# Intro to R: Part 2 - Data Visualization with Base R

### 2024-09-06

Data Visualization in R
BCBB Summer R Seminar
Mina Peyton

Shift + Command + Return == run current chunk

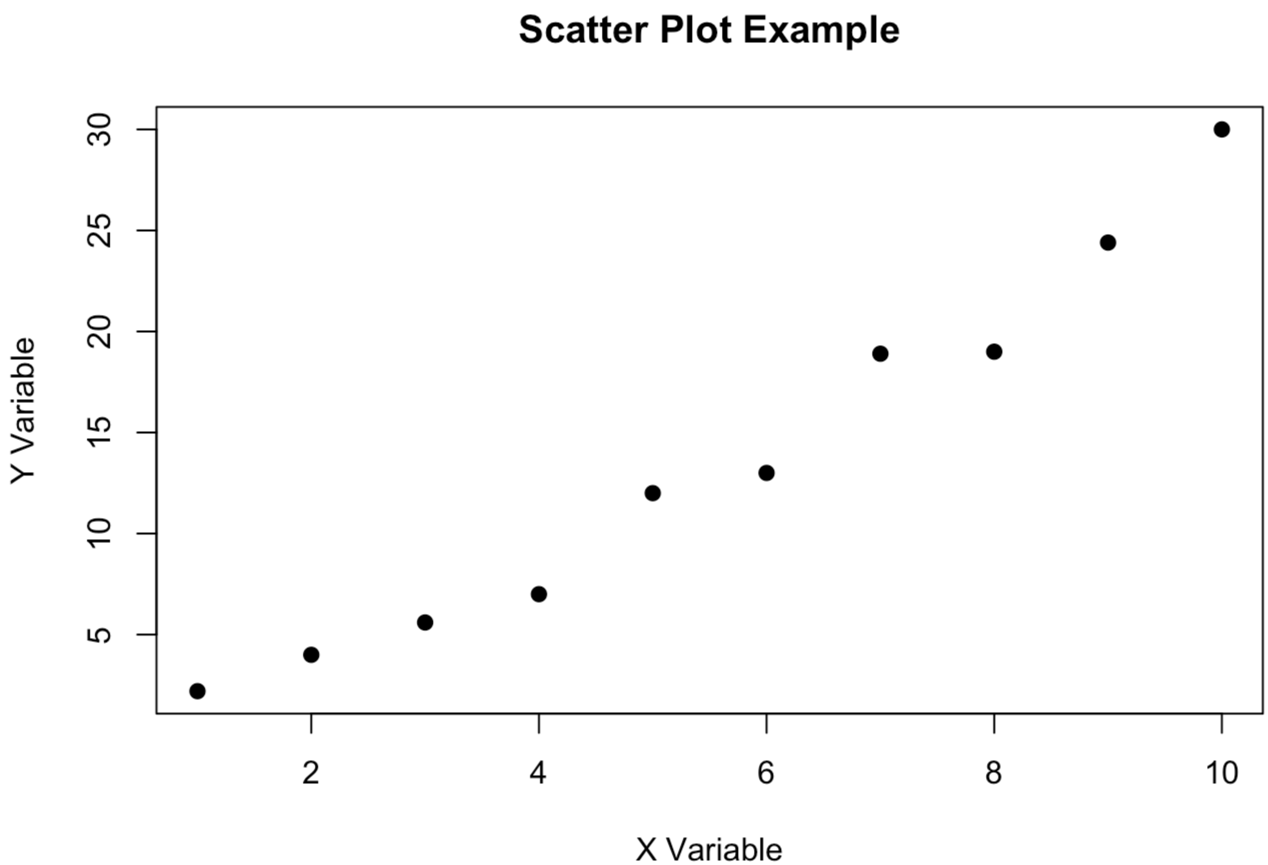
There are a few different plotting systems in R, we will cover the Base R plotting system and the ggplot2 package.

Base R Graphics - basic plots that use high-level plotting functions
Based_R-charts (https://r-charts.com/base-r/)

# 1. Scatter plots (simple and with fit line)

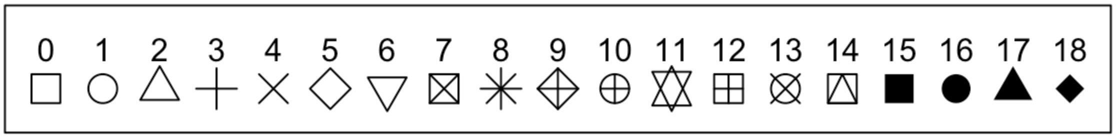
Displays values for typically two variables as points on a Cartesian coordinate system. Each point on the scatter plot represents an observation in your data, with its position determined by the values of the two variables being plotted

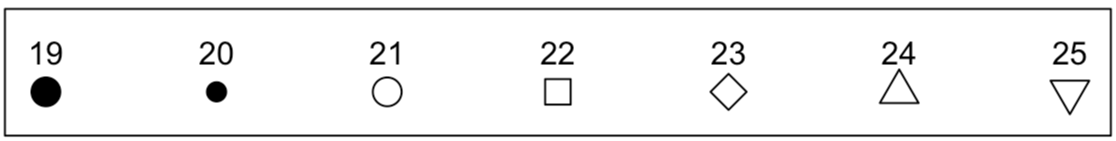
```
# Example data (continuous data)
x <- 1:10
y <- c(2.2, 4, 5.6, 7, 12, 13, 18.9, 19, 24.4, 30)

# Create a scatter plot
plot(x, y, main = "Scatter Plot Example", xlab = "X Variable",
     ylab = "Y Variable", pch = 19)
```




Display pch (plotting 'character') values

```
par(mfrow = c(2, 1))

# Plot the first set (pch = 0 to 18)
plot(0:18, rep(1, 19), xlim = c(0, 18), ylim = c(0.5, 2), type = "n",
     xlab = "", ylab = "", xaxt = "n", yaxt = "n")
for (i in 0:18) {
  points(i, 1, pch = i, cex = 2)
  text(i, 1.5, labels = i)}

# Plot the second set (pch = 19 to 25)
plot(19:25, rep(1, 7), xlim = c(19, 25), ylim = c(0.5, 2), type = "n",
     xlab = "", ylab = "", xaxt = "n", yaxt = "n")
for (i in 19:25) {
  points(i, 1, pch = i, cex = 2)
  text(i, 1.5, labels = i)}
```

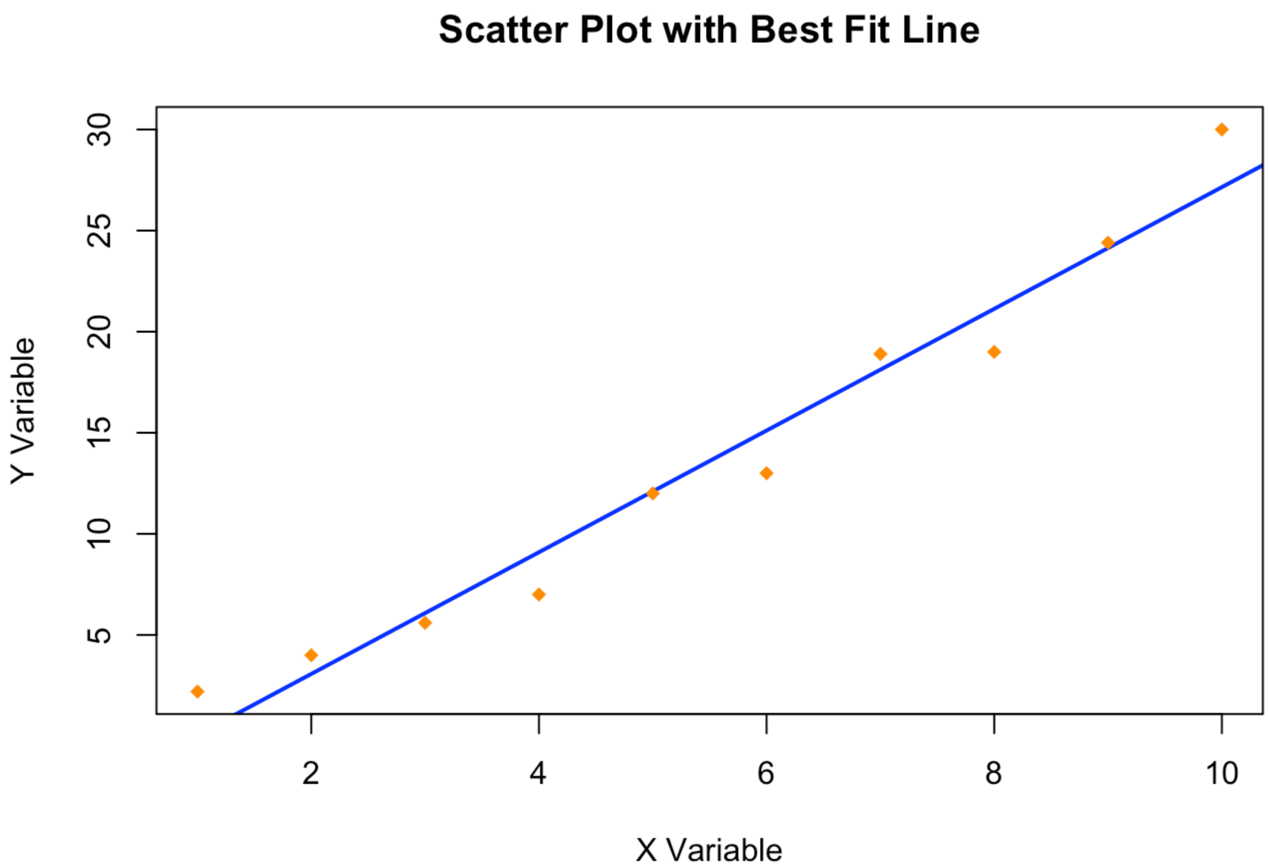



```
# Reset par to default
par(mfrow = c(1, 1), mar = c(5,4,4,2))
graphics.off()
```

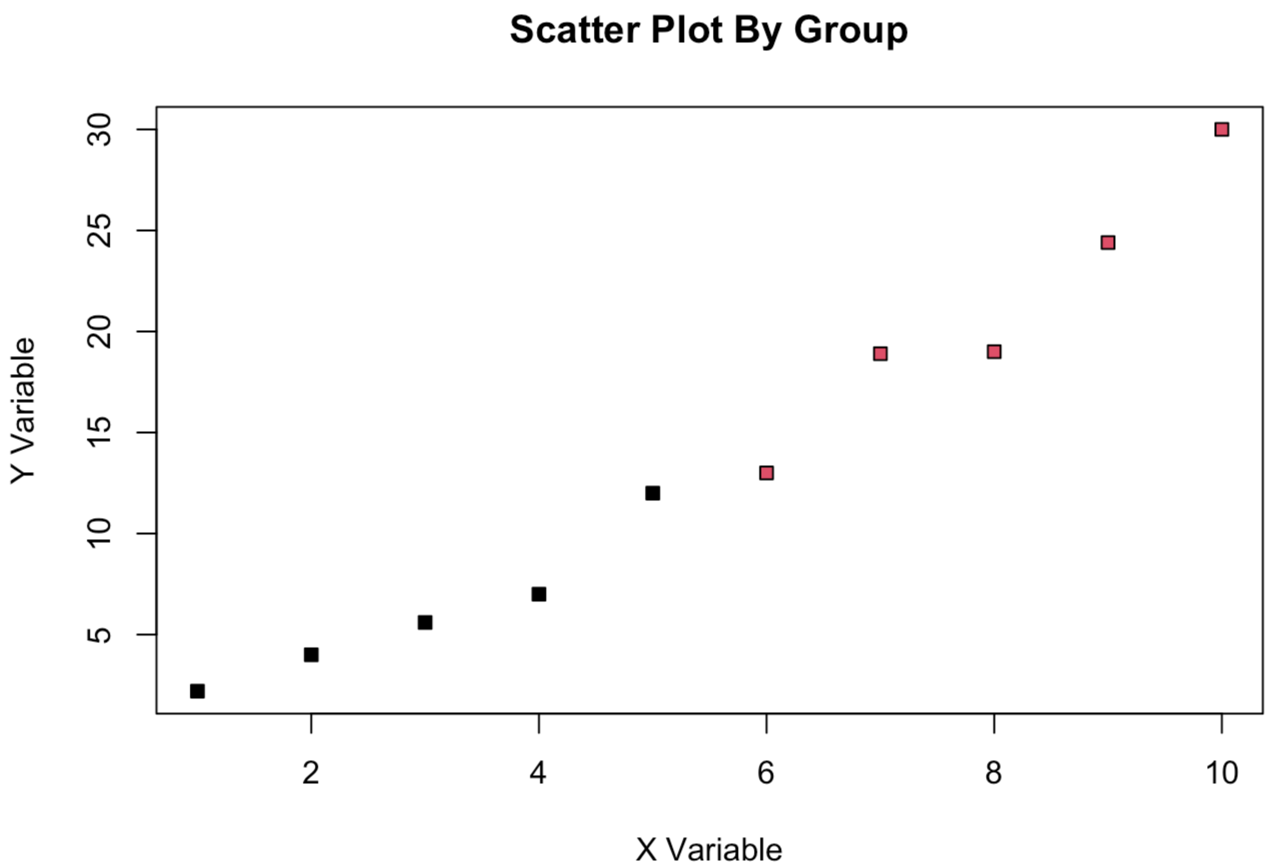
Add best fit line (regression line)
- Fit a linear model with lm()
- Use the abline() function after fitting a linear model

```
fit <- lm(y ~ x) # fit a linear model

plot(x, y, main = "Scatter Plot with Best Fit Line", xlab = "X Variable",
     ylab = "Y Variable", pch = 18, col = "darkorange",
     cex = 1, abline(fit, col = "blue", lwd = 2))
```




Scatterplot with color for group/category

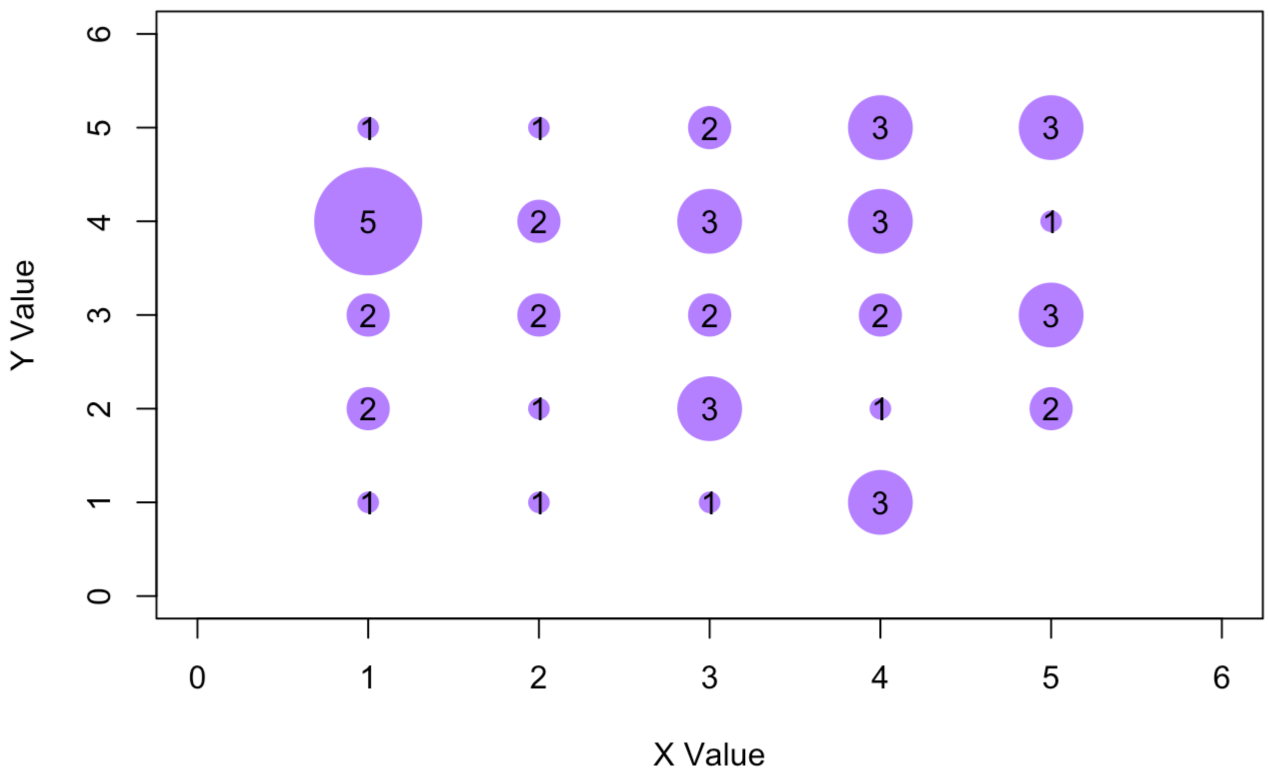
```
data = as.data.frame(cbind(x,y))
data$type = as.factor(c(rep("A", 5), rep("B", 5)))

plot(data$x, data$y, main = "Scatter Plot By Group", xlab = "X Variable",
     ylab = "Y Variable", pch = 22,
     cex = 1, bg = data$type)
```




```
# bg = a vector of background colors for open plot symbols, see points.
# Note: this is not the same setting as par("bg").
```

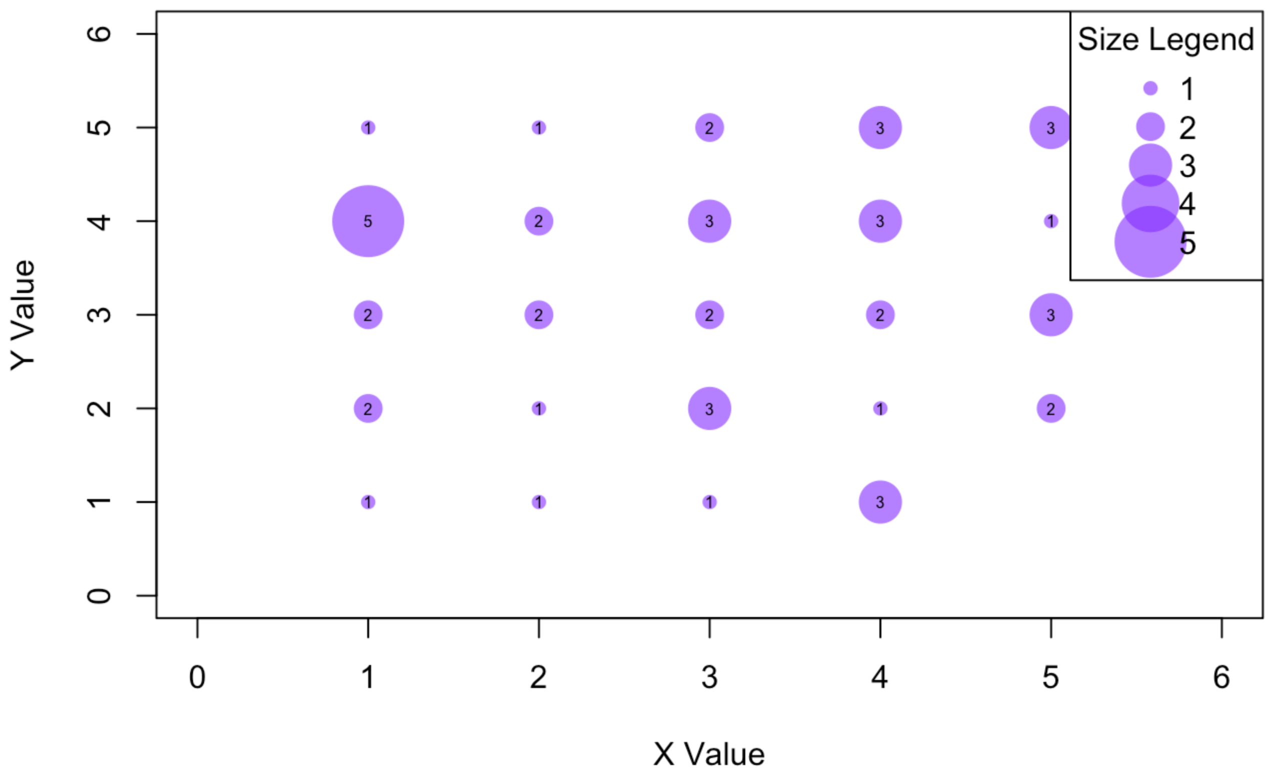
Example data (discrete data)
Create 2 discrete variables

```
xx <- sample(1:5, size = 50, replace = TRUE)
yy <- sample(1:5, size = 50, replace = TRUE)

## counts of discrete values
xy_table <- xyTable(xx,yy)

# plot
plot(xy_table$x , xy_table$y , cex=xy_table$number*1.5, pch=16 , col=rgb(0.5,0,1,0.6),
     xlab= "X Value" , ylab= "Y Value" , xlim=c(0,6) , ylim=c(0,6) )
text(xy_table$x , xy_table$y , xy_table$number)
```



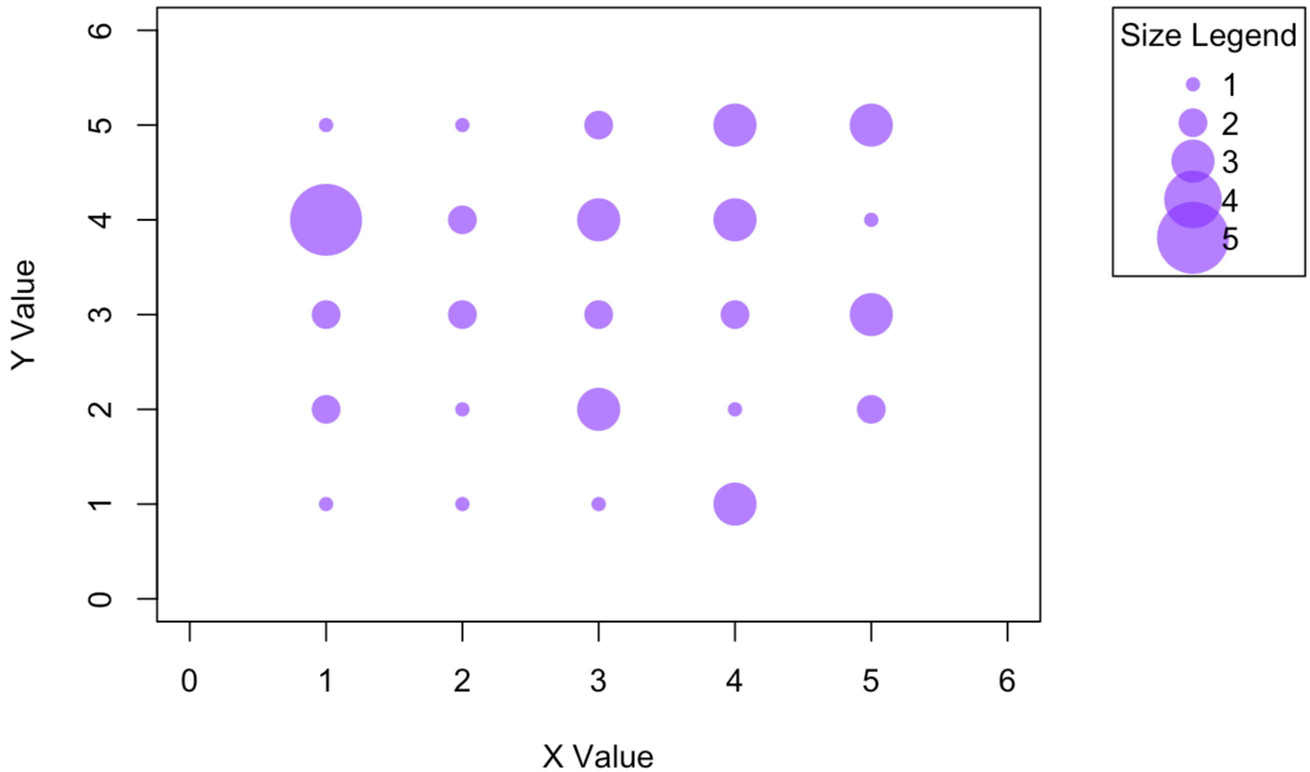

```
# rgb() function creates a color in R by specifying the levels of
# Red, Green, and Blue (RGB) components, along with an optional alpha
#value for transparency
```

```
plot(xy_table$x , xy_table$y , cex=xy_table$number, pch=16 , col=rgb(0.5,0,1,0.6) ,
     xlab= "X Value" , ylab= "Y Value" , xlim=c(0,6) , ylim=c(0,6) )
text(xy_table$x , xy_table$y , xy_table$number, cex = 0.5)
legend("topright",
       legend = c("1", "2", "3", "4", "5"),  # Descriptive labels for sizes
       pch = 16,  # Use the same point type (pch = 16)
       pt.cex = c(1, 2, 3, 4, 5),  # Different sizes for the points in the legend
       col = rgb(0.5,0,1,0.6),  # Same color as the plot
       title = "Size Legend")  # Add a title to the legend
```




Adjust margins to place the legend outside of the plot

```
# default 5,4,4,2
par(mar=c(5, 4, 4, 8), xpd=TRUE) # bottom, left, top, right margins
# par() = parameters of plot
# xpd = FALSE, plotting clipped to the plot region
# xpd = TRUE = plotting is clipped to the figure region

plot(xy_table$x , xy_table$y , cex=xy_table$number, pch=16 , col=rgb(0.5,0,1,0.6) ,
     xlab= "X Value" , ylab= "Y Value" , xlim=c(0,6) , ylim=c(0,6) )
# text(xy_table$x , xy_table$y , xy_table$number, cex = 0.5)

#add legend outside of plot
legend("topright",
       legend = c("1", "2", "3", "4", "5"),  # Descriptive labels for sizes
       pch = 16,  # Use the same point type (pch = 16)
       pt.cex = c(1, 2, 3, 4, 5),  # Different sizes for the points in the legend
       col = rgb(0.5,0,1,0.6),  # Same color as the plot
       title = "Size Legend", # Add a title to the legend
       inset = c(-0.3,0))
```

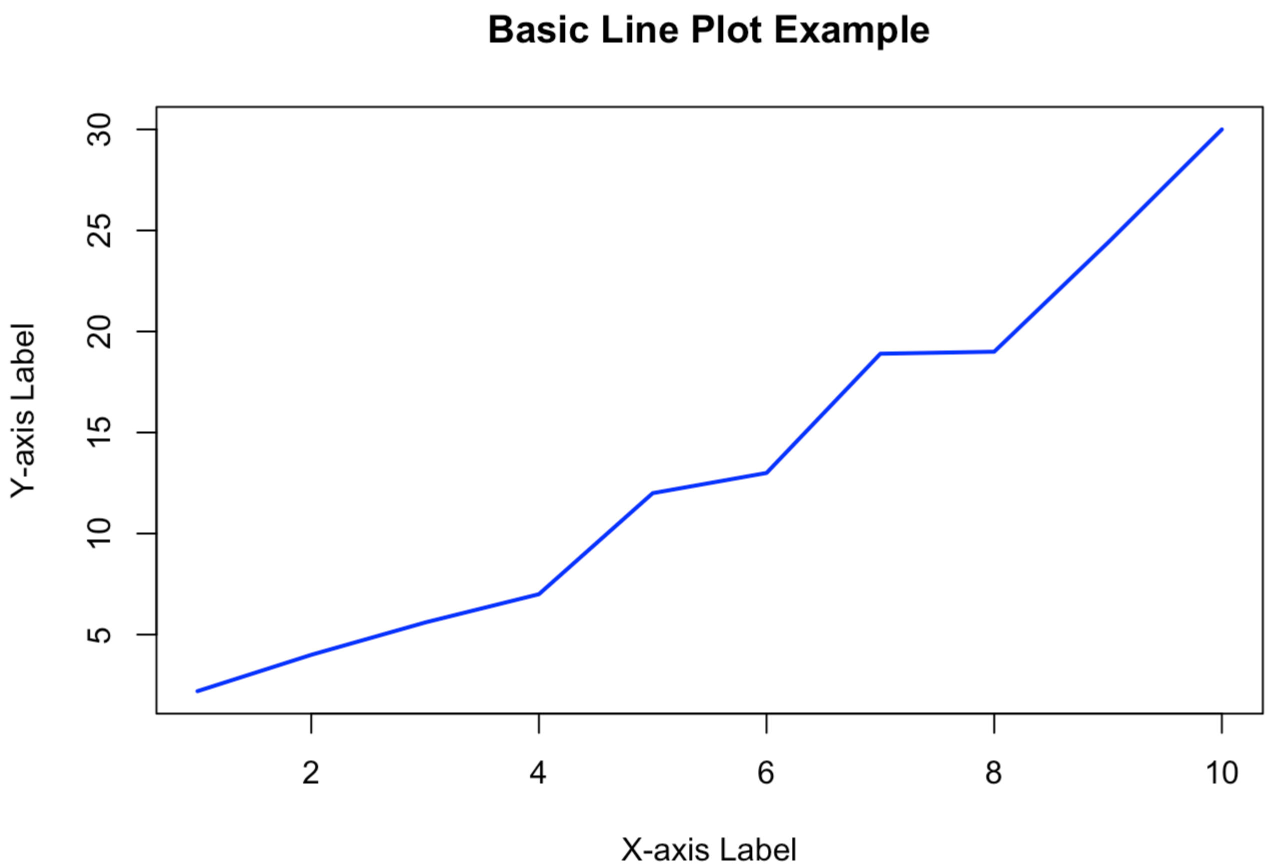



```
# inset = inset distances from the margins as a fraction of the plot
# inset = c(-0.3, 0) this moves the legend 30% of the plot width to the left
# of the top-right corner of the plot.

# Reset par to default
par(mfrow = c(1, 1), mar = c(5,4,4,2))
graphics.off()
```

# 2. Line plots

Example data (continuous data)

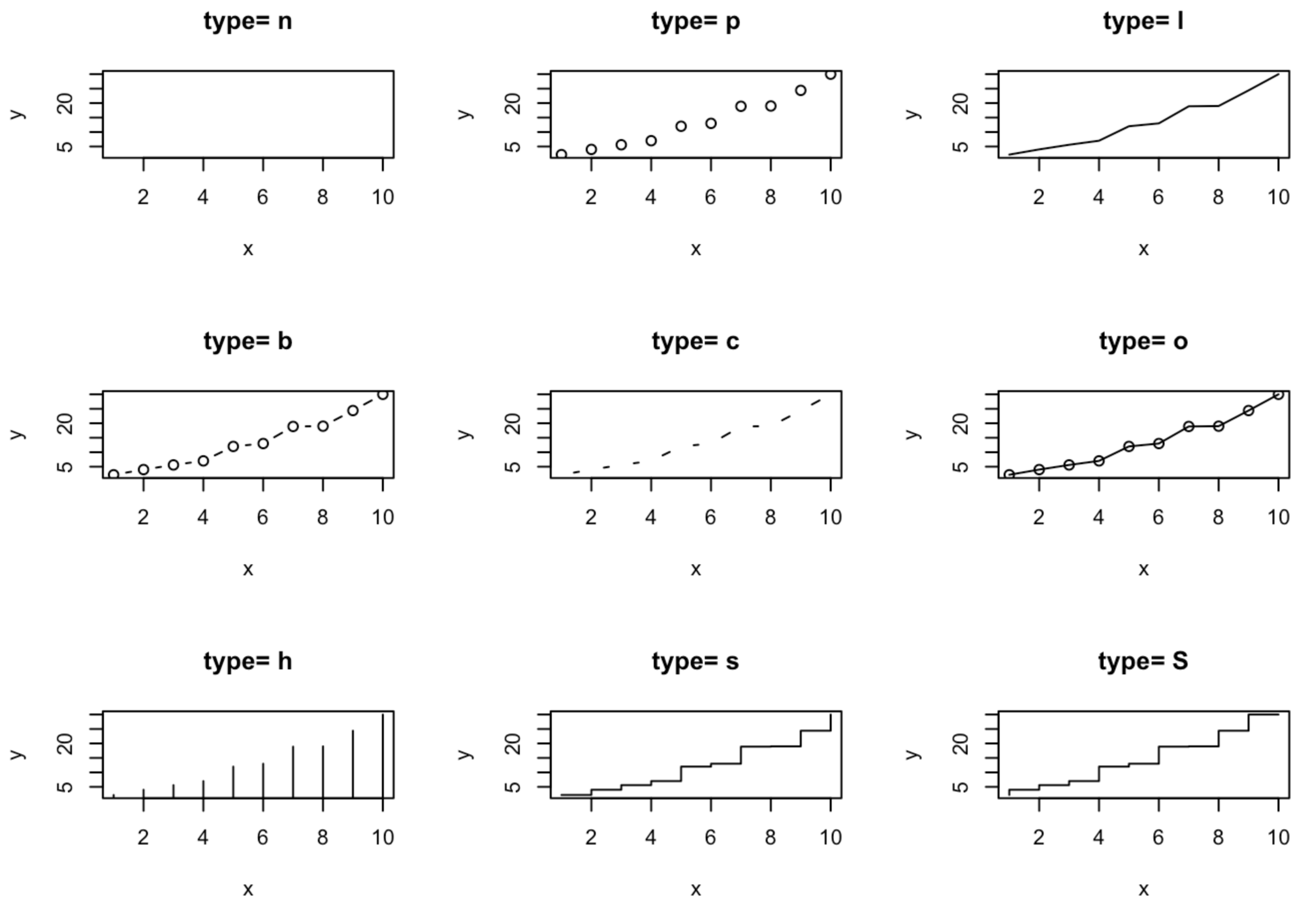
```
x <- 1:10
y <- c(2.2, 4, 5.6, 7, 12, 13, 18.9, 19, 24.4, 30)

plot(x, y, type = "l", col = "blue", lwd = 2,
     main = "Basic Line Plot Example",
     xlab = "X-axis Label",
     ylab = "Y-axis Label")
```




Line parameters:
type = line plot type
lty = line styles
lwd = line width
col = line color
pch = symbols

Display all line plot types:
“n” No plotting
“p” Points
“l” Lines
“b” Lines and points
“c” Lines without the part of the points
“o” Lines and points (overplotted)
“h” Histogram-like
“s” Stairs (first line horizontal)
“S” Stairs (first line vertical)

```
par(pch=21, col="black") # plotting symbol and color
# for pch symbols: https://r-charts.com/base-r/pch-symbols/
par(mfrow=c(3,3)) # all plots on one page
line_types = c("n","p","l","b","c","o","h","s","S")

for(i in 1:length(line_types)){
  heading = paste("type=",line_types[i])
  plot(x, y, main=heading,
       type=line_types[i])}
```

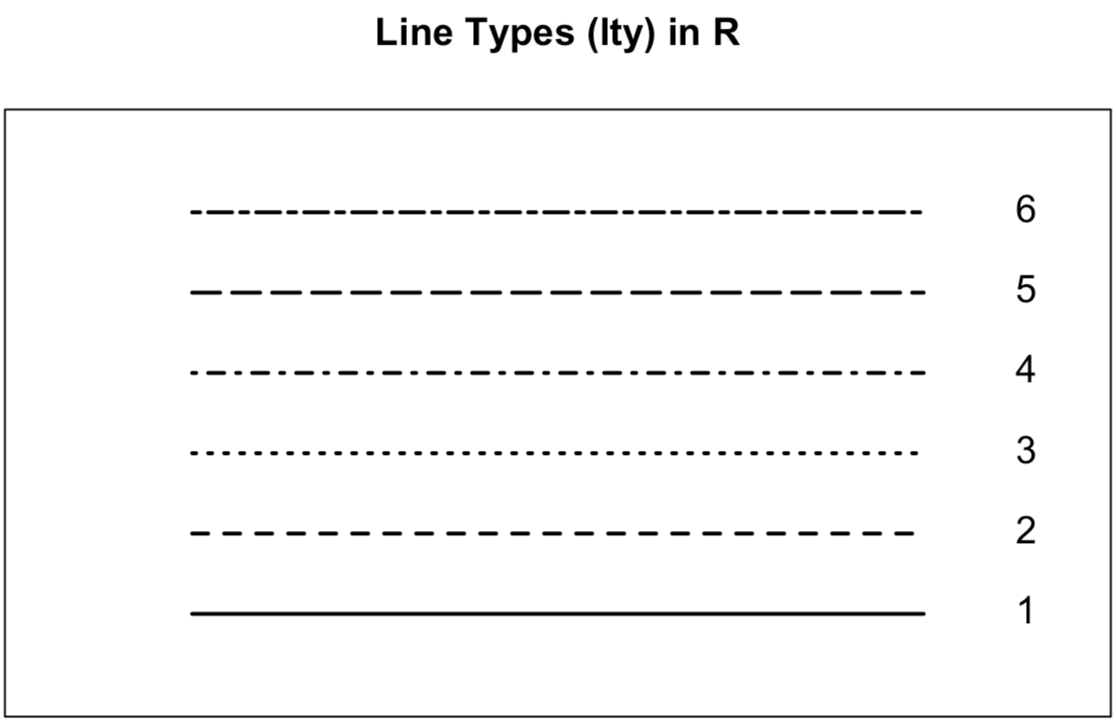



```
par(mfrow = c(1,1))
graphics.off()
```

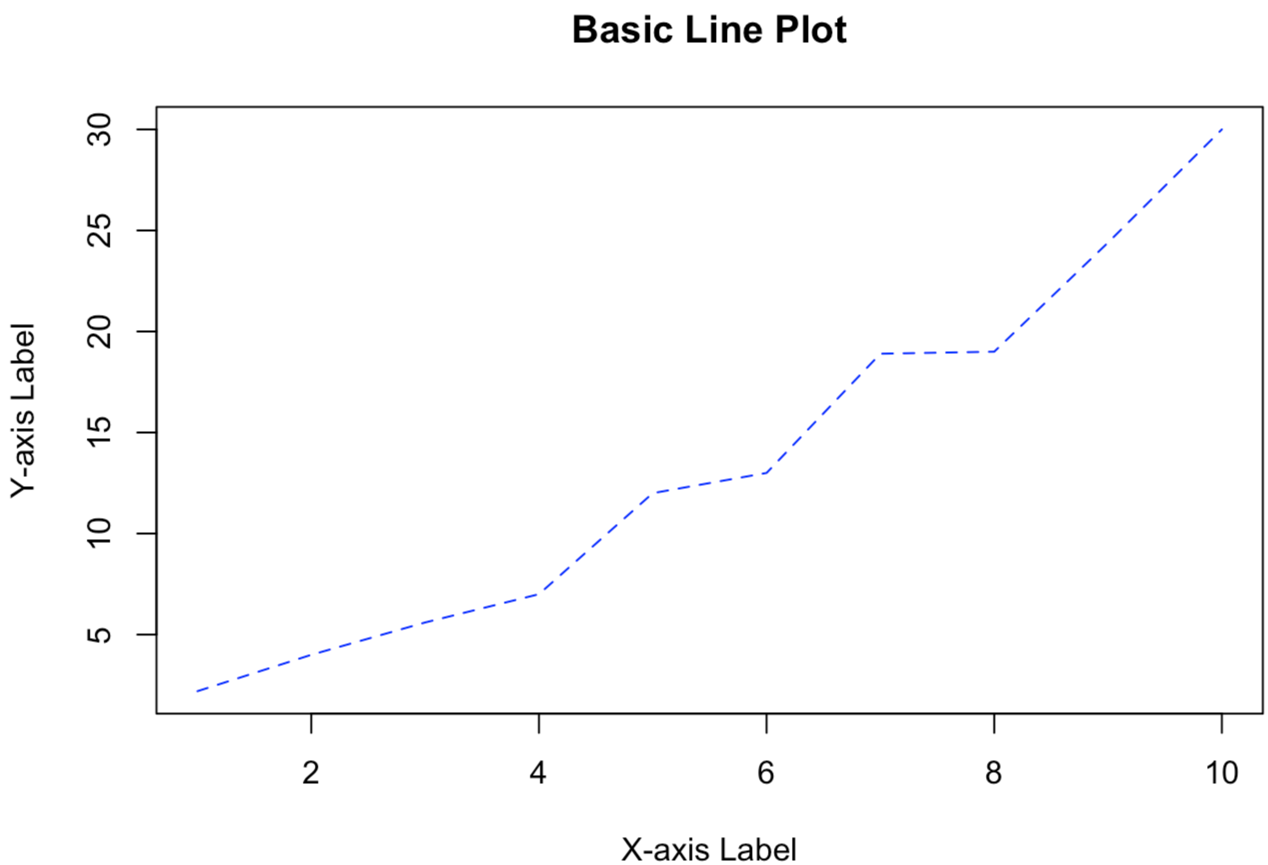
Display line style types

R’s base graphics has 6 standard line types

```
lty_values <- 1:6  # R's base graphics has 6 standard line types

# Set up the plot area
plot(1, type = "n", xlim = c(0, 7), ylim = c(0, 7), xlab = "", ylab = "",
     xaxt = "n", yaxt = "n", main = "Line Types (lty) in R")

# Loop through each lty value and plot it
for (i in lty_values) {
  # Draw the line with corresponding lty
  lines(c(1, 6), c(i, i), lty = i, lwd = 2)

  # Add the label with the lty number
  text(6.5, i, labels = i, cex = 1.2, pos = 4)}
```

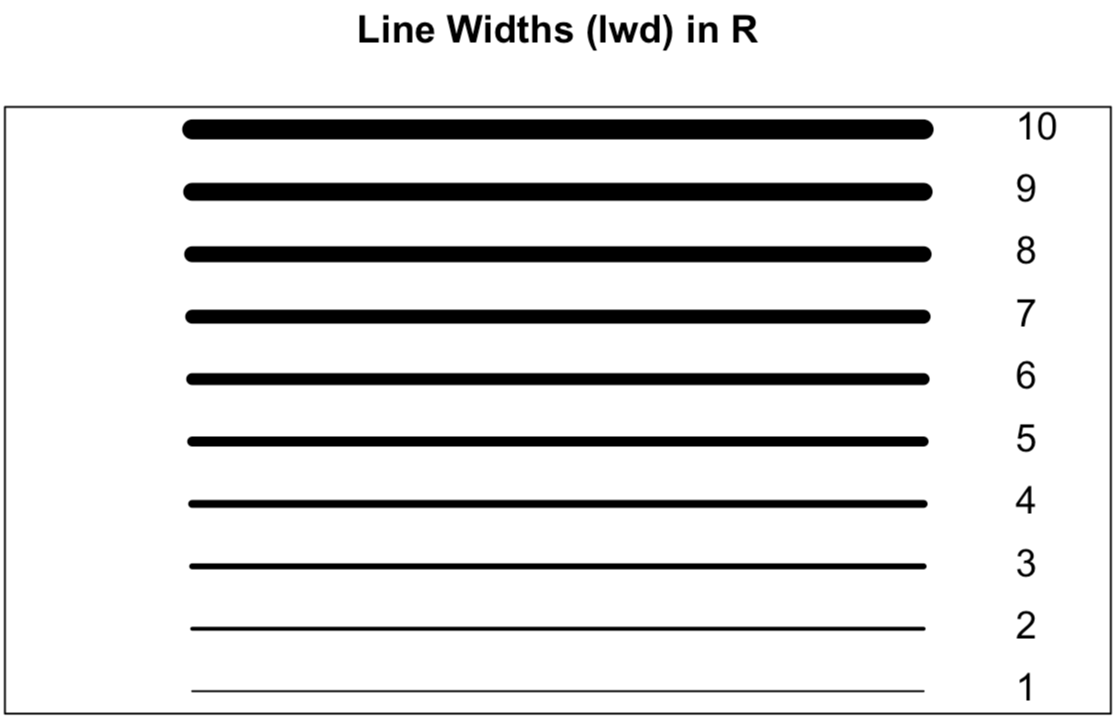



```
plot(x, y, type = "l", col = "blue", lty = 2,
     main = "Basic Line Plot",
     xlab = "X-axis Label",
     ylab = "Y-axis Label")
```

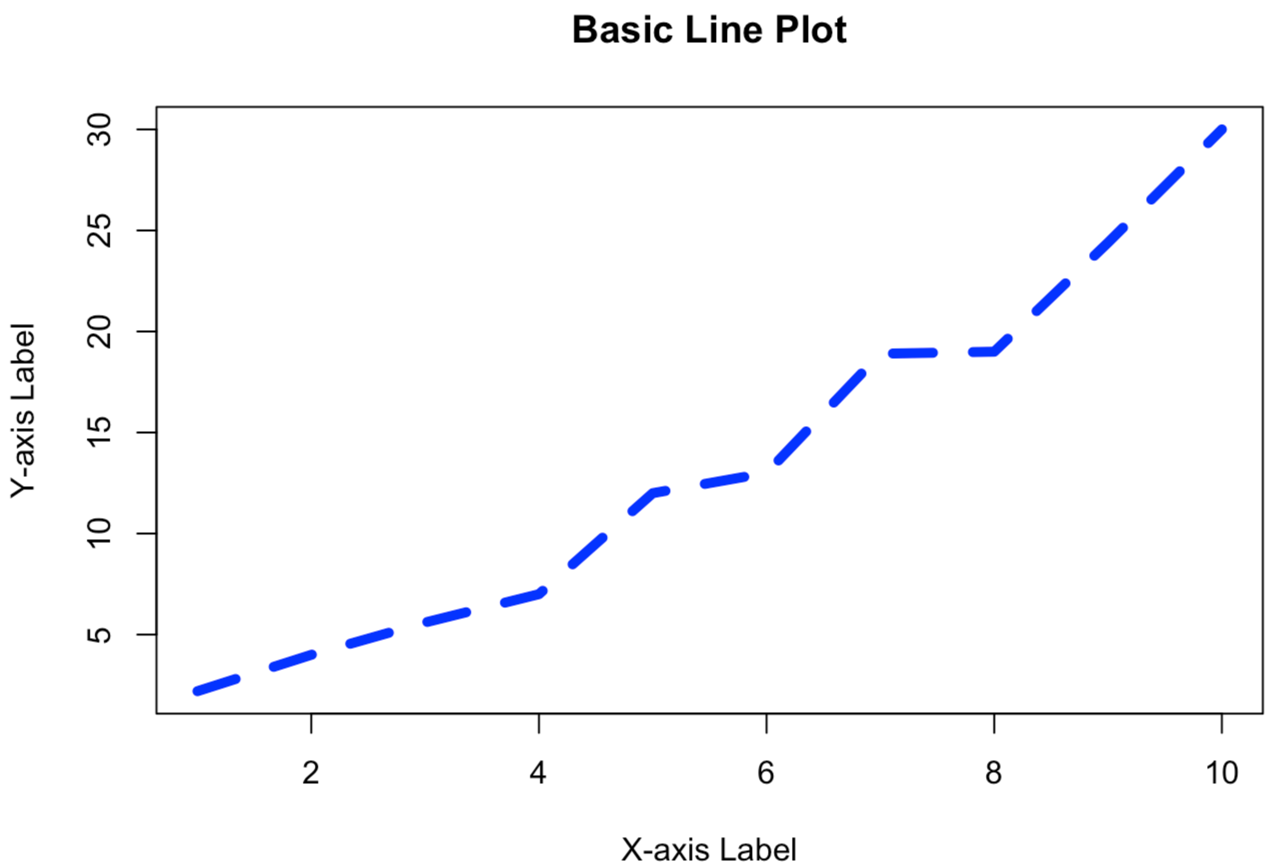



Display a range of lwd values

```
lwd_values <- seq(1, 10, by = 1)  # Varying line widths from 1 to 10 in steps of 1

# Set up the plot area
plot(1, type = "n", xlim = c(0, 7), ylim = c(1, length(lwd_values)), xlab = "", ylab =
"",
     xaxt = "n", yaxt = "n", main = "Line Widths (lwd) in R")

# Loop through each lwd value and plot it
for (i in seq_along(lwd_values)) {
  # Draw the line with corresponding lwd
  lines(c(1, 6), c(i, i), lwd = lwd_values[i])

  # Add the label with the lwd value
  text(6.5, i, labels = lwd_values[i], cex = 1.2, pos = 4)}
```




```
plot(x, y, type = "l", col = "blue", lwd = 5, lty = 2,
     main = "Basic Line Plot",
     xlab = "X-axis Label",
     ylab = "Y-axis Label")
```




### Framingham datasest

```
df <- read_excel("framingham.xlsx")
str(df) # structure function to display the internal structure of an
```

```
## tibble [4,240 × 16] (S3: tbl_df/tbl/data.frame)
##  $ male           : num [1:4240] 1 0 1 0 0 0 0 0 1 1 ...
##  $ age            : num [1:4240] 39 46 48 61 46 43 63 45 52 43 ...
##  $ education      : chr [1:4240] "4" "2" "1" "3" ...
##  $ currentSmoker  : num [1:4240] 0 0 1 1 1 0 0 1 0 1 ...
##  $ cigsPerDay     : chr [1:4240] "0" "0" "20" "30" ...
##  $ BPMeds         : chr [1:4240] "0" "0" "0" "0" ...
##  $ prevalentStroke: num [1:4240] 0 0 0 0 0 0 0 0 0 0 ...
##  $ prevalentHyp   : num [1:4240] 0 0 0 1 0 1 0 0 1 1 ...
##  $ diabetes       : num [1:4240] 0 0 0 0 0 0 0 0 0 0 ...
##  $ totChol        : chr [1:4240] "195" "250" "245" "225" ...
##  $ sysBP          : num [1:4240] 106 121 128 150 130 ...
##  $ diaBP          : num [1:4240] 70 81 80 95 84 110 71 71 89 107 ...
##  $ BMI            : chr [1:4240] "26.97" "28.73" "25.34" "28.58" ...
##  $ heartRate      : chr [1:4240] "80" "95" "75" "65" ...
##  $ glucose        : chr [1:4240] "77" "76" "70" "103" ...
##  $ TenYearCHD     : num [1:4240] 0 0 0 1 0 0 1 0 0 0 ...
```

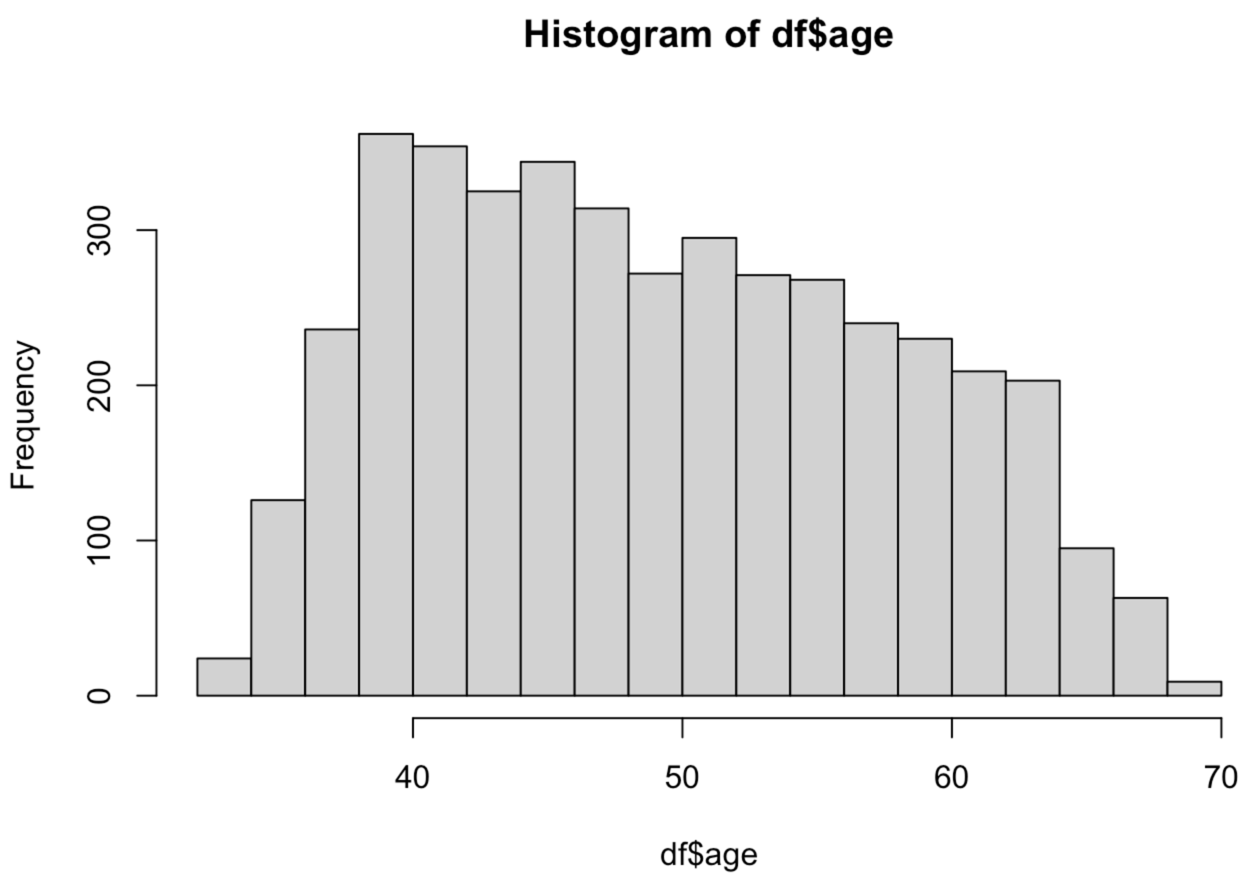
```
# R object
```

# 3. Density plots - understand the distribution of the data

- histograms
- kernal density plots

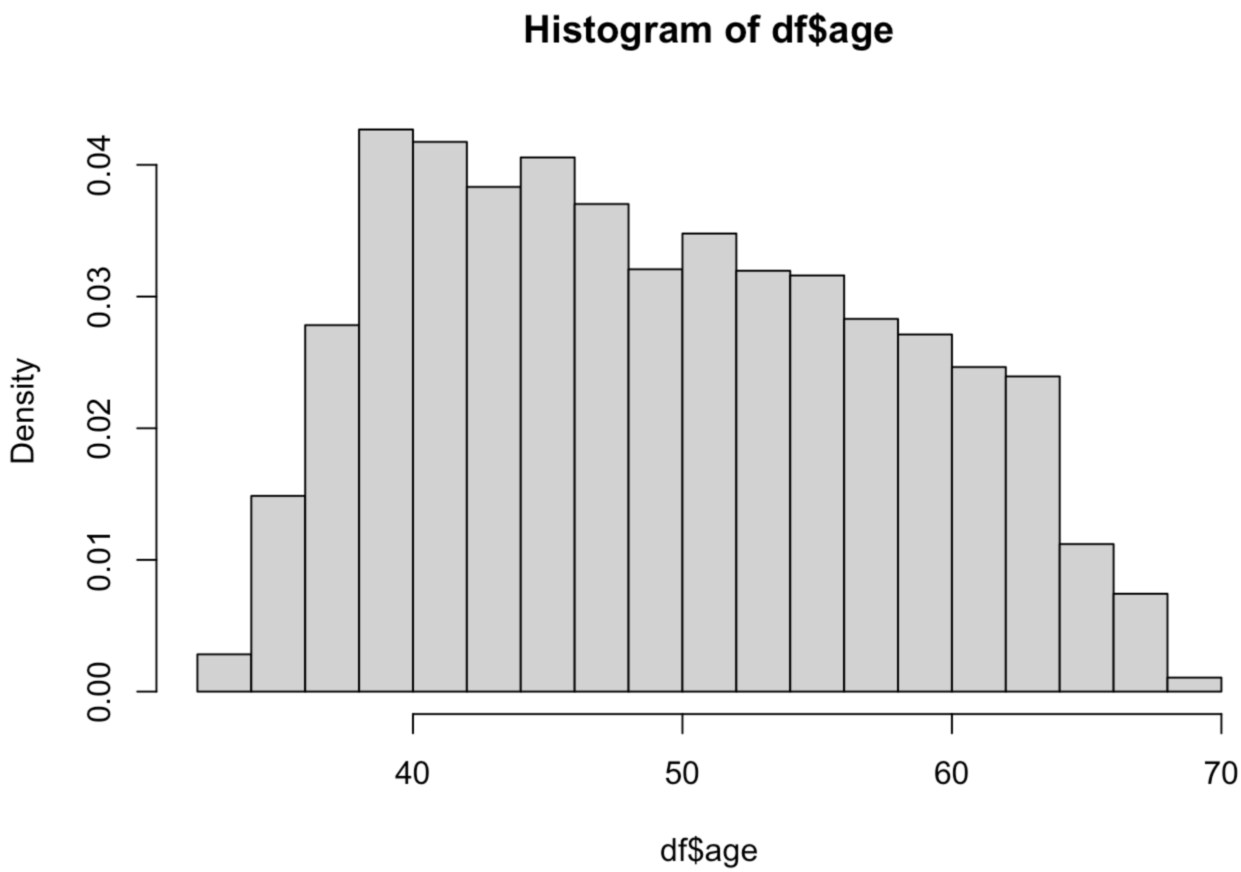
Simple Histogram

```
hist(df$age)
```




```
# frequency = counts component: the number of data points in each bin/category
```

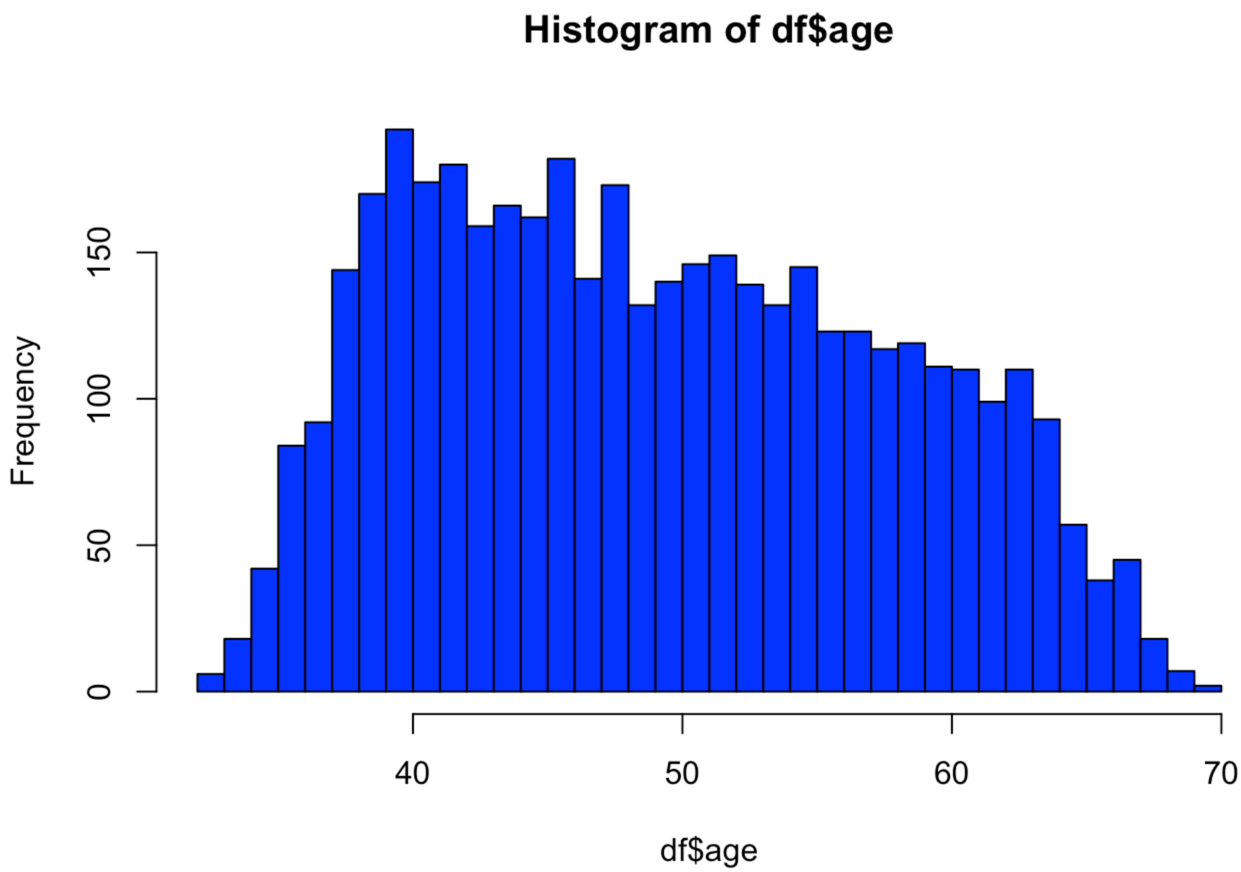
```
hist(df$age, freq = FALSE)
```



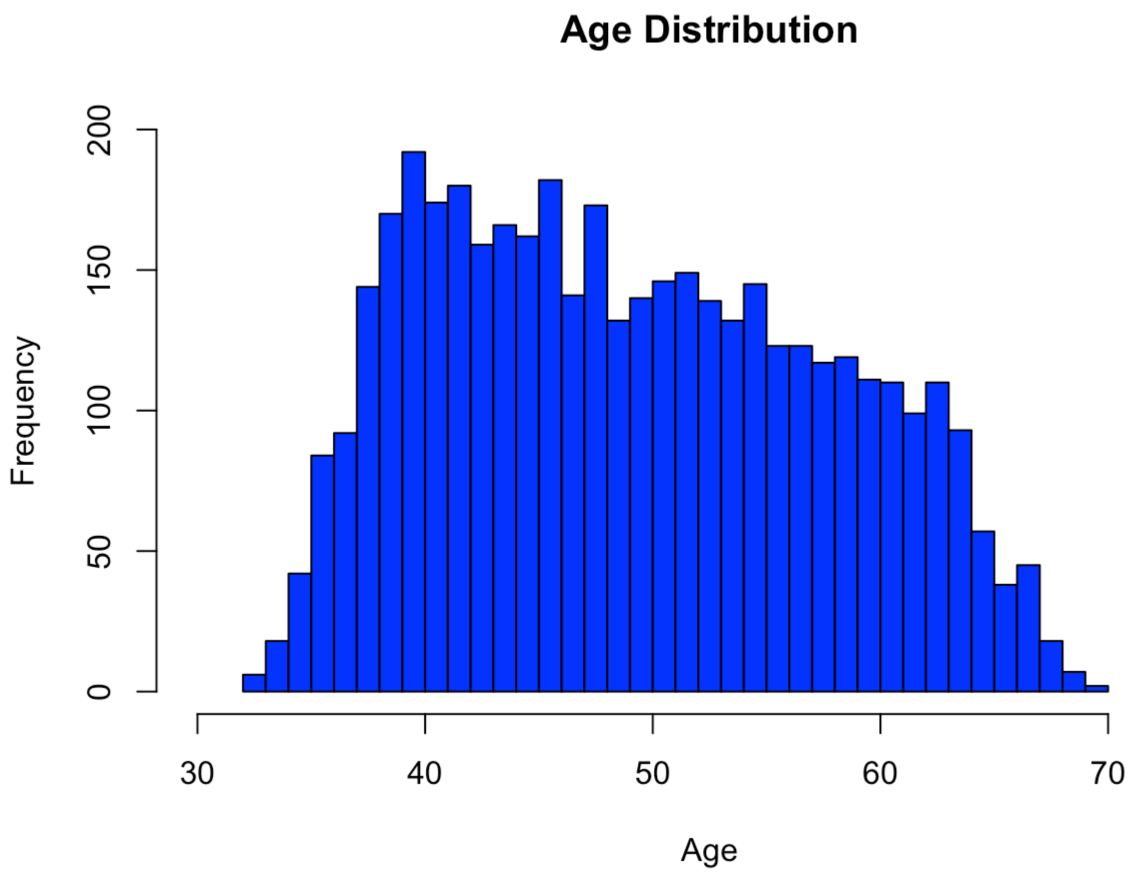

```
# density = modified relative frequency (absolute frequency/total number
# of values for the variable) that represents
# the percentage of the data that belongs to a specific category. The
# area of the entire histogram is always equal to 1.
```

Colored Histogram with Different Number of Bins

```
hist(df$age, breaks=30, col="blue")
```




```
# add labels and change the range of x and y values
hist(df$age, breaks=30, col="blue", xlab="Age",
     main="Age Distribution",
     xlim = c(30,75),
     ylim = c(0,200))
```

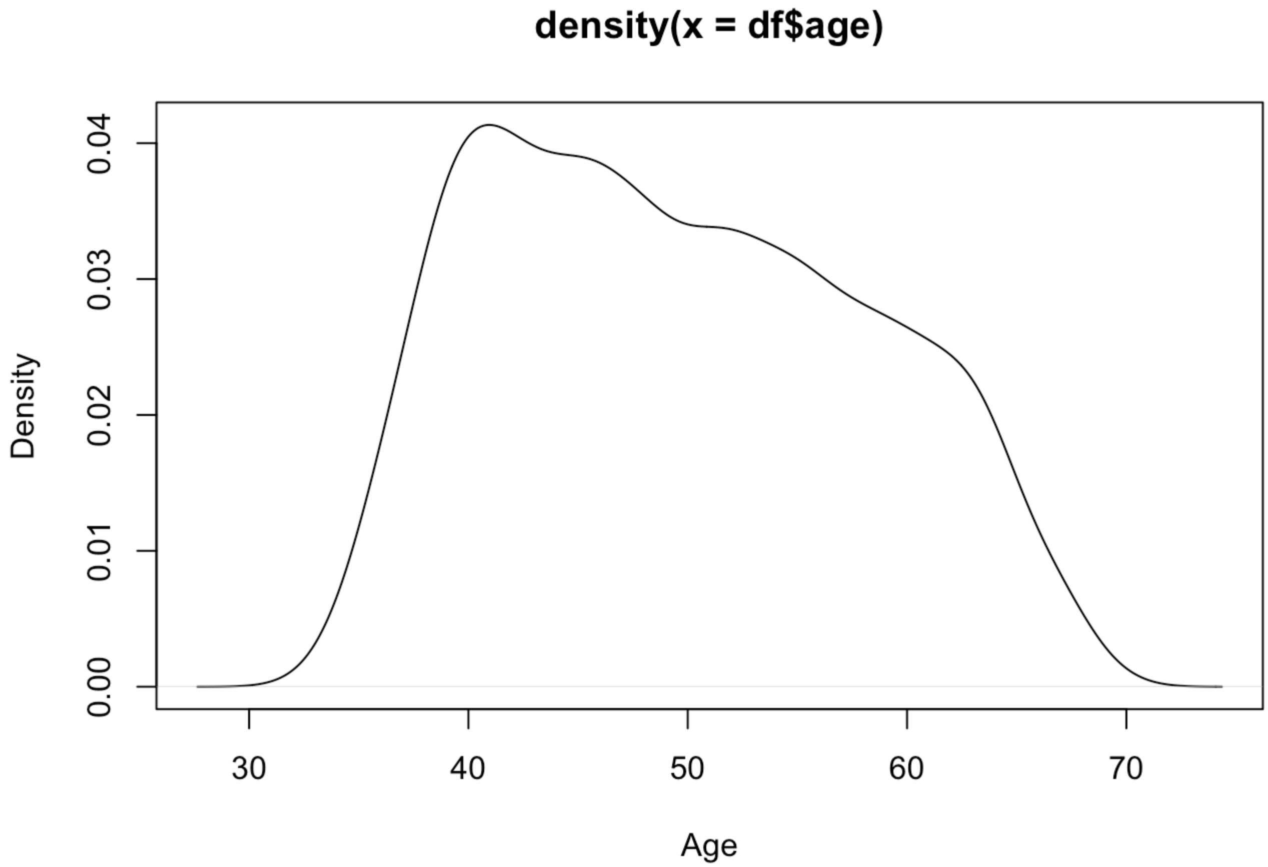



Kernal density plot = smooth version of the histogram

```
density(df$age) # default = gaussian
```

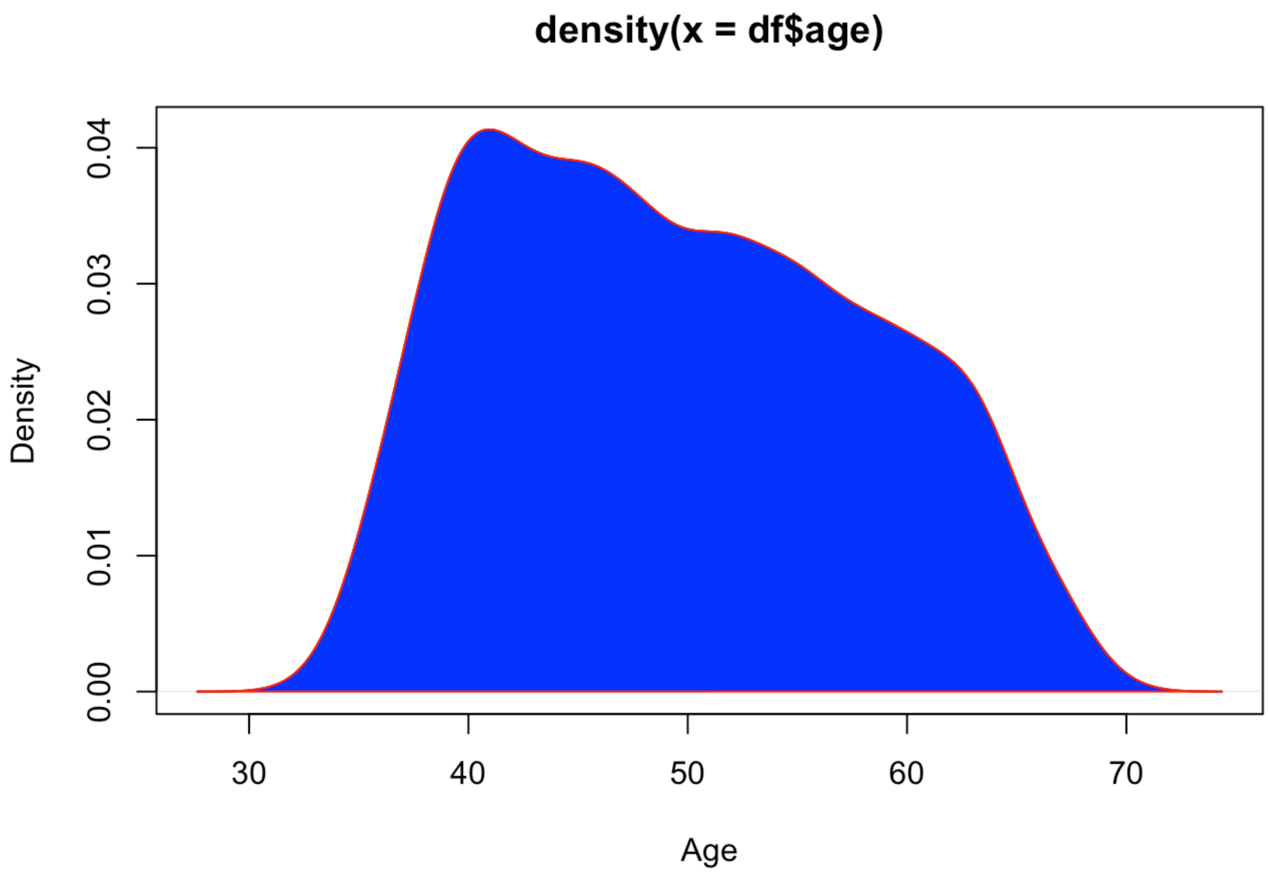
```
##
## Call:
##  density.default(x = df$age)
##
## Data: df$age (4240 obs.);    Bandwidth 'bw' = 1.452
##
##        x               y
##  Min.   :27.64   Min.   :1.180e-06
##  1st Qu.:39.32   1st Qu.:4.287e-03
##  Median :51.00   Median :2.592e-02
##  Mean   :51.00   Mean   :2.137e-02
##  3rd Qu.:62.68   3rd Qu.:3.399e-02
##  Max.   :74.36   Max.   :4.135e-02
```

```
# computes the kernal density estimates
plot(density(df$age), xlab = "Age")
```




Color fill the kernal density plot

```
d = density(df$age)
plot(density(df$age), xlab = "Age")
polygon(density(df$age), col="blue", border="red")
```

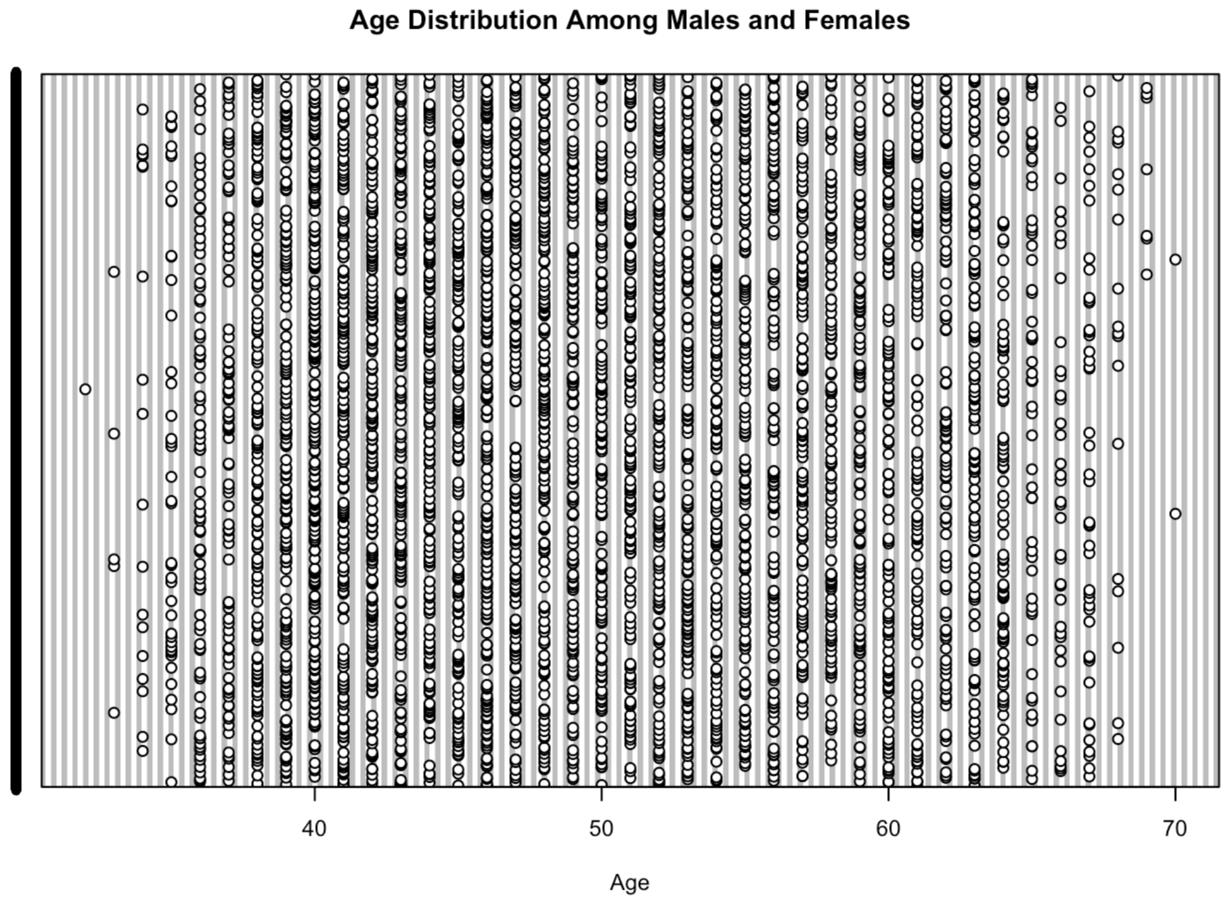



# 4. Dotplots = another way to visualize the distribution of small datasets

Useful when you want to see individual data points and their distribution without the aggregation that occurs in histograms or kernel density plots

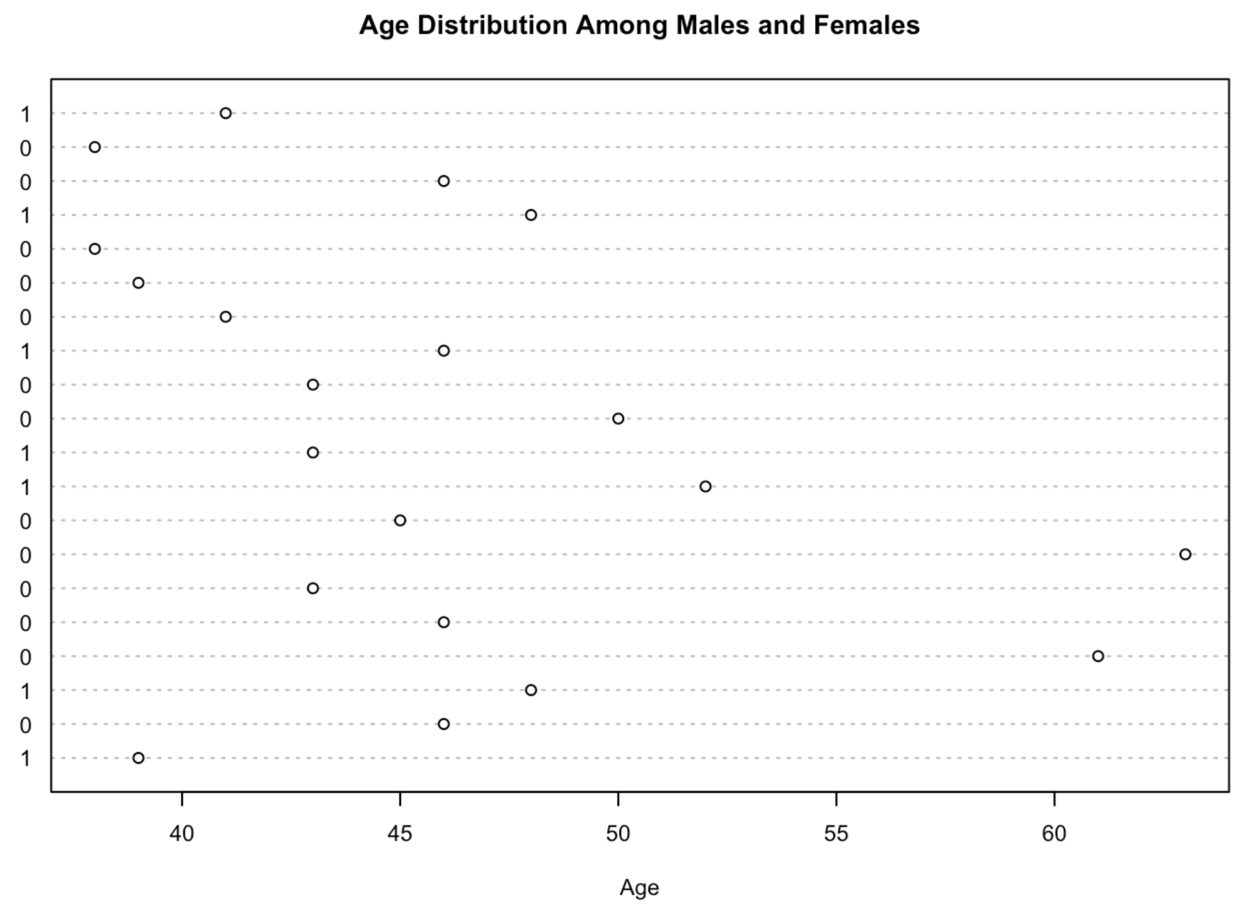
Simple Dotplot

```
dotchart(df$age,labels=df$male,cex=.7,
         main="Age Distribution Among Males and Females",
         xlab="Age")
```

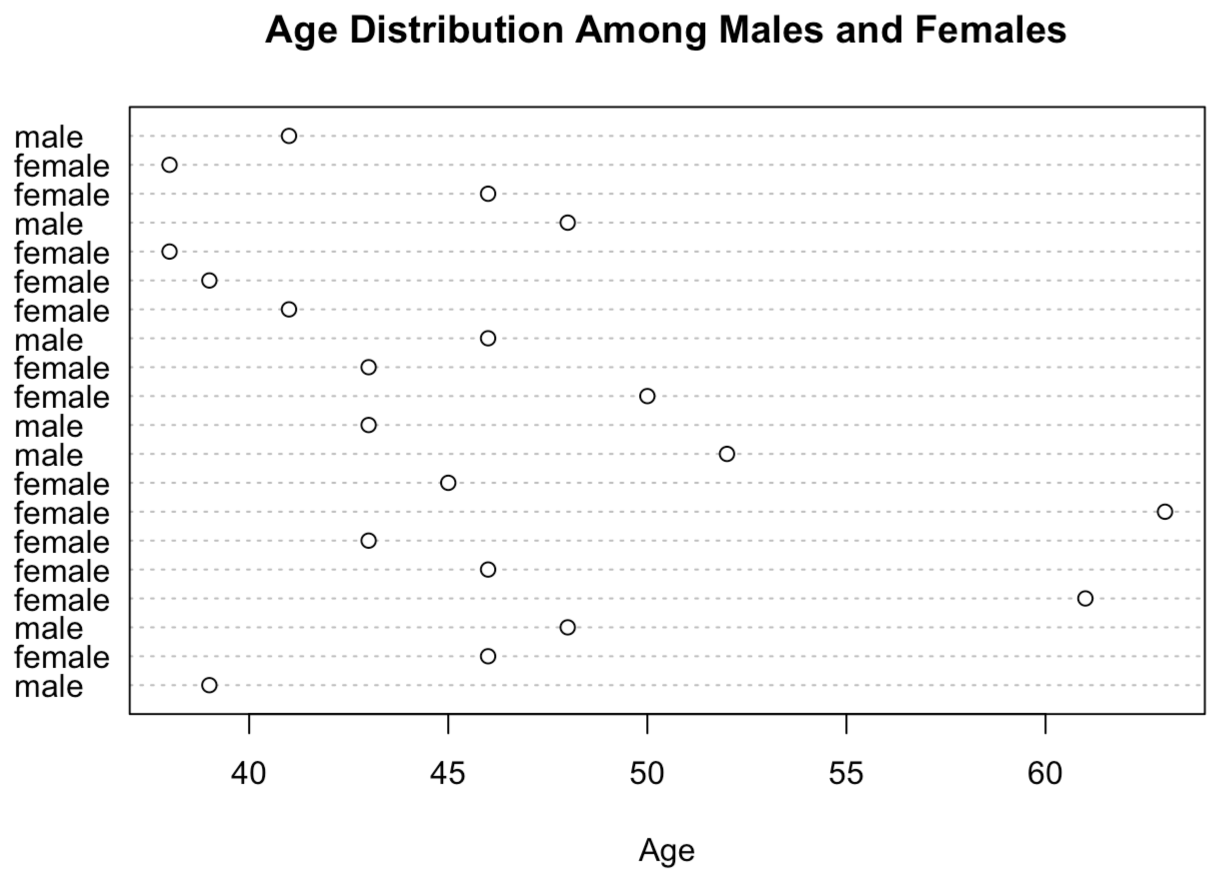



Take a subset of the data

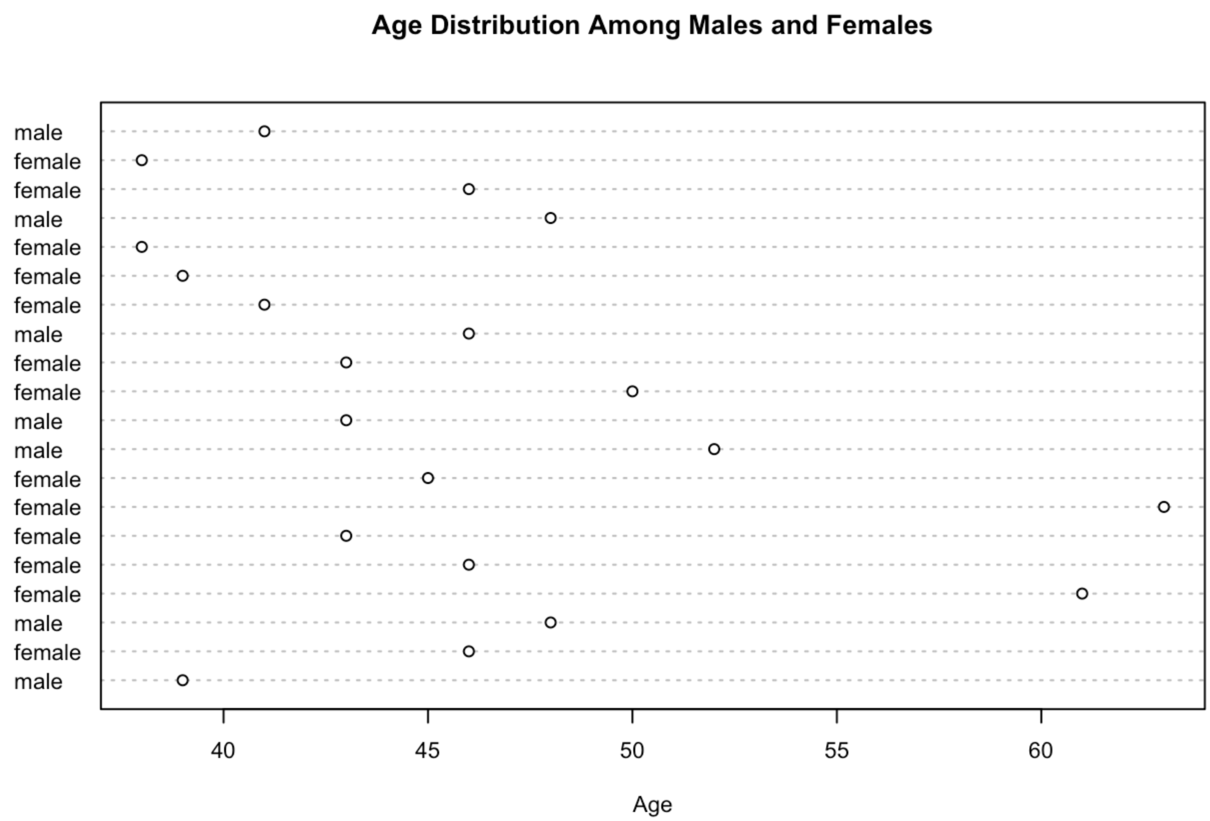
```
df_sub = head(df,20)
dotchart(df_sub$age,labels=df_sub$male,cex=.7,
         main="Age Distribution Among Males and Females",
         xlab="Age")
```




```
###
df_sub$gender = ifelse(df_sub$male == 0, "female", "male")
dotchart(df_sub$age,labels=df_sub$gender,
         main="Age Distribution Among Males and Females",
         xlab="Age")
```




```
dotchart(df_sub$age,labels=df_sub$gender,cex=.7,
         main="Age Distribution Among Males and Females",
         xlab="Age")
```

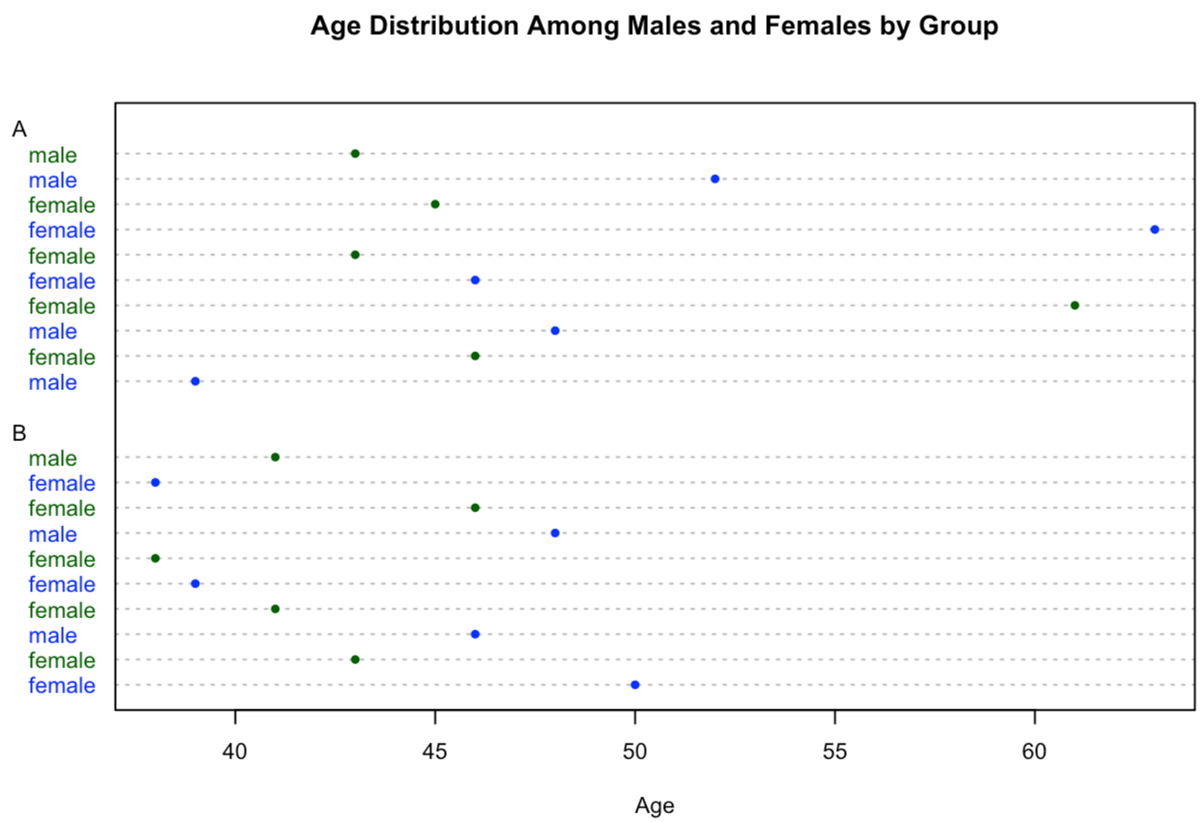



```
# cex = character size to be used, value smaller than one can help avoid
# label overlap
```

Randomly assign “Control” and “Test”

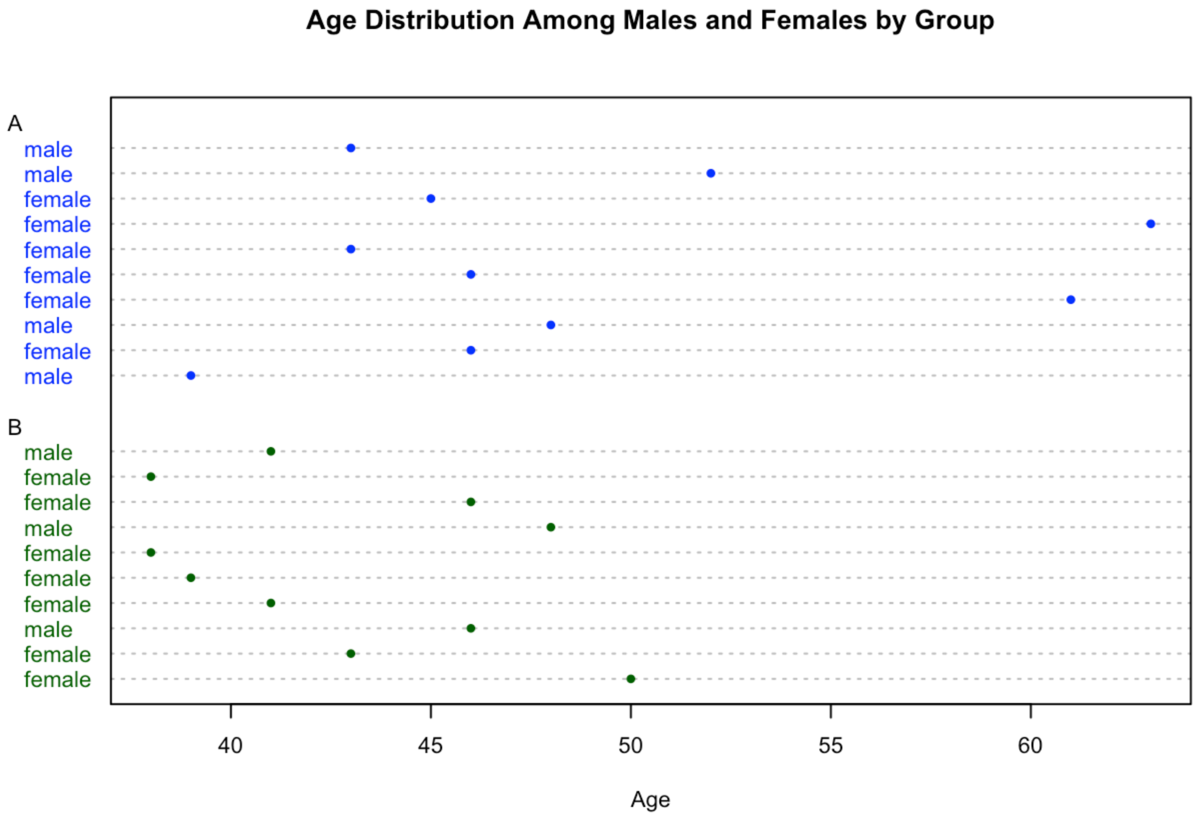
```
df_sub$group <- sample(c("Control", "Test"), size = nrow(df_sub), replace = TRUE)
# size = number of items to choose
# replace = should sampling be with replacement?
# determines whether sampling is done with or without replacement
# vector you're sampling from can be selected multiple times.
# This means that after an item is selected, it's "put back" into the
# pool of possible selections, so it can be picked again in subsequent
# draws
# replace = TRUE: You can select the same item multiple times
# (sampling with replacement).
# replace = FALSE: Each item can be selected only once
# (sampling without replacement).
```

```
df_sub$group = as.factor(c(rep("A", 10), rep("B", 10)))

dotchart(df_sub$age,labels=df_sub$gender,cex=.7,
        main="Age Distribution Among Males and Females by Group",
        xlab="Age", gcolor="black", groups= df_sub$group,
        col = c("blue", "darkgreen"),
        pch = 20)
```

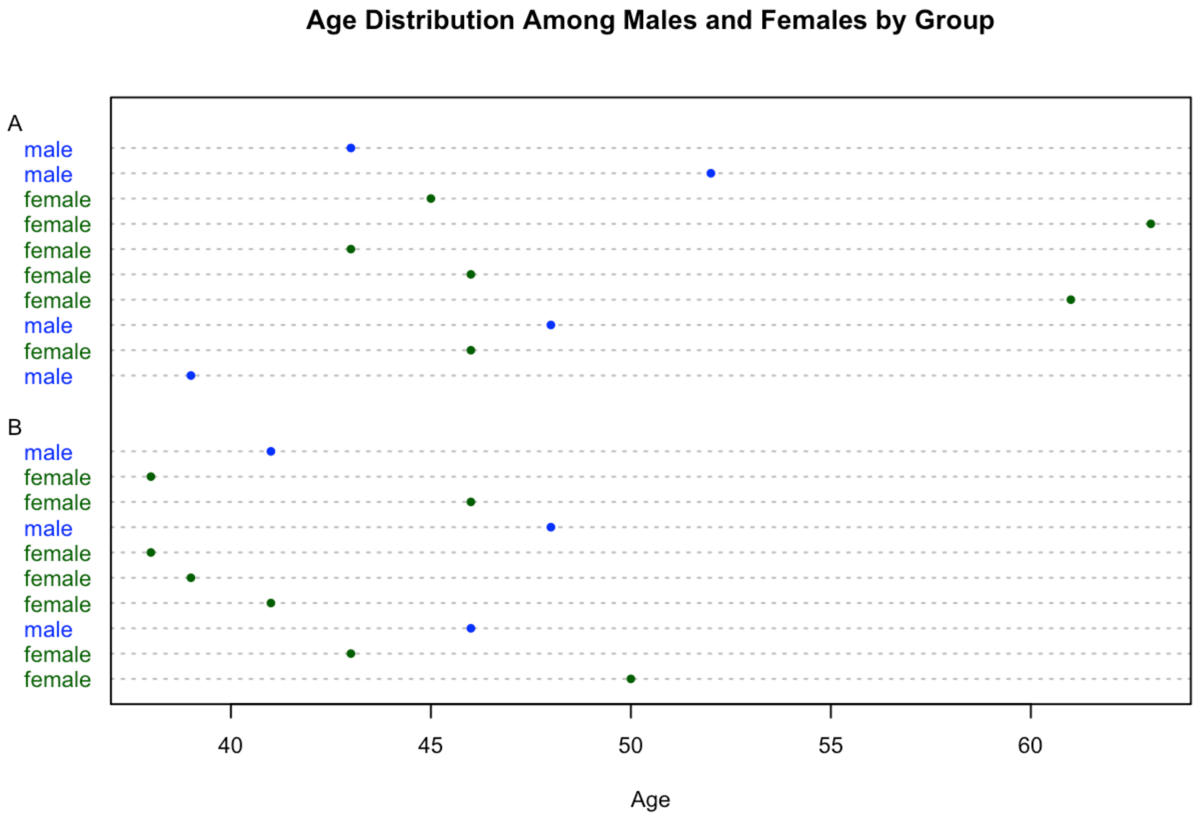



```
df_sub$color[df_sub$group == "A"] = "blue"
df_sub$color[df_sub$group == "B"] = "darkgreen"
```

```
dotchart(df_sub$age,labels=df_sub$gender,cex=.7,
        main="Age Distribution Among Males and Females by Group",
        xlab="Age", gcolor="black", groups= df_sub$group,
        col = df_sub$color,
        pch = 20)
```




```
df_sub$color[df_sub$gender == "male"] = "blue"
df_sub$color[df_sub$gender == "female"] = "darkgreen"

dotchart(df_sub$age,labels=df_sub$gender,cex=.7,
         main="Age Distribution Among Males and Females by Group",
         xlab="Age", gcolor="black", groups= df_sub$group,
         col = df_sub$color,
         pch = 20)
```

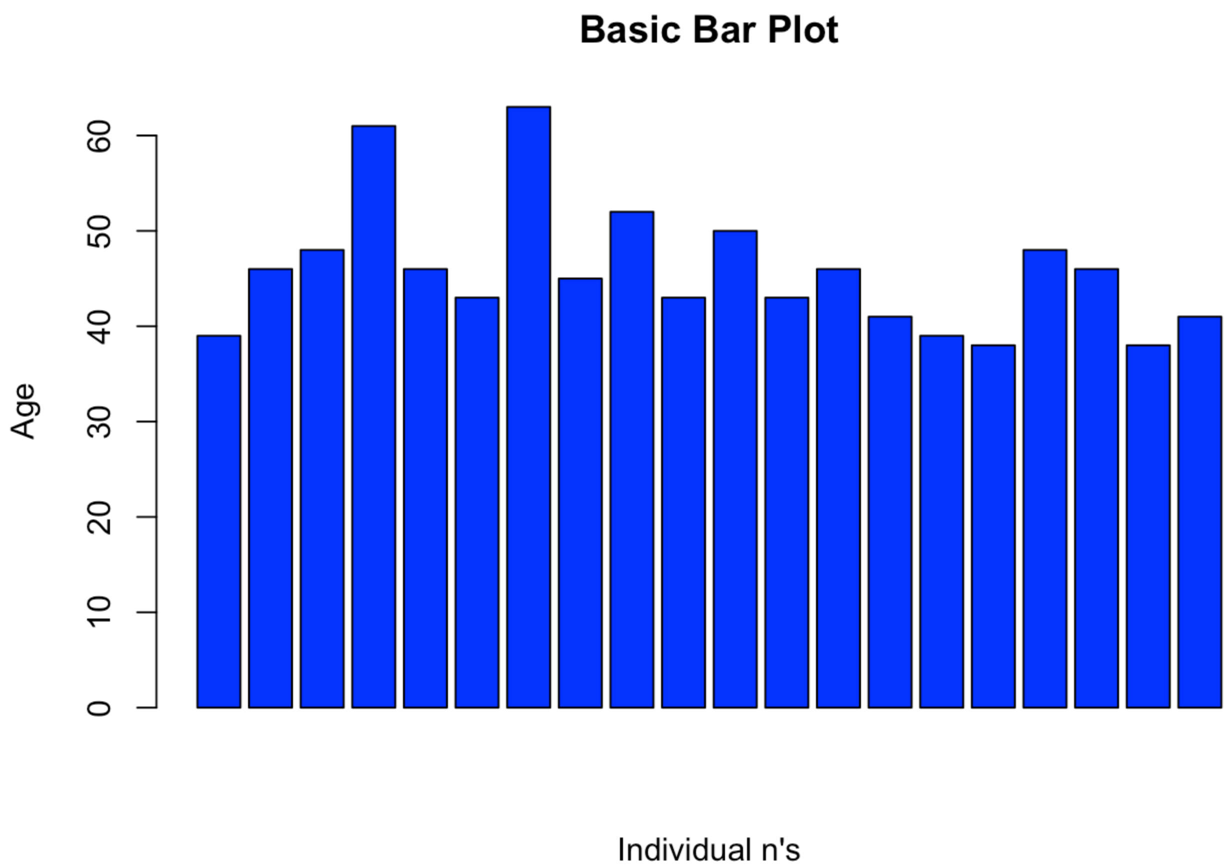



# 5. Bar plots (simple, stacked, and grouped)

Used for comparing quantities across different categories or groups
- Simple bar plot
- Stacked bar plot
- Grouped bar plot

Simple bar plot

```
barplot(df_sub$age,
        col = "blue",
        main = "Basic Bar Plot", xlab = "Individual n's",
        ylab = "Age")
```



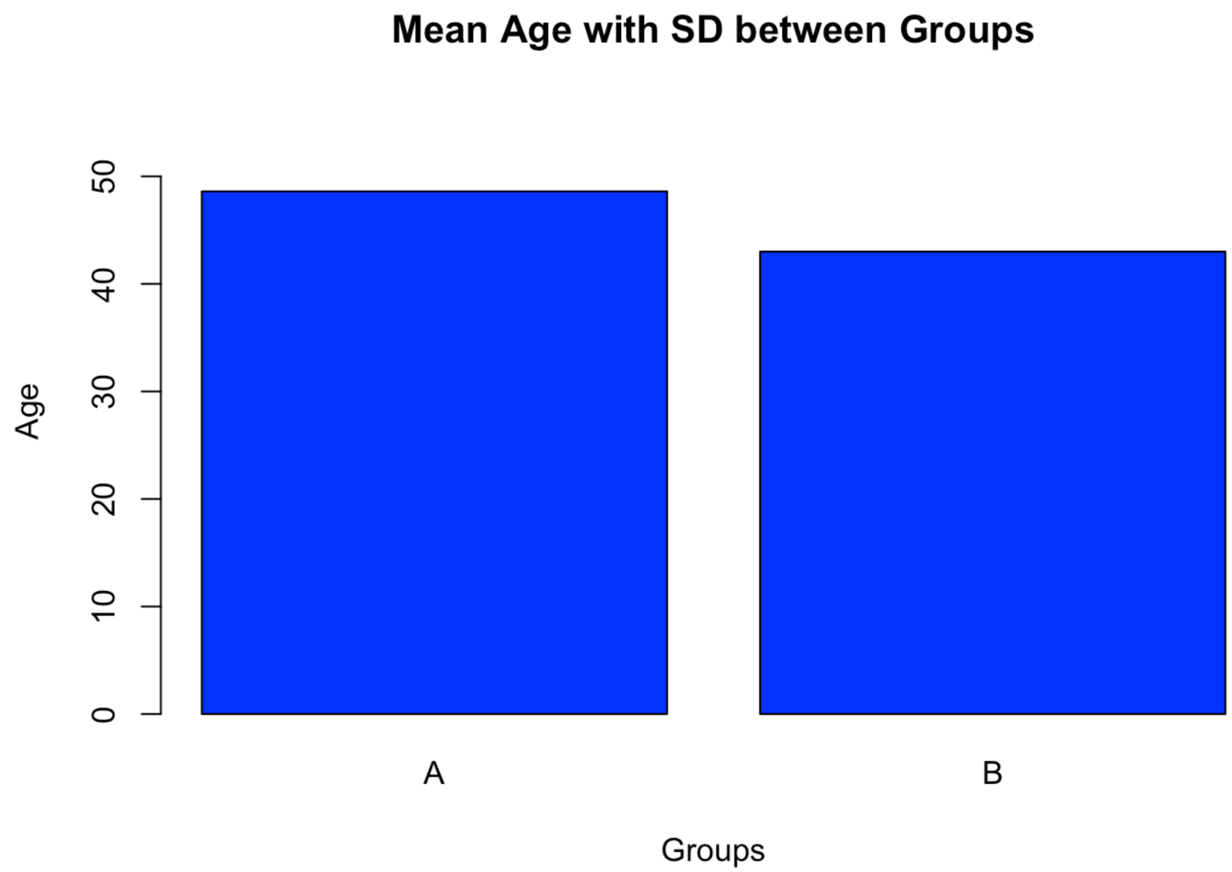

```
# bar for each n
```

Calculate mean, standard deviation, and standard error of the mean for each group/category

```
mean_values <- tapply(df_sub$age, df_sub$group, mean)
sd_values <- tapply(df_sub$age, df_sub$group, sd)
sem_values <- sd_values/sqrt(nrow(df_sub))
# tapply = "table apply", it applies a function to subsets of a vector,
# split by a given factor or grouping variable.
# useful when you want to perform calculations (like mean, sum,
# standard deviation, etc.) on different groups within your data
```

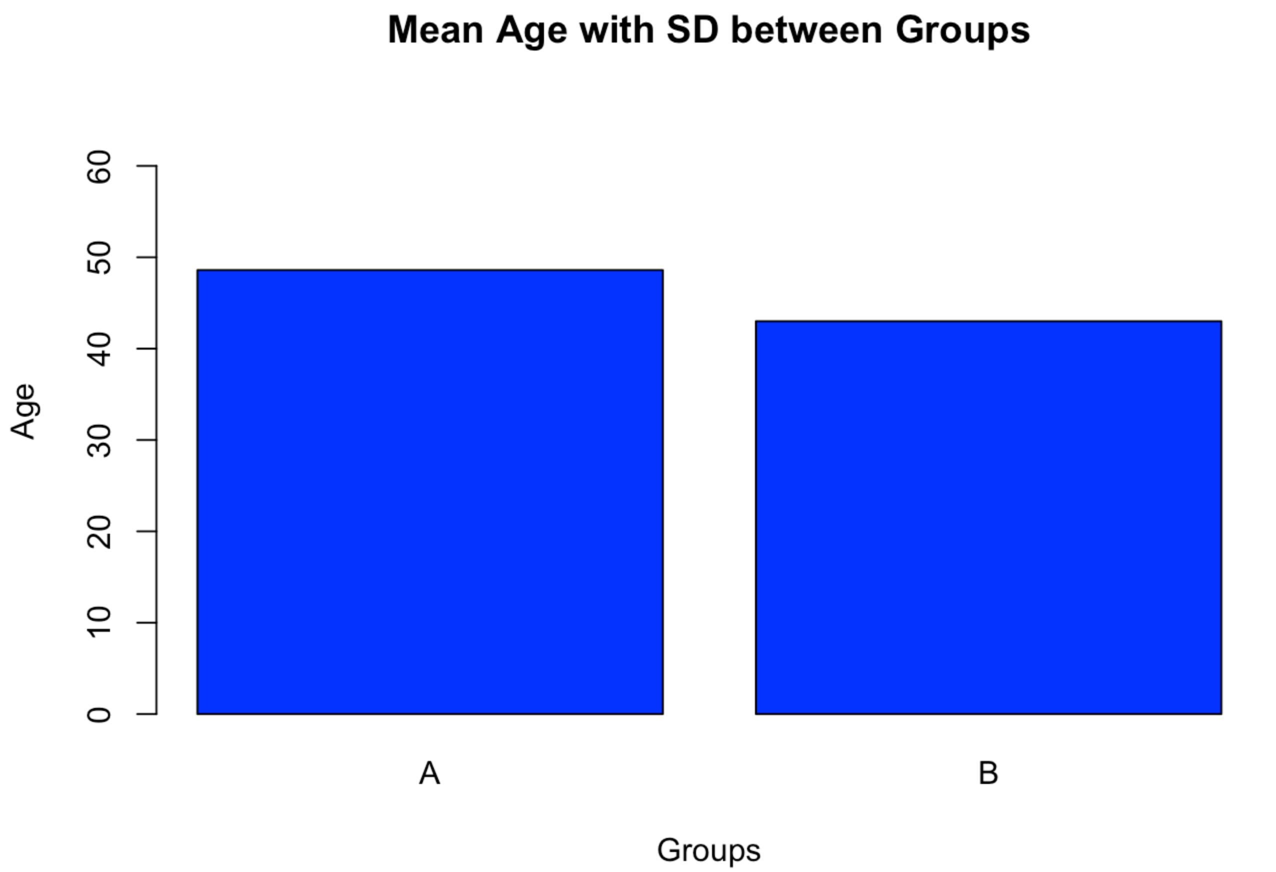
Basic barplot

```
bp_means <- barplot(mean_values,col = "blue",
                         ylim = c(0, max(mean_values + sd_values)),
                         main = "Mean Age with SD between Groups",
                         xlab = "Groups", ylab = "Age")
```



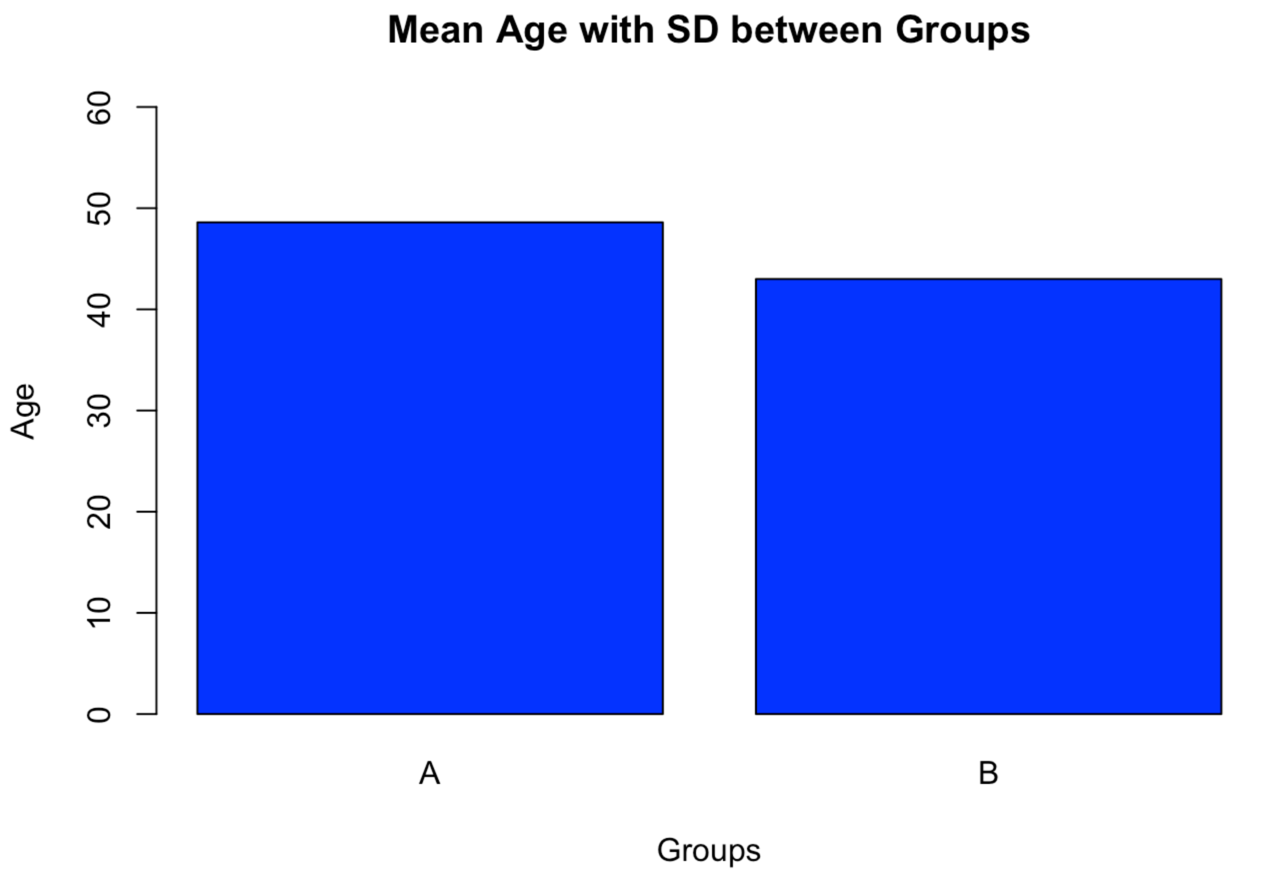

Barplot with adjusted y-scale

```
bp_means = barplot(mean_values, col = "blue",
                         ylim = c(0, max(mean_values + sd_values)+10),
                         main = "Mean Age with SD between Groups",
                         xlab = "Groups", ylab = "Age")
```



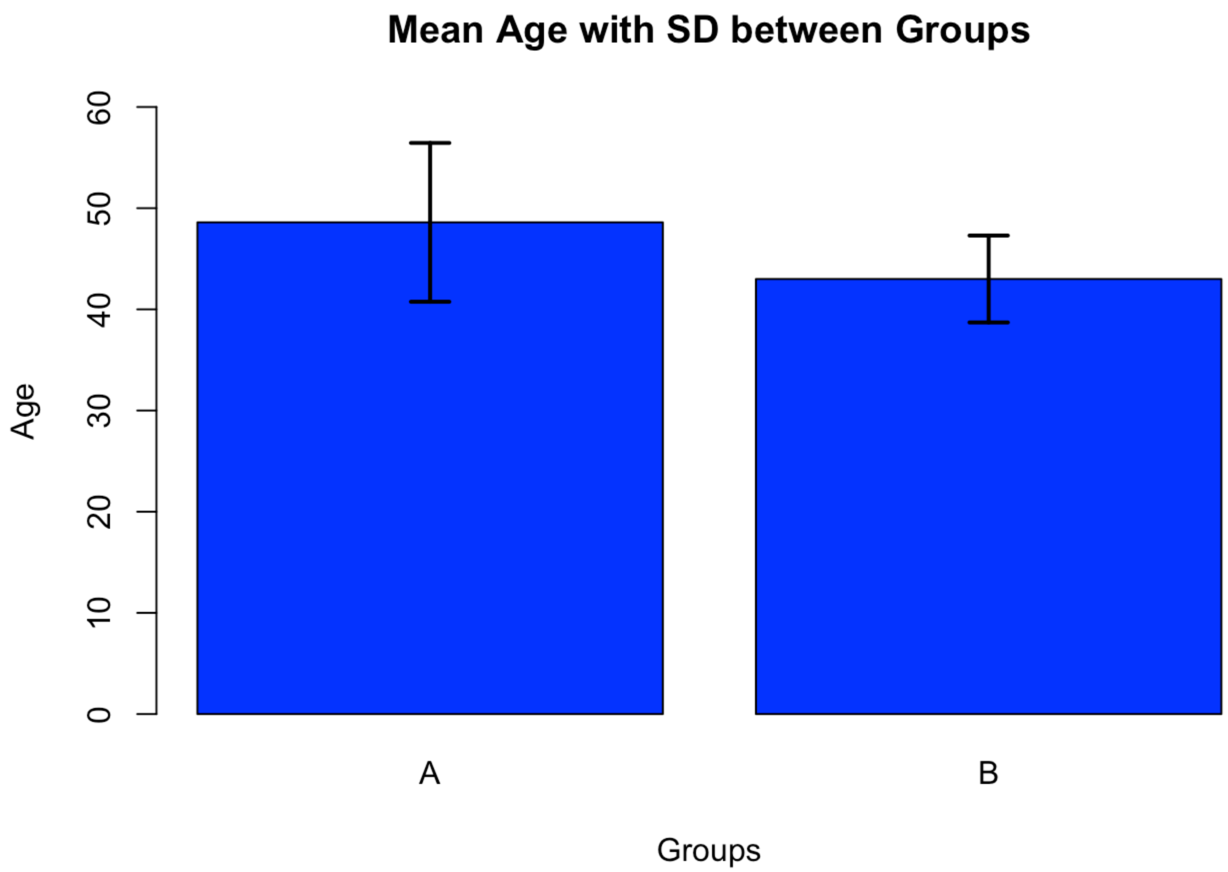

Automatically adjust y-axis scale with function pretty()

```
bp_means <- barplot(mean_values,col = "blue",
                         ylim = range(pretty(c(0, max(mean_values + sd_values)))),
                         main = "Mean Age with SD between Groups",
                         xlab = "Groups", ylab = "Age")
```




Add error bars to the bar plot

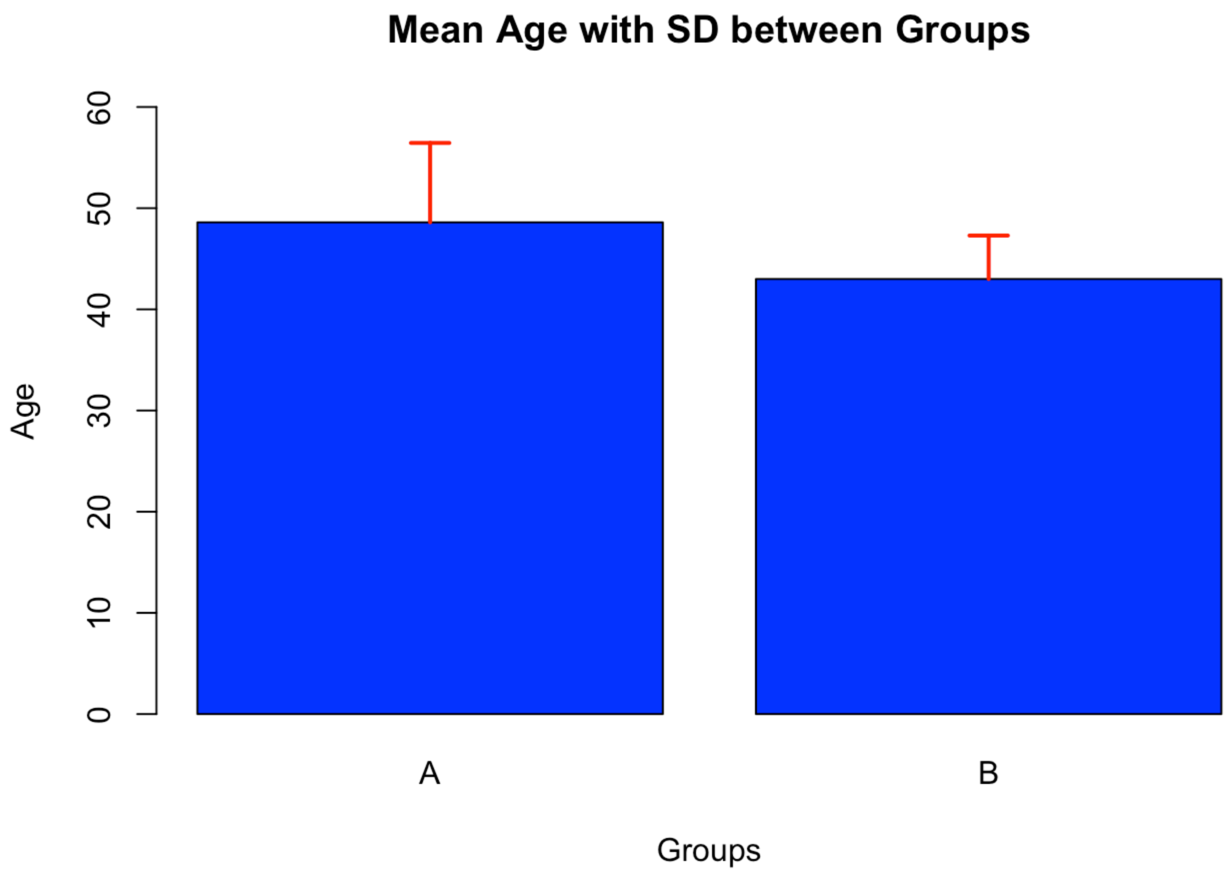
```
bp_means <- barplot(mean_values,col = "blue",
                         ylim = range(pretty(c(0, max(mean_values + sd_values)))),
                         main = "Mean Age with SD between Groups",
                         xlab = "Groups", ylab = "Age")
arrows(bp_means, mean_values - sd_values, bp_means, mean_values + sd_values,
       angle = 90, code = 3, length = 0.1, lwd = 2, col = "black")
```




```
# code = 1 down - SD only, 2 - up + SD only, 3 - both directions
```

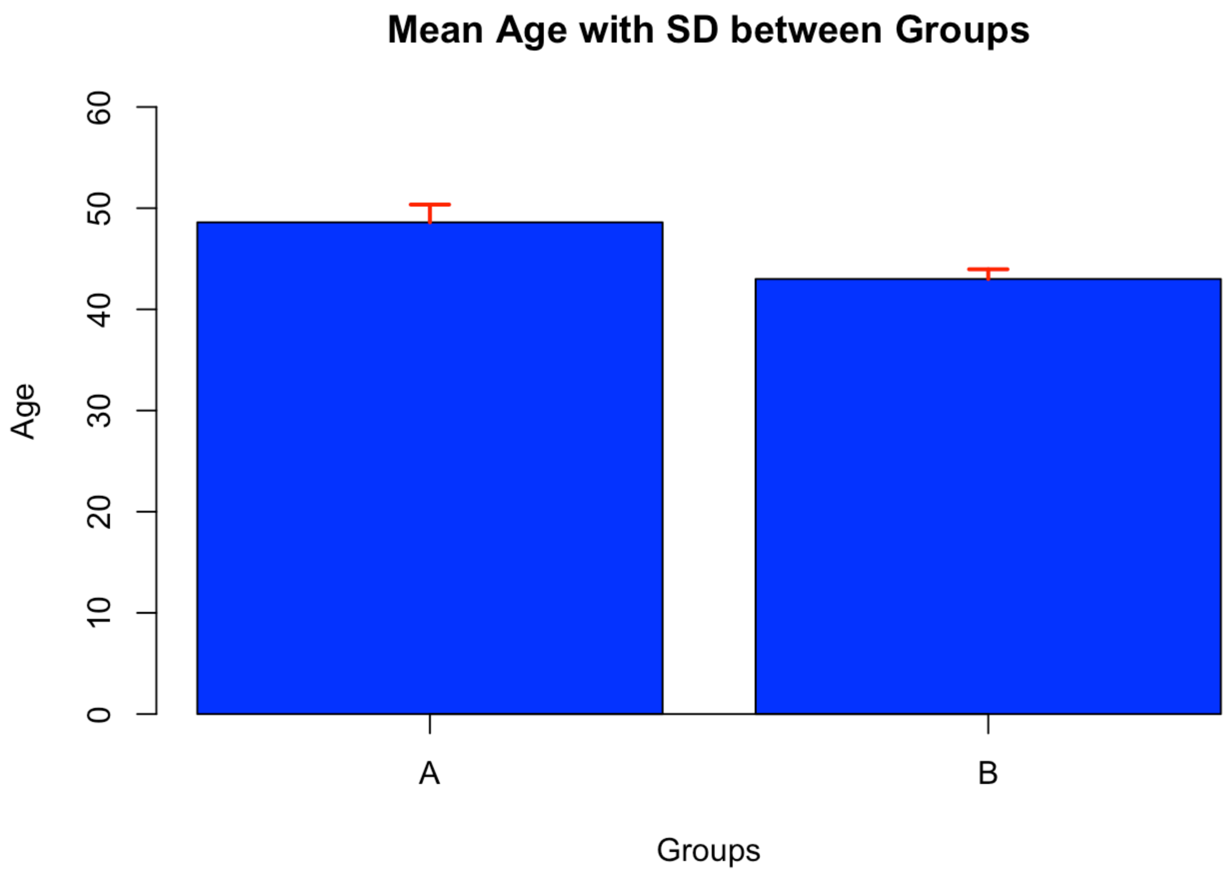
Error bars - one direction

```
bp_means <- barplot(mean_values,col = "blue",
                    ylim = range(pretty(c(0, max(mean_values + sd_values)))),
                    main = "Mean Age with SD between Groups",
                    xlab = "Groups", ylab = "Age")
arrows(bp_means, mean_values, bp_means, mean_values + sd_values,
       angle = 90, code = 2, length = 0.1, lwd = 2, col = "red")
```



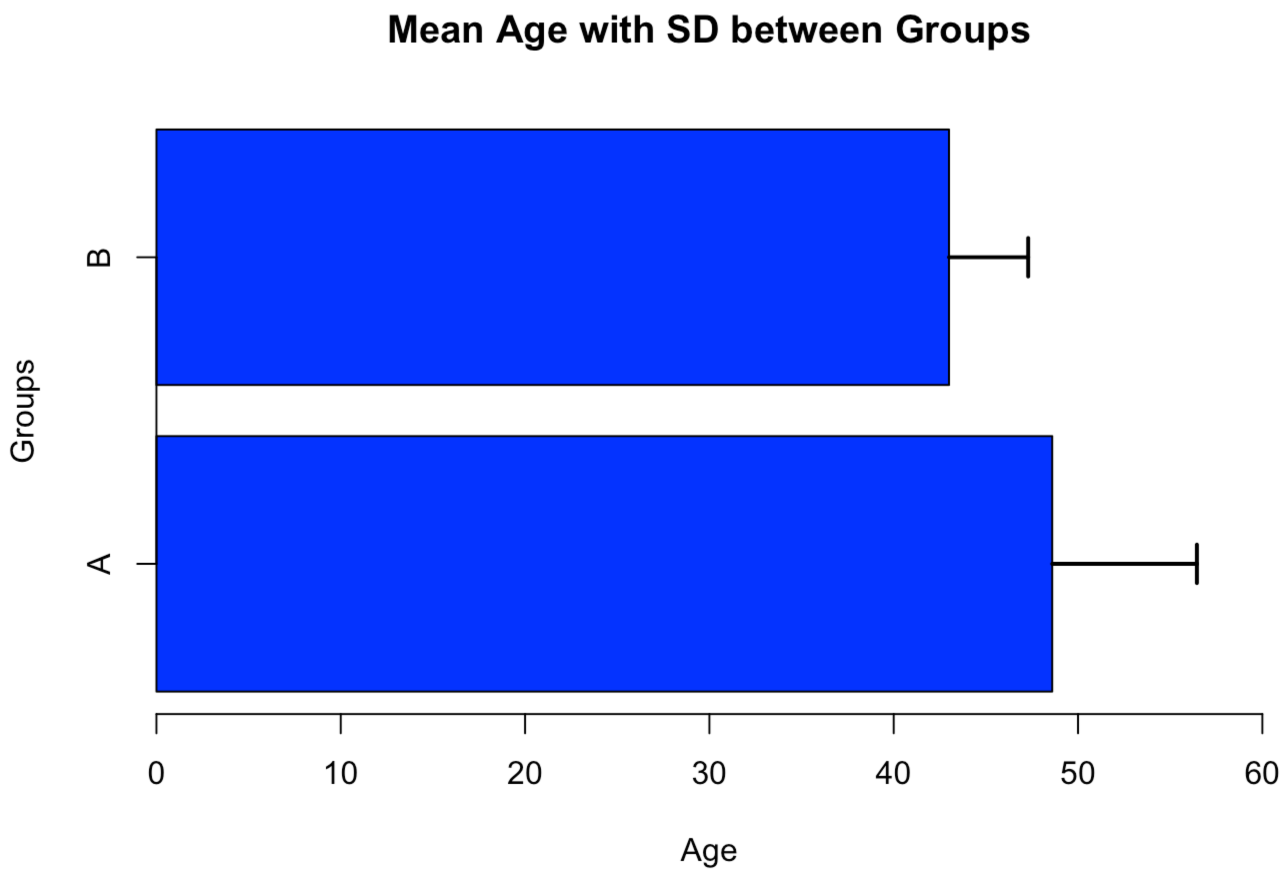

Plot with SEM instead of SD

```
bp_means <- barplot(mean_values,col = "blue",
                    ylim = range(pretty(c(0, max(mean_values + sd_values)))),
                    main = "Mean Age with SD between Groups",
                    xlab = "Groups", ylab = "Age", axis.lty = 1)
arrows(bp_means, mean_values, bp_means, mean_values + sem_values,
       angle = 90, code = 2, length = 0.1, lwd = 2, col = "red")
```




Horizontal barplot **switch x and y

```
bp_means <- barplot(mean_values, col = "blue",
                    xlim = range(pretty(c(0, max(mean_values + sd_values)))),
                    main = "Mean Age with SD between Groups",
                    xlab = "Age", ylab = "Groups", horiz = TRUE,
                    axis.lty = 1)

arrows(mean_values, bp_means, mean_values + sd_values, bp_means,
       angle = 90, code = 2, length = 0.1, lwd = 2, col = "black")
```

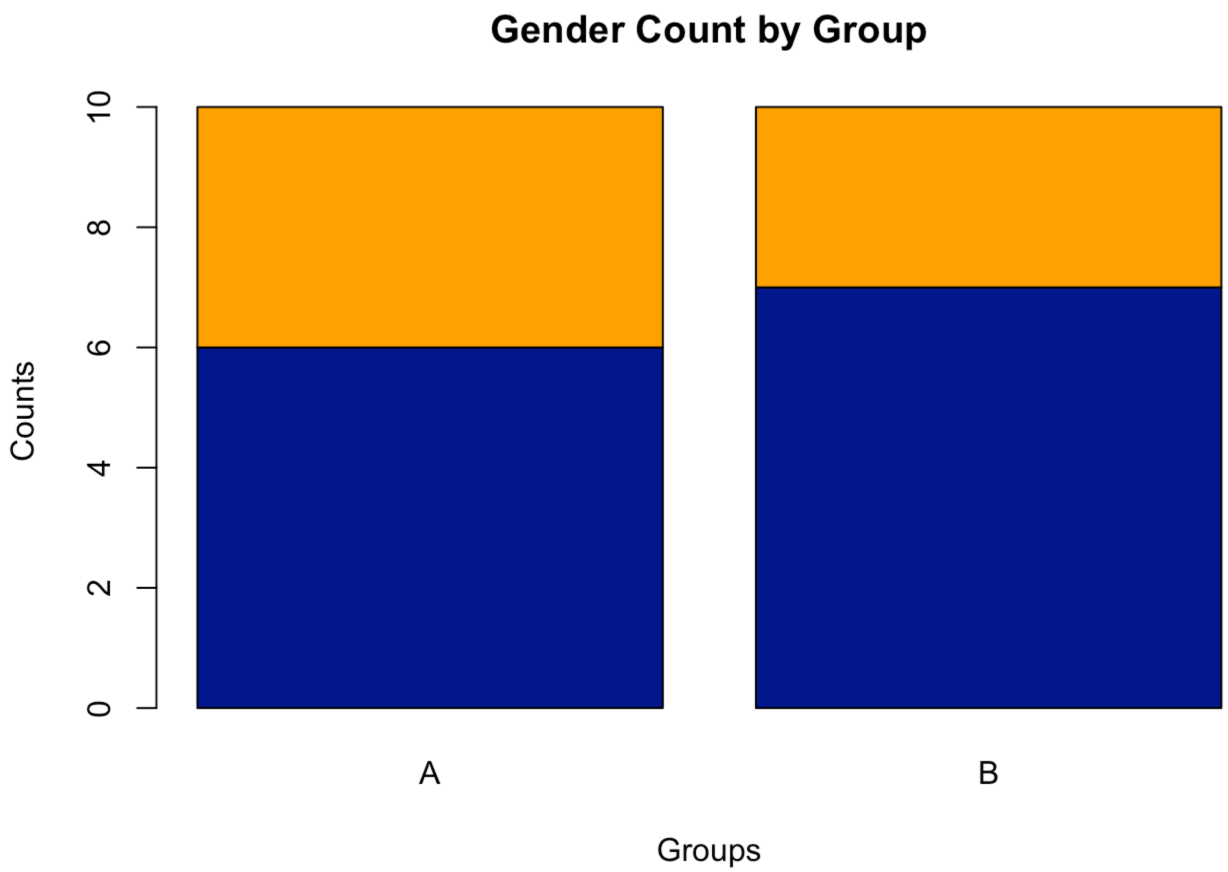



Stacked barplot

```
gender_counts = table(df_sub$gender,df_sub$group)
# builds a contingency table of the counts at each
# combination of the factor levels
gender_counts
```

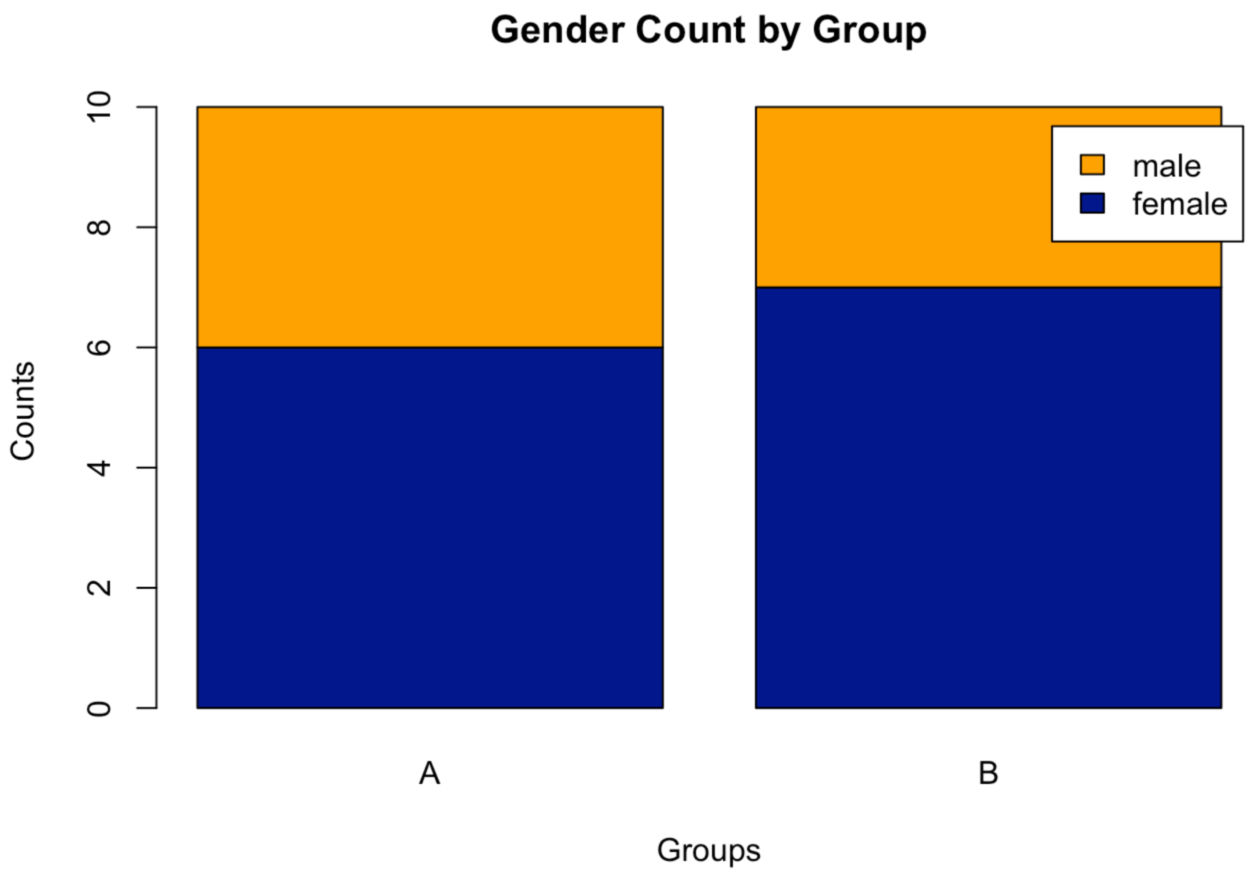
```
##
##          A B
##   female 6 7
##   male   4 3
```

```
barplot(gender_counts, main="Gender Count by Group",
        xlab="Groups",ylab = "Counts", col=c("darkblue","orange"))
```



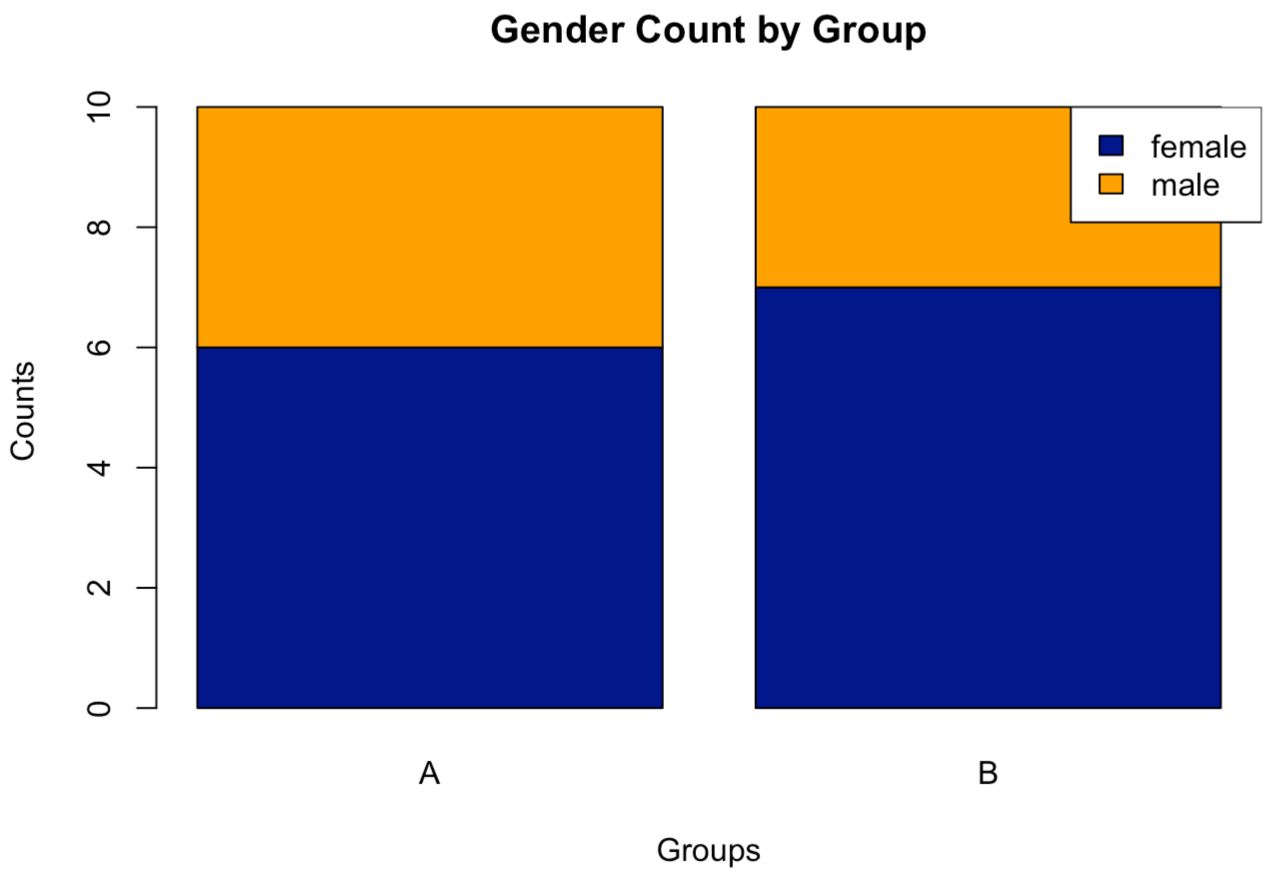

Stacked barplot with legend

```
barplot(gender_counts, main="Gender Count by Group",
        xlab="Groups",ylab = "Counts", col=c("darkblue","orange"),
        legend = rownames(gender_counts))
```



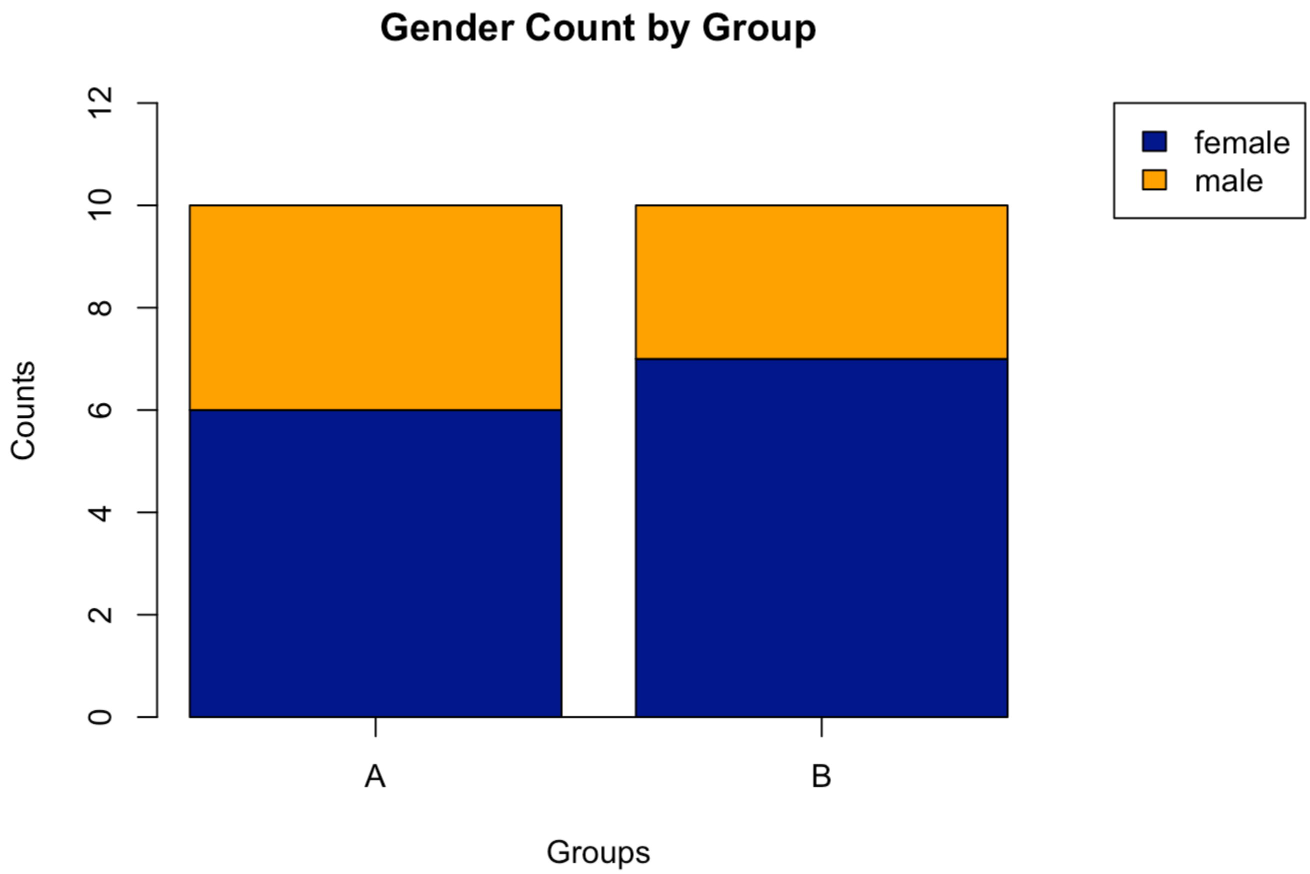

Add ‘legend’ with legend()

```
barplot(gender_counts, main="Gender Count by Group",
        xlab="Groups",ylab = "Counts", col=c("darkblue","orange"))
legend("topright", legend = rownames(gender_counts),
       fill = c("darkblue","orange"))
```




Adjust margins, to place legend outside of plot

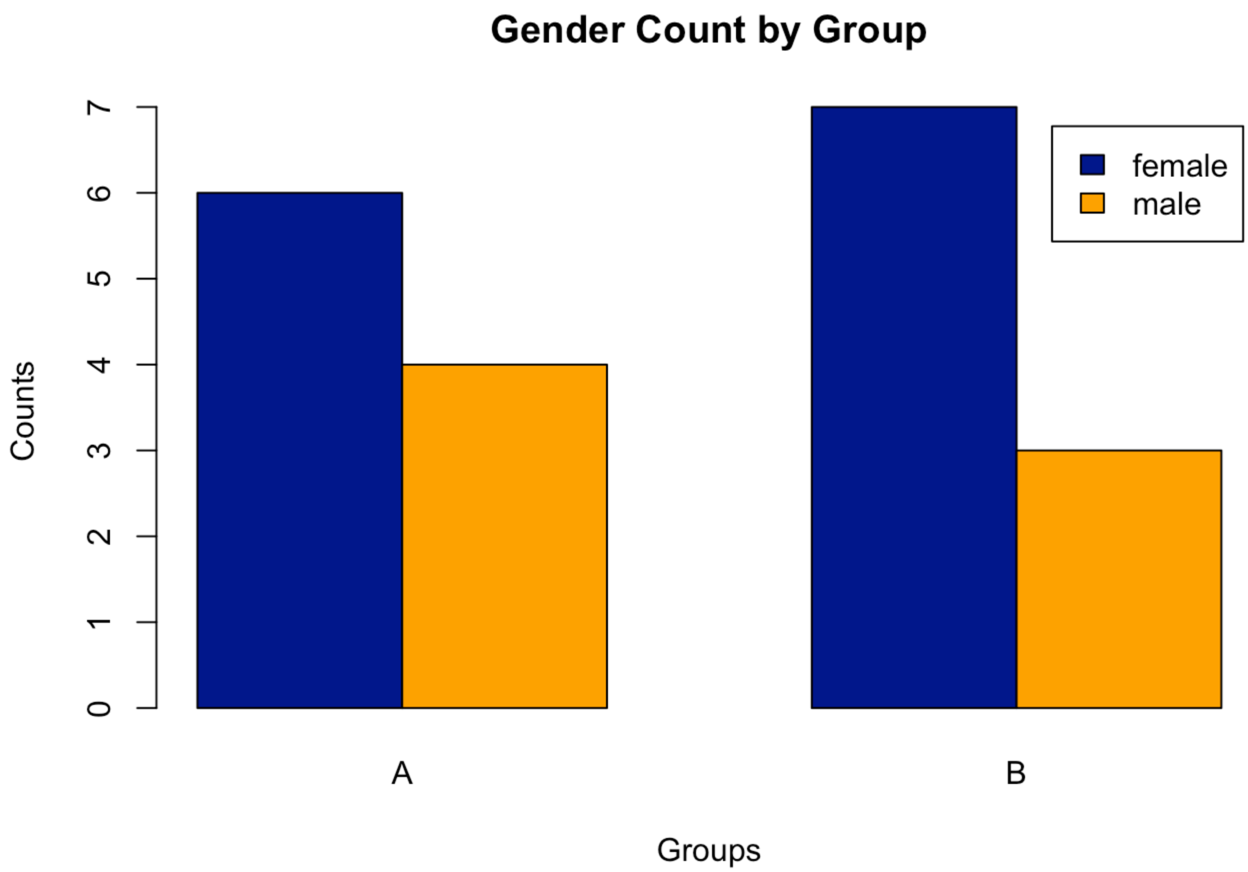
```
# default 5,4,4,2
par(mar=c(5, 4, 4, 8), xpd=TRUE) # bottom, left, top, right margins
# par() = parameters of plot
# xpd = FALSE, plotting clipped to the plot region
# xpd = TRUE = plotting is clipped to the figure region

barplot(gender_counts, main="Gender Count by Group",
        xlab="Groups",ylab = "Counts", col=c("darkblue","orange"),
        ylim = c(0,12), axis.lty = 1)

#add legend outside of plot
legend("topright", legend = rownames(gender_counts),
       fill = c("darkblue","orange"),
       inset=c(-0.3, 0))
```




```
# inset = inset distances from the margins as a fraction of the plot
# inset = c(-0.3, 0) this moves the legend 30% of the plot width to the left
# of the top-right corner of the plot.
```

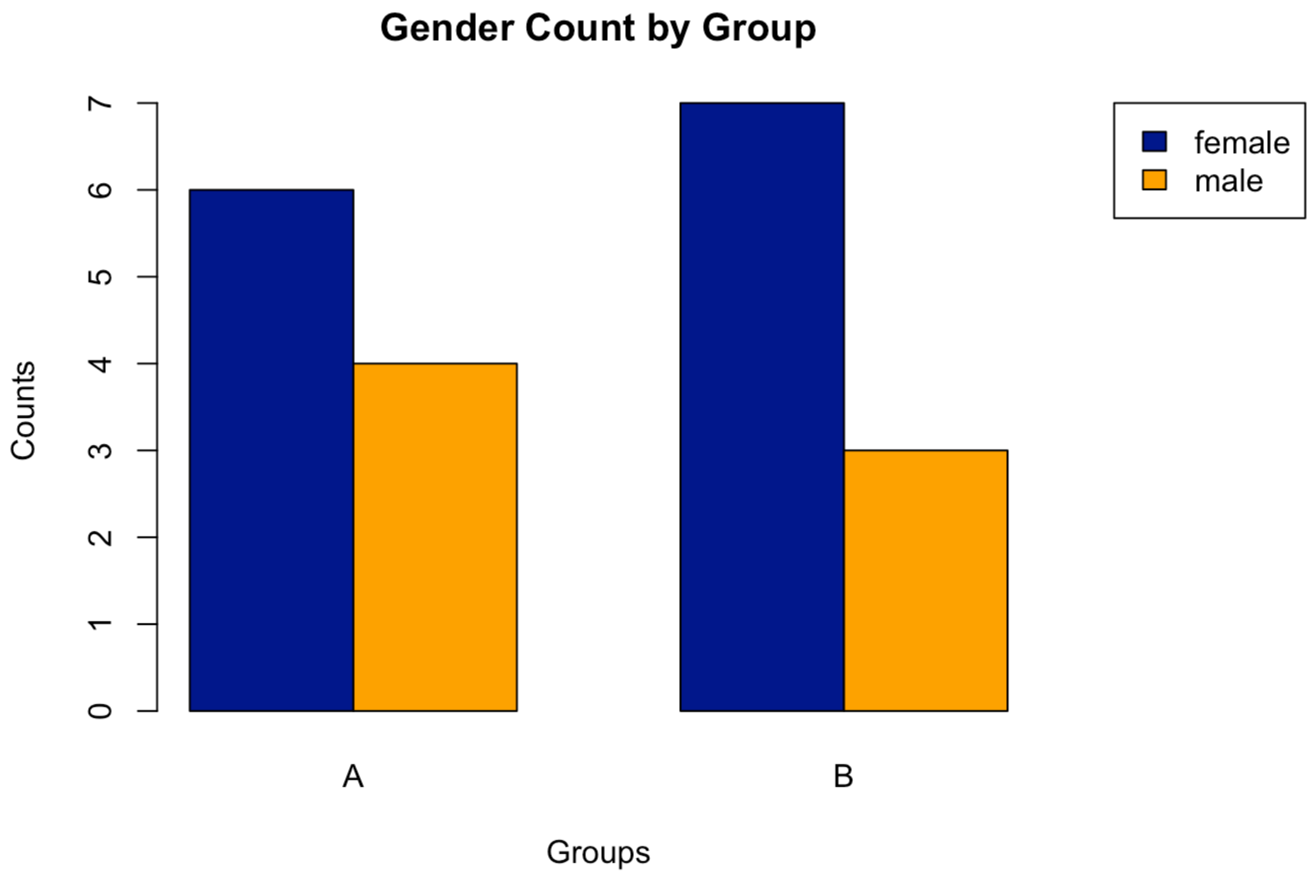
Grouped barplot

```
barplot(gender_counts, main="Gender Count by Group",
        xlab="Groups",ylab = "Counts", col=c("darkblue","orange"),
        legend = rownames(gender_counts), beside=TRUE)
```




Adjust par(mar()) settings

```
par(mar=c(5, 4, 4, 8), xpd=TRUE)
barplot(gender_counts, main="Gender Count by Group",
        xlab="Groups",ylab = "Counts", col=c("darkblue","orange"),
        beside = TRUE)

legend("topright", legend = rownames(gender_counts),
       fill = c("darkblue","orange"),
       inset=c(-0.3, 0))
```

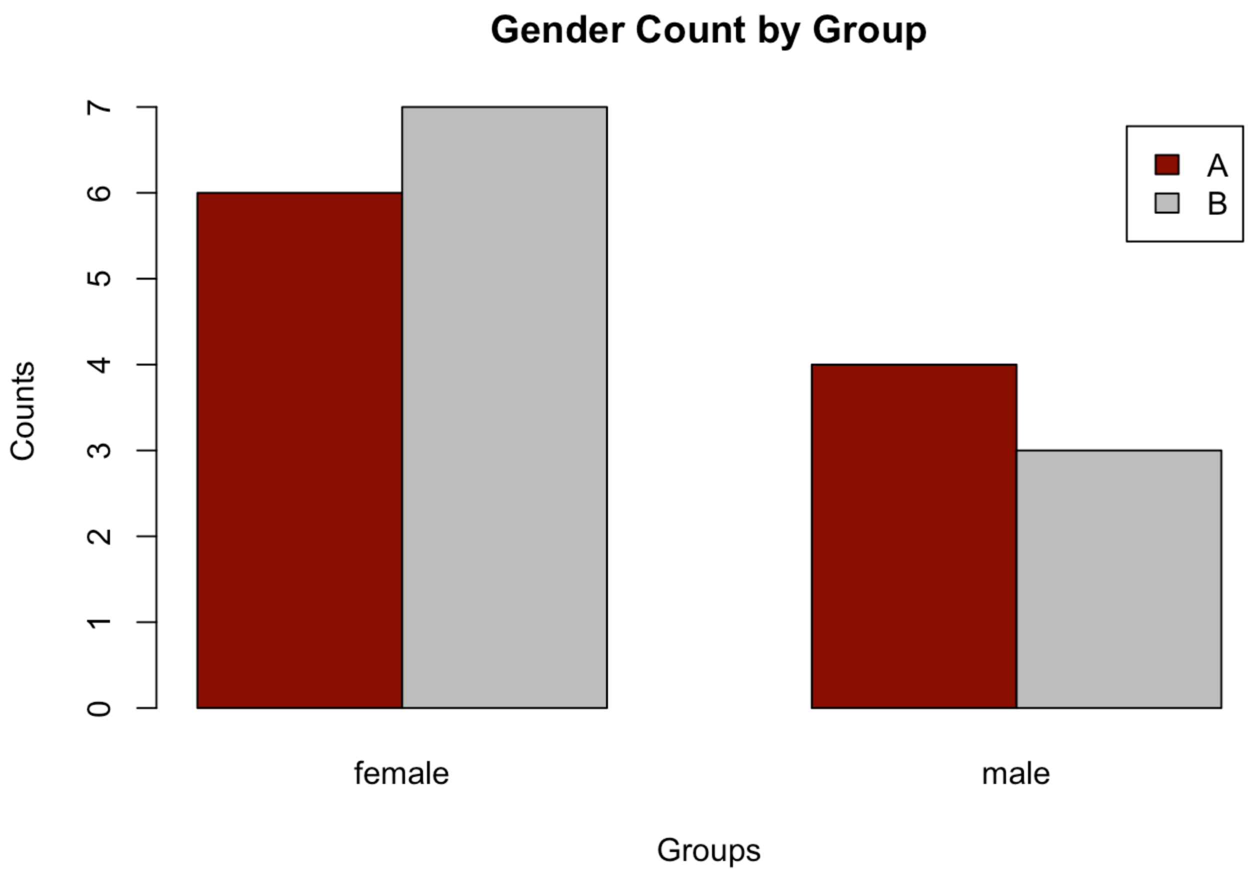



Transposed data, color = group

```
gender_counts
```

```
##
##          A B
##   female 6 7
##   male   4 3
```

```
barplot(t(gender_counts), main="Gender Count by Group",
        xlab="Groups", ylab = "Counts",col=c("darkred","grey"),
        legend = colnames(gender_counts), beside = TRUE)
```




```
# t() function transposes matrices and data frames
```

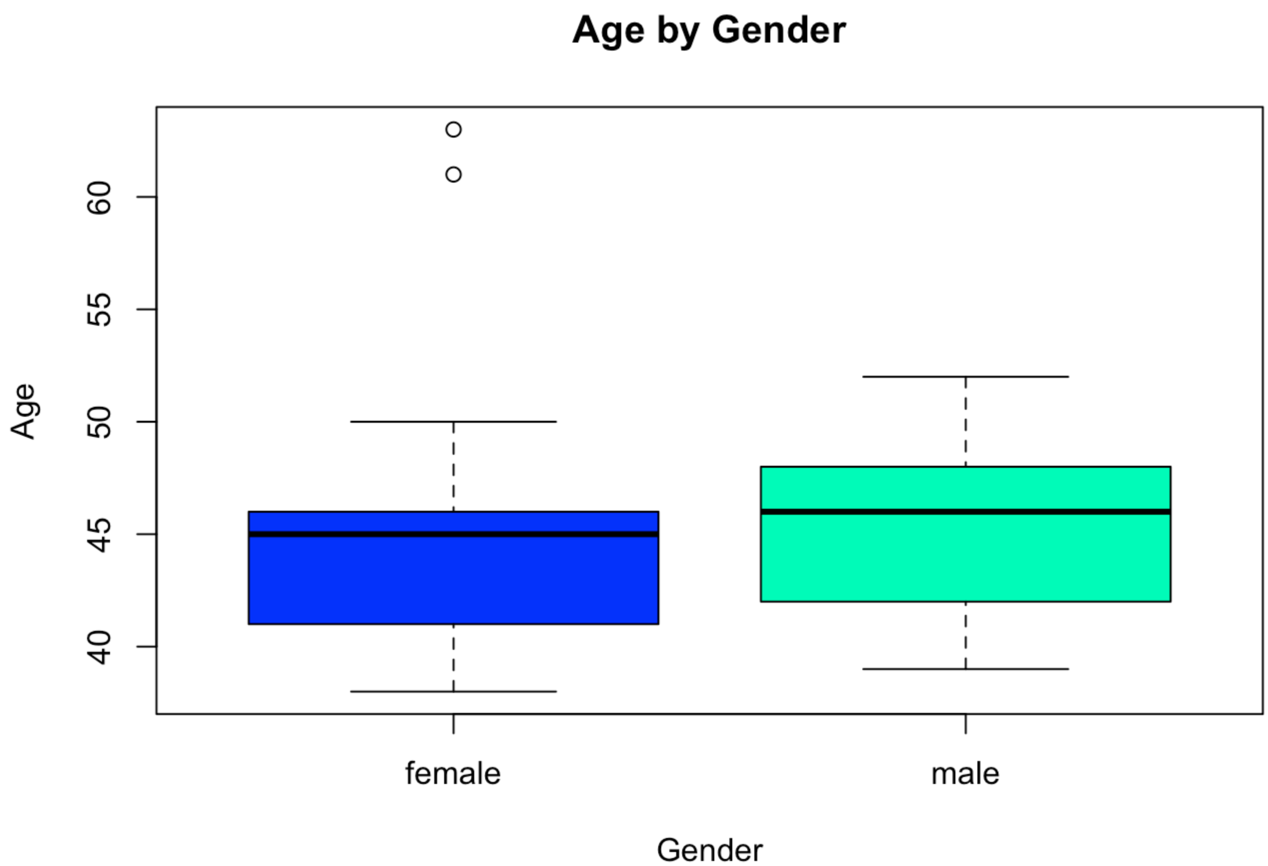
# 6. Boxplots

Box-and-whisker plot, is a graphical representation of the distribution of a dataset that shows its central tendency, spread, and potential outliers. It provides a summary of the data through five key summary statistics: the minimum, first quartile (Q1), median, third quartile (Q3), and maximum

Formula: y ~ group, where y is a numeric vector of data values to be split into groups according to the grouping variable (usually a factor)

Basic boxplot

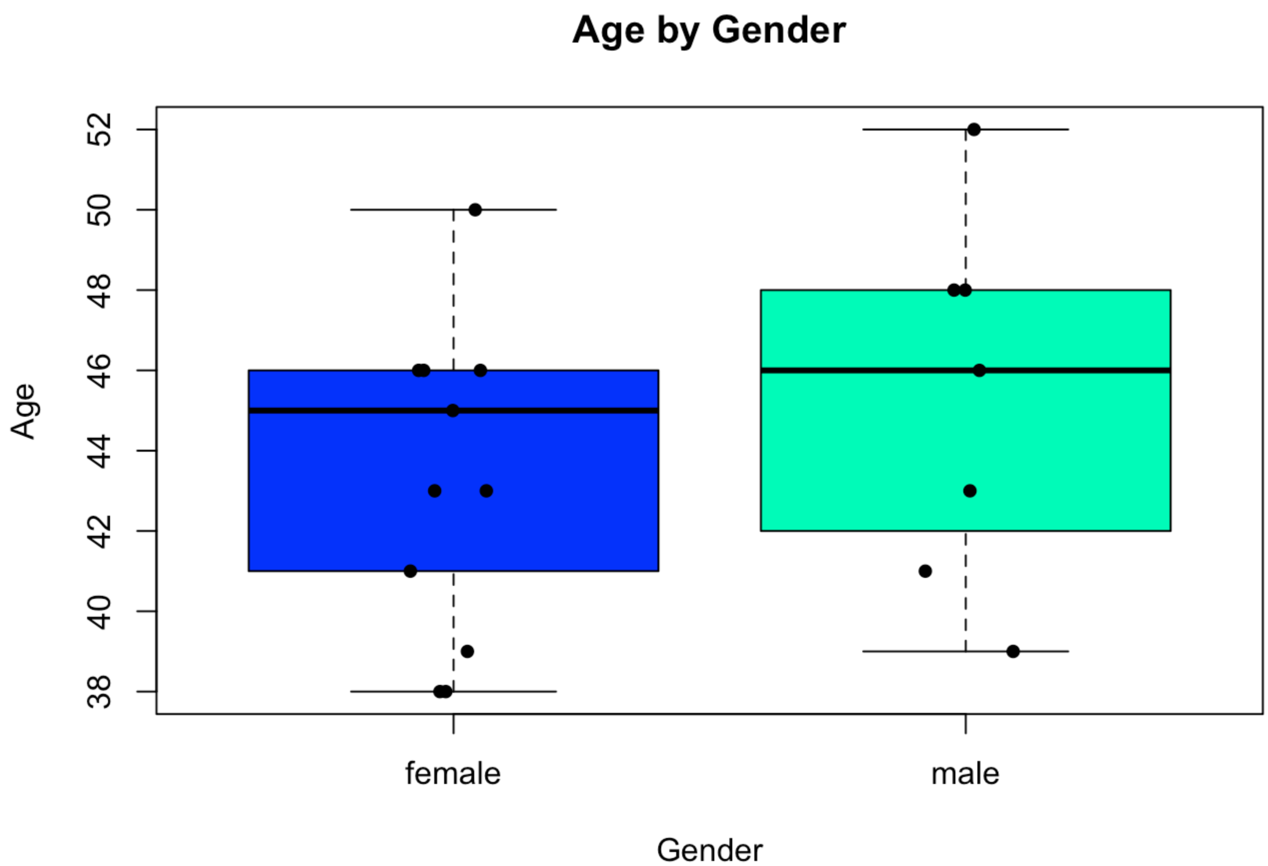
```
boxplot(age~gender,data=df_sub, main="Age by Gender",
        xlab="Gender", ylab="Age", col = c("#032cfc", "#03fcba"))
```




```
# col = "color_name"
# col = rgb(R,G,B, transparency)
# col = hex code
```

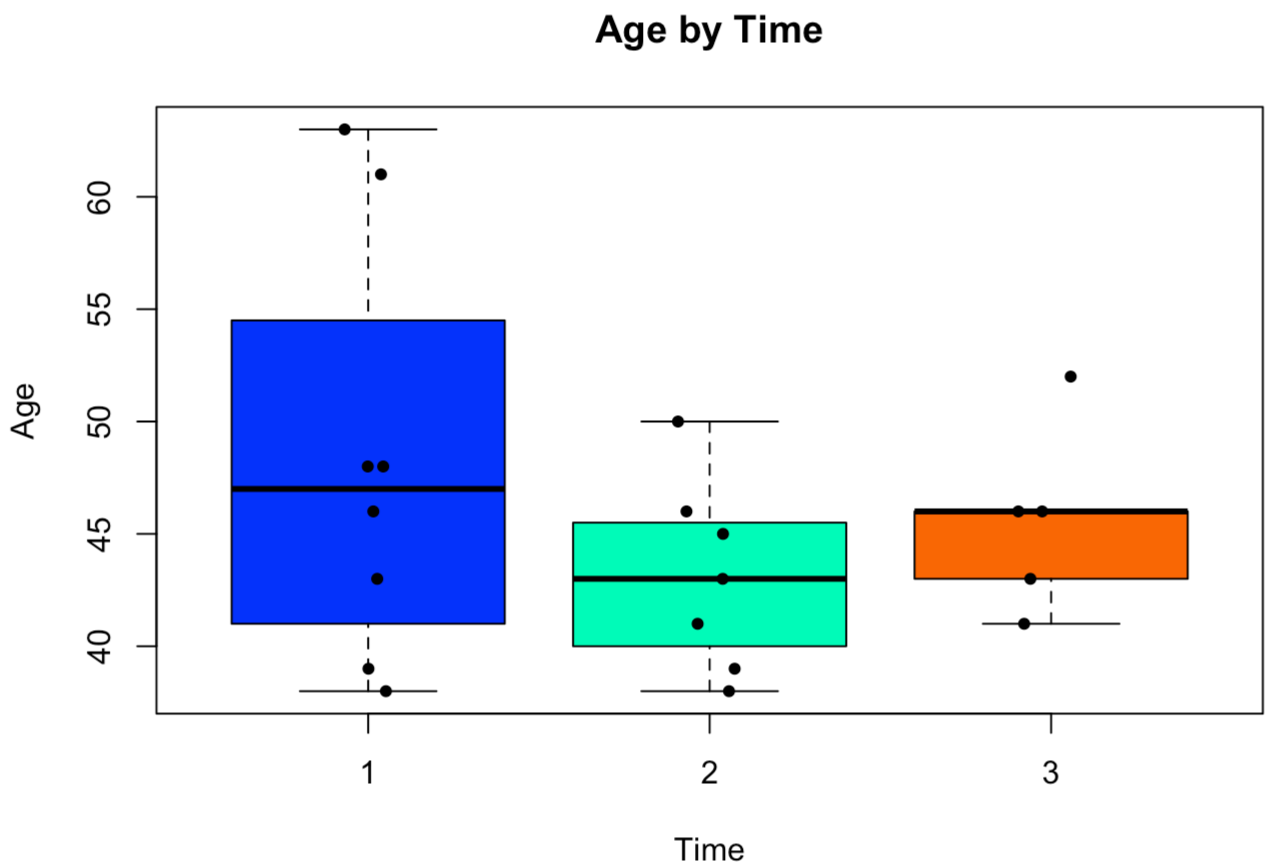
Boxplot with scatter points, remove outliers

```
boxplot(age~gender,data=df_sub, main="Age by Gender",
        xlab="Gender", ylab="Age", col = c("#032cfc", "#03fcba"),
        outline = FALSE)
# Add data points
stripchart(age ~ gender,
           data = df_sub,
           method = "jitter",
           pch = 19,
           col = "black",
           vertical = TRUE,
           add = TRUE,
           cex = 0.8)
```



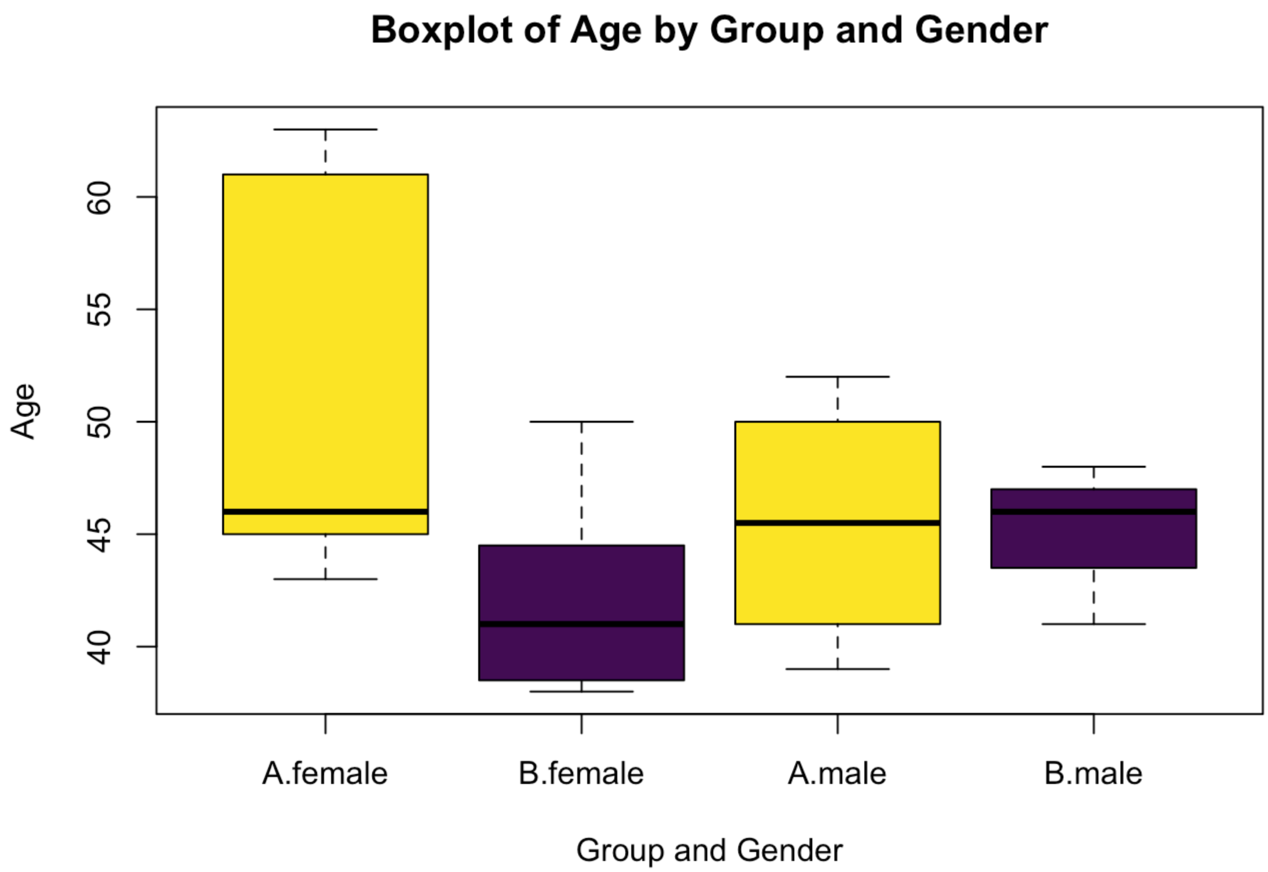

# Add new variable ‘time’

```
set.seed(092024)
df_sub$time = as.factor(sample(1:3, size = 20, replace = TRUE))
boxplot(age~as.factor(time),data=df_sub, main="Age by Time",
        xlab="Time", ylab="Age", col = c("#032cfc", "#03fcba", "#fc6b03"),
        outline = FALSE)

# Add data points
stripchart(age ~ time,
           data = df_sub,
           method = "jitter",
           pch = 19,
           col = "black",
           vertical = TRUE,
           add = TRUE,
           cex = 0.7)
```




Boxploty of age by gender and group

```
boxplot(age ~ interaction(group, gender), data = df_sub,
        col = viridis(3)[as.integer(interaction(df_sub$group, df_sub$gender))],
        xlab = "Group and Gender",
        ylab = "Age",
        main = "Boxplot of Age by Group and Gender")
```




# 7. Pie chart

Create 'time_counts' dataframe

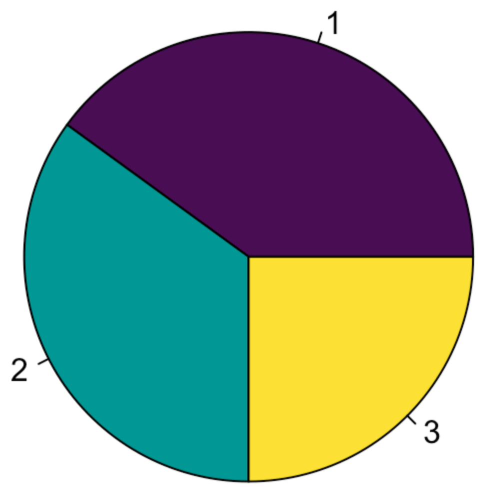
```
# Create a table from the data
time_counts <- table(df_sub$time)
time_counts
```

```
##
## 1 2 3
## 8 7 5
```

Pie charts with different color palettes

Viridis

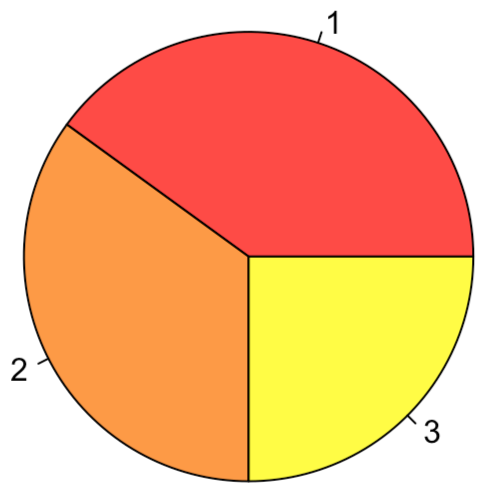
```
pie(time_counts,
    col = hcl.colors(length(time_counts), "viridis"))
```




```
# hcl.colors(n, palette name)
# palette names: hcl.pals()
# alpha = transparenc level: 0-1
# rev = logical to indicate if the colors should be reversed
```

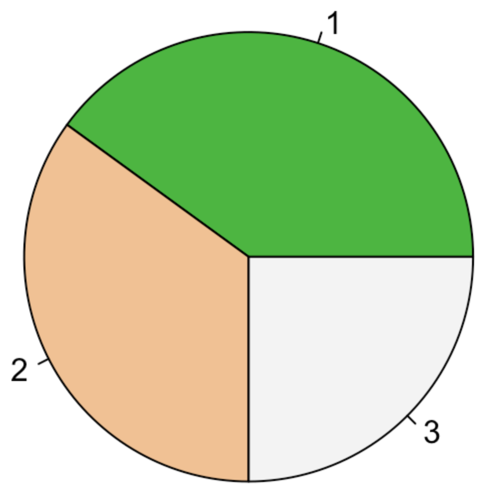
heat.colors

```
pie(time_counts,
    col = heat.colors(length(time_counts), alpha = 0.8))
```



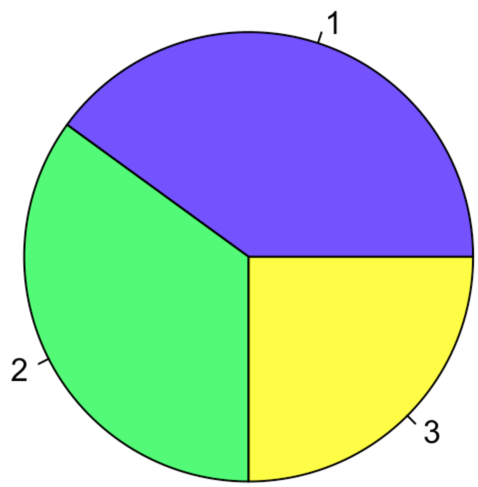

terrain.colors

```
pie(time_counts,
    col = terrain.colors(length(time_counts), alpha = 0.8))
```



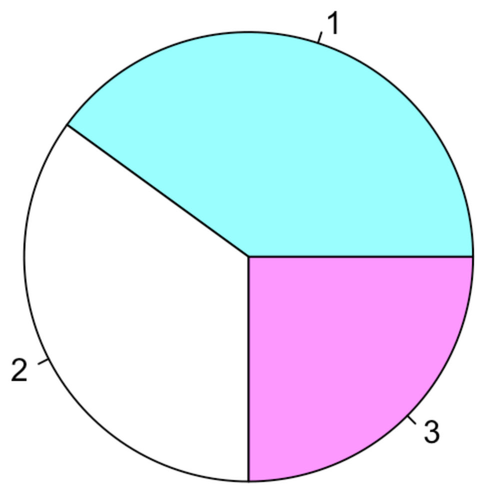

topo.colors

```
pie(time_counts,
    col = topo.colors(length(time_counts), alpha = 0.8))
```



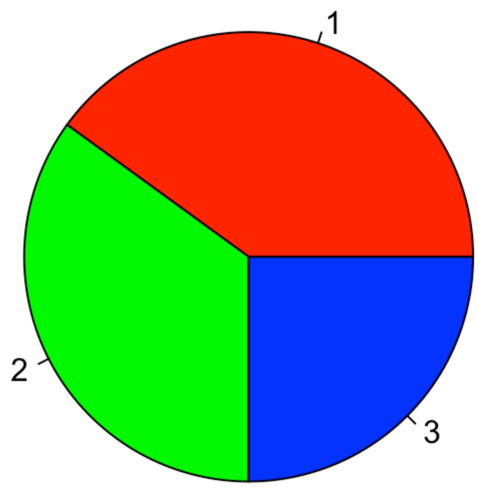

cm.colors

```
pie(time_counts,
    col = cm.colors(length(time_counts), alpha = 0.8))
```




rainbow

```
pie(time_counts,
    col = rainbow(length(time_counts), s = 1, v = 1))
```




```
# s = saturation
# v = value
```

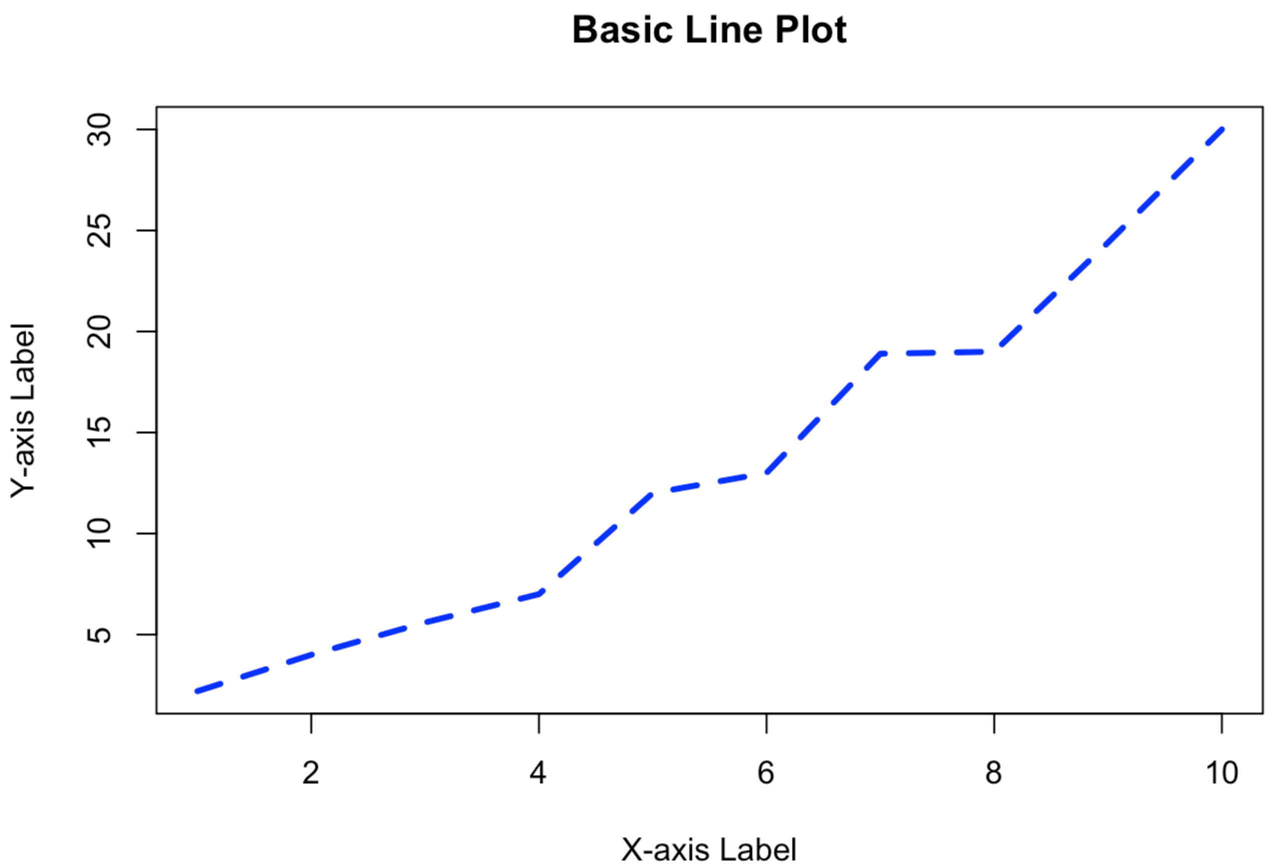
**Save the df_sub dataframe**

```
write.csv(df_sub, "framingham_sub.csv", row.names = FALSE)
# row.names = FALSE removes column index that is generated, as default
# is TRUE
```

# Practice Session

1. What happens when we plot lwd = 2, lty = 4, and pch = 19?

```
plot(x, y, type = "l", col = "blue", lwd = 3, lty = 2, pch = 19,
     main = "Basic Line Plot",
     xlab = "X-axis Label",
     ylab = "Y-axis Label")
```




2. Using tidyr format, how would you create a grouped barplot with Base R?

```
count_data = df_sub %>%
group_by(group, gender) %>%
 count(gender)
count_data
```

```
## # A tibble: 4 × 3
## # Groups:   group, gender [4]
##   group gender     n
##   <fct> <chr>  <int>
## 1 A     female     6
## 2 A     male       4
## 3 B     female     7
## 4 B     male       3
```

```
gender_counts
```

```
##
##          A B
##   female 6 7
##   male   4 3
```

3. Convert ‘count_data’ to wide format to create bar plot with Base R

4. Create a grouped barplot and add legend

# Save a plot in R

1. Export menu in the Plots tab (save as image, pdf, or copy to clipboard)
2. Save as pdf

```
# Open graphical device and set parameters for output
pdf("my_plot.pdf",         # File name
    width = 8, height = 7, # Width and height in inches
    bg = "white")

# Creating the plot
plot(x,y)

# Closing the graphical device
dev.off()
```

```
## pdf
##   2
```