Visualization using R

objectives for graphical excellence

1.Show, and even reveal, the data:
The data should tell a story, especially a story hidden in large masses of data. However, reveal the data in context, so the story is correctly told.
2. Induce the viewer to think of the substance of the data: The format of the graph should be so natural to the data, that it hides itself and lets data shine.
3. Avoid distorting what the data have to say: Statistics can be used to lie. In the name of simplifying, some crucial context could be removed leading to distorted communication.
4. Make large data sets coherent: By giving shape to data, visualizations can help bring the data together to tell a comprehensive story.
5. Encourage the eyes to compare different pieces of data: Organize the chart in ways the eyes would naturally move to derive insights from the graph.
6. Reveal the data at several levels of detail: Graphs leads to insights, which raise further curiosity, and thus presentations should help get to the root cause.
7. Serve a reasonably clear purpose – informing or decision-making.
8. Closely integrate with the statistical and verbal descriptions of the dataset: There should be no separation of charts and text in presentation. Each mode should tell a complete story. Intersperse text with the map/graphic to highlight the

1. Histogram

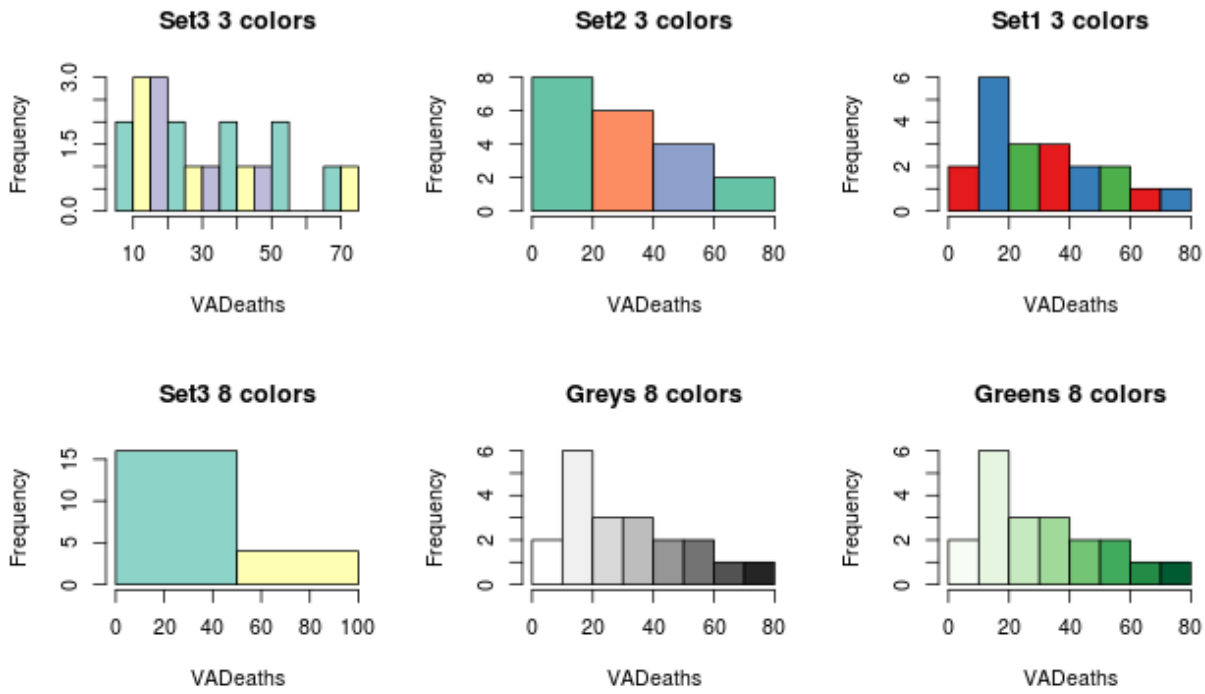
Histogram is basically a plot that breaks the data into bins (or breaks) and shows frequency distribution of these bins. You can change the breaks also and see the effect it has data visualization in terms of understandability.

Let me give you an example.

Note: We have used par(mfrow=c(2,5)) command to fit multiple graphs in same page for sake of clarity(see the code below).

The following commands show this in a better way. In the code below, the *main* option sets the Title of Graph and the *col* option calls in the color pallete from RColorBrewer to set the colors.

```
library(RColorBrewer)
data(VADeaths)
par(mfrow=c(2,3))
hist(VADeaths,breaks=10, col=brewer.pal(3,"Set3"),main="Set3 3
colors")
hist(VADeaths,breaks=3 ,col=brewer.pal(3,"Set2"),main="Set2 3
colors")
hist(VADeaths,breaks=7, col=brewer.pal(3,"Set1"),main="Set1 3
colors")
hist(VADeaths,,breaks= 2, col=brewer.pal(8,"Set3"),main="Set3 8
colors")
hist(VADeaths,col=brewer.pal(8,"Greys"),main="Greys 8 colors")
hist(VADeaths,col=brewer.pal(8,"Greens"),main="Greens 8 colors")
```



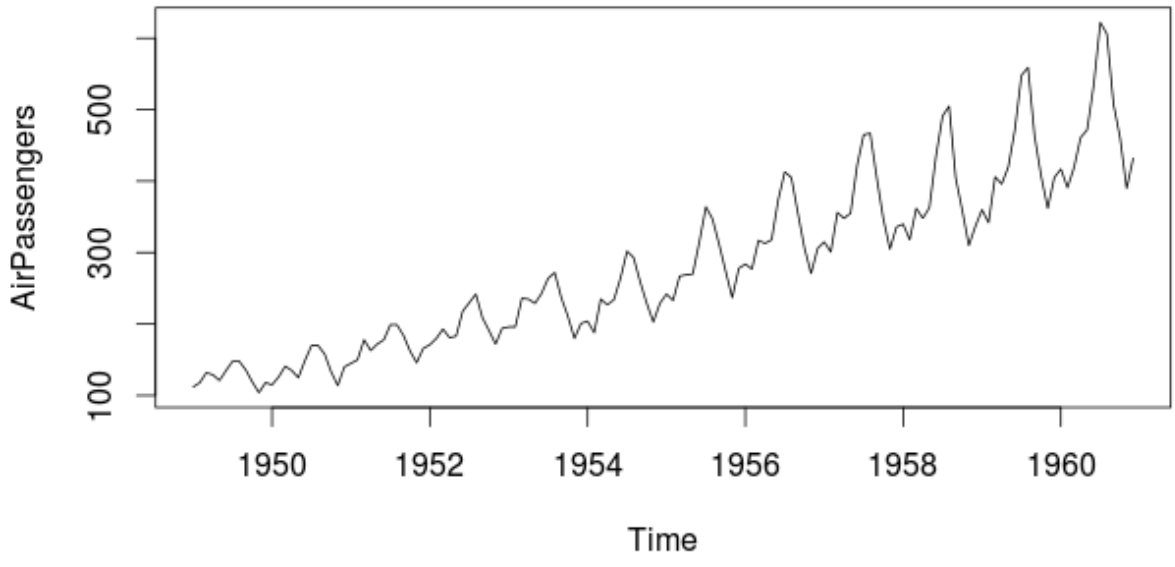
Notice, if number of breaks is less than number of colors specified, the colors just go to extreme values as in the “Set 3 8 colors” graph. If number of breaks is more than number of colors, the colors start repeating as in the first row.

2. Bar/ Line Chart

Line Chart

Below is the line chart showing the increase in air passengers over given time period. Line Charts are commonly preferred when we are to analyse a trend spread over a time period. Furthermore, line plot is also suitable to plots where we need to compare relative changes in quantities across some variable (like time). Below is the code:

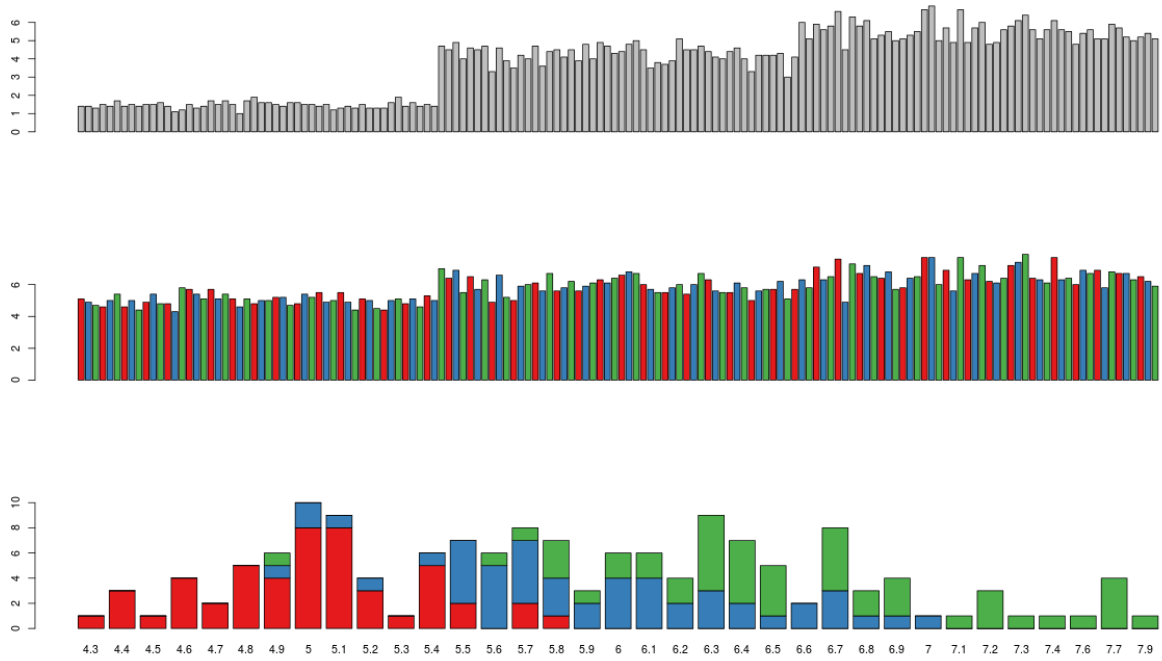
```
plot(AirPassengers,type="l")  #Simple Line Plot
```



Bar Chart

Bar Plots are suitable for showing comparison between cumulative totals across several groups. Stacked Plots are used for bar plots for various categories. Here's the code:

```
barplot(iris$Petal.Length) #Creating simple Bar Graph
barplot(iris$Sepal.Length,col  = brewer.pal(3,"Set1"))
barplot(table(iris$Species,iris$Sepal.Length),col  =
brewer.pal(3,"Set1")) #Stacked Plot
```



3. Box Plot (including group-by option)

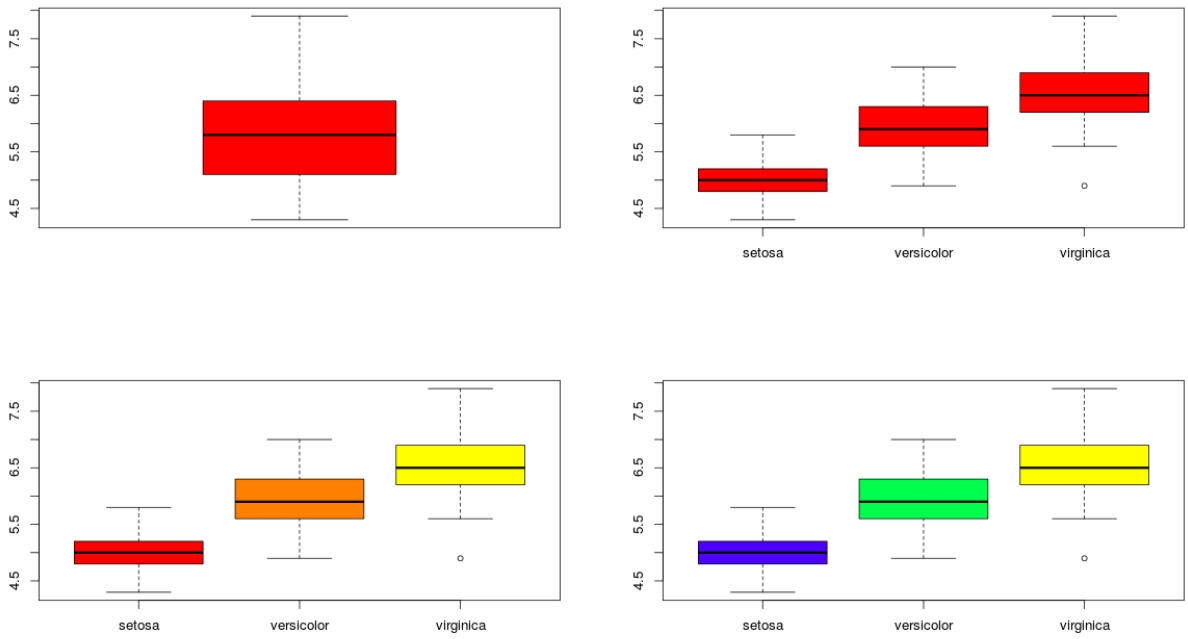
Box Plot shows 5 statistically significant numbers- the minimum, the 25th percentile, the median, the 75th percentile and the maximum. It is thus useful for visualizing the spread of the data is and deriving inferences accordingly. Here's the basic code:

```
boxplot(iris$Petal.Length~iris$Species) #Creating Box Plot
between two variable
```

Let's understand the code below:

In the example below, I have made 4 graphs in one screen. By using the ~ sign, I can visualize how the spread (of Sepal Length) is across various categories (of Species). In the last two graphs I have shown the example of color palettes. A color palette is a group of colors that is used to make the graph more appealing and helping create visual distinctions in the data.

```
data(iris)
par(mfrow=c(2,2))
boxplot(iris$Sepal.Length,col="red")
boxplot(iris$Sepal.Length~iris$Species,col="red")
oxplot(iris$Sepal.Length~iris$Species,col=heat.colors(3))
boxplot(iris$Sepal.Length~iris$Species,col=topo.colors(3))
```



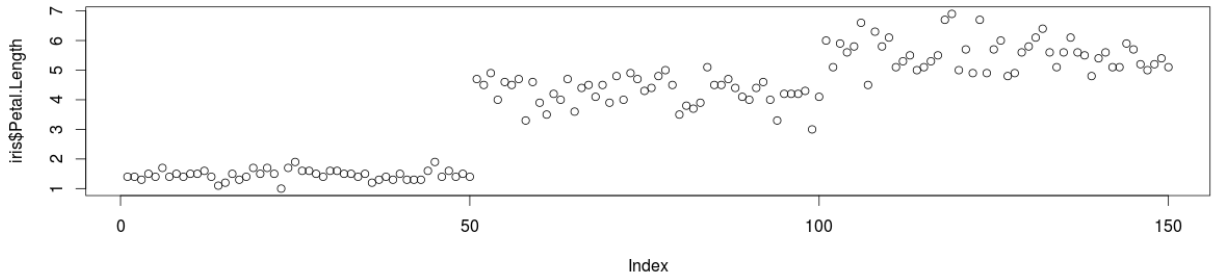
To learn more about the use of color palletes in R, visit here.

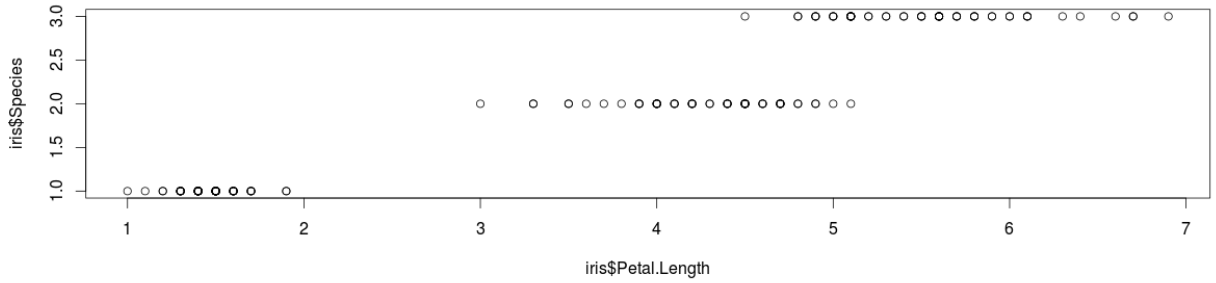
4. Scatter Plot (including 3D and other features)

Scatter plots help in visualizing data easily and for simple data inspection.

Here's the code for simple scatter and multivariate scatter plot:

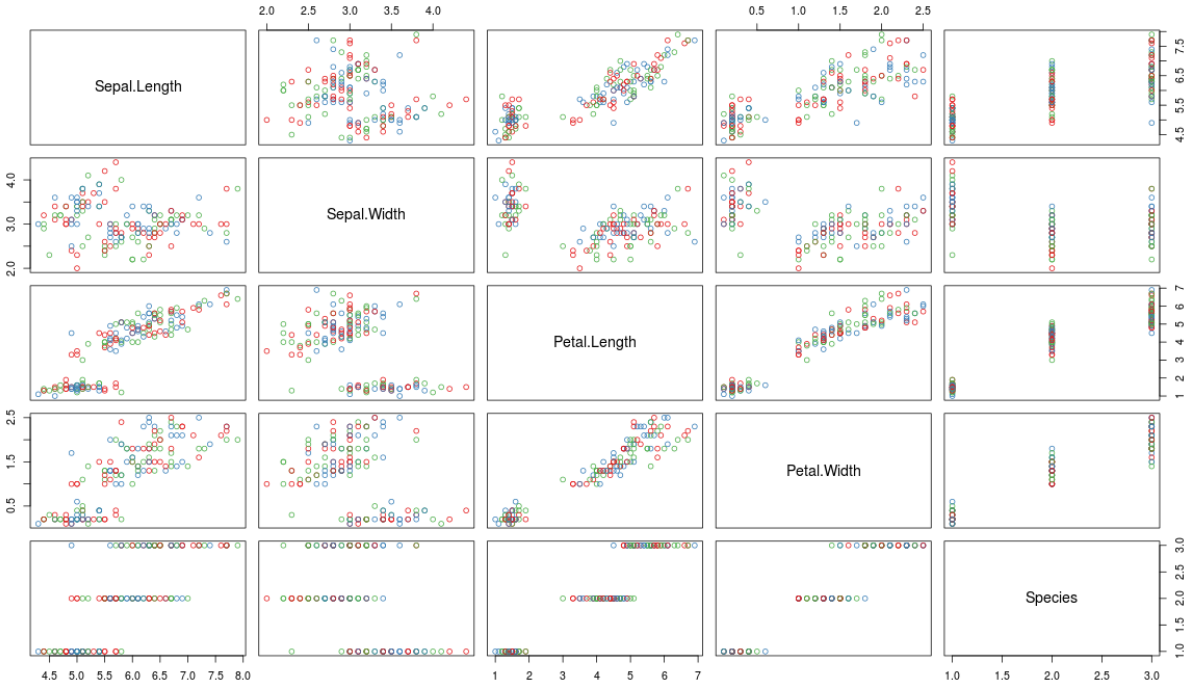
```
plot(x=iris$Petal.Length) #Simple Scatter Plot
plot(x=iris$Petal.Length,y=iris$Species) #Multivariate Scatter
Plot
```





Scatter Plot Matrix can help visualize multiple variables across each other.

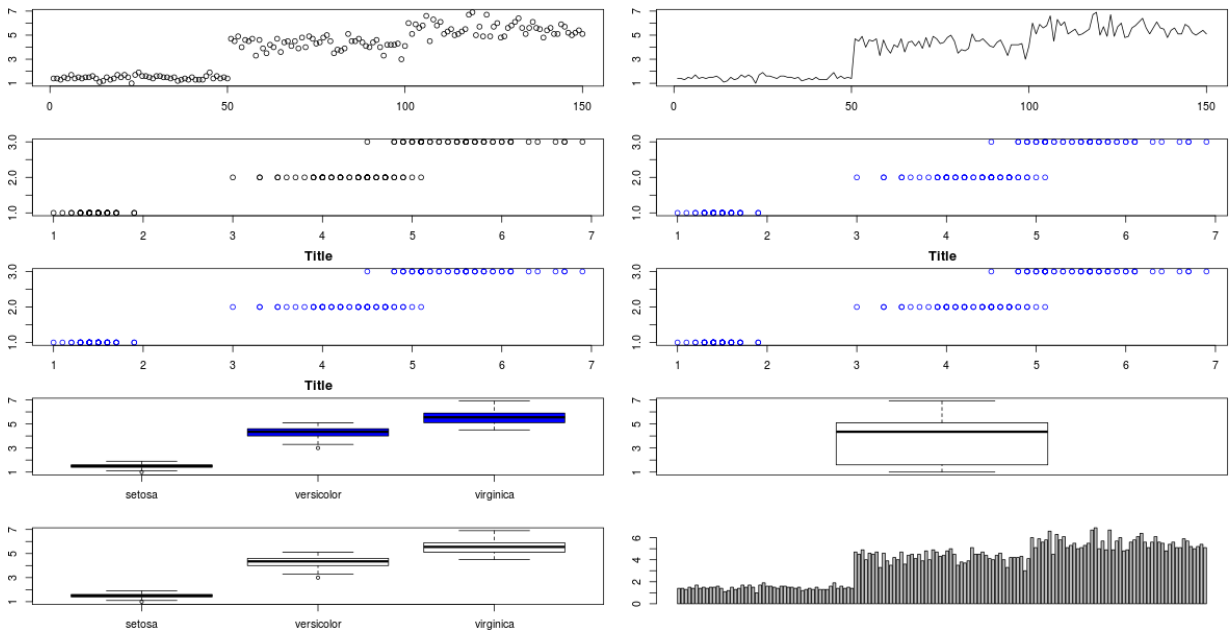
```
plot(iris,col=brewer.pal(3,"Set1"))
```



You might be thinking that I have not put pie charts in the list of basic charts. That's intentional, not a miss out. This is because data visualization professionals frown on the usage of pie charts to represent data. This is because of the human eye cannot visualize circular distances as accurately as linear distance. Simply put- anything that can be put in a pie chart is better represented as a line graph. However, if you like pie-chart, use:

```
pie(table(iris$Species))
```

Here'a a complied list of all the graphs that we have learnt till here:



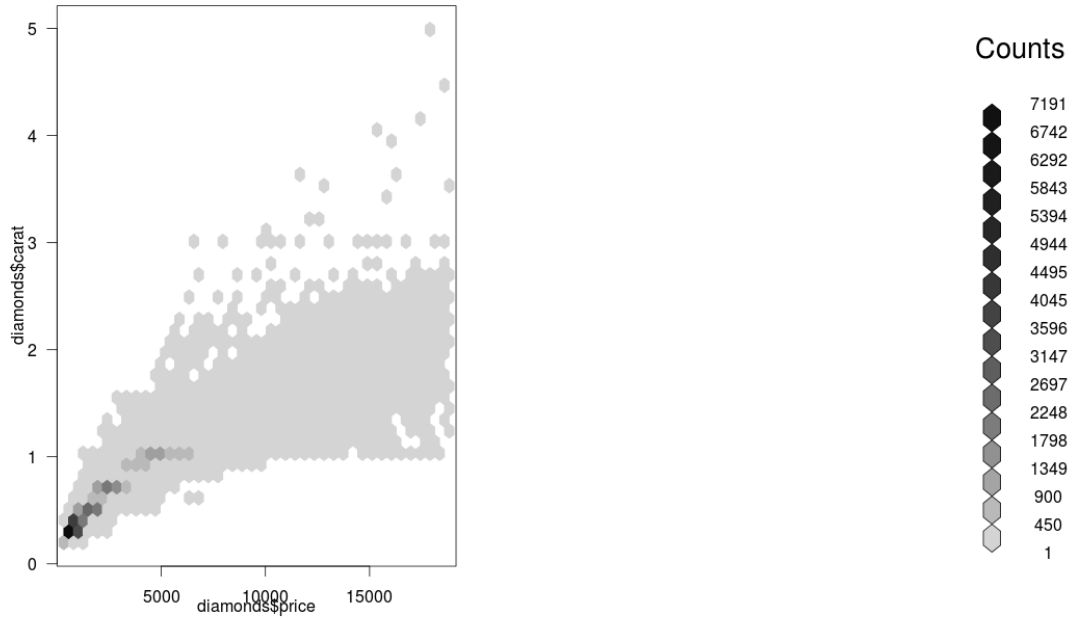
You might have noticed that in some of the charts, their titles have got truncated because I've pushed too many graphs into same screen. For changing that, you can just change the 'mfrow' parameter for par .

Advanced Visualizations

What is Hexbin Binning ?

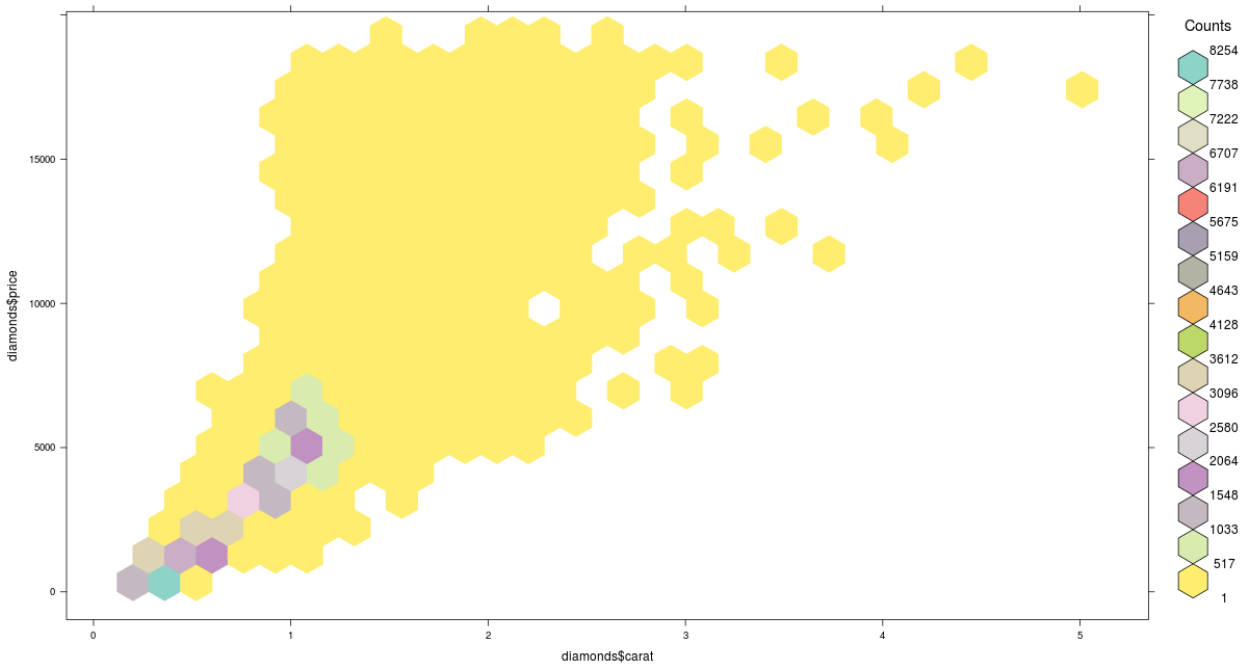
We can use the hexbin package in case we have multiple points in the same place (overplotting). Hexagon binning is a form of bivariate histogram useful for visualizing the structure in datasets with large n. Here's the code:

```
>library(hexbin)
>a=hexbin(diamonds$price,diamonds$carat,xbins=40)
>library(RColorBrewer)
>plot(a)
```



We can also create a color palette and then use the hexbin plot function for a better visual effect. Here's the code:

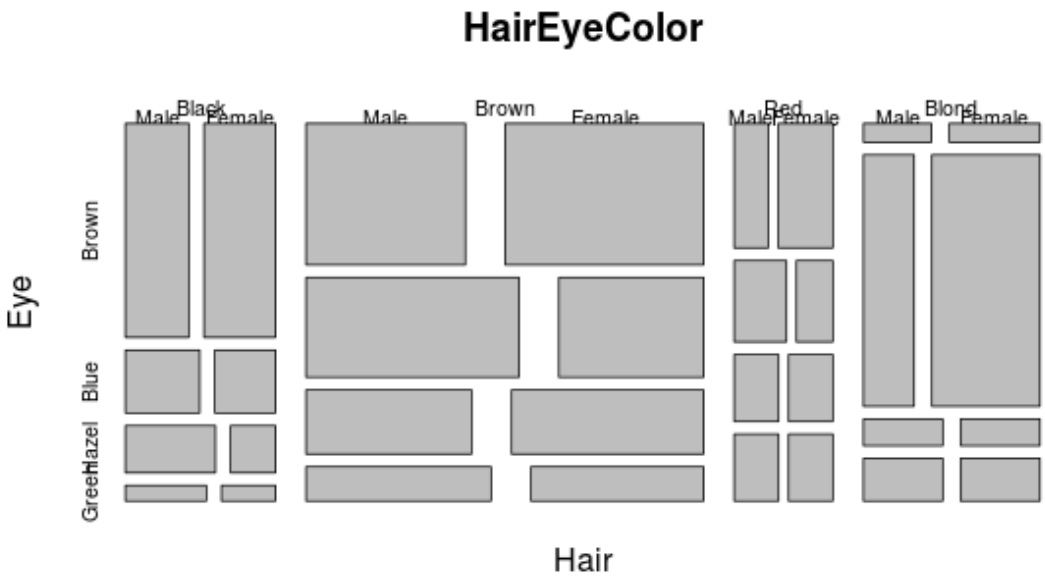
```
>library(RColorBrewer)
>rf <- colorRampPalette(rev(brewer.pal(40,'Set3')))
>hexbinplot(diamonds$price~diamonds$carat, data=diamonds,
colramp=rf)
```



Mosaic Plot

A mosaic plot can be used for plotting categorical data very effectively with the area of the data showing the relative proportions.

```
> data(HairEyeColor)
 > mosaicplot(HairEyeColor)
```



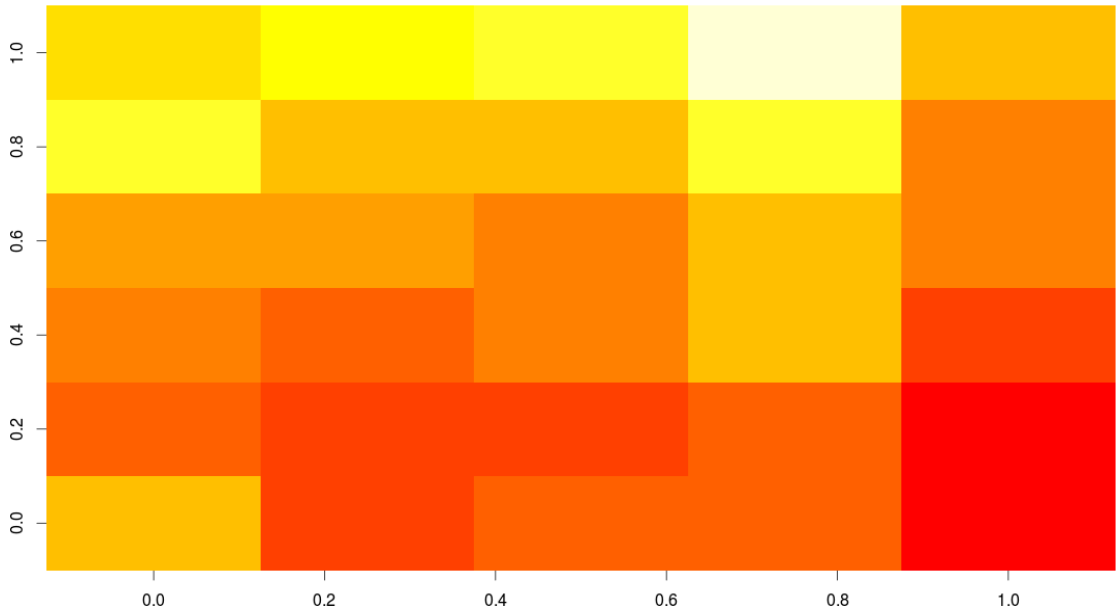
Heat Map

Heat maps enable you to do exploratory data analysis with two dimensions as the axis and the third dimension shown by intensity of color. However you need to convert the dataset to a matrix format. Here's the code:

```
> heatmap(as.matrix(mtcars))
```

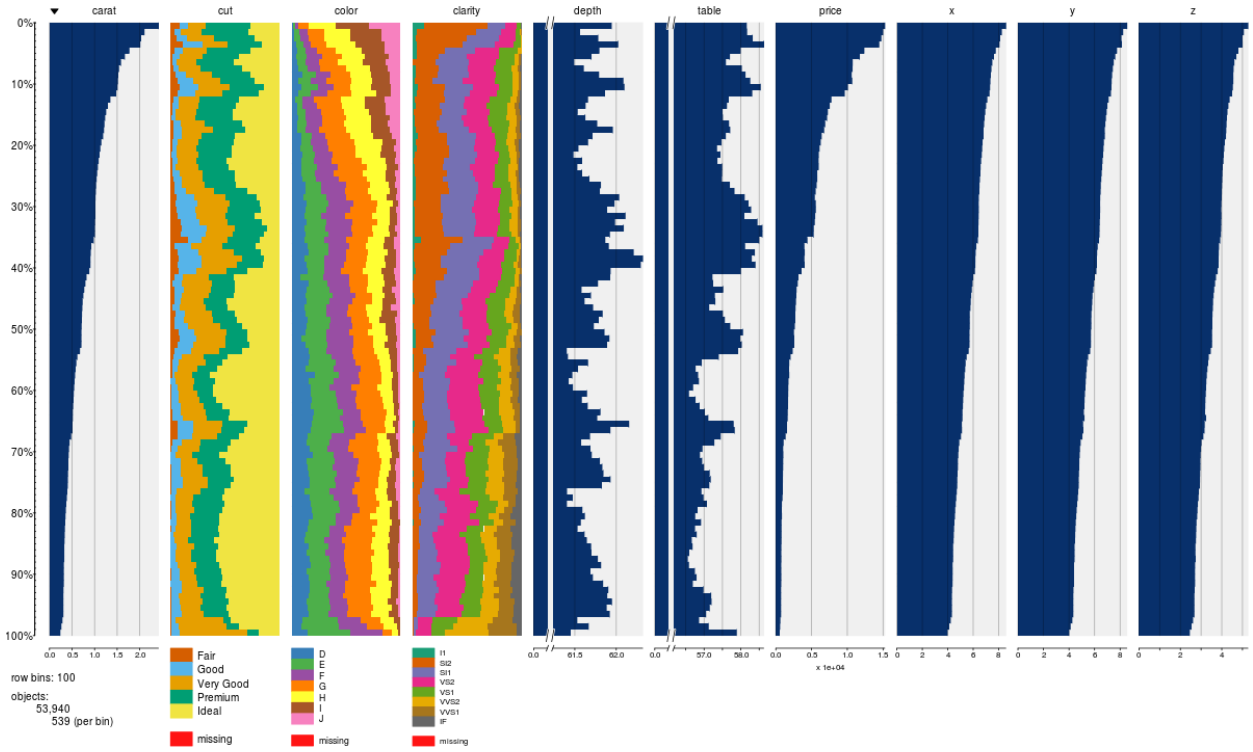
You can use image() command also for this type of visualization as:

```
> image(as.matrix(b[2:7]))
```



How to summarize lots of data ?

You can use tableplot function from the tabplot package to quickly summarize a lot of data

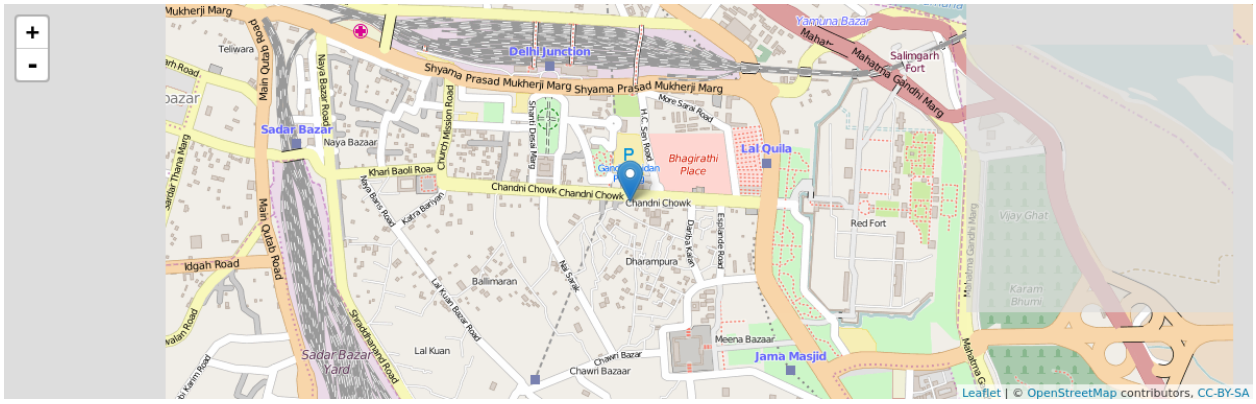


Map Visualization

The latest thing in R is data visualization through Javascript libraries. Leaflet is one of the most popular open-source JavaScript libraries for interactive maps. It is based at https://rstudio.github.io/leaflet/

You can install it straight from github using:

```
devtools::install_github("rstudio/leaflet")
```



The code for the above map is quite simple:

```
library(magrittr)
library(leaflet)
m <- leaflet() %>%
addTiles() %>%  # Add default OpenStreetMap map tiles
addMarkers(lng=77.2310, lat=28.6560, popup="The delicious food
of chandni chowk")
m  # Print the map
```

3D Graphs

One of the easiest ways of impressing someone by R's capabilities is by creating a 3D graph in R without writing ANY line of code and within 3 minutes. Is it too much to ask for?

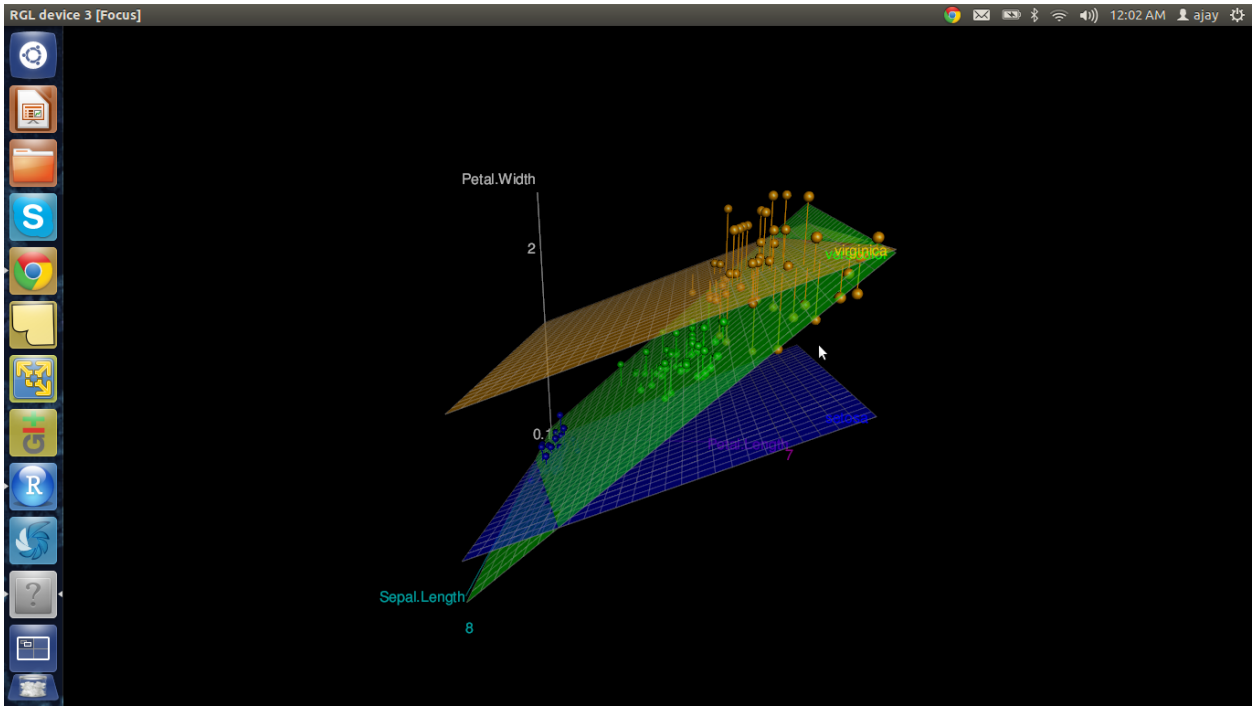
We use the package R Commander which acts as Graphical User Interface (GUI). Here are the steps:

- Simply install the Rcmdr package
- Use the 3D plot option from within graphs

The code below is not typed by the user but automatically generated.

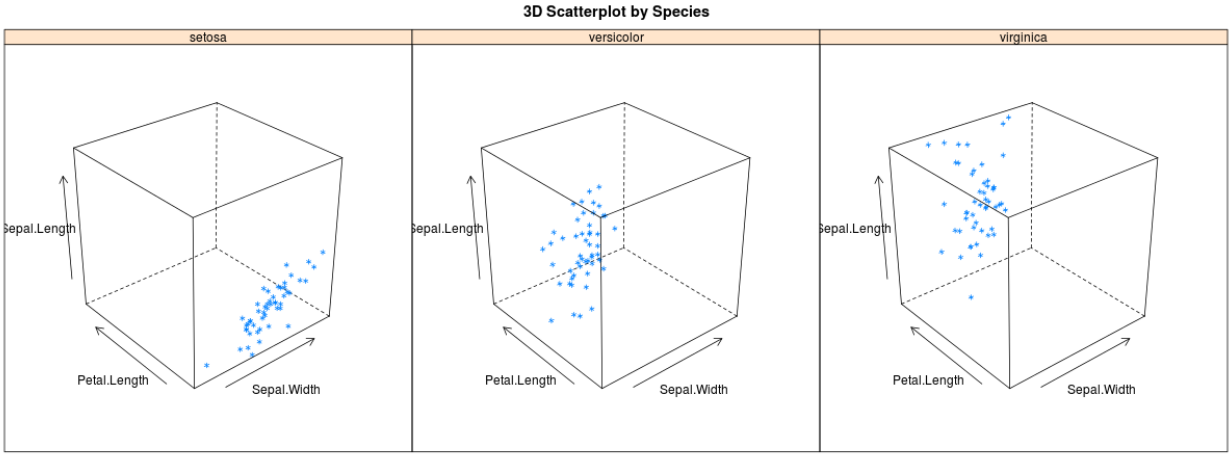
Note: When we interchange the graph axes, you should see graphs with the respective code how we pass axis labels using xlab, ylab, and the graph title using Main and color using the col parameter.

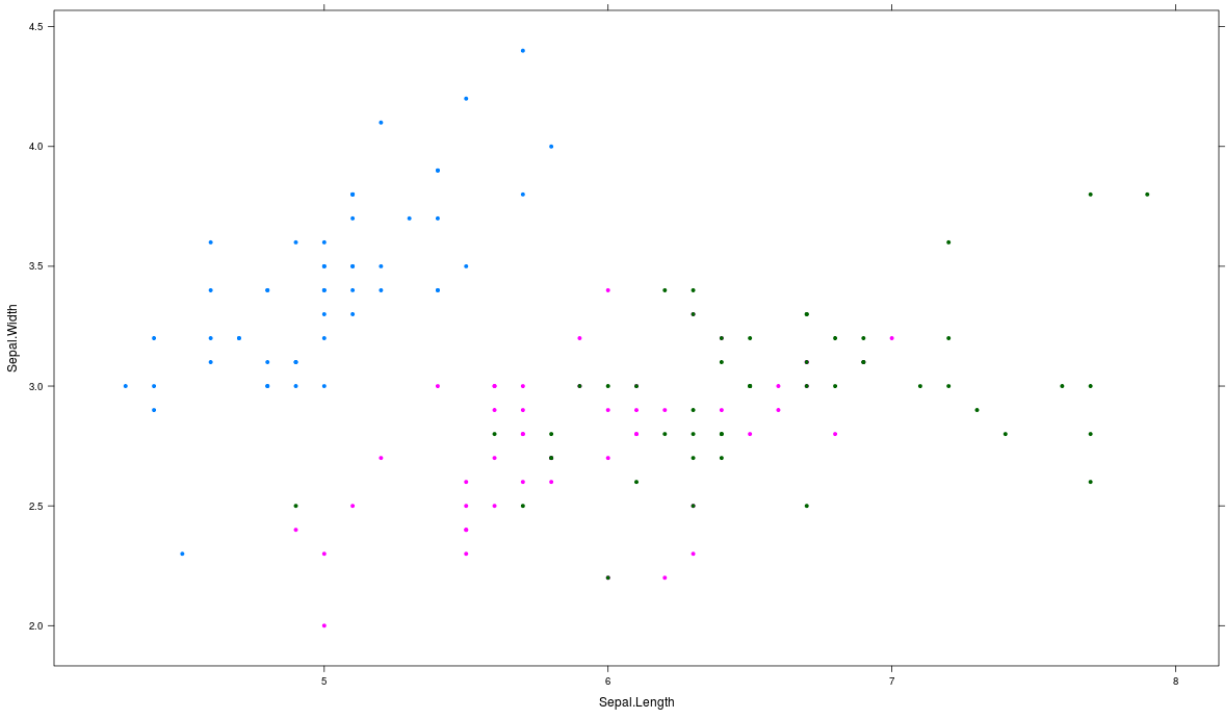
```
>data(iris, package="datasets")
>scatter3d(Petal.Width~Petal.Length+Sepal.Length|Species,
data=iris, fit="linear"
>residuals=TRUE, parallel=FALSE, bg="black", axis.scales=TRUE,
grid=TRUE, ellipsoid=FALSE)
```



You can also make 3D graphs using Lattice package . Lattice can also be used for xyplots. Here's the code:

```
>attach(iris)# 3d scatterplot by factor level
>cloud(Sepal.Length~Sepal.Width*Petal.Length|Species, main="3D
Scatterplot by Species")
>xyplot(Sepal.Width ~ Sepal.Length, iris, groups = iris$Species,
pch= 20)
```

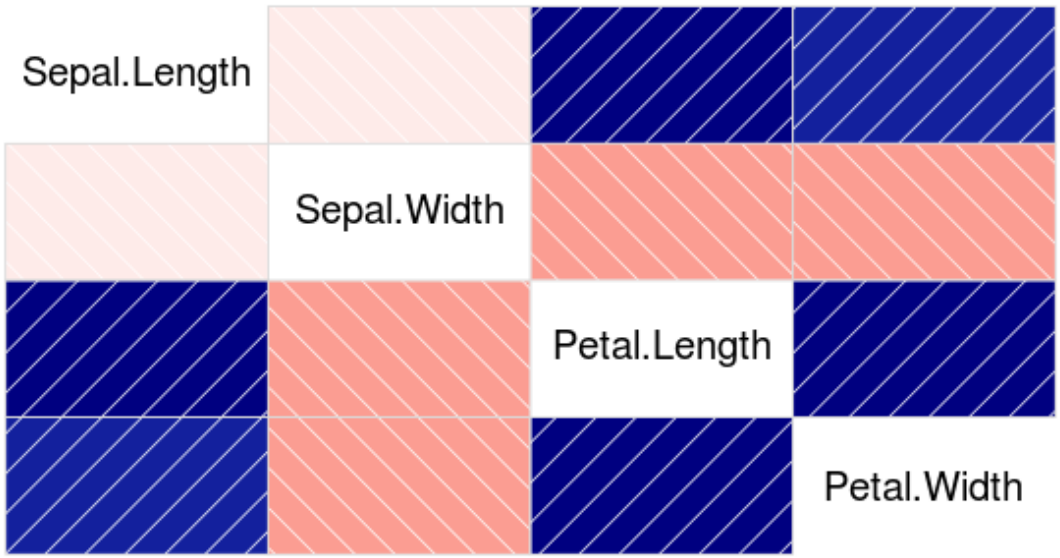




Correlogram (GUIs)

Correlogram help us visualize the data in correlation matrices. Here's the code:

```
> cor(iris[1:4])
 Sepal.Length Sepal.Width Petal.Length Petal.Width
 Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
 Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
 Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
 Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
 > corrgram(iris)
```



Simulation

Simulation is a method used to examine the "what if" without having *real* data. We just make it up! We can use pre-programmed functions in R to simulate data from different probability distributions or we can design our own functions to simulate data from distributions not available in R.

Normal Distribution

Profiling

profiling attempts to determine the validity of groups or distributions of values in a particular record field.

Individual Value Profiling: Syntactic Profilin
Profiling for syntactic constraints involves simply checking whether data values are in (or not in) the set of permissible values. At the specific example levels, a good choice for understanding syntactic type profiling is a random subset of values that satisfy the syntactic constraints along with a random subset of values that do not satisfy the syntactic constraints. From these examples, you often can make reasonable decisions as to whether a different syntactic type might be a better fit for a set of data

Individual Value Profiling: Semantic Profiling
Semantic type constraints correspond to the meaning or interpretation of field values: values are valid if their interpretations satisfy the constraints. For example, suppose that a dataset has a field corresponding to the biological age, in years, of a customer. For some customers, suppose that we don't have a reported age, and the age field for the records corresponding to these customers contains the value -1. Although syntactically not a valid age in years, semantically this value can be interpreted as a Boolean indicator for whether or not the customer reported their age. Deriving a new field corresponding to reported age might be the variable we need to analyze how customers' willingness to report information predicts overall satisfaction.

Profiling semantic type constraints often requires deriving a new record field that explicitly encodes the semantic interpretation of a source field. This explicit encoding can then be syntactically typed, where validity can be determined by testing whether the value is in the valid set. Thus, as with syntactic types, the most appropriate summary statistics correspond to the percentage of field values that are valid, invalid, and empty/null

Set-Based Profiling

Set-based profiling focuses on the shape and extent of the distribution of values found within a single record field or in the range of relationships between multiple record fields. For numeric fields, distributional profiling often builds from a simple histogram of the set of values and might involve comparing that histogram against a known distribution like a Poisson or Gaussian probability distribution.

Statistical Analysis with R

DESCRIPTIVE STATISTICS IN R

a. Write a R script to find basic descriptive statistics using summary, str, quartile function

on mtcars & cars datasets.

b.Write a R script to find subset of dataset by using subset (), aggregate () functions on iris dataset.

Descriptive statistics is a branch of statistics that deals with the summary and analysis of data. R is a powerful tool for descriptive statistics, as it provides a variety of built-in functions and packages for data analysis.

A) Write a R script to find basic descriptive statistics using summary, str, quartile function on mtcars & cars datasets.

1. The summary() function provides a summary of the distribution of each variable in the dataset. Here's an example using the mtcars dataset:

```
# Load the mtcars dataset

data(mtcars)


# View the first few rows of the dataset

head(mtcars)


# Use the summary function to view basic descriptive statistics

summary(mtcars)
```

Output:

```
      mpg             cyl             disp             hp
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
```

```
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
 Median :19.20   Median :6.000   Median :196.3   Median :123.0
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
       drat             wt             qsec             vs
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
       am              gear            carb
 Min.   :0.0000   Min.   :3.000   Min.   :1.000
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
 Median :0.0000   Median :4.000   Median :2.000
 Mean   :0.4062   Mean   :3.688   Mean   :2.812
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

R script that finds basic descriptive statistics using summary, str, and quartile functions on the mtcars and cars datasets:

```
# Load the mtcars dataset
data(mtcars)
```

```
# View the first few rows of the dataset
head(mtcars)

# Use the summary function to view basic descriptive statistics
summary(mtcars)

# Use the str function to view the structure of the dataset
str(mtcars)

# Use the quantile function to view the quartiles of the dataset
quantile(mtcars)

# Load the cars dataset
data(cars)

# View the first few rows of the dataset
head(cars)

# Use the summary function to view basic descriptive statistics
summary(cars)

# Use the str function to view the structure of the dataset
str(cars)
```

Use the quantile function to view the quartiles of the dataset

quantile(cars)

The summary() function provides a summary of the distribution of each variable in the dataset, while the str() function provides information about the structure of the dataset. The quantile() function is used to view the quartiles of the dataset.

```
      mpg              cyl              disp              hp
Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
Median :19.20   Median :6.000   Median :196.3   Median :123.0
Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
      drat             wt             qsec             vs
Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
Median :3.695   Median :3.325   Median :17.71   Median :0.0000
Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
      am              gear            carb
Min.   :0.0000   Min.   :3.000   Min.   :1.000
1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
```

Median :0.0000 Median :4.000 Median :2.000

Mean :0.4062 Mean :3.688 Mean :2.812

3rd Qu.:1.0000 3rd Qu.:

Visualization tools in R (Ggplot , Lattice)

VISUALIZATIONS

a. Find the data distributions using box and scatter plot.

b. Find the outliers using plot.

Find the data distributions using box and scatter plot:

To create a box plot and a scatter plot in R, we can use the ggplot2 package. Here's an example:

1.Install and load the ggplot2 package:

```
install.packages("ggplot2")

library(ggplot2)
```

2.Create a data frame:

```
data <- data.frame(x = rnorm(100, 0, 1), y = rnorm(100, 0, 1))
```

Here, we create a data frame with two columns x and y, each containing 100 random numbers generated from a normal distribution with mean 0 and standard deviation 1.

3.Create a box plot:

```
ggplot(data, aes(y = x)) +
  geom_boxplot()
```

This will create a box plot of the x column.

4. Create a scatter plot:

```
ggplot(data, aes(x = x, y = y)) +
  geom_point()
```

This will create a scatter plot of the x and y columns.

Find the outliers using plot:

To find the outliers in a data set using a plot in R, we can use the boxplot() function. Here's an example:

1.Create a data frame:

```
data <- data.frame(x = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20))
```

Here, we create a data frame with one column x, containing 11 numbers, where the last number (20) is an outlier.

2.Create a box plot:

```
boxplot(data$x)
```

This will create a box plot of the x column.

3.Identify the outliers:

In the box plot, the outliers are represented as individual points outside the whiskers. In this example, we can see that the value 20 is an outlier.

Alternatively, we can use the boxplot.stats() function to programmatically identify the outliers:

```
stats <- boxplot.stats(data$x)
outliers <- stats$out
```

Here, stats$out returns the outliers of the x column.

We can then visualize the outliers by adding them as points to the box plot:

```
boxplot(data$x)
```

```
points(rep(1, length(outliers)), outliers, pch = 19)
```

This will add the outliers as points to the box plot.