


Day 2 | Hands-on Exercises: Data Visualization

In R, plots are generated using graphics device. Two types of plotting commands are available in R, **High level** and **Low level**. The high level plotting commands are used to directly plot on the graphics device while the low level plotting commands are used to plot on already existing plots (Generated using high level commands).

Type	Function	Purpose (Parameters specific to the corresponding plot)
High level	plot()	Scatter plot, Line plot
	hist()	Histogram (<i>breaks, border, labels, freq</i>)
	barplot()	Bar plot (<i>names.arg, horiz, border, space, beside</i>)
	boxplot()	Draw Box plot (<i>horizontal, notch, names, border</i>)
	pie()	Pie chart (<i>labels</i>)
	pairs()	Multivariate scatter plot
	dotchart()	Dot chart
	plot(density())	Density plot
	venn()	Venn diagram using gplots package.
	plot3d	3D scatter plot using rgl package.
	heatmap.2()	Draw heatmap using gplots package. (<i>Colv, Rowv</i>)
Low-level	lines()	Add lines to plot. <i>lines(x-coordinates, y-coordinates, lty, lwd, col)</i>
	points()	Add points to plot. <i>points(x-coordinates, y-coordinates, pch, col)</i>
	text()	Add texts to plot. <i>text(x-coordinates, y-coordinates, texts, pos, offset)</i>
	legend()	Add legends to plot. <i>legend(x-coordinates, y-coordinates, legends, colors, lwd, pch)</i>
Graphics parameters <i>par()</i> function	pch	Defines points. Try ?pch to get more helps. <i>pch=15</i> 
	lty	Line type (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash)
	lwd	Line width (a <i>positive</i> number, defaulting to 1). <i>Lwd=2</i> .

Day 2 | Hands-on Exercises: Data Visualization

	xlab, ylab	Add labels to x-axis and y-axis. <i>xlab="value", ylab="Freq"</i>
	main, sub	Add title and subtitle to plot. <i>main="Main title", sub="Sub title"</i>
	xlim, ylim	Set the x-axis and y-axis range. <i>xlim=c(0,100), ylim=c(-10,10)</i>
	cex	Magnify the text and points on the plot relative to default 1. <i>cex=1.2 will magnify the symbols and text on the plot by 20%.</i>
	cex.lab	Magnify the x-axis and y-axis labels. <i>cex.lab=1.2</i>
	cex.axis	Magnify the ticks marks and values on the tick mark. <i>cex.axis=1.2</i>
	cex.main, cex.sub	Magnify the main title and subtitles respectively. <i>cex.main=1.2, cex.sub=1.2</i>
	bty	Box type. If bty is one of "o" (the default), "l", "7", "c", "u", or "]" the resulting box resembles the corresponding upper case letter.
	col	Colors to lines, points. Explore colors() function to get inbuilt function names. rainbow(n) to get a vector of n contiguous colors. <i>col="red"</i>
	col.axis	Color the axis tick mark labels. <i>col.axis="red"</i>
	las	Style of axis tick mark labels. (0: parallel to axis, 1: always horizontal, 2: always perpendicular to axis, 3: always vertical)
	mfrow	Multi panel figures. <i>mfrow=c(3,2)</i> will draw 6 figures divided as 3 rows and 2 columns

Notes

Scatter Plot

Let's draw a simple scatter plot (Fig. 1-4)

```
1 x=sample(100,50)
2 plot(x)
3 plot(x,type="l") # line
4 plot(x,type="o") # both
5 plot(x,type="h") # vertical
6 plot(x,type="n") # nothing
7 plot(x,type="s") # stairs
```

- Line1: Create a vector x, 50 elements, values between 0 and 100.
- Line2: Plot the value of x against the index.
- Line3: *type="l"* specifies draw a line plot. *type="p"* for points, *type="o"* for both points and lines. *type="h"* for vertical lines, *type="n"* draws nothing, *type="s"* draws stairs.

Add title/subtitle/x-axis label/y-axis label to plot.

```
1 plot(x,
2 type="l",
3 xlab="Index",
4 ylab="Value",
5 main="Random plot",
6 sub="subtitle");
```

- We give extra graphics parameters inside the plot function to add annotations.
- *xlab="Index"* will change x-axis labels
- *ylab="Value"* will change y-axis labels
- *main="Random plot"* will change plot title.
- *sub="subtitle"* will ass subtitle to the plot.

Increase line thickness/Assign line color/Assign colors to points. (Fig. 5)

```
plot(x,type="l",xlab="Index",
ylab="Value",main="Random plot",
lwd=2);
-----
plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=2,col="red");
-----
colors();
rainbow(5);
```

- *type="l"* will draw a line plot.
- *lwd=2* will increase line width twice.
- *col="red"* will change the line color to red.
- *colors()*; function to see all inbuilt color names.
- *rainbow(5)*; will give 5 consecutive color values which can be used for multiple colors.

Fig. 6

```
plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=2,
col=c("red","blue","green"));
-----
plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=2,col=rainbow(50));
```

- There are 50 elements in x. so 50 points in plot.
- *col=c("red","green","blue")* is a vector of 3 elements.
- The color vector will be recycled to color the 50 points. So points will look like alternate red/green/blue.
- *col=rainbow(50)* will generate 50 consecutive colors which will be assigned to 50 points on the plot.

Day 2 | Hands-on Exercises: Data Visualization

```
veccol=c(rep("red",times=20),
rep("blue",times=20),
rep("green",times=10))

plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=2,col=veccol)
```

- Create a vector of colors where red is repeated 20 times followed by blue 20 times and green 10 times.
- Using `c()` and `rep()` function we created `veccol` vector.
- `col=veccol` assigns 50 colors of `veccol` vector to 50 points on the plot.

Different point types, line types (Fig. 7)

```
plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=2,col=c("red","blue","green"),
pch=15);
-----
plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=1,col=c("red","blue","green"),
pch=15,lty=2);
```

- `pch=15` will change the plotting symbol to filled square.
- Try `?pch` to explore more about points.
- `lty=2` will change the line type. (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash)



Rescale the x-axis and y-axis

```
plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=1,col=c("red","blue","green"),
pch=15,lty=2,
ylim=c(0,150),xlim=c(0,100));
```

- `ylim=c(0,150)` will scale y-axis from 0 to 150.

- `xlim` and `ylim` parameters are used to change the default x and y axis ranges.
- We have to give the minimum and maximum values of x and y-axis.
- `xlim=c(0,100)` will scale x-axis from 0 to 100.

Magnification of labels and symbols

```
plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=1,col=c("red","blue","green"),
pch=15,lty=2,cex.lab=1.5);
-----
plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=1,col=c("red","blue","green"),
pch=15,lty=2,cex.lab=1.5,
cex=1.5,cex.axis=1.2,cex.main=1.5);
```

- `cex.lab=1.5` will magnify the x-axis and y-axis labels by 50%. Default `cex.lab=1`.
- `cex.axis=1.2` magnifies the labels on the tick-marks of axis.
- `cex.main=1.2` magnifies the title
- `cex.sub=1.2` magnifies the subtitle.
- `cex=1.2` magnifies the symbol/text on the plots

Day 2 | Hands-on Exercises: Data Visualization

```
plot(x,cex=1.5,cex.lab=1.2,
cex.axis=1.2,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=1,col=c("red","blue","green"),
pch=15,lty=2,bty="n")
-----
plot(x,cex=1.5,cex.lab=1.2,
cex.axis=1.2,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=1,col=c("red","blue","green"),
pch=15,lty=2,col.axis="red",bty="n")
-----
plot(x,cex=1.5,cex.lab=1.2,
cex.axis=1.2,type="o",xlab="Index",
ylab="Value",main="Random plot",lwd=1,
col=c("red","blue","green"),pch=15,
lty=2,las=1,col.lab="brown",bty="n")
```

- *bty="n"* will not draw box around the plot.
- If *bty* is one of "o" (the default), "l", "7", "c", "u", or "]" the resulting box resembles the corresponding upper case letter.
- *col.axis="red"* will color the axis labels on the tick marks to red color
- *col.lab="brown"* will color the x-axis and y-axis labels to brown.
- Style of axis tick mark labels is given by *las=1*. (0: parallel to axis, 1: always horizontal, 2: always perpendicular to axis, 3: always vertical)

Low level plotting functions (lines and legend, Fig. 8)

```
plot(x,type="o",xlab="Index",
ylab="Value",main="Random plot",
lwd=1,col=c("red","blue","green"),
pch=15,lty=2,ylim=c(0,150));
-----
lines(c(0,60),c(50,50),lty=2);
lines(c(-10,60),c(100,100),lty=2);
-----
legend(0,155,c("Day1","Day2","Day3"),
lwd=2,col=c("red","blue","green"));
-----
legend(40,155,c("Day1","Day2","Day3"),
lwd=2,col=c("red","blue","green"));
```

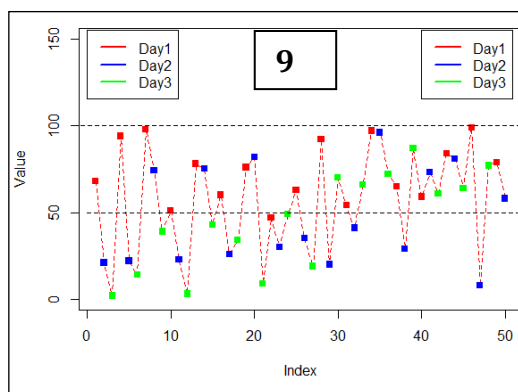
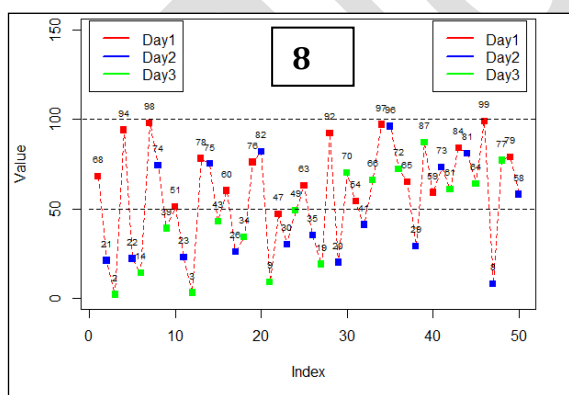
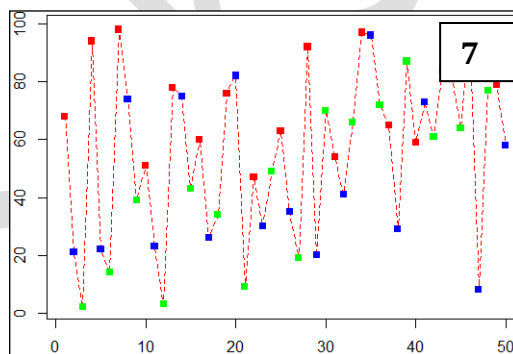
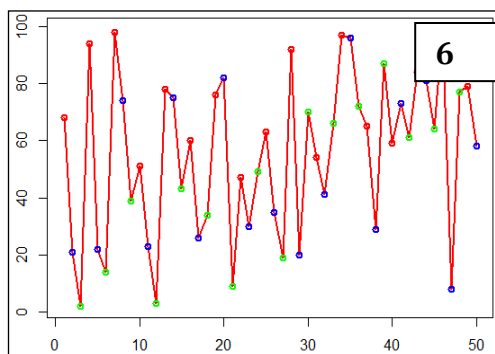
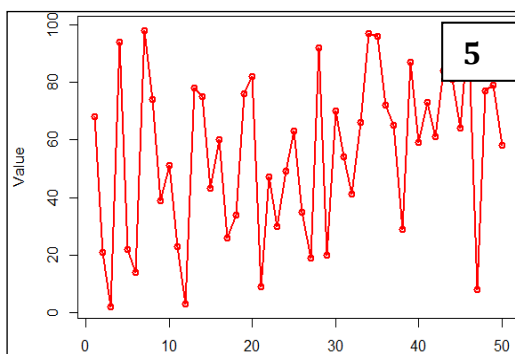
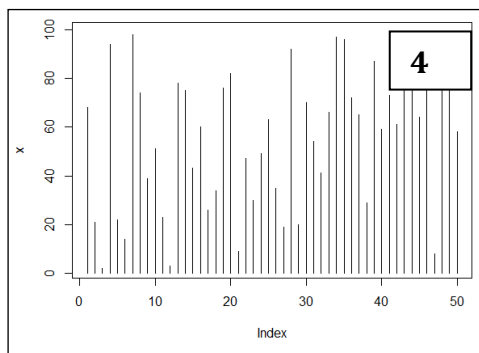
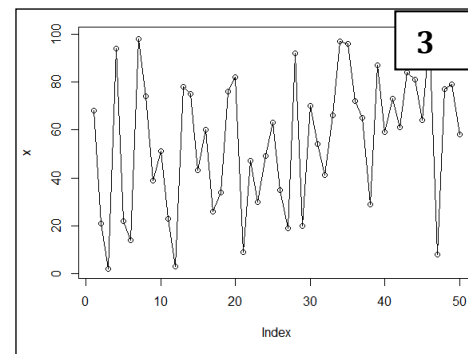
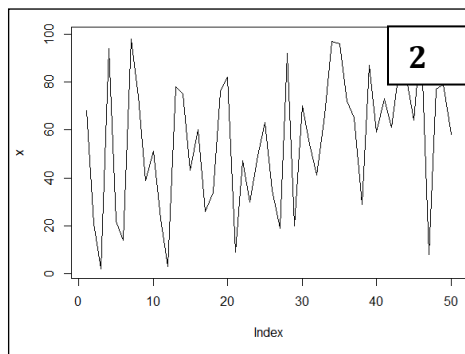
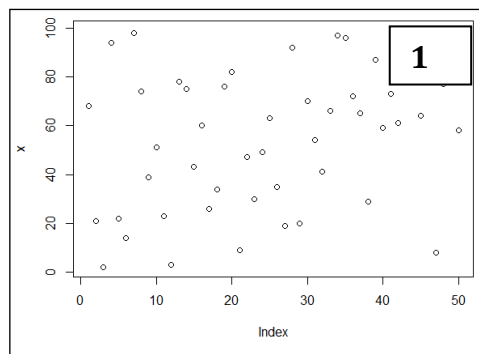
- *lines()* draws lines on a plot. To draw lines we need x and y coordinates of points. The syntax: pass x-coordinates of all points as 1st argument and all y-coordinates as 2nd argument. *lty* and *lwd* can also be used.
- *legend()* draws legends on a plot. To draw legend we need to specify where to draw legend by specifying x and y coordinates as 1st and 2nd argument. The legends to be written as 3rd argument. Colors of legends are passed by *col* parameter while *lwd* sets line width.

Adding texts to existing plot (Fig. 9)

```
text(1:50,x,labels=x);
text(1:50,x,labels=x,pos=3,cex=0.6);
text(1:50,x,labels=x,pos=3,offset=0.8,cex=0.6);
```

- *text()* adds texts to existing plots. To add text, we need x- and y- coordinates which we pass as 1st and 2nd arguments respectively. Third argument (*labels*) is the texts that are to be written on the plot.
- Since we want to write the values of x on the points, the x and y coordinates of text will be exactly same as the points i.e. x-coordinates will be 1 to 50 while y coordinates will be value of x itself. The text to be added will also be x.
- *pos=3* will add the text on the top of point. *pos=1/2/3/4* means *below/left/top/right* to point
- *Offset* is used to put space between point and text. *cex=0.6* decrease the text size.

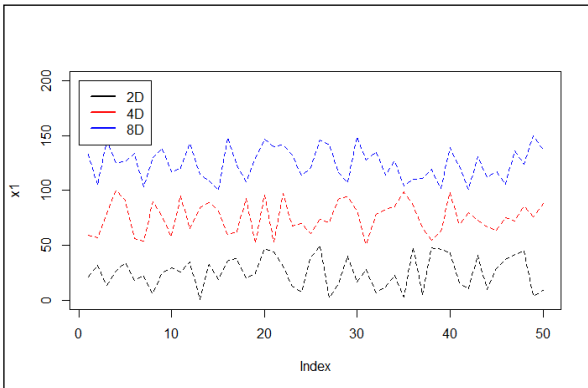
Day 2 | Hands-on Exercises: Data Visualization



Day 2| Hands-on Exercises: Data Visualization

Line Plots

Draw a line plot.



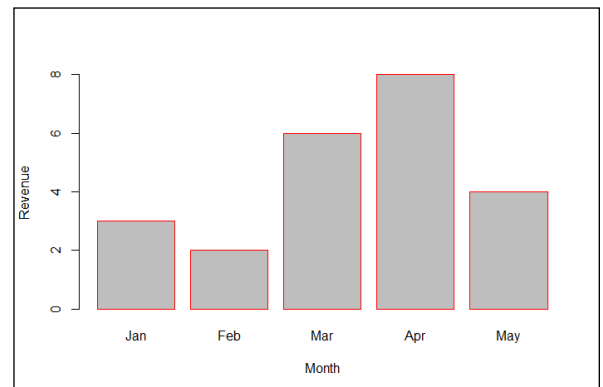
```
2 x1=sample(1:50,50)
3 x2=sample(50:100,50)
4 x3=sample(100:150,50)
5 -----
6 plot(x1,type="l",lty=2)
7 lines(x2,type="l",col="red",lty=2);
8 lines(x3,type="l",col="blue",lty=2);
9 -----
10 plot(x1,type="l",lty=2,ylim=c(0,200));
11 lines(x2,type="l",col="red",lty=2);
12 lines(x3,type="l",col="blue",lty=2);
13 -----
14 legend(0,200,c("2D","4D","8D"),
15 lwd=2,col=c("black","red","blue"));
```

- Create 3 vectors x1, x2, x3 with 50 elements with different range values.
- To visualize x1, x2 and x3 as lines, we need to first draw x1 using high-level `plot()` command then x2 and x3 can be drawn using low-level `lines()` command by adding lines to the already existing plots.
- Lines 6-8 will plot 3-lines but we can't see x2 and x3 lines since the `plot()` in Line6 was executed using x1 values whose y-axis margin lies between 0-50 but x2 values between 50-100 and x3 between 100-150. So x2 and x3 will be out-of margin on the plot.
- To correctly see x2 and x3, we first sets the `ylim` to 0-200 (Line10) and redraw.

Simple Bar plots

Revenue of a company from Jan-May...

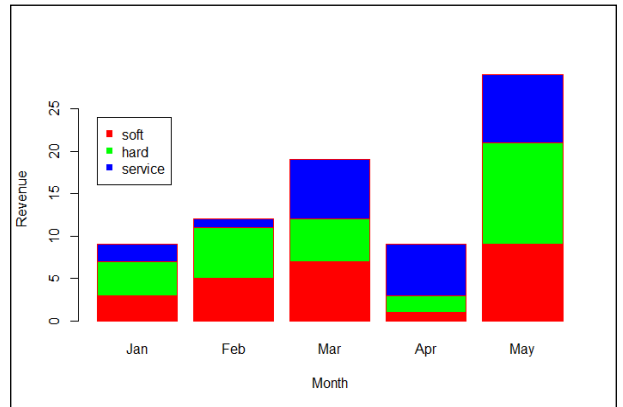
```
2 x <- c(3,2,6,8,4);
3 barplot(x);
4 -----
5 barplot(x,
6 names.arg=c("Jan","Feb","Mar","Apr","May"),
7 xlab="Month",ylab="Revenue",border="red");
8 -----
9 barplot(x,horiz=TRUE,
10 names.arg=c("Jan","Feb","Mar","Apr","May"),
11 ylab="Month",xlab="Revenue",border="red");
```



- Here x can be considered as heights of bars in a bar plot.
- Using `names.arg=c()` we can name the bars.
- `border="red"` will give red border to bars.
- Line5-7 will generate the output as shown in Figure.
- `horiz=TRUE` will make *horizontal bar* plot. Note: `xlab` and `ylab` changed accordingly.

Stacked Bar plots

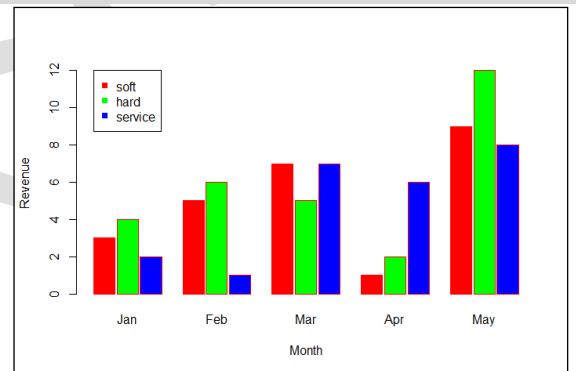
```
1 A <- matrix(  
2 c(3,5,7,1,9,4,6,5,2,12,2,1,7,6,8),  
3 nrow=3,ncol=5,byrow=TRUE);  
4  
5 barplot(A);  
6 -----  
7 barplot(A,  
8 names.arg=c("Jan","Feb","Mar","Apr","May"),  
9 xlab="Month",ylab="Revenue",border="red");  
10 -----  
11 barplot(A,col=c("red","green","blue"),  
12 names.arg=c("Jan","Feb","Mar","Apr","May"),  
13 xlab="Month",ylab="Revenue",border="red");  
14 -----  
15 legend(x=0.2,y=24,  
16 legend=c("soft","hard","service"),  
17 pch=15,col=c("red","green","blue"));
```



- When argument to `barplot()` is a matrix, a stacked bar plot will be generated.
- Using `names.arg` assign name to each bar, `col` parameter to color each stack.

Grouped Bar Plots

```
barplot(A,beside=T,  
space=c(0.1,1),  
col=c("red","green","blue"),  
names.arg=c("Jan","Feb","Mar","Apr","May"),  
xlab="Month",ylab="Revenue",border="red");  
  
legend(x=1,y=12,  
legend=c("soft","hard","service"),  
pch=15,col=c("red","green","blue"));
```

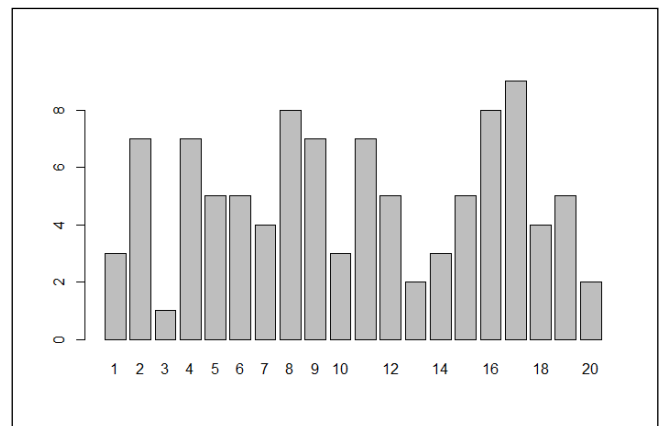


- The argument `beside=TRUE` will convert stacked to grouped bar plot
- `space=c(0.1, 1)` specifies to add 0.1 spacing between bars and spacing of 1 between each groups.

Calculate frequency and then visualize using bar plot.

```
1 x=sample(20,100,replace=T)  
2 t=table(x)  
3 barplot(t)
```

- `x` is vector of 100 elements with values between 0 and 20.
- `Table()` function categorizes a vector and count the number in each category.
- Here table function creates 20 category and count its frequency which we consider as height in bar plot.

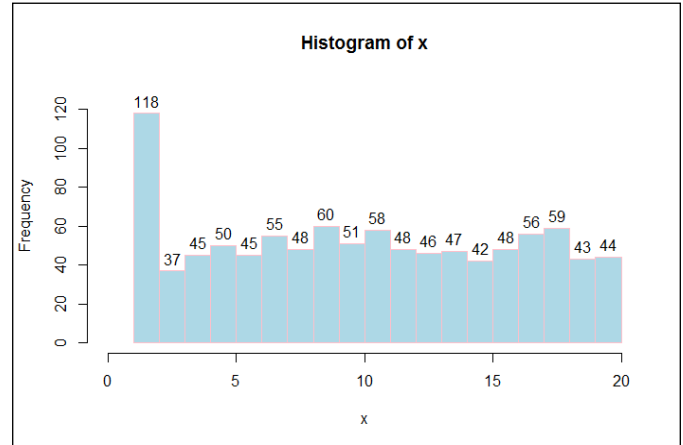


Histogram (? hist for more details)

```

1 x=sample(20,1000,replace=T)
2 hist(x)
3 hist(x,col="grey",labels=T)
4 hist(x,col="grey",labels=T,ylim=c(0,130));
5 -----
6 hist(x,labels=TRUE,ylim=c(0,130),
7 col = "lightblue", border = "pink" );
8 -----
9 hist(x,breaks=20,xlim=c(0,20)
10 labels=TRUE,ylim=c(0,130),
11 col = "lightblue", border = "pink");
12 -----
13 hist(x,freq=F,breaks=20,labels=F,
14 ylim=c(0,0.12),col="lightblue",
15 border="pink",xlim=c(0,20));

```



- *labels=T* will add label on the bars. *col* will fill the bars with colors, *border* is to give border color of bar. *Breaks=20* will assign number of histogram bins to 20.
- *Freq=F* will plot probability instead of frequency. Note: Probability lies between 0 to 1. So we have to change *ylim* accordingly.

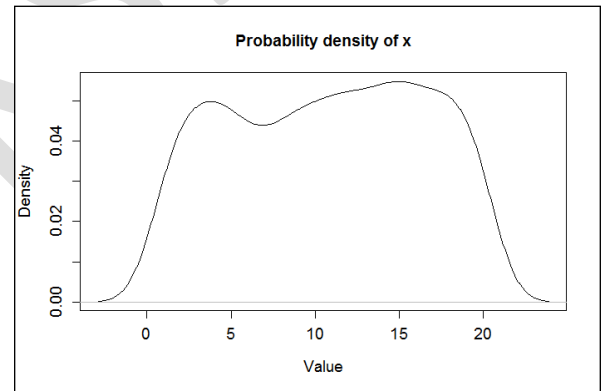
Density plot (?density for more details)

Plot density of single distribution

```

1 x=sample(20,1000,replace=T)
2 plot(density(x));
3
4 plot(density(x),xlab="Value",
5 main="Probability density of x",
6 cex.lab=1.2,cex.axis=1.2);

```



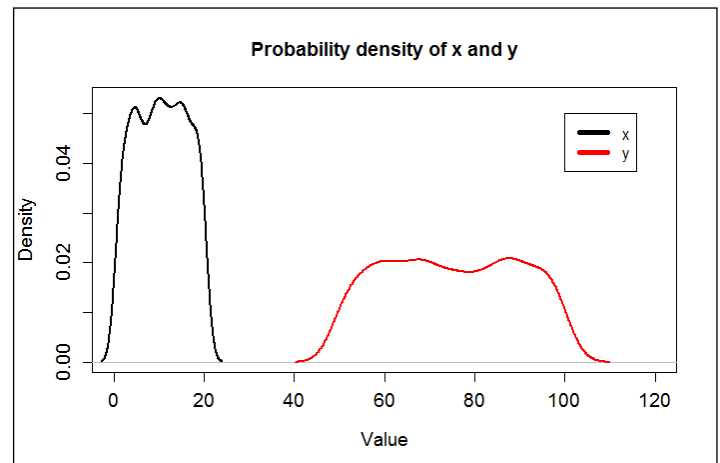
density(x) function computes kernel density estimates of *x* which is plotted by *plot(density(x))*.

Plot density plot of two distributions

```

1 x=sample(20,1000,replace=T)
2 y=sample(50:100,1000,replace=T)
3
4 plot(density(x));
5 lines(density(y));
6 -----
7 plot(density(x),xlim=c(0,100));
8 lines(density(y));
9 -----
10 plot(density(x),xlim=c(0,120),
11 lwd=2,cex.lab=1.2,cex.axis=1.2,
12 xlab="Value",main="Prob. density");
13
14 lines(density(y),lwd=2,col="red");
15
16 legend(100,0.05,c("x","y"),
17 col=c("black","red"),lwd=5);

```

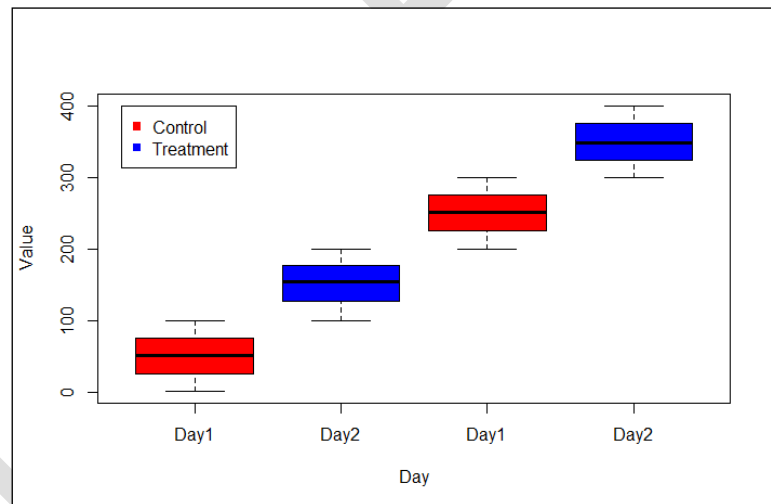


Day 2 | Hands-on Exercises: Data Visualization

- x and y are two distributions whose density are estimated using density(x) and density(y).
- Line4-5: plots the density of x and y but since the x-ranges of x and y are different (x values between 0-20 while y between 50 and 100), we can't see distribution of y in this plot.
- Line7-8 we change xlim between 0 and 100 so that both x and y densities are within plot-margin.
- Line10-11, x by defaults get black while y gets red color (Line14).

Box plots

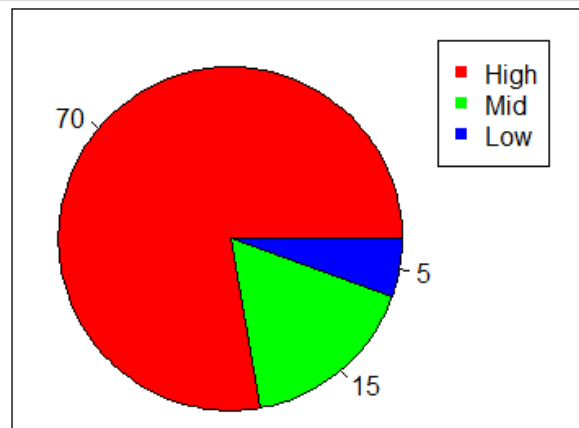
```
1 x1=sample(1:100,1000,replace=T)
2 x2=sample(100:200,1000,replace=T)
3 x3=sample(200:300,1000,replace=T)
4 x4=sample(300:400,1000,replace=T)
5 m=cbind(x1,x2,x3,x4);
6 head(m)
7
8 boxplot(m);
9 -----
10 boxplot(m,
11 col=c("red","blue","red","blue"),
12 xlab="Day",ylab="Value");
13 -----
14 boxplot(m,
15 names=c("Day1","Day2","Day1","Day2"),
16 col=c("red","blue","red","blue"),
17 xlab="Day",ylab="Value");
18 -----
19 legend(0.5,400,
20 c("Control","Treatment"),
21 col=c("red","blue"),pch=15);
```



- Create a matrix of 1000*4 dimensions. Let's assume that these are the gene expression values of 1000 genes measured across two days among control and treatment sample. Columns represents: Control-Day1, Control-Day2, Treatment-Day1 and Treatment-Day2.
- Boxplot() gives the distribution of data. *col* parameters sets colors to each box. *names* parameter sets names below each box.

Pie chart

```
1 x=c(70,15,5);
2 pie(x);
3 -----
4 pie(x,labels=x,
5 col=c("red","green","blue"));
6
7 legend(0.97,0.92,
8 c("High","Mid","Low"),
9 col=c("red","green","blue"),pch=15)
```



- 1st argument is the proportion values, *labels* will assign labels to each category, *col* sets color to each category.

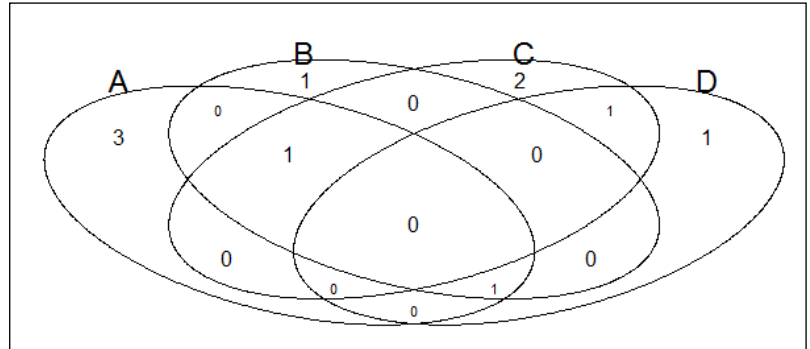
Day 2 | Hands-on Exercises: Data Visualization

Venn diagram

```

1 library(gplots)
2 x1=1:5
3 x2=4:6
4 x3=c(4,8:10);
5 x4=c(7,5,10);
6 x5=c(7,10);
7
8 v1=venn( list(x1,x2,x3) );
9 v2=venn( list(x1,x2,x3,x4) );
10 v2=venn( list(x1,x2,x3,x4,x5) );

```



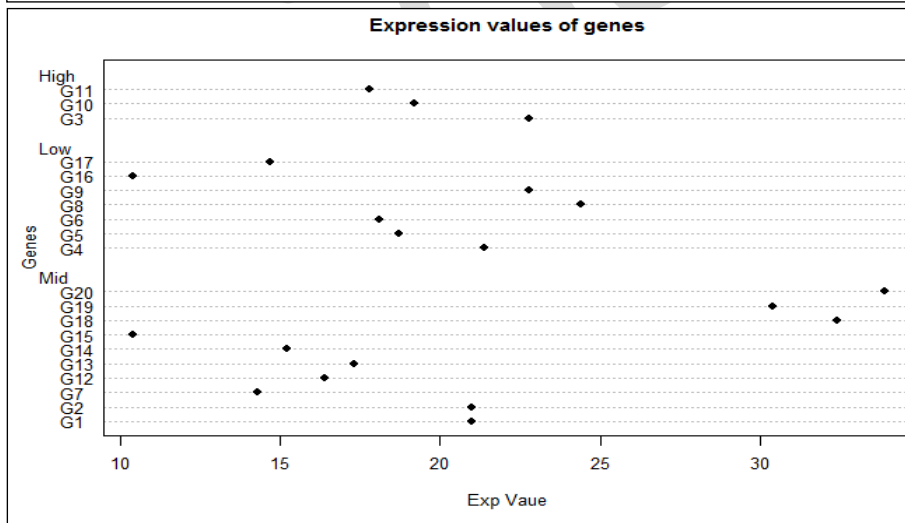
- We use `venn()` function of `gplots` library to draw venn diagram.
- `venn()` accepts a list of vector to be used for diagram.

Dot chart

```

1 data=read.table("Genexp_dot.txt",header=T)
2
3 dotchart(data$Exp,xlab="Exp Vaue",
4 ylab="Genes",main="Expression values of genes");
5 -----
6 dotchart(data$Exp,
7 labels=data$Gene,
8 xlab="Exp Vaue",ylab="Genes",
9 main="Expression values of genes");
10 -----
11 dotchart(data$Exp,
12 cex=0.8,labels=data$Gene,
13 xlab="Exp Vaue",ylab="Genes",
14 main="Expression values of genes");
15 -----
16 dotchart(data$Exp,pch=16,
17 group=data$Level,cex=0.8,labels=data$Gene,
18 xlab="Exp Vaue",ylab="Genes",
19 main="Expression values of genes");

```



- Dot chart is another way of data visualization.
- You are provided with gene expression data consists of 3 columns, Gene name, its expression value and its Gene class (High/Middle/Low level of gene expression)
- Line1 reads data using `read.table()`
- Line3-4: `dotchart` of expression values
- Line6-9: We add the Gene names to each dot using `labels=data$Gene`.
- Line11-14: We use `cex=0.8` to prevent overlap between gene names.
- Line16-19: We divide the data into 3 groups as High/Mid/Low gene expression classes and for each group we redraw the dot chart.

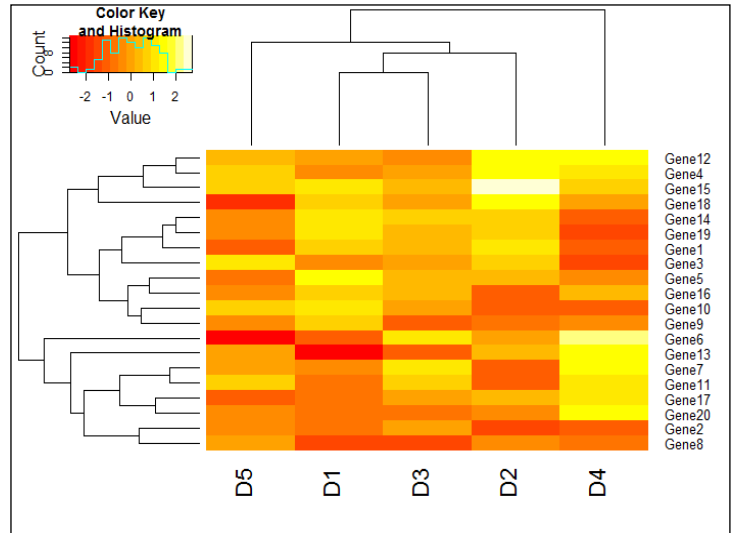
The dot chart can be useful to see how a given value is distributed across a set of groups.

Heat map

```

1 m1=read.table("GenExp_heatmap.txt")
2 m1=as.matrix(m1);
3
4 library("gplots")
5 heatmap.2(m1)
6 heatmap.2(m1,trace="none")
7
8 heatmap.2(m1,trace="none",
9 cellnote=m,notecol="black");
10
11 heatmap.2(m1,trace="none",
12 cellnote=m,notecol="black",
13 Colv=FALSE);

```



- We use `heatmap.2()` of `gplots` package to draw heatmap.
- We first read the data as data frame. Since the `heatmap.2()` function accepts matrix, we convert data frame to matrix using `as.matrix()` function.
- `trace="none"` will prevent to draw tract of color key. `cellnote=m` will allow us to write the values of matrix `m` in each cell. `notecol="black"` will set the color of notes in each cell as black.
- `Colv=FALSE` will prevent the ordering of column as per hierarchical clustering. So the dendrogram won't be displayed. `Rowv=FALSE` will prevent clustering of rows.

Scatter plot Matrices

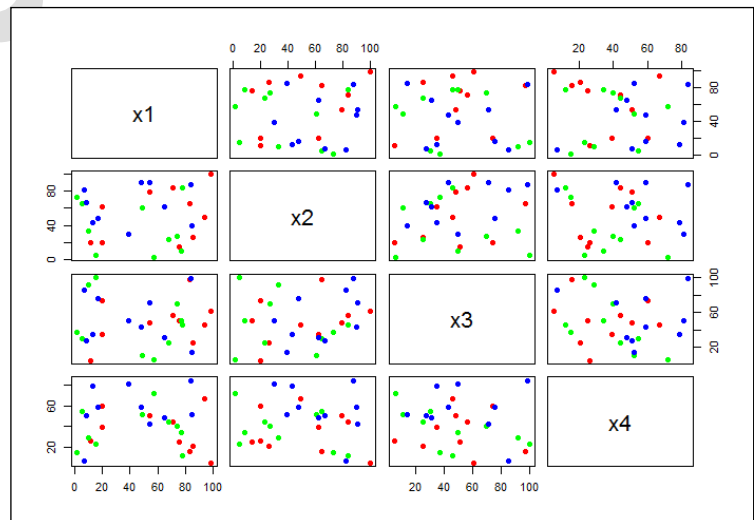
```

x1=sample(1:100,30,replace=T)
x2=sample(1:100,30,replace=T)
x3=sample(1:100,30,replace=T)
x4=sample(1:100,30,replace=T)
m=data.frame(x1,x2,x3,x4);

pairs(m)

pairs(m,pch=16,
col=rep(c("red","green","blue"),
each=10))

```

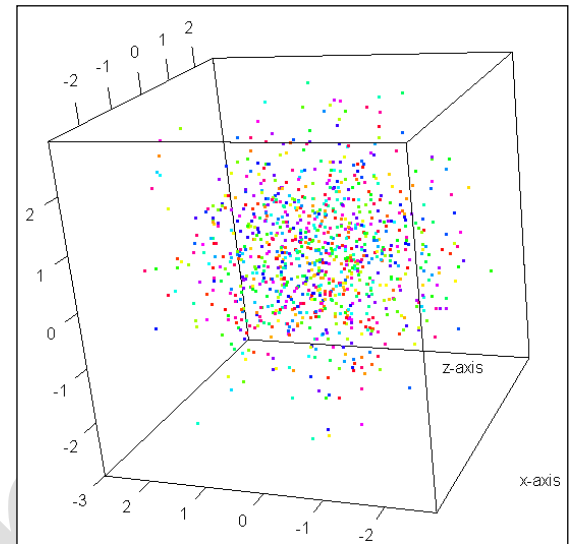


- `m` is a matrix of dimension 30×4 .
- `pairs()` function is useful to draw a matrix of scatterplots.
- This is useful to get a global view of data distribution.
- `pch` is used to change symbol while `col` is a vector of 30 colors corresponding to 30 observations (1st 10 observations are colored red followed by next 10 as green followed by last 10 as blue).

3D Scatter plot

```
library(rgl);  
x=rnorm(1000);  
y=rnorm(1000);  
z=rnorm(1000);  
plot3d(x,y,z,  
size=3,col=rainbow(1000),  
xlab="x-axis",ylab="y-axis",  
zlab="z-axis");
```

- Use *rgl* package to draw 3d scatterplot.
- *size=3* will change size of points, *col* parameter is to change the color of points.
- *?plot3d* for more details

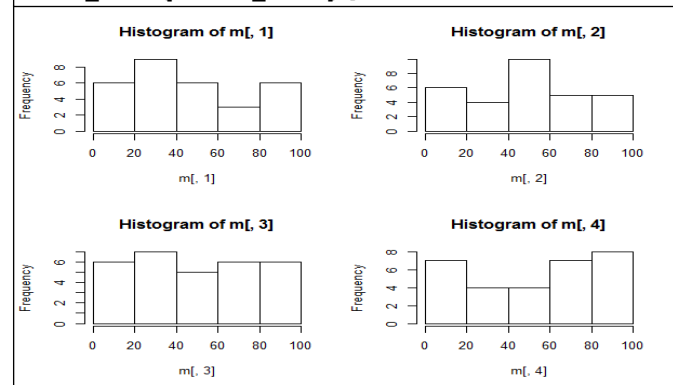


Multi panel plots

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

- By setting the *mfrow* parameter to number of rows and columns, multi-panel plots can be generated.
- Line1: we set the *mfrow* parameter to `c(2,2)` i.e. 2 rows and 2 columns and pass *mfrow* in *par()* function. *par()* after modifying the graphics parameter, returns original state of graphics device which we store in *oldpar*. In Line6, we reset the graphics device to older state using *oldpar*.
- Line2-4 will generate 4 plots, filled row-wise (because of *mfrow*). To fill the plots column wise we can use *mfcol* parameter.

```
1 oldpar=par(mfrow=c(2,2));  
2 hist(m[,1])  
3 hist(m[,2])  
4 hist(m[,3])  
5 hist(m[,4])  
6 par(oldpar);
```



Some other ways of generating multi-panel plots are using *layout()* and *split.screen()* functions and setting *fig* parameter in *par()* function.