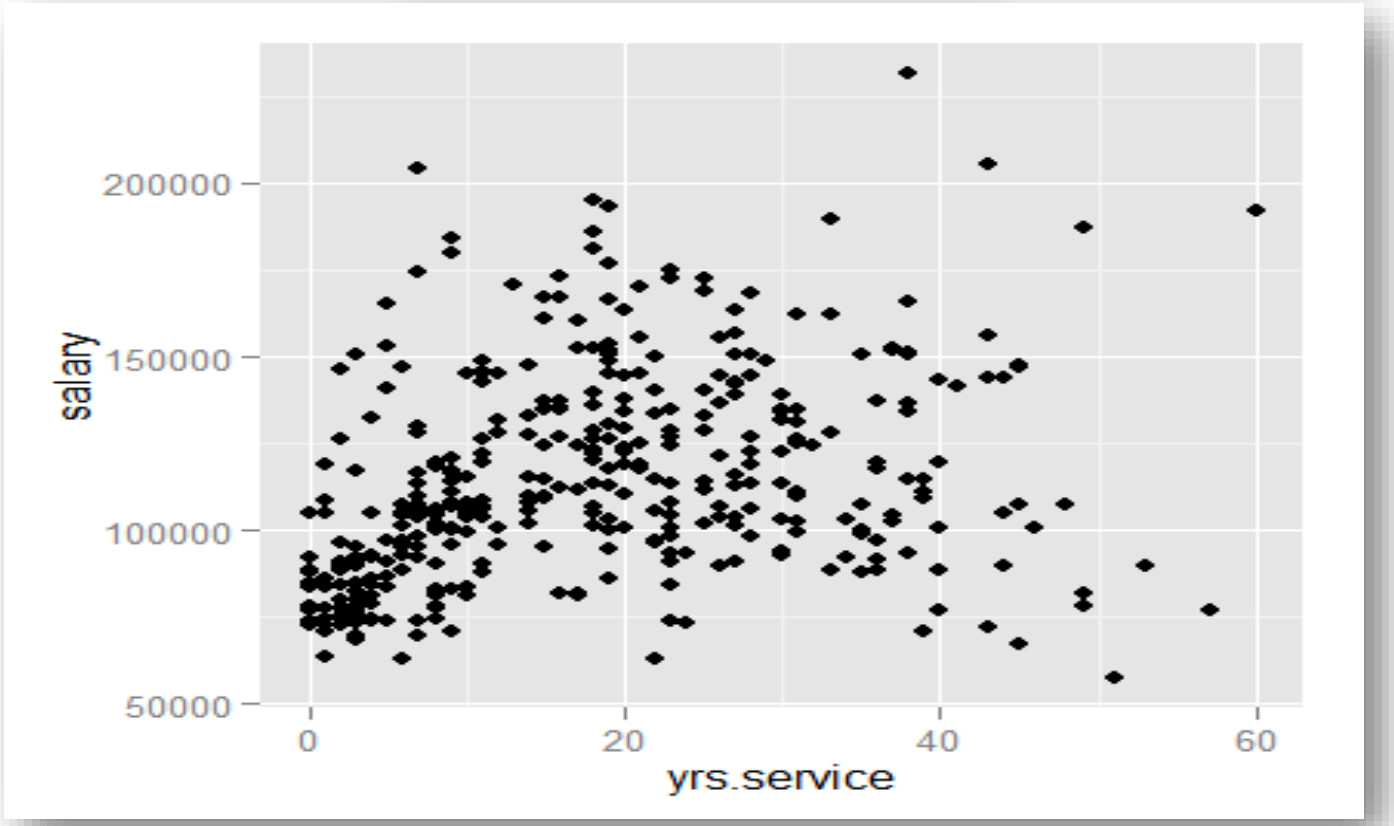
geom_*() functions

Geom	Graph Type
geom_point()	Scatter Plot
geom_line()	Line Graph
geom_histogram()	Histogram
geom_density()	Density Plot
geom_smooth()	Regression Line
geom_boxplot()	Boxplot
geom_bar()	Bar PLot

Scatter Plot

- Scatter plot can be generated with geom_point() function

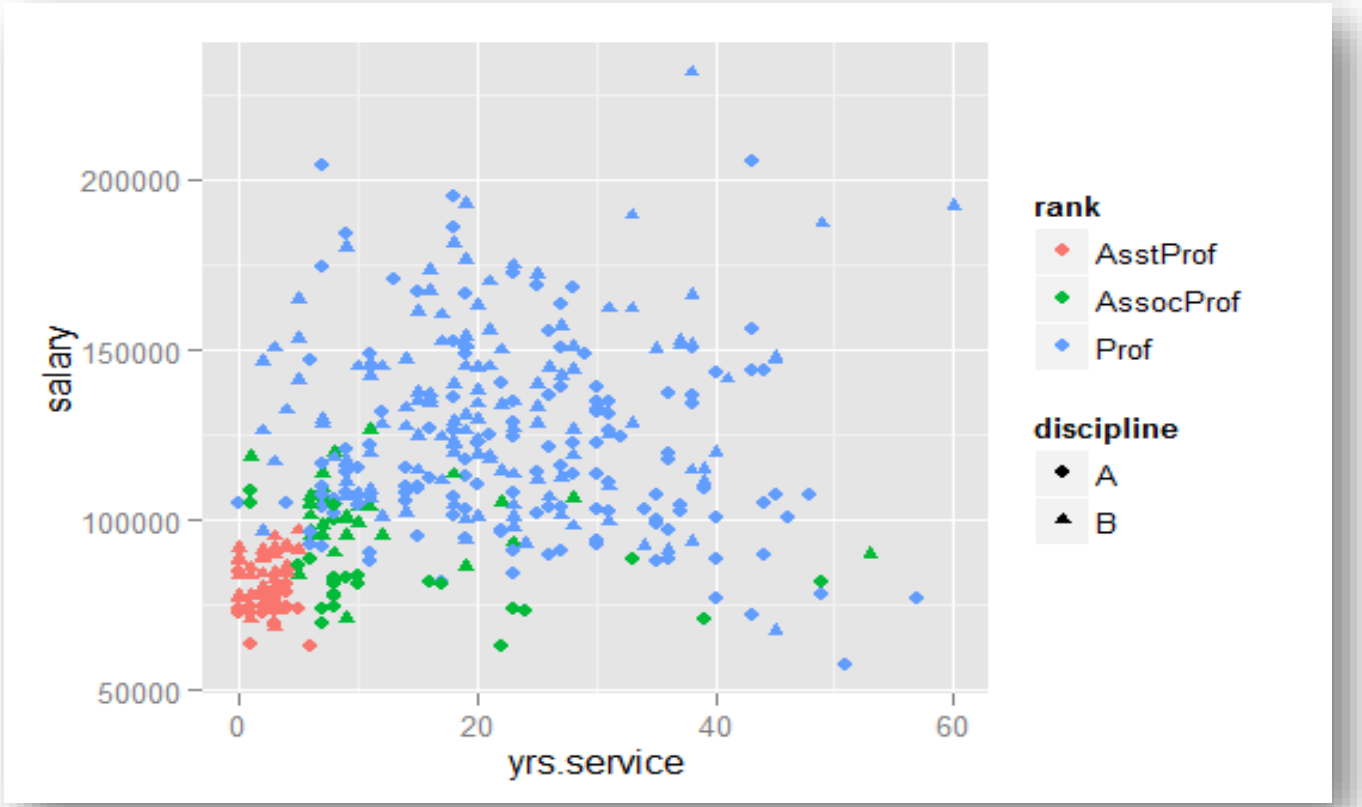
```
p+geom_point()
```



Sane's STATS Academy of Statistics

Grouping in Scatter Plot

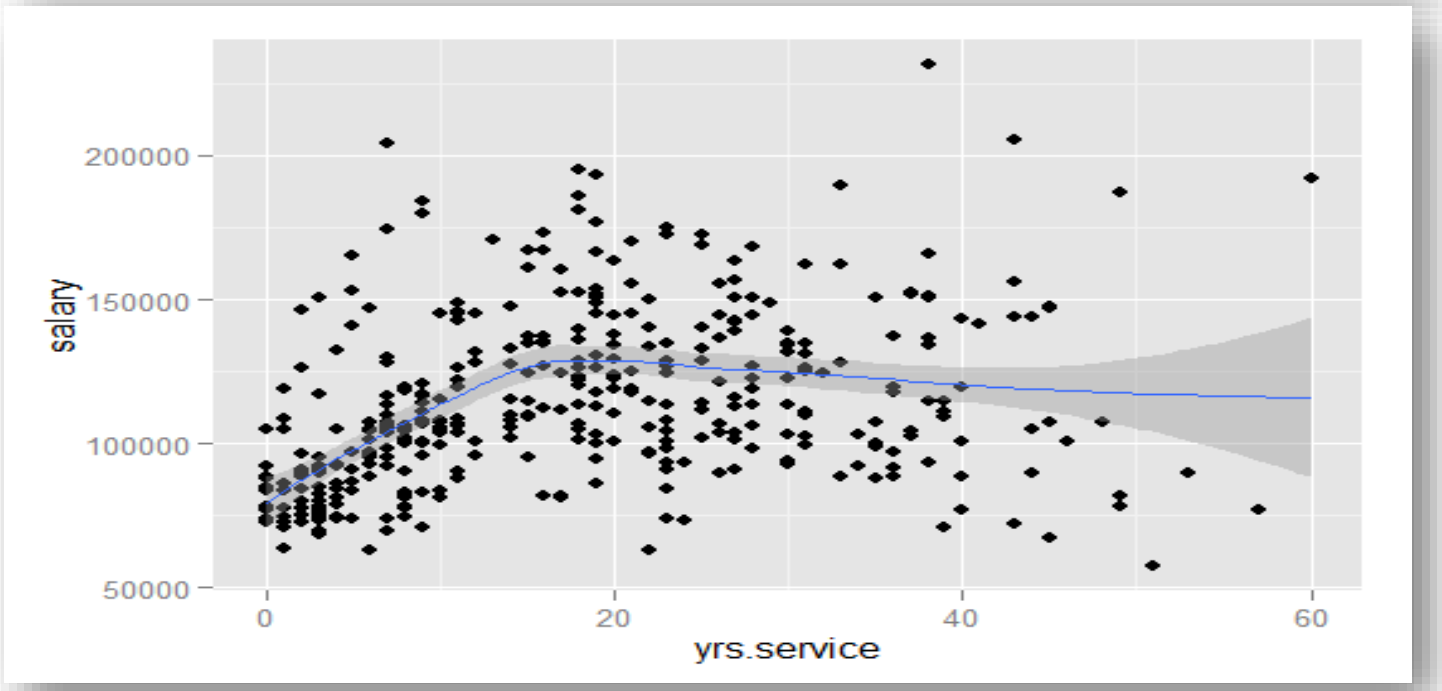
```
ggplot(Salaries,aes(yrs.service,salary,color=rank,shape=discipline))+
 geom_point()
```



Smoothening

- By default, the smoothening happens with loess method.
- We need to specify lm method if we want linear regression line

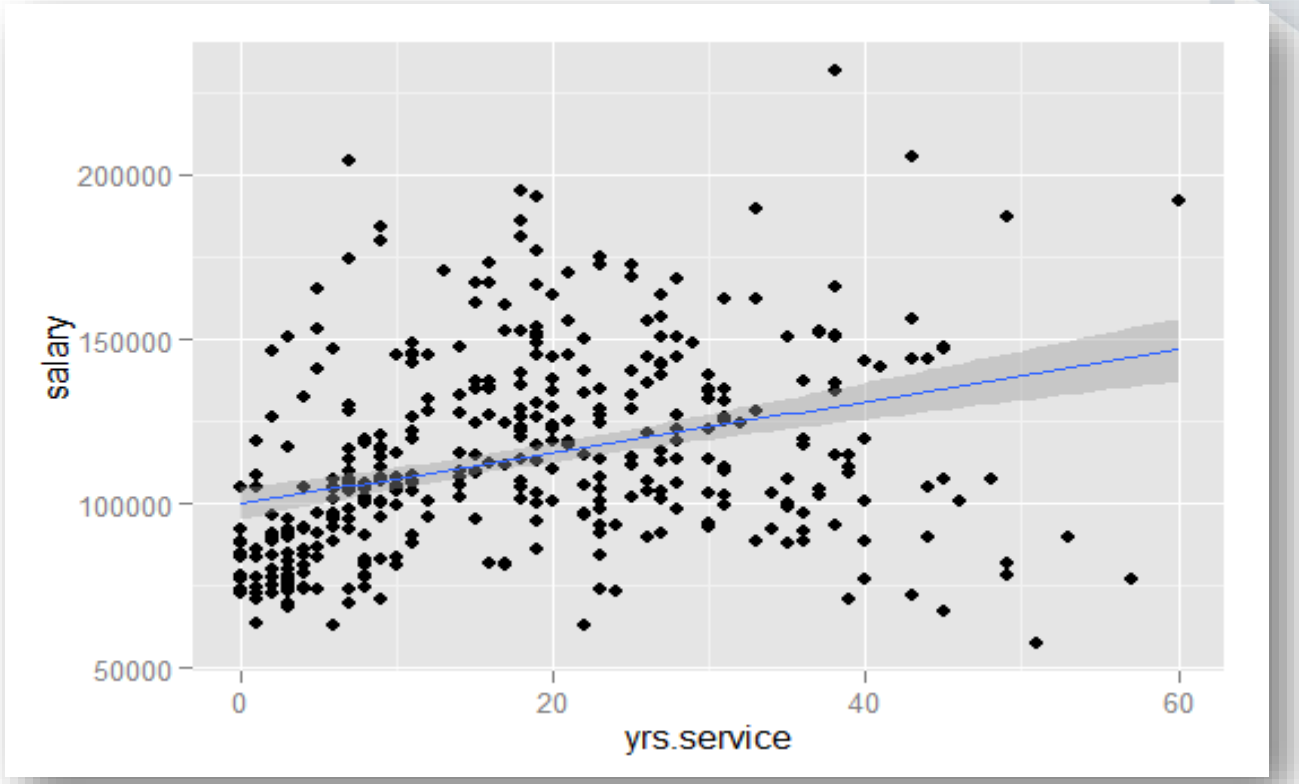
```
> p+geom_point()+geom_smooth()
geom_smooth: method="auto" and size of largest group is <1000, so using loess.
Use 'method = x' to change the smoothing method.
```



Sane's STATS Academy of Statistics

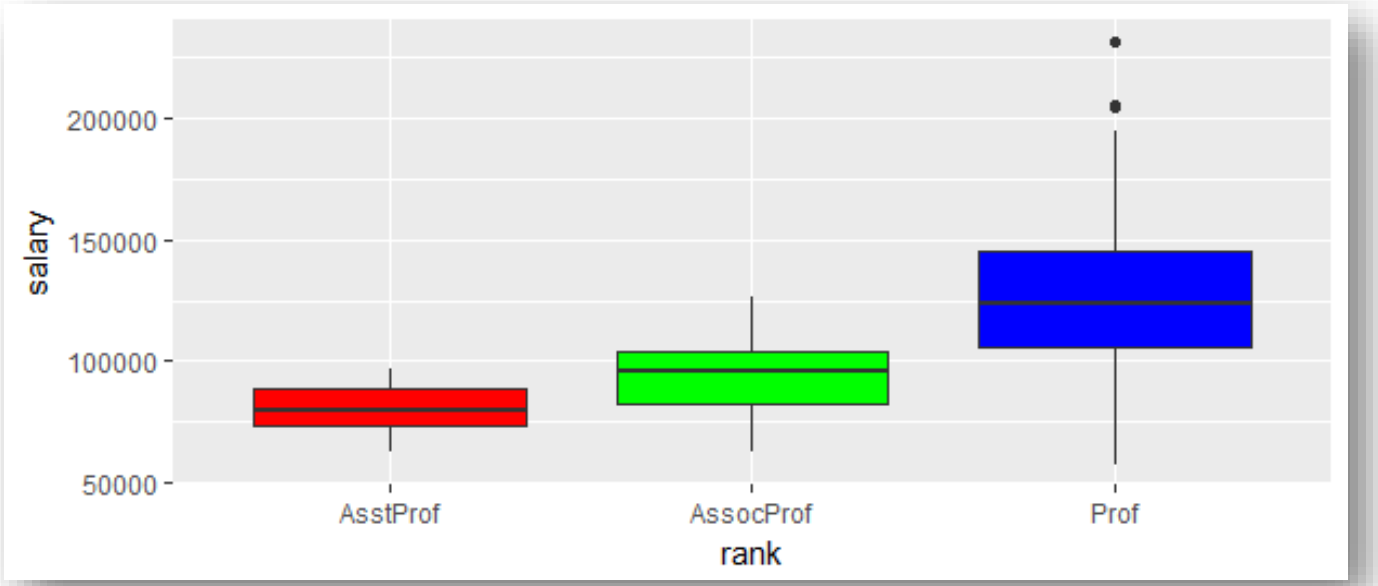
Smoothening

```
p+geom_point()+geom_smooth(method="lm")
```



Boxplot

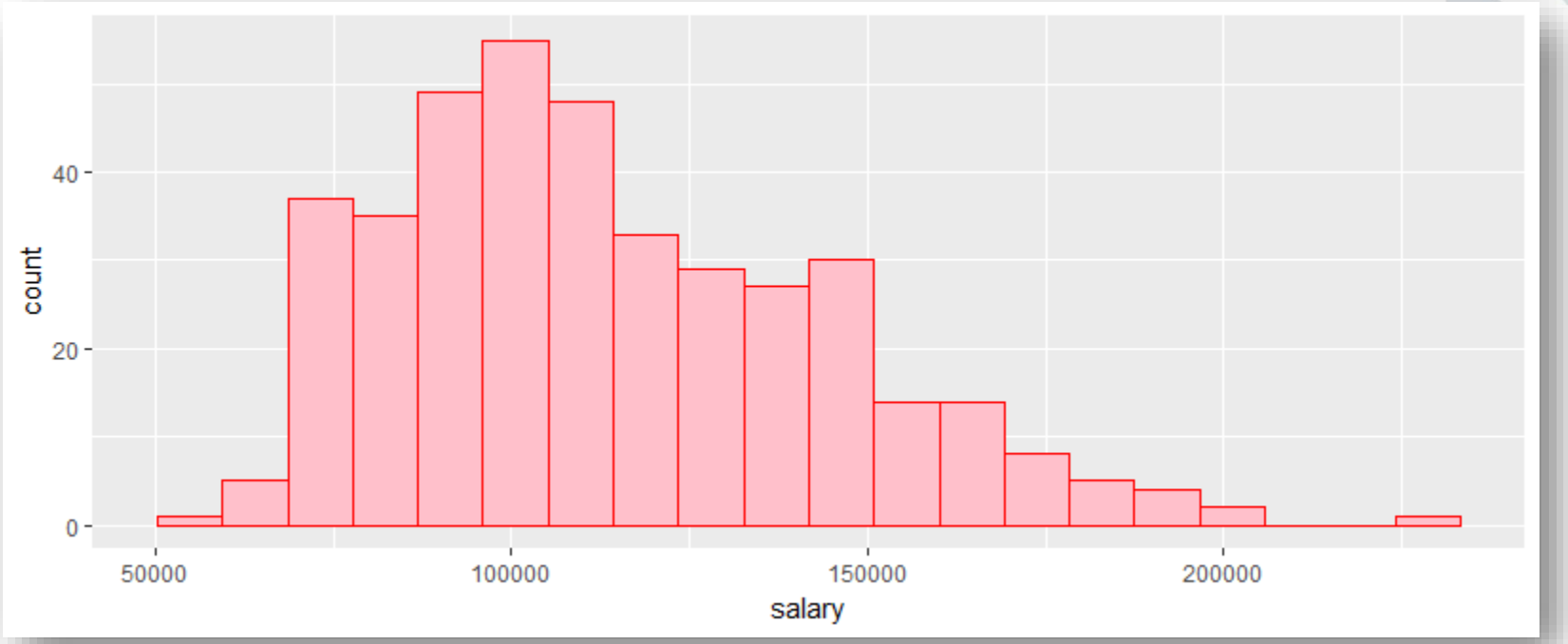
```
ggplot(Salaries, aes(x=rank,y=salary))+
  geom_boxplot(fill=c("red","green","blue"))
```



Sane's STATS Academy of Statistics

Histogram

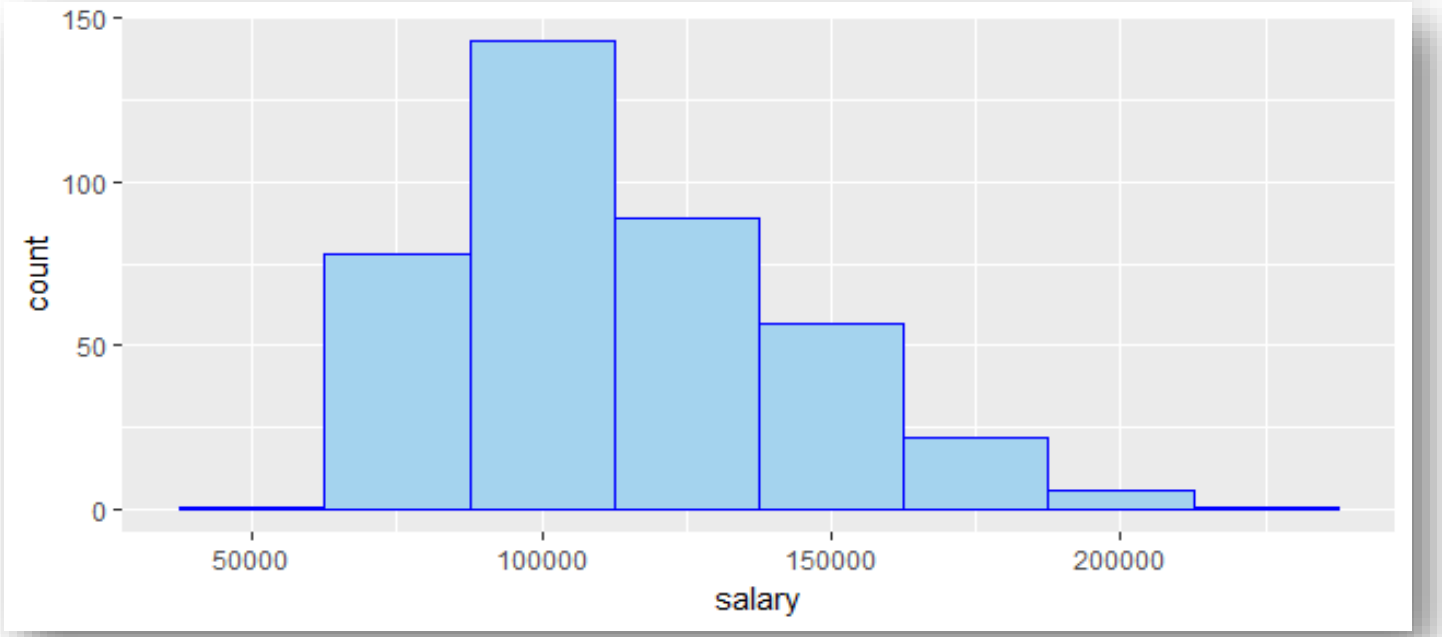
```
ggplot(Salaries,aes(x=salary)) +
  geom_histogram(bins=20,fill="pink",color="red")
```



Sane's STATS Academy of Statistics

Histogram

```
ggplot(Salaries,aes(x=salary)) +
  geom_histogram(binwidth = 25000,fill="lightskyblue2",color="blue")
```



Sane's STATS Academy of Statistics

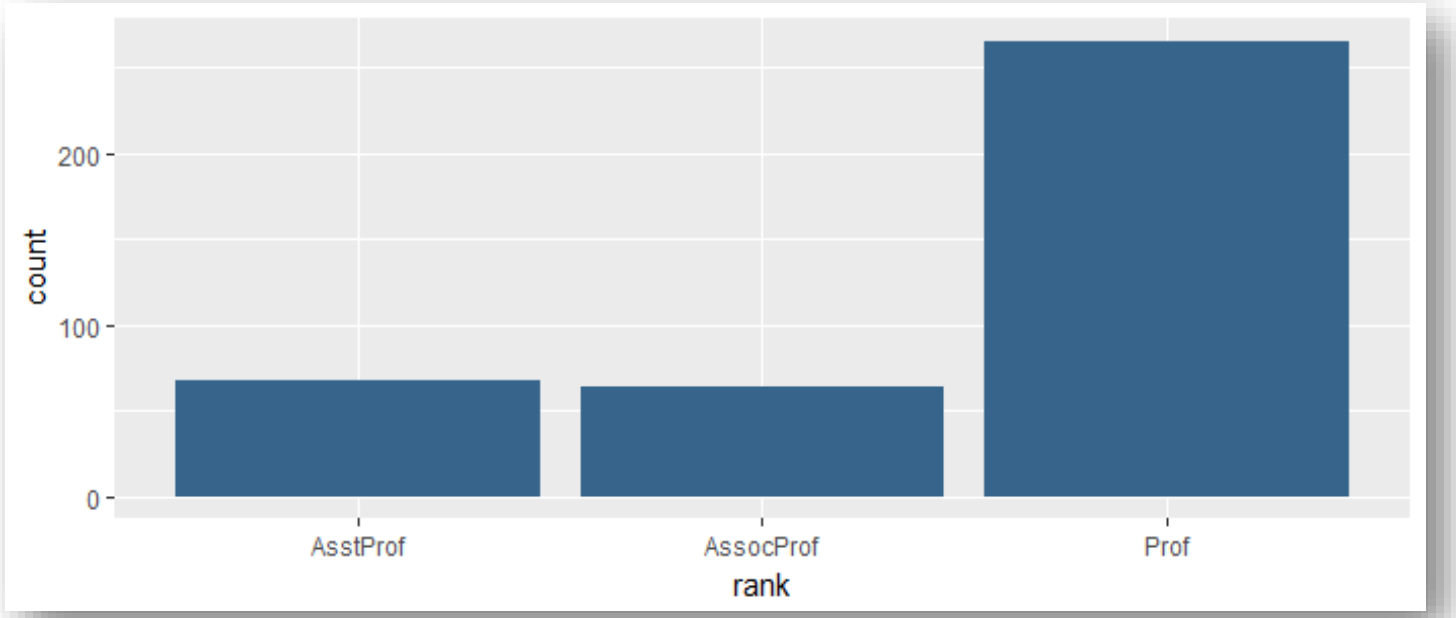
Density Plot

```
ggplot(Salaries, aes(x=salary))+geom_density()
```

Sane's STATS Academy of Statistics

Barplot

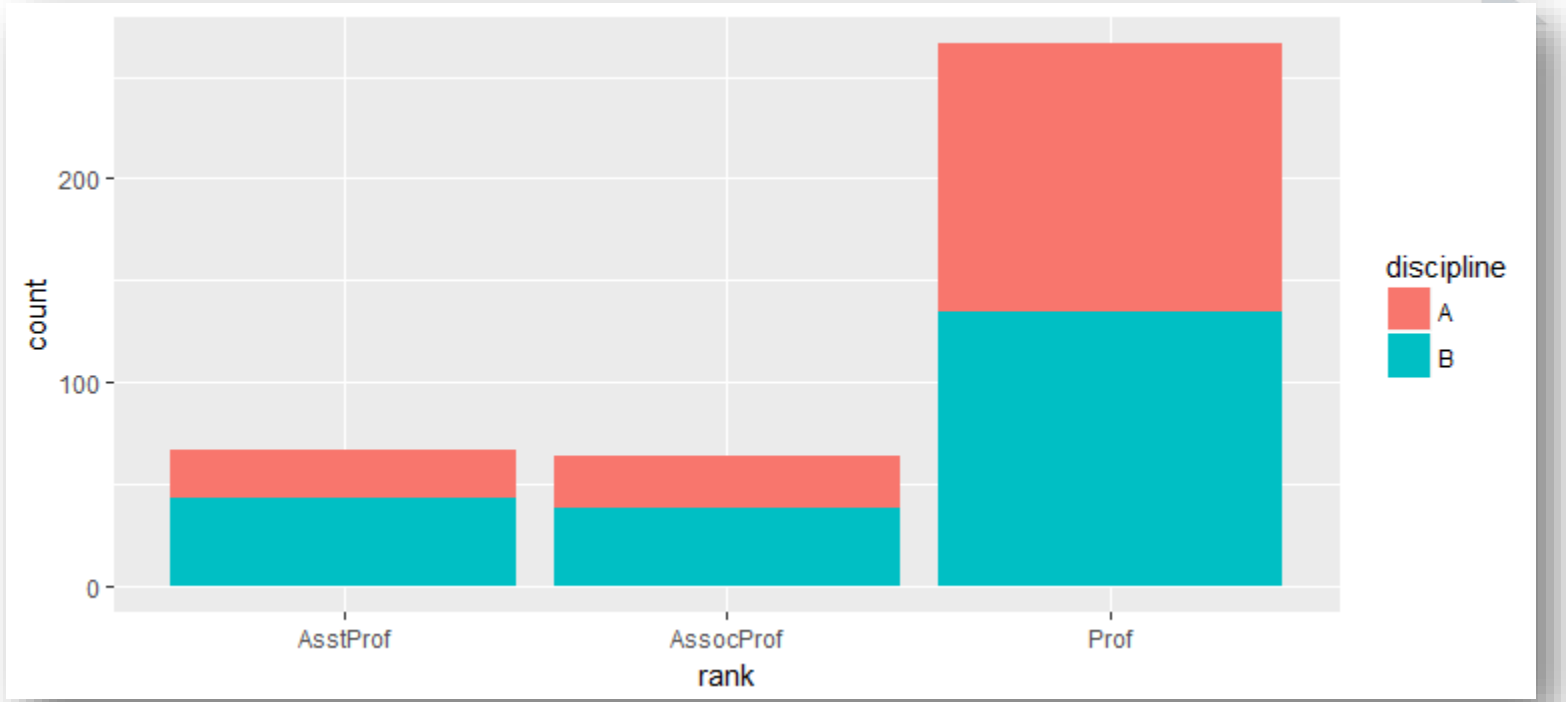
```
ggplot(Salaries, aes(rank))+geom_bar(fill="steelblue4")
```



Sane's STATS Academy of Statistics

Barplot

```
ggplot(Salaries, aes(rank,fill=discipline))+geom_bar()
```



Sane's STATS Academy of Statistics

Facets

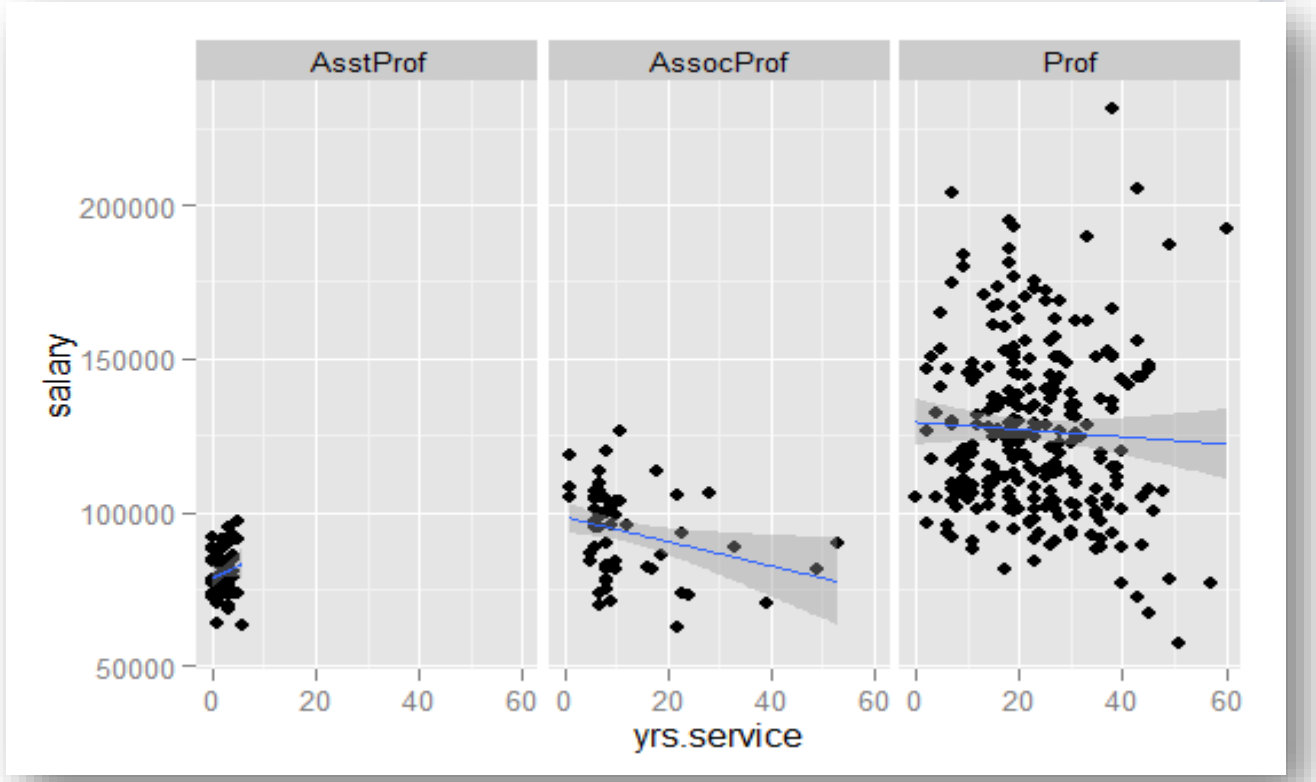
- Sometimes, the relationship become clearer if the graphs are shown side by side
- These are also called trellis graphs

Syntax	Effect
facet_grid(rowvar ~ colvar)	Separate plots for each combination of rowvar and colvar in grid form
facet_grid(rowvar ~ .)	Separate plots of each category in rowvar into a single column
facet_grid(. ~ colvar)	Separate plots of each category in colvar into a single row

Sane's STATS Academy of Statistics

Facet Examples

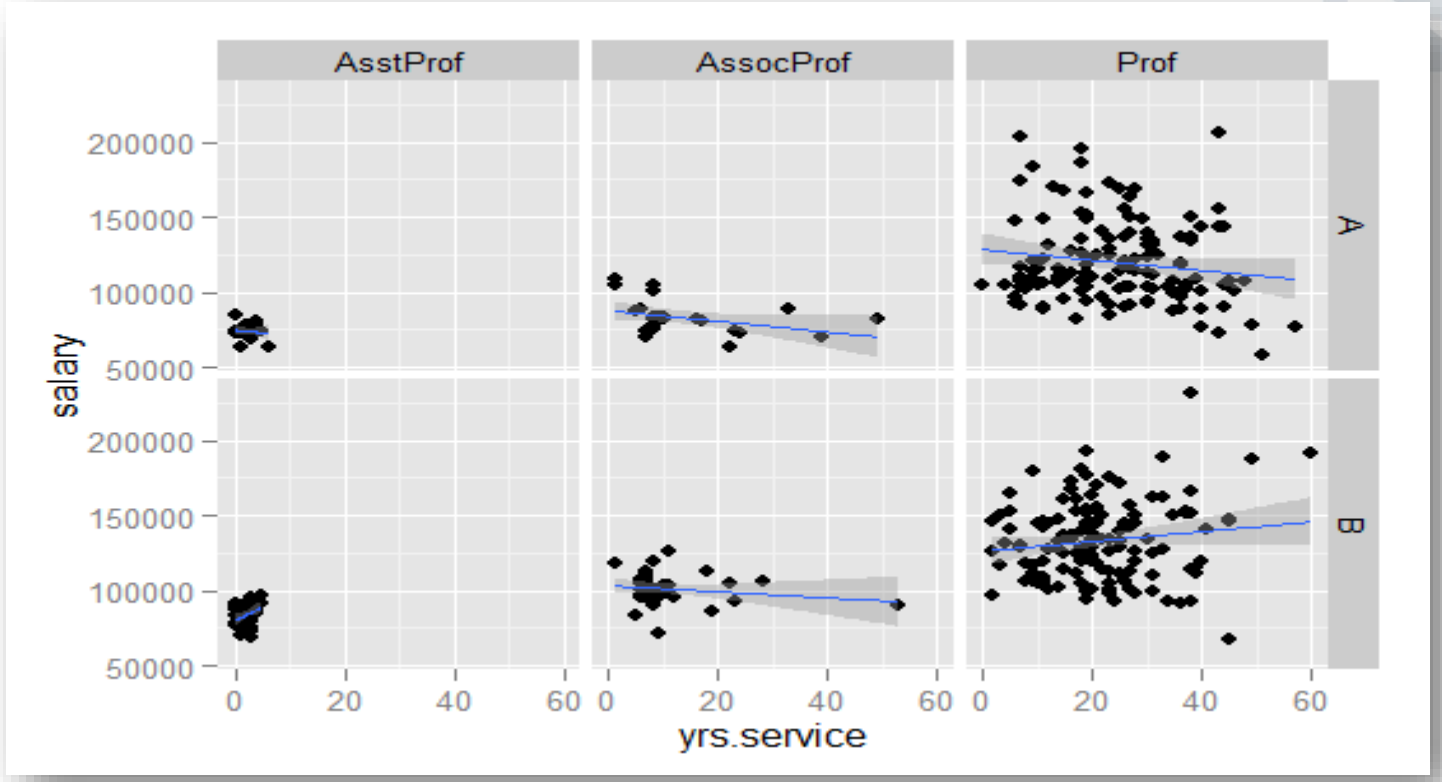
```
p+geom_point()+geom_smooth(method="lm")+facet_grid(.~rank)
```



Sane's STATS Academy of Statistics

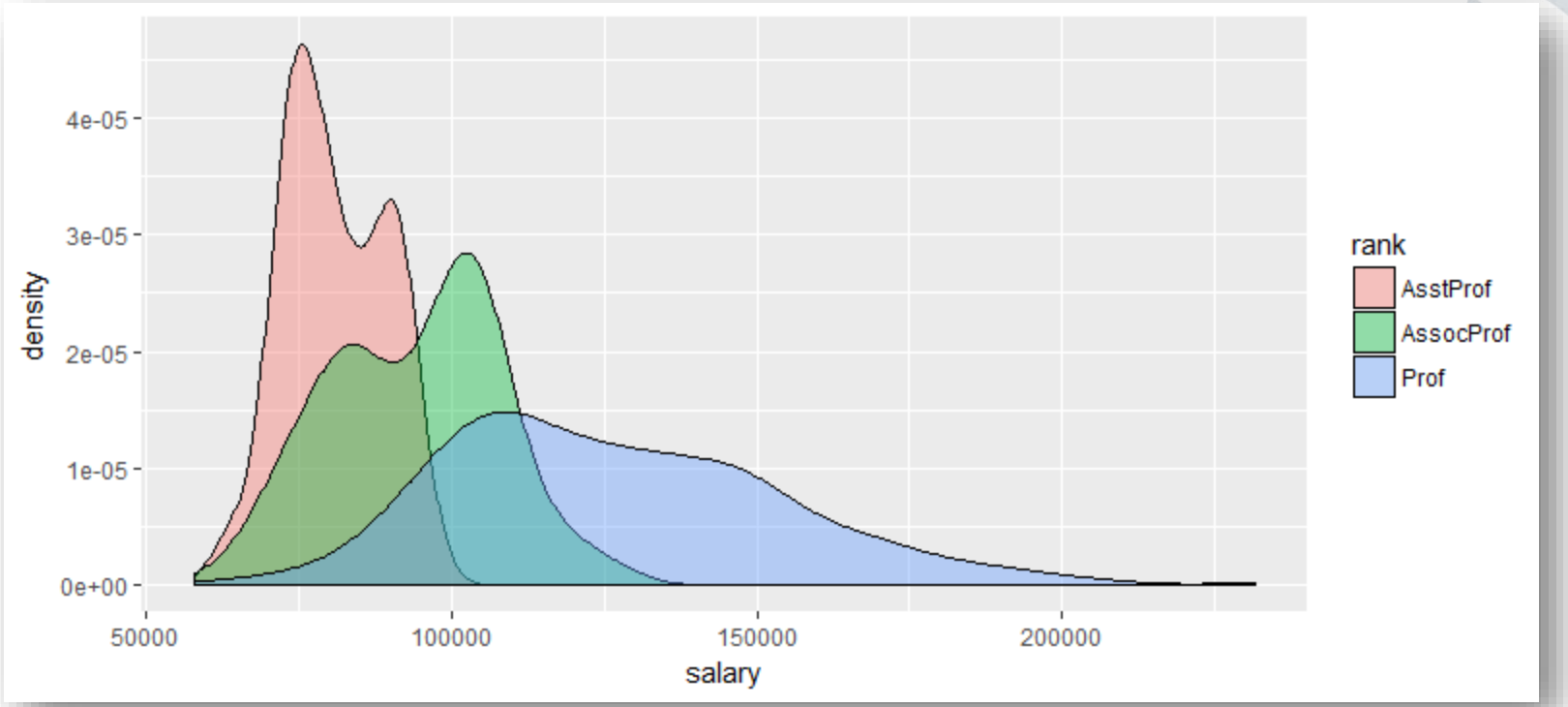
Facet Examples

```
p+geom_point()+geom_smooth(method="lm")+facet_grid(discipline~rank)
```



Boxplot

```
ggplot(Salaries, aes(x=salary,fill=rank))+geom_density(alpha=0.4)
```



Rendering labels

```
> p+geom_point(aes(color=rank), size=2, alpha=3/4)+labs(title="Plot by Rank")
+labs(x="Years in Service",y="Salary")
```