ASDM Workshop 1: Basic Statistics and Data Visualization with R

- Charith Silva, Dr.Mo Saraee

Part 1: Basic Statistics with R

R has a rich set of functions.

The functions which are already created or defined in the programming framework are known as a built-in function. R provides the various built-in mathematical and statistical functions to perform the mathematical calculation.

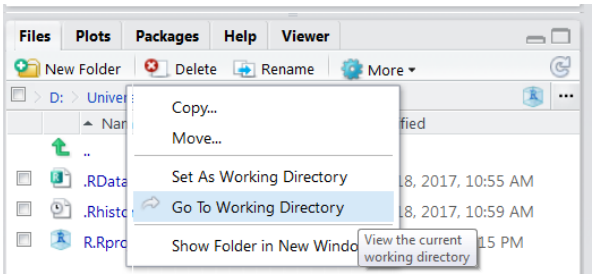
You may work through the workshop as follows:

1. Start the RStudio.

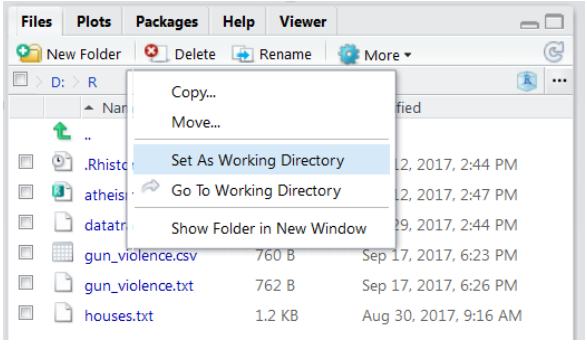
2. Change the working directory

   `File` → `More` → `Go To Working Directory…`

   In the **`Go To Working Directory`** dialogue, navigate to and select the folder where you saved your data file eg: `F:\ASDM\Week1`. Click **OK.**

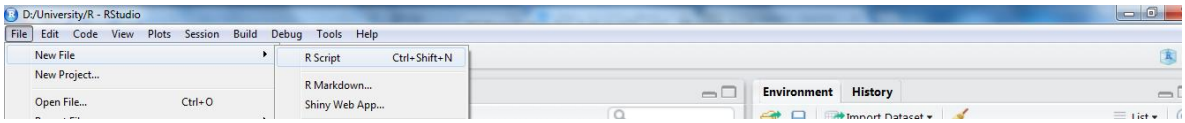


Then **`Set As Working Directory`**



3. Open a new R script window:

   `File → New File → R script`



4. Create a dataset.

```
x <- c(4, 8, 23, 2, 16, 7)
```

or

```
x = c(4, 8, 23, 2, 16, 7)
```

5. R has various built-in functions for calculating simple descriptive statistics from a vector of data. A vector is a variable with one or more values of the same type.

```
mean(x) # arithmetic mean
median(x) # middle value
length(x) # number of elements in a vector or list
range(x) # largest and smallest value
sd(x) # standard deviation
var(x) # variance
```

```
> mean(x) # arithmetic mean
[1] 10
> median(x) # middle value
[1] 7.5
> length(x) # number of elements in a vector or list
[1] 6
> range(x) # largest and smallest value
[1]  2 23
> sd(x) # standard deviation
[1] 7.974961
> var(x) # variance
[1] 63.6
```

6. To get help with any function, use the help() function or the ? operator, along with the name of the function.

```
help(max)
?max
```

7. Quick way to explore data is to use the summary() function. This function gives you a number of descriptive statistics for each continuous variable (range, quartiles, mean, median)

```
summary(x)
```

Part 2: Round Numbers in R

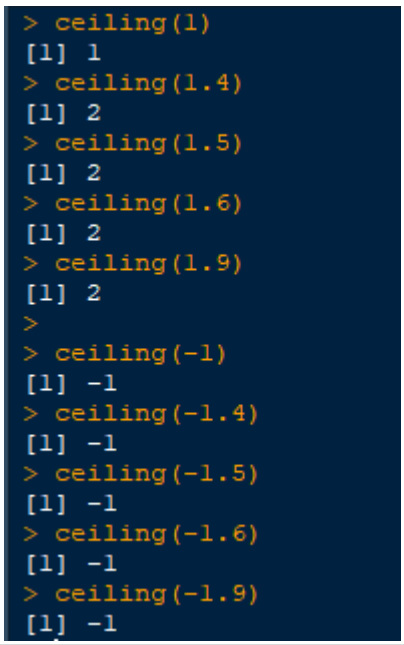
R is more than just a statistical programming language. It's a powerful tool for all kinds of data manipulation. Let's say you want to round a number to the nearest whole number because decimal values are not significant to you. There are several ways to round a numerical value. Rounding a numerical value means replacing it by another value that is approximately equal but has a shorter, simpler, or more explicit representation; Various sorts of rounding can be done easily in R eg: rounding up, rounding down, rounding to the nearest integer.

See the examples below to understand how it works in R.

Round a Single value to a specified number

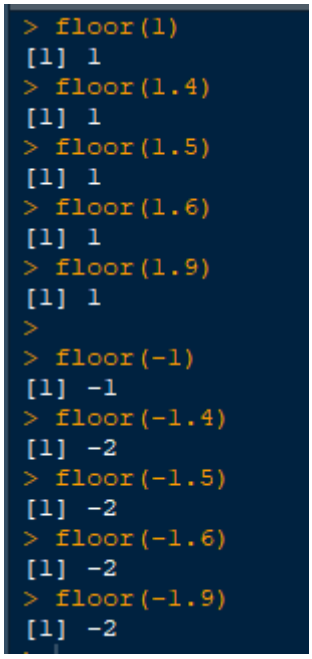
2.1 - **ceiling()** function takes a single numeric argument and returns a value containing the smallest integers not less than the corresponding element.

```
ceiling(1)
ceiling(1.4)
ceiling(1.5)
ceiling(1.6)
ceiling(1.9)

ceiling(-1)
ceiling(-1.4)
ceiling(-1.5)
ceiling(-1.6)
ceiling(-1.9)
```



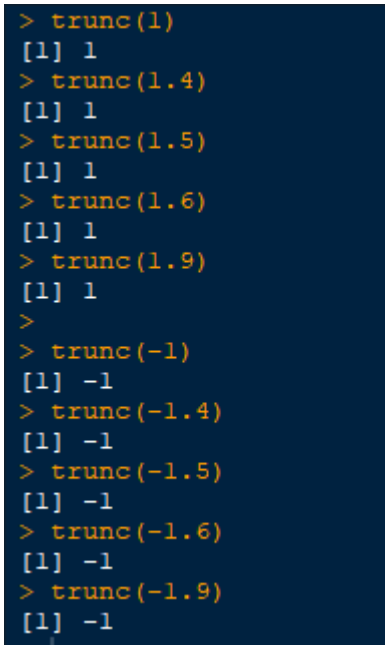
2.2 - **floor()** function takes a single numeric argument and returns a value containing the largest integers not greater than the corresponding element.

```
floor(1)
floor(1.4)
floor(1.5)
floor(1.6)
floor(1.9)

floor(-1)
floor(-1.4)
floor(-1.5)
floor(-1.6)
floor(-1.9)
```



2.3 - **trunc()** function takes a single numeric argument x and returns a value containing the integers formed by truncating the values in x toward 0.

```
trunc(1)
trunc(1.4)
trunc(1.5)
trunc(1.6)
trunc(1.9)

trunc(-1)
trunc(-1.4)
trunc(-1.5)
trunc(-1.6)
trunc(-1.9)
```



2.4 - **round()** function rounds the values in its first argument to the specified number of decimal places (default 0).

```
round(1)
round(1.4)
round(1.5)
round(1.6)
round(1.9)

round(-1)
round(-1.4)
round(-1.5)
round(-1.6)
round(-1.9)
```

```
> round(1)
[1] 1
> round(1.4)
[1] 1
> round(1.5)
[1] 2
> round(1.6)
[1] 2
> round(1.9)
[1] 2
>
> round(-1)
[1] -1
> round(-1.4)
[1] -1
> round(-1.5)
[1] -2
> round(-1.6)
[1] -2
> round(-1.9)
[1] -2
```

```
round(1.949,digits = 0)
round(1.949,digits = 1)
round(1.949,digits = 2)
round(-1.949,digits = 0)
round(-1.949,digits = 1)
round(-1.949,digits = 2)
```

```
> round(1.949,digits = 0)
[1] 2
> round(1.949,digits = 1)
[1] 1.9
> round(1.949,digits = 2)
[1] 1.95
>
> round(-1.949,digits = 0)
[1] -2
> round(-1.949,digits = 1)
[1] -1.9
> round(-1.949,digits = 2)
[1] -1.95
```

2.5 - **signif()** function rounds the values in its first argument to the specified number of significant digits.

```
signif(1.949,digits = 0)
signif(1.949,digits = 1)
signif(1.949,digits = 2)

signif(-1.949,digits = 0)
signif(-1.949,digits = 1)
signif(-1.949,digits = 2)
```

```
> signif(1.949,digits = 0)
[1] 2
> signif(1.949,digits = 1)
[1] 2
> signif(1.949,digits = 2)
[1] 1.9
>
> signif(-1.949,digits = 0)
[1] -2
> signif(-1.949,digits = 1)
[1] -2
> signif(-1.949,digits = 2)
[1] -1.9
```

Part 3: Exploring data graphically

One of the great strengths of R is the graphics capabilities. In most situations you will benefit from first looking at your data graphically. This is to

- Detect any large outliers (for example data entry mistakes)
- Assess what the approximate distribution of the data is
- See the main patterns in the data.

There are many data sets that are automatically available within R, and there are more that are available as part of the packages. You can list the data sets by their names and then load a data set into memory to be used in your statistical analysis.

3.1 - Use **data()** function to list the data sets available in available packages

```
data()
```

```
Data sets in package 'datasets':

AirPassengers                           Monthly Airline Passenger Numbers 1949-1960
BJsales                                 Sales Data with Leading Indicator
BJsales.lead (BJsales)                  Sales Data with Leading Indicator
BOD                                     Biochemical Oxygen Demand
CO2                                     Carbon Dioxide Uptake in Grass Plants
ChickWeight                             Weight versus age of chicks on different diets
DNase                                   Elisa assay of DNase
EuStockMarkets                          Daily Closing Prices of Major European Stock Indices, 1991-1998
Formaldehyde                            Determination of Formaldehyde
HairEyeColor                            Hair and Eye Color of Statistics Students
Harman23.cor                            Harman Example 2.3
Harman74.cor                            Harman Example 7.4
Indometh                                Pharmacokinetics of Indomethacin
InsectSprays                            Effectiveness of Insect Sprays
JohnsonJohnson                          Quarterly Earnings per Johnson & Johnson Share
LakeHuron                               Level of Lake Huron 1875-1972
LifeCycleSavings                        Intercountry Life-Cycle Savings Data
Loblolly                                Growth of Loblolly pine trees
Nile                                    Flow of the River Nile
Orange                                  Growth of Orange Trees
OrchardSprays                           Potency of Orchard Sprays
PlantGrowth                             Results from an Experiment on Plant Growth
Puromycin                               Reaction Velocity of an Enzymatic Reaction
Seatbelts                               Road Casualties in Great Britain 1969-84
Theoph                                  Pharmacokinetics of Theophylline
```

3.2 - **cars** data set contain the speed of cars and the distances taken to stop.

Note that the data were recorded in the 1920s.

```
cars
```

```
> cars
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
```

```
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
33    18   56
34    18   76
35    18   84
36    19   36
37    19   46
38    19   68
39    20   32
40    20   48
41    20   52
42    20   56
43    20   64
44    22   66
45    23   54
46    24   70
47    24   92
48    24   93
49    24  120
50    25   85
```

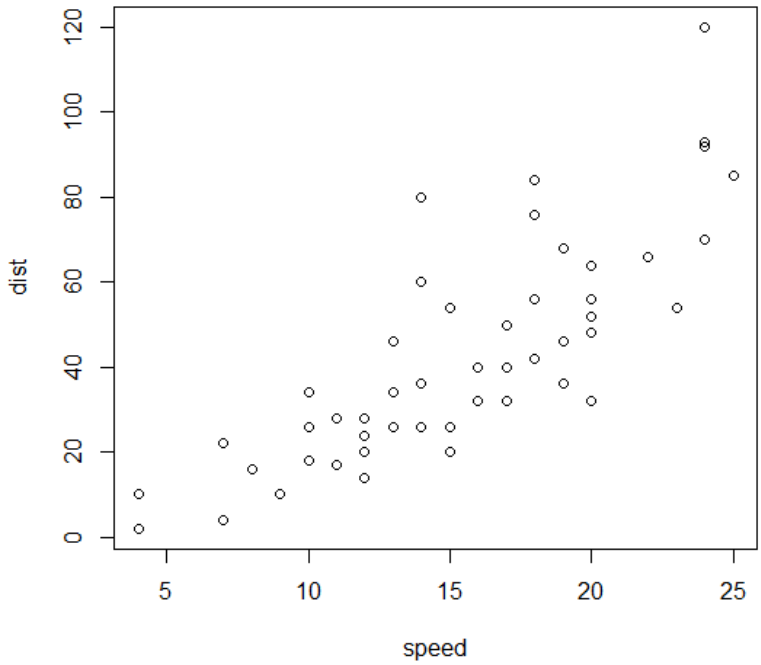
3.3 - Explore cars data is to use the summary() function.

```
summary(cars)
```

```
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
>
```

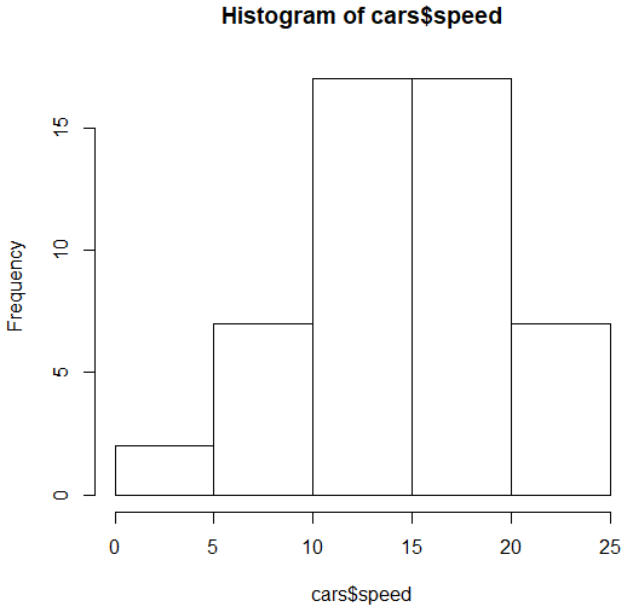
3.4 - A quick graphical check of the data is provided by the simple command plot() that will open a new window displaying plots of all pair-wise variable combinations.

```
plot(cars)
```

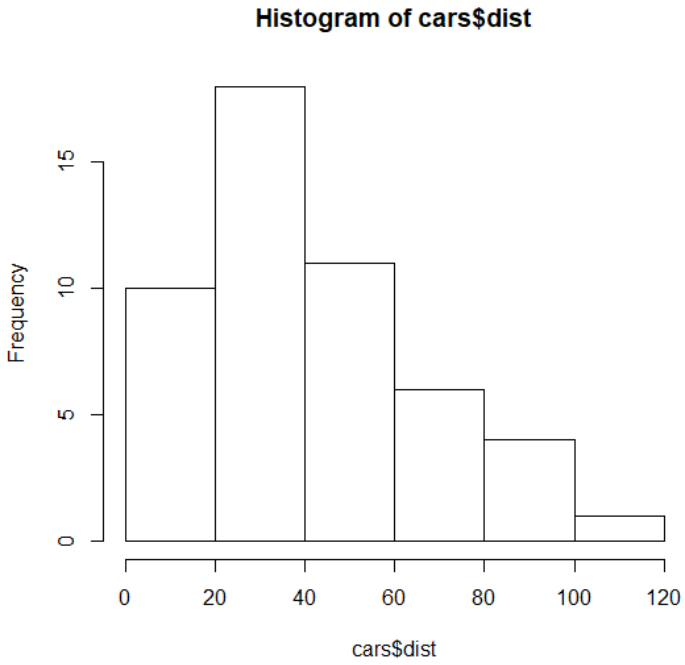


3.5 - The command **`hist()`** produces a histogram displaying data values on the x-axis against their frequencies on the y-axis allowing you to judge the distribution of the data. The command **`hist()`** is applied to individual variables (columns) of the data, that are given by the name of the data frame followed by a dollar sign and the name of the variable (column).

```
hist(cars$speed)
```

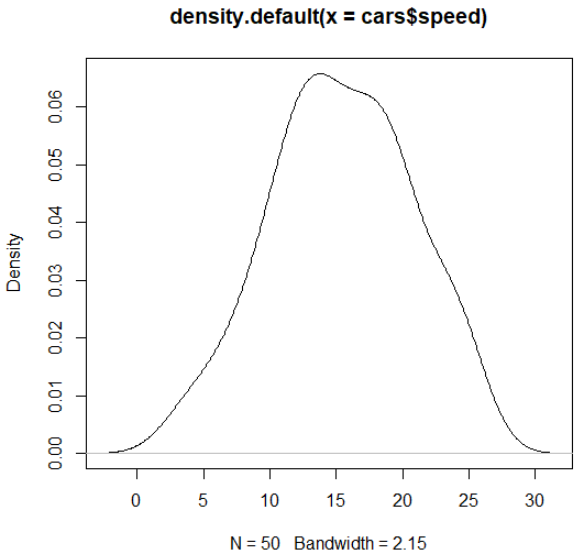


```
hist(cars$dist)
```

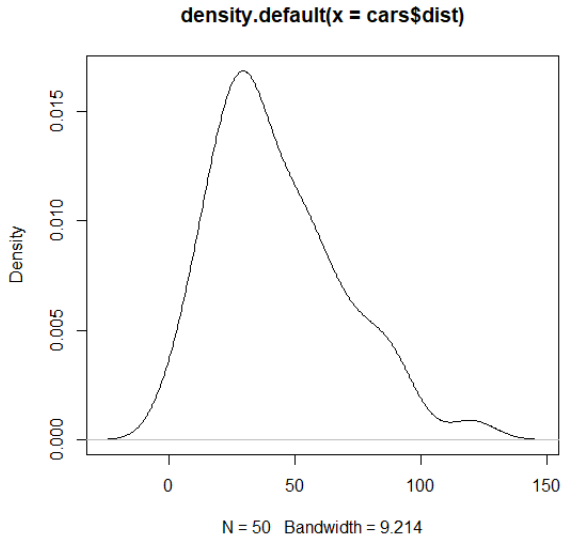


3.6 - Kernal density plots are usually a much more effective way to view the distribution of a variable. Create the plot using plot(density(x)) where x is a numeric vector.

```
plot(density(cars$speed))
```




```
plot(density(cars$dist))
```



Part 4: Data Visualization with R

We will use two external datasets (eg: ***"Thaitourism1.csv"***, ***"Thaitourism2.csv")*** in this part of the workshop***.*** You can download these two datasets from the Week 1 Workshop folder on the Blackboard. Save those two files to a working folder on your **F: drive** (eg: F:\ASDM\Week1).

Foreign tourists entering Thailand make a major contribution to the Thai economy. However, in recent years, the nationalities and number of foreign tourist has changed. This dataset provides a good starting point for identifying the recent trends and for forecasting expected numbers in the near future. However, the tourism industry is highly sensitive to political and environmental events. (source : data.world). Two data sets contain data related to monthly number of tourist visas issued to people separated by region and nationality, for the period of 2010-2016.

1. Import the first data file called "**Thaitourism1.csv**" and create a dataframe called "**Thai_tourist**" to practice the graphic related functions.

```
Thai_tourist <- read.csv("Thaitourism1.csv", header= TRUE)
```

2. Inspect the dataset in R

Once the file has been imported to R we often want to do few things to explore the dataset:

```
names(Thai_tourist)
head(Thai_tourist)
tail(Thai_tourist)
str(Thai_tourist)
summary(Thai_tourist)
```

3. Import the second data file called "**Thaitourism2.csv**" and create a dataframe called "**Thai_tourist_full**".

```
Thai_tourist_full <- read.csv("Thaitourism2.csv", header= TRUE)
```

4. Inspect the dataset in R

```
names(Thai_tourist_full)
head(Thai_tourist_full)
```

```
tail(Thai_tourist_full)
str(Thai_tourist_full)
summary(Thai_tourist_full)
```

5. Filter the **Thai_tourist** data frame and create a new data frame called **Thai_2016** to contain only 2016 data.

```
Thai_2016<-Thai_tourist[Thai_tourist$Year==2016,]
Thai_2016
```

6. Filter the **Thai_tourist_full** data frame and create a new data frame called **Thai_UK** to contain only UK tourists data.

```
Thai_UK<-Thai_tourist_full[Thai_tourist_full$nationality=="UnitedKingdom",]
Thai_UK
```

Visualizing the data via graphics can be important at the beginning stages of data analysis to understand basic properties of the data, to find simple patterns in data, and to suggest possible modeling strategies.

The R language has a number of different in-built plotting functions which draw different styles of plot. All of these functions have many options which can dramatically change the appearance of the plot, and we'll look at these later so even if there isn't a plot which immediately looks the way you want, you can probably find one which can be adapted. The particular plot function you need will depend on the number of variables you want to plot and the pattern you wish to highlight.

- Plots with two variables
- Plots for a single sample
- Multivariate plots

For each plot type there are a range of specific options which can be added to the required arguments to change the appearance of the plot.

In the section below we will go through the different plot types.

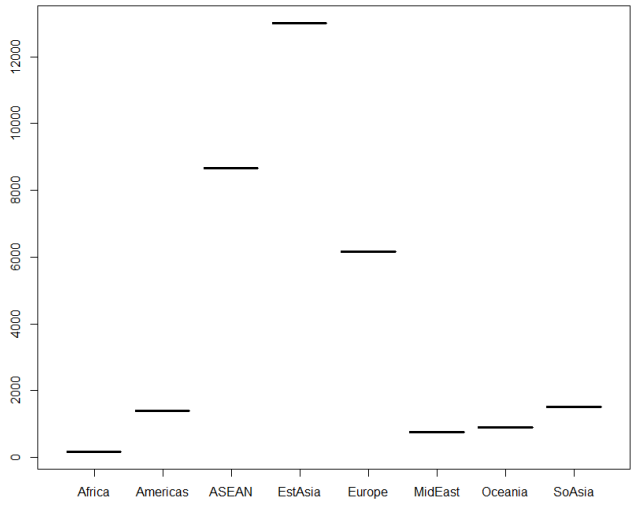
7. **plot() function**

The plot function is the generic x-y plotting chart type. Its default appearance is as a scatterplot, but by changing the way the points are plotted and joined you can make it into a line graph or a point and line graph. With two variables (typically the response variable on the y axis and the explanatory variable on the x axis), the kind of plot you should produce depends upon the nature of your explanatory variable. The common uses of plot you'd pass in 2 vectors of values (x and y).

The **type** option determines what sort of plot you get

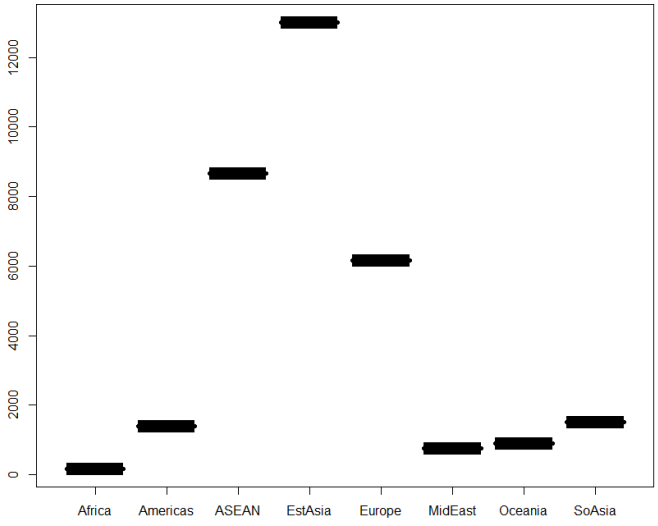
- **type="p"** gives you a scatter plot (the default)

```
plot(Thai_2016$Region,Thai_2016$Tourists_1000s,type="p")
```



- **type="l"** gives you a line plot. For line plots **lwd** sets the line width (thickness)

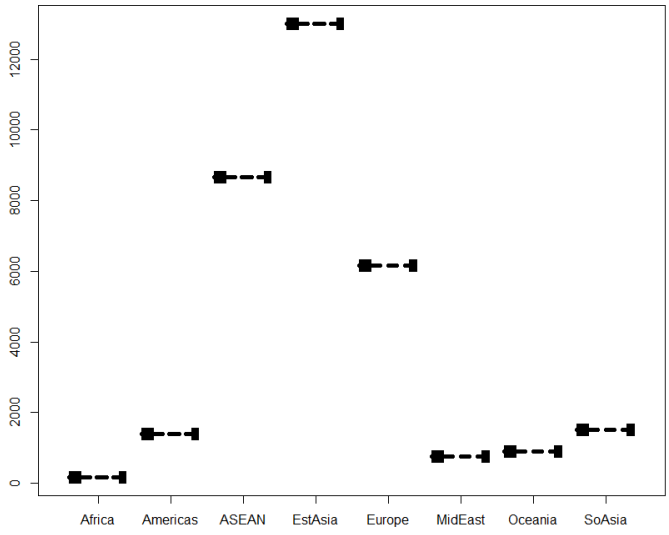
```
plot(Thai_2016$Region,Thai_2016$Tourists_1000s,type="l",lwd=5)
```



- **lty** sets the line type.

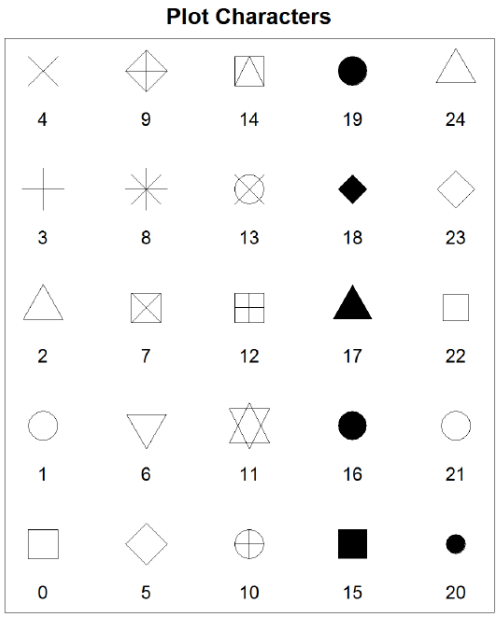
1. Solid line
2. Dashed line
3. Dotted line
4. Dot and dash line
5. Long dash line
6. Long then short dash line

```
plot(Thai_2016$Region,Thai_2016$Tourists_1000s,type="p",lwd=5,lty=3)
```

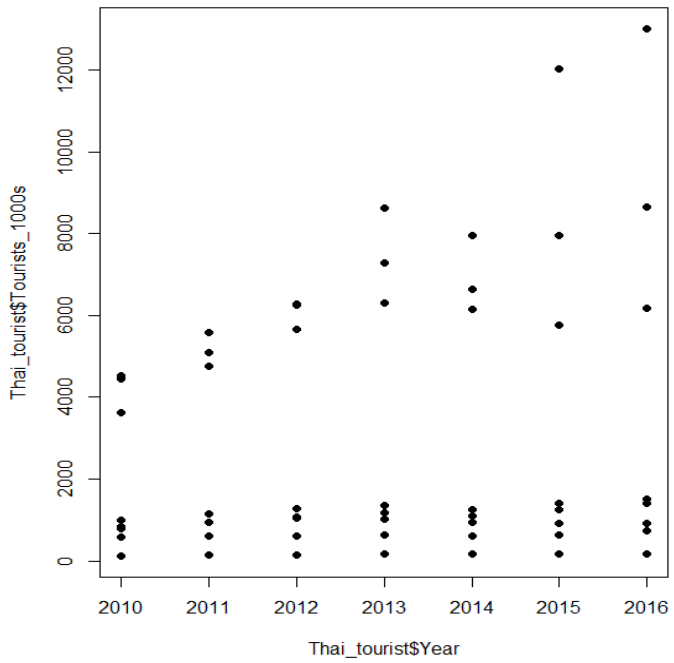


- **Plotting character**

For plots which use a specific character to indicate a data point you can choose from a range of different point shapes to use. These are set using the **pch** (point character) option and take a single number as its value.

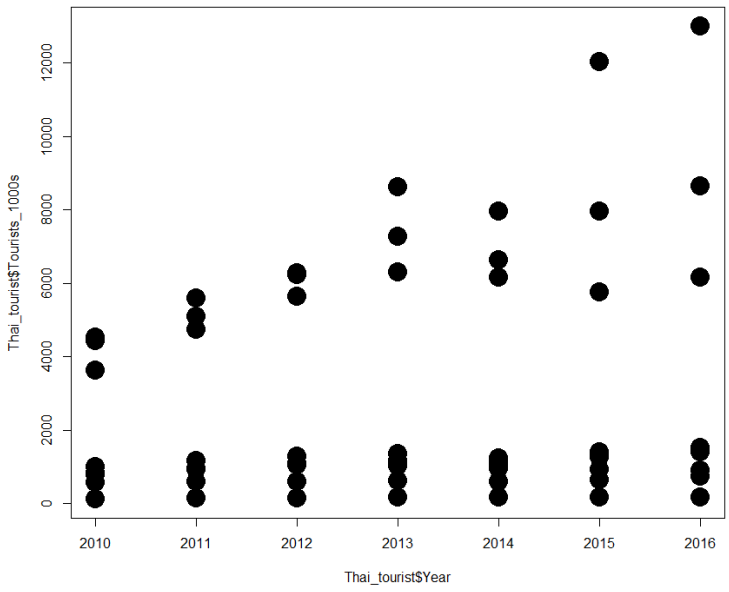


```
plot(Thai_tourist$Year,Thai_tourist$Tourists_1000s,pch = 19)
```



The size of the plot characters can be changed using the **cex** (character expansion) option.

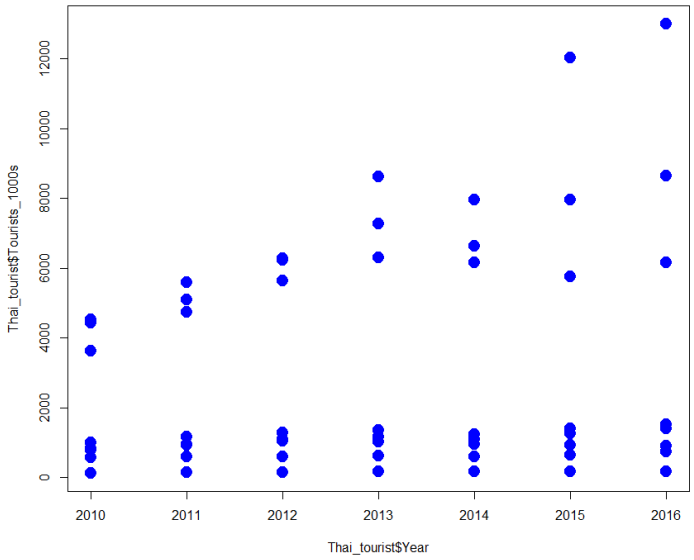
```
plot(Thai_tourist$Year,Thai_tourist$Tourists_1000s,pch = 19,cex=3)
```



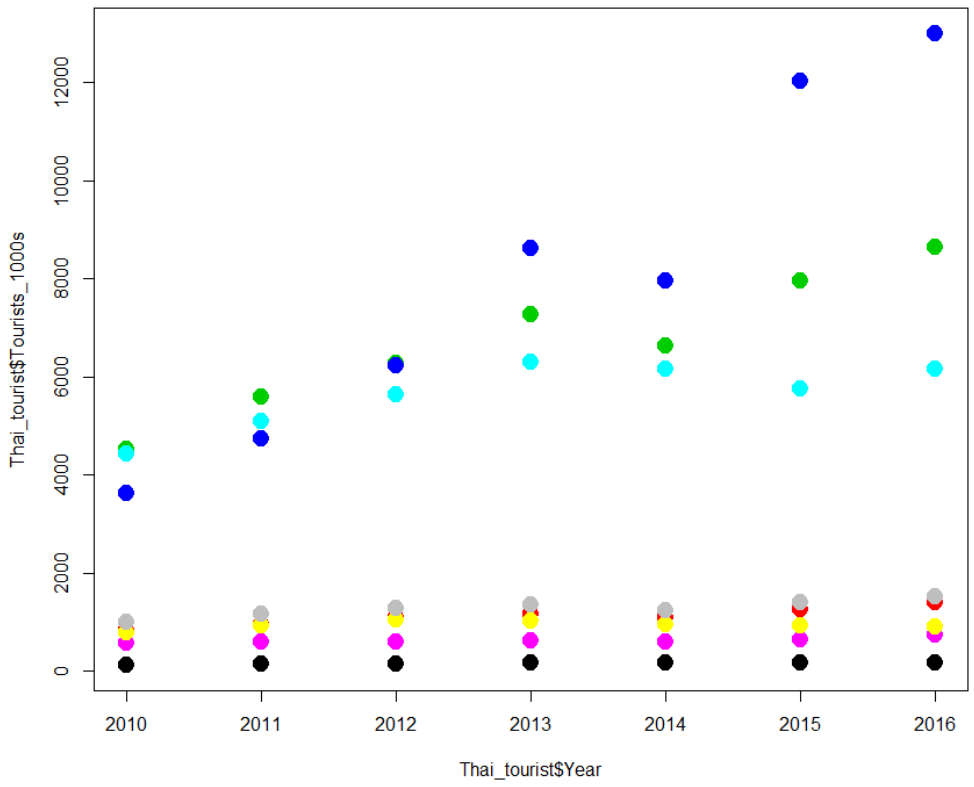
- **Specifying colours**

Colours can be manually specified either through a direct RGB hexadecimal string or through the use of a controlled set of colour names defined within the language.

```
plot(Thai_tourist$Year,Thai_tourist$Tourists_1000s,pch = 19,cex=2,col="blue")
```



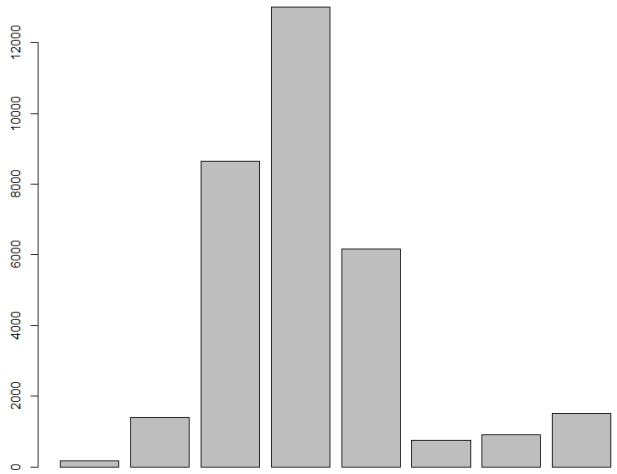

```
plot(Thai_tourist$Year,Thai_tourist$Tourists_1000s,
      pch = 19,cex=2,col=Thai_tourist$Region)
```



8. barplot() function

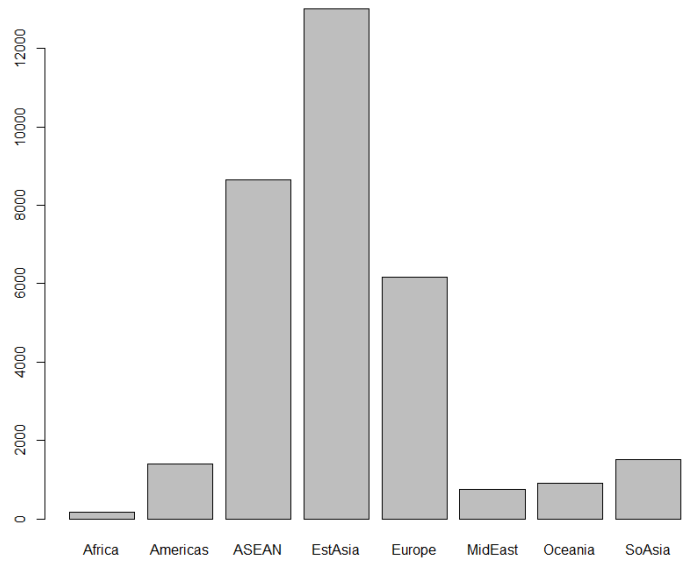
The barplot is useful for summarizing categorical data. By varying the options applied you can create standard bar graphs, stacked bar graphs, or have groups of side by side bars. The graphs can be constructed either horizontally or vertically.

```
barplot(Thai_2016$Tourists_1000s)
```



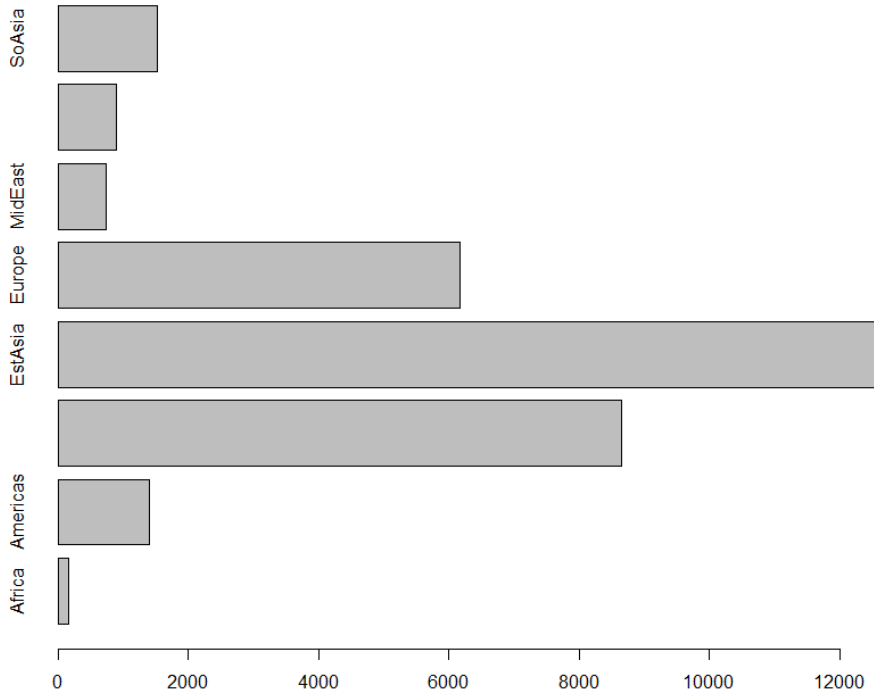
- The **names.arg** option is used to set the category names. These will also be taken from the names associated with the data vector if it has them.

```
barplot(Thai_2016$Tourists_1000s,names.arg=Thai_2016$Region)
```



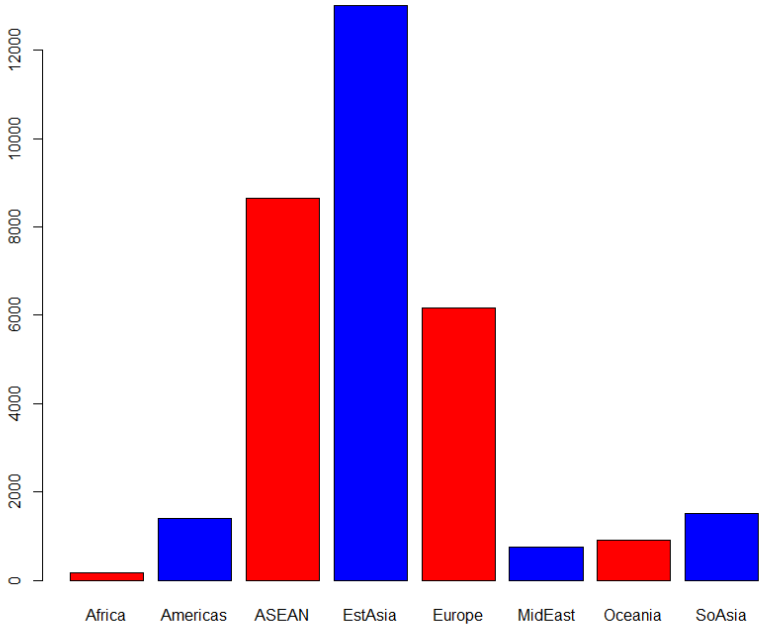
- If you want your bars to run horizontally then you set **horiz** to TRUE.

```
barplot(Thai_2016$Tourists_1000s,names.arg=Thai_2016$Region,horiz= TRUE)
```



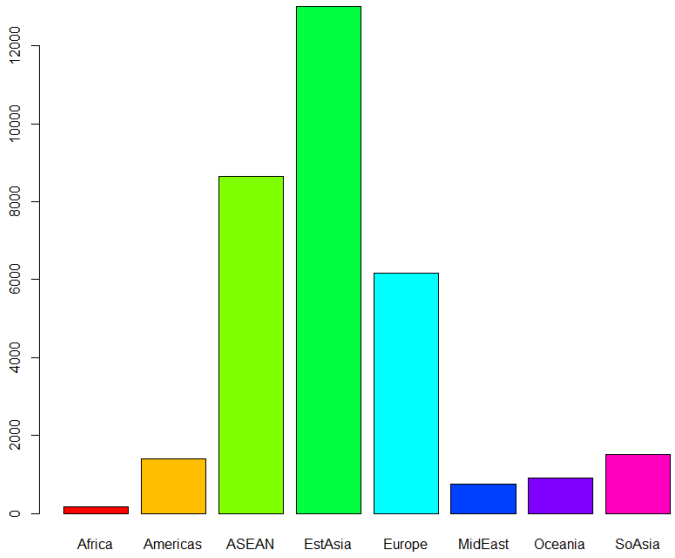
- **Colours** can be specified either by name (e.g **col = "red"**) or as a hexadecimal RGB triplet (such as **col = "#FFCC00"**).

```
barplot(Thai_2016$Tourists_1000s,names.arg=Thai_2016$Region,col=c("red","blue"))
```



rainbow() function can be used to create a vector of n contiguous colours.

```
barplot(Thai_2016$Tourists_1000s,names.arg=Thai_2016$Region,col=rainbow(8))
```

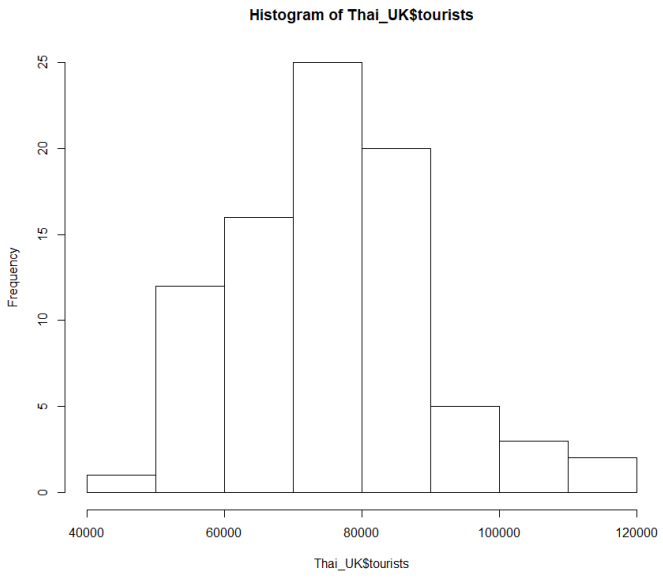


9. hist() function

A histogram is useful to look at when we want to see more detail on the full distribution of the data. The hist function draws histograms which bin your data and then count the occurrences of data points within each bin. It is a natural way to summarise the distribution of a single continuous dataset.

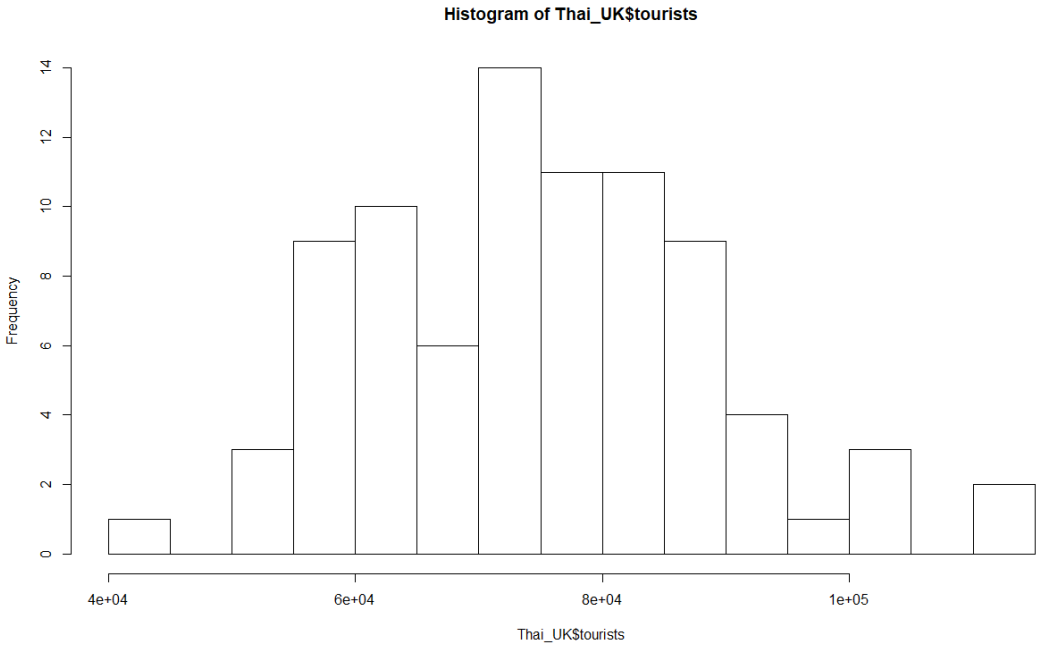
- The **hist** function only takes a single vector of values for its data.

```
hist(Thai_UK$tourists)
```



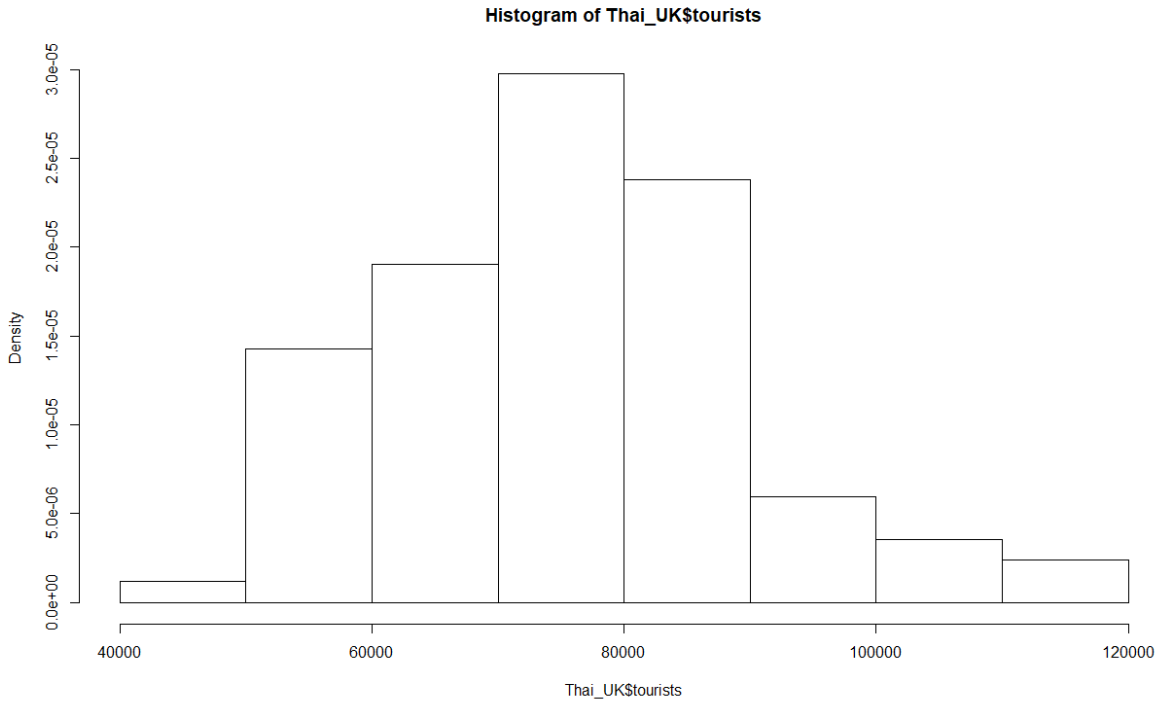
- To control the number of categories into which the data is broken you use the **breaks** argument. Most simply this is a single value indicating the total number of categories

```
hist(Thai_UK$tourists,breaks=16)
```



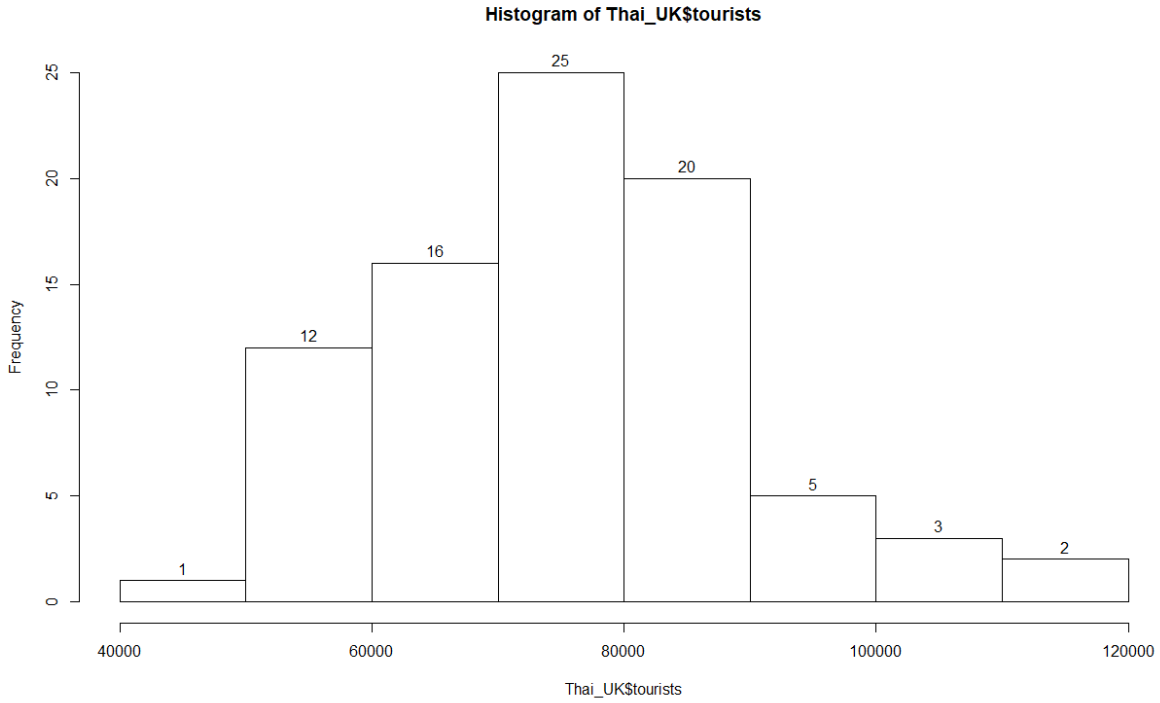
- By default the plot shows the frequency with which data falls into each category, and the y axis is the number of observations. Instead of counting the number of datapoints per bin, R can give the probability densities using the **freq=FALSE** option:

```
hist(Thai_UK$tourists,freq=FALSE)
```



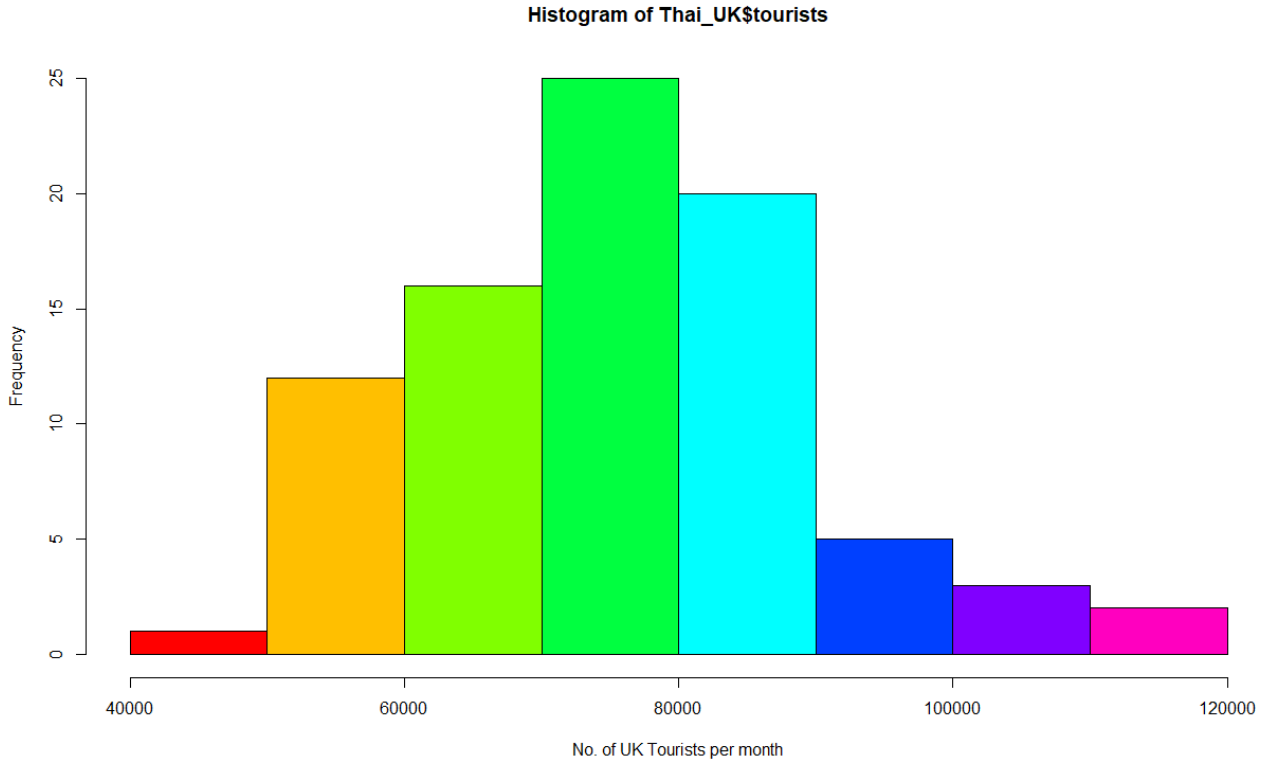
- If you want to put text above each of the bars then you can pass a character vector in the labels argument.

```
hist(Thai_UK$tourists, labels = TRUE)
```



- Axis labels for the x and y axes can be set using the **xlab** and **ylab** options. These

```
hist(Thai_UK$tourists, xlab="No. of UK Tourists per month",col=rainbow(8))
```



8. boxplot()

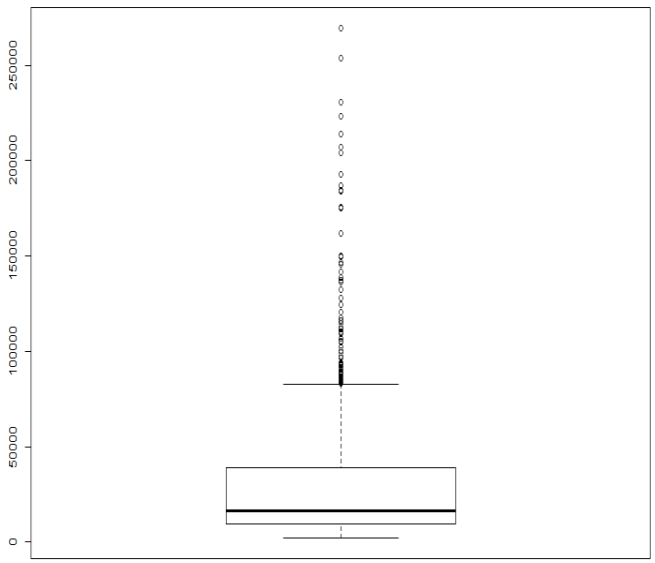
Boxplots are a visual representation of the five-number summary plus a bit more information. In particular, boxplots commonly plot outliers that go beyond the bulk of the data. It plots out the median, interquartile range and makes an estimate of the ends of the distribution and shows individual outliers which fall beyond this. From the boxplot, we can see that there are a few points on both the high and the low end that appear to be outliers.

- Data for boxplot can either be a single numeric vector, a list of numeric vectors or a formula.

```
#below code crates a new dataframe called Thai_Europe to hold only data from Europe Region.

#subset() function return subsets of vectors, matrices or data frames which meet conditions. The subset( ) function is the easiest way to select variables and observations from large dataset.

#droplevels() function is used to drop unused levels from factors in a data frame.

Thai_Europe <-droplevels(subset(Thai_tourist_full, region=="Europe"))
```

- Plotting single numeric vector

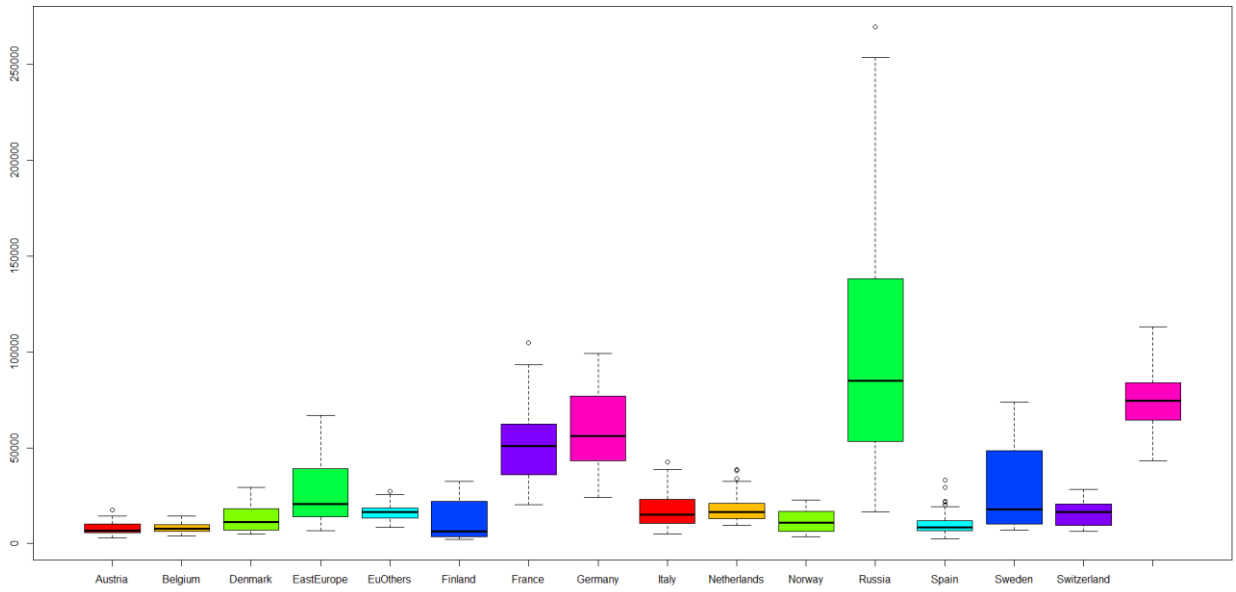
```
boxplot(Thai_Europe$tourists, data=Thai_Europe)
```



- Data is represented with a box
- The ends of the box are at the first and third quartiles, i.e., the height of the box is IQR
- The median is marked by a line within the box
- Whiskers: two lines outside the box extended to Minimum and Maximum
- Outliers: points beyond a specified outlier threshold, plotted individually

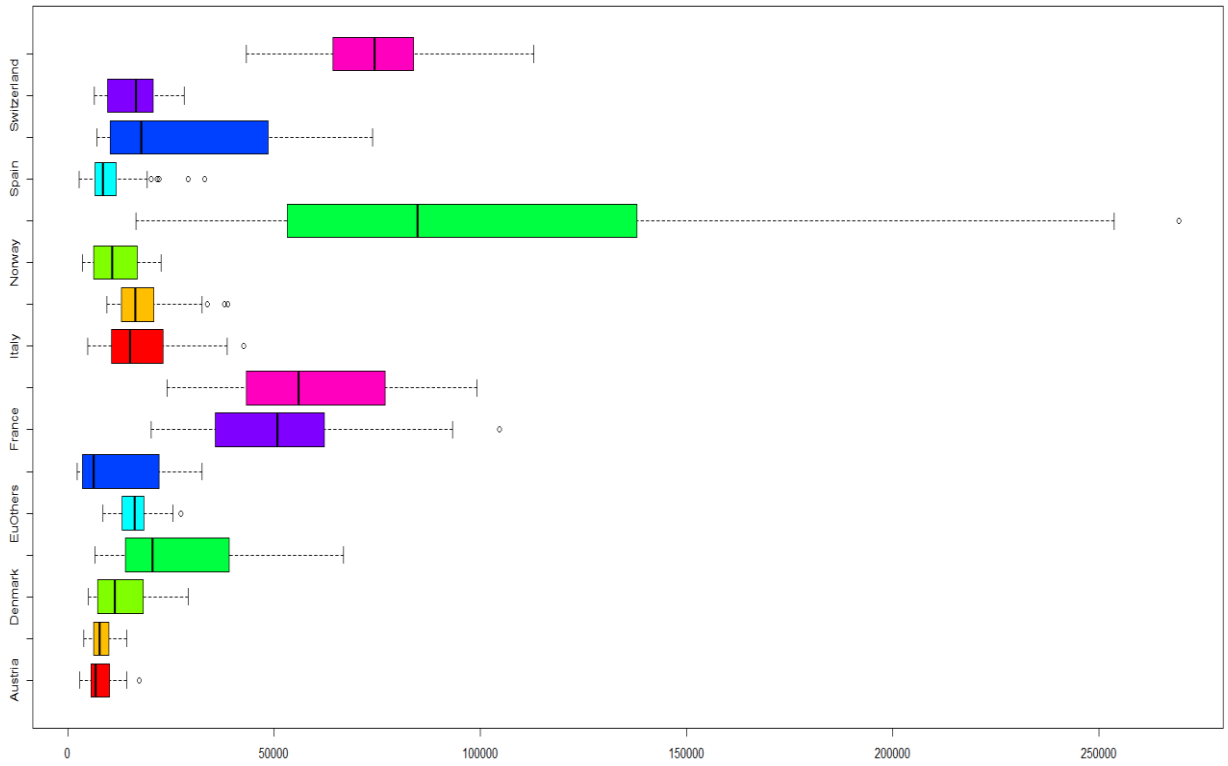
- Plotting boxplot with a formula

```
boxplot(tourists~nationality, data=Thai_Europe,col=rainbow(8))
```



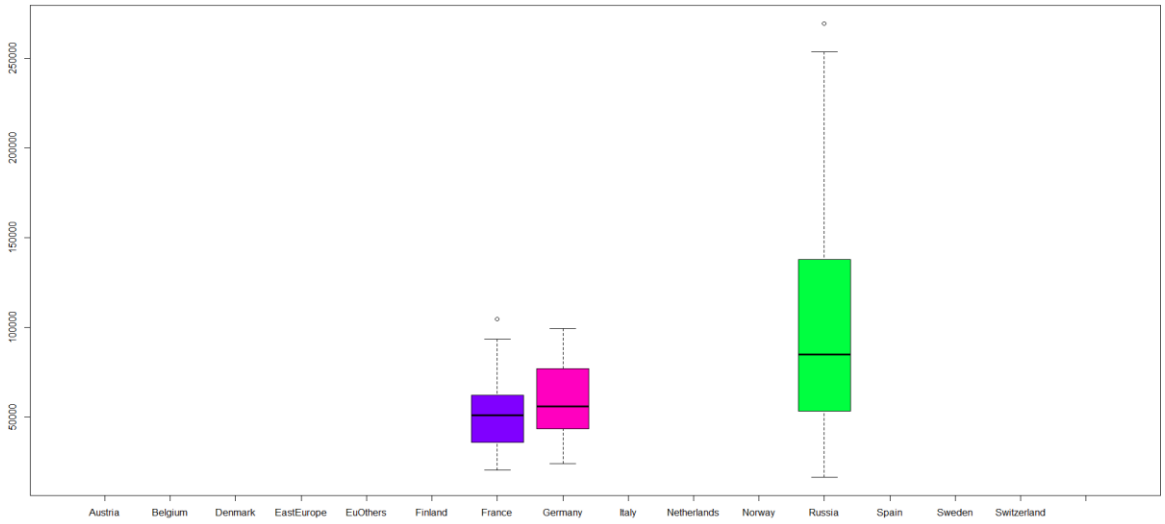
- The boxplot is vertical by default but can be plotted sideways by setting the **horizontal** argument.

```
boxplot(tourists~nationality, data=Thai_Europe,col=rainbow(8),horizontal=TRUE)
```



- **Subset** an optional vector specifying a subset of observations to be used for plotting

```
boxplot(tourists~nationality, data=Thai_Europe,col=rainbow(8),
                subset=nationality %in% c("France","Russia","Germany"))
```

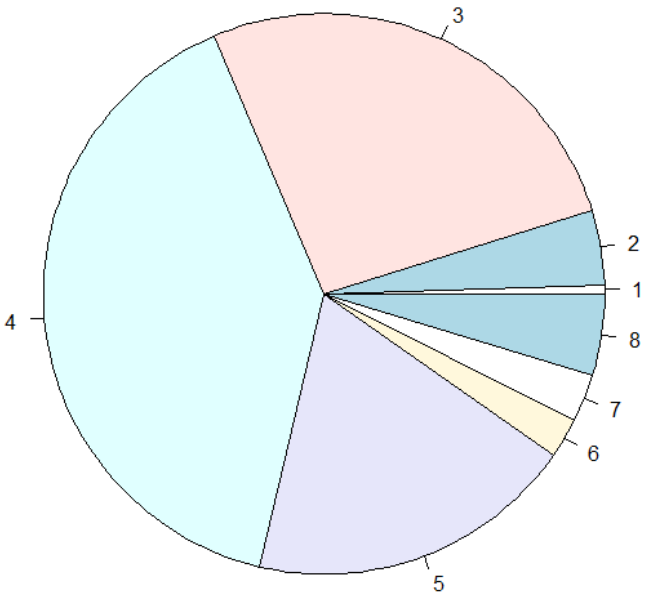


9. pie() function

The pie function draws a pie chart to summarise the categorical division of a dataset into subgroups. The additional parameters are used to control labels, color, title etc.

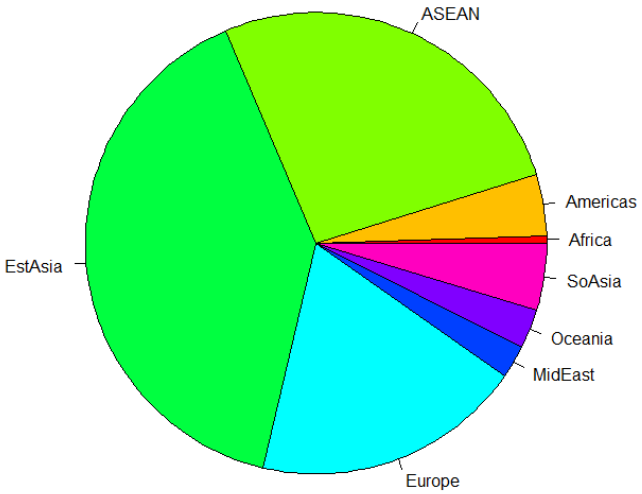
- Data must be a numeric vector of positive values.

```
pie(Thai_2016$Tourists_1000s)
```



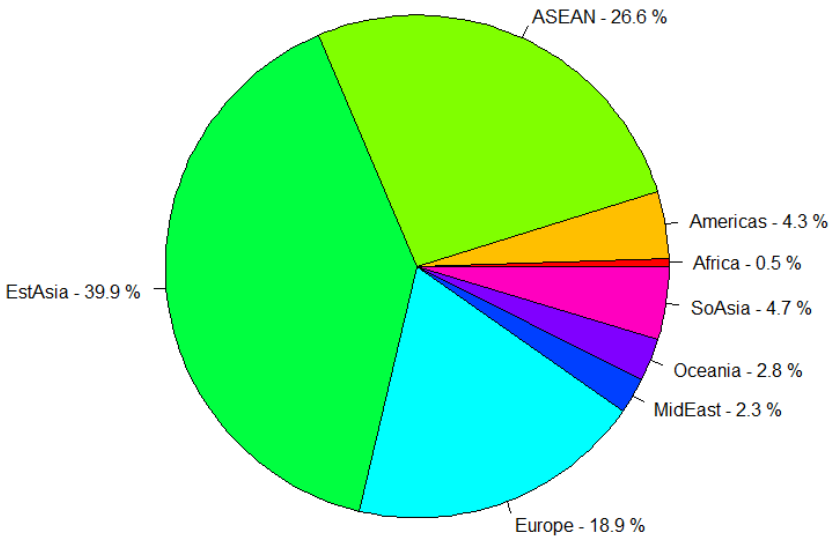
Labels are taken from the data names but can be overridden by the **labels** argument. If the labels are big then you might need to reduce the size of the pie by setting the radius argument to a lower value.

```
pie(Thai_2016$Tourists_1000s,labels =Thai_2016$Region,col=rainbow(8))
```



- Add percentage to the labels

```
percent <- round(100*Thai_2016$Tourists_1000s/sum(Thai_2016$Tourists_1000s), 1)
percent <- paste(Thai_2016$Region, "-",percent,"%") # add percents to labels
# paste0 function is simply concatenates the vector with space separator.

pie(Thai_2016$Tourists_1000s, labels = percent, col = rainbow(8))
```



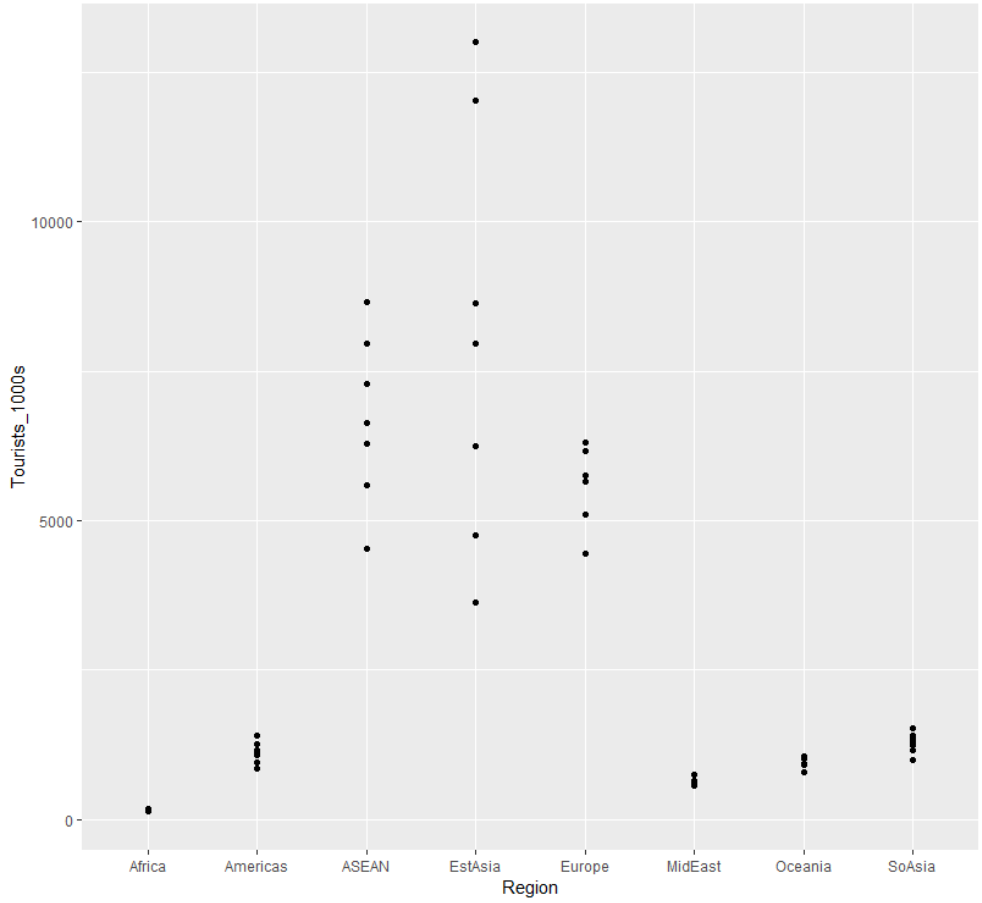
10. ggplot2

The ggplot2 package offers a powerful graphics language for creating elegant and complex plots. Its popularity in the R community has exploded in recent years. The qplot function in ggplot2 is what you use to “quickly get some data on the screen”. It works much like the plot() function in base graphics system. The **qplot()** function can be used to create the most common graph types. While it does not expose **ggplot**'s full power, it can create a very wide range of useful plots. There are additional functions in ggplot2 that allow you to make arbitrarily sophisticated plots.

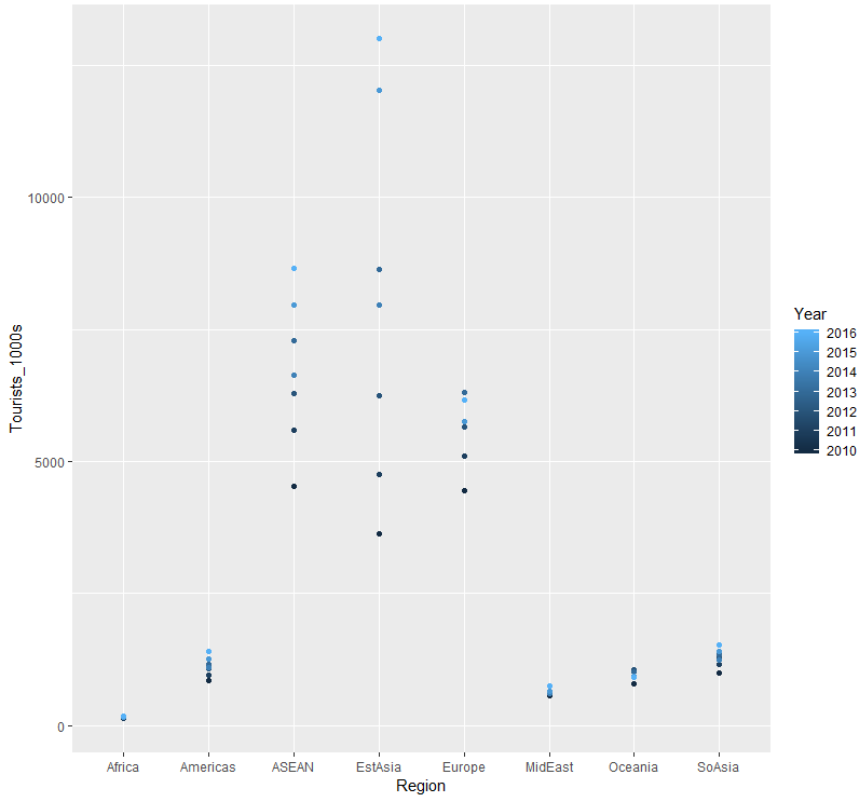
```
#run this line, if you do not have the packages installed
install.packages("ggplot2")

#load libraries into R session
library(ggplot2)
```

- scatterplot with qplot() function

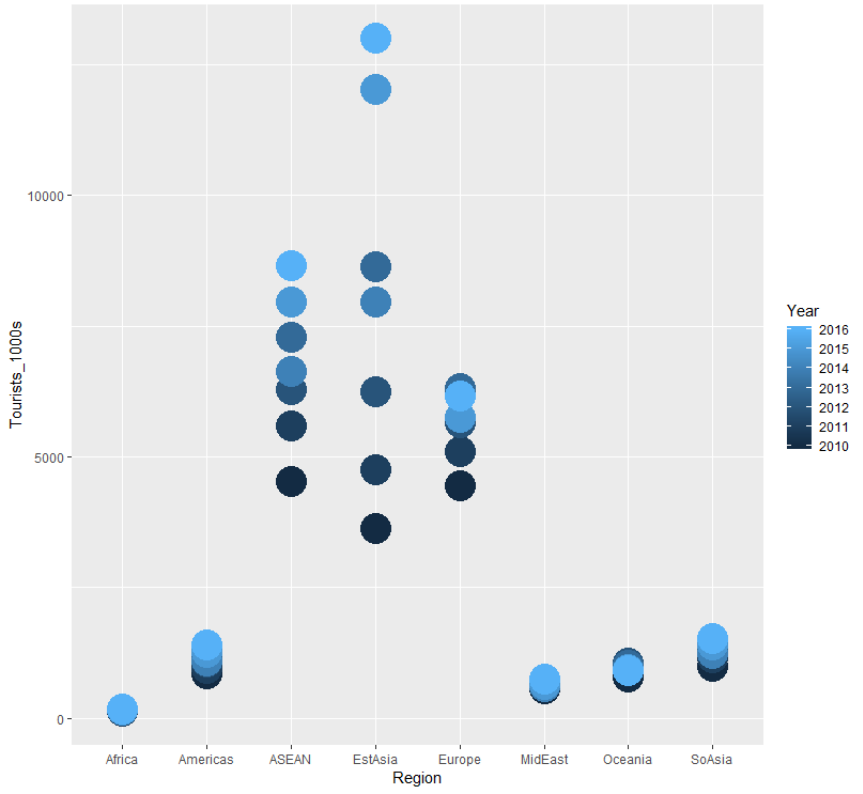
```
qplot(Region,Tourists_1000s,data=Thai_tourist)
```



```
qplot(Region,Tourists_1000s,data=Thai_tourist, color = Year)
```

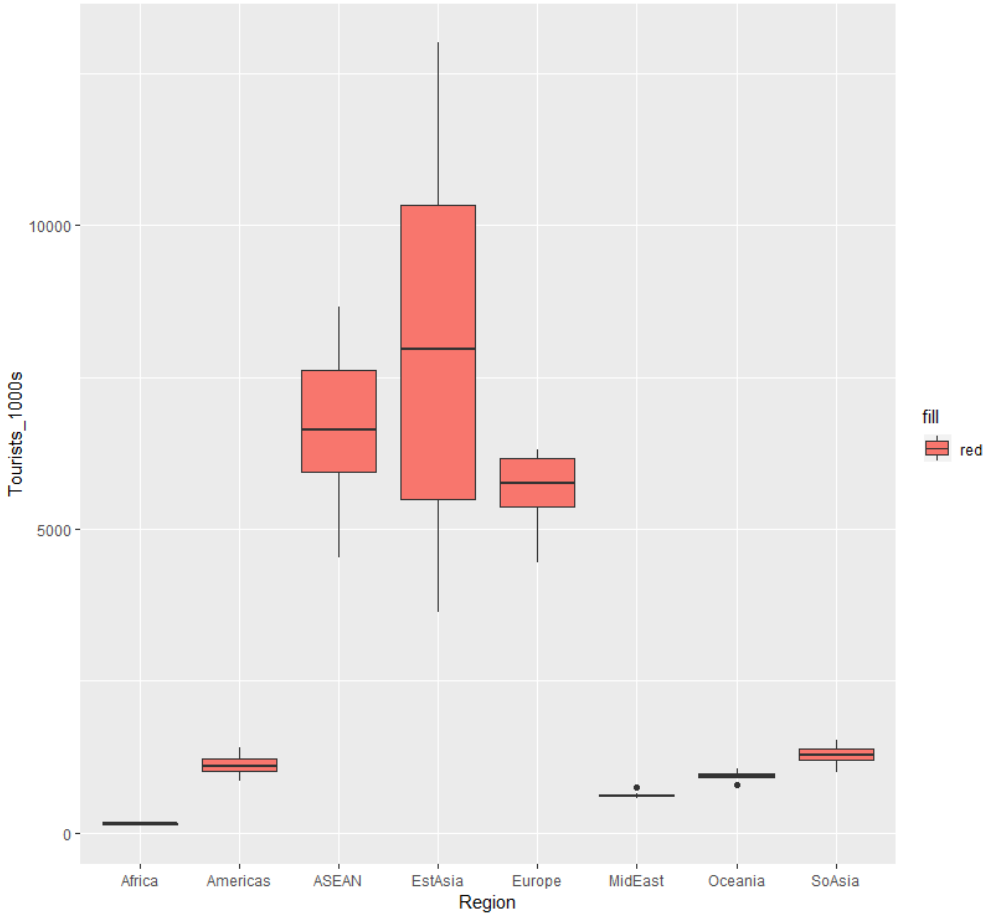



```
qplot(Region,Tourists_1000s,data=Thai_tourist, color = Year, size = I(10))
```



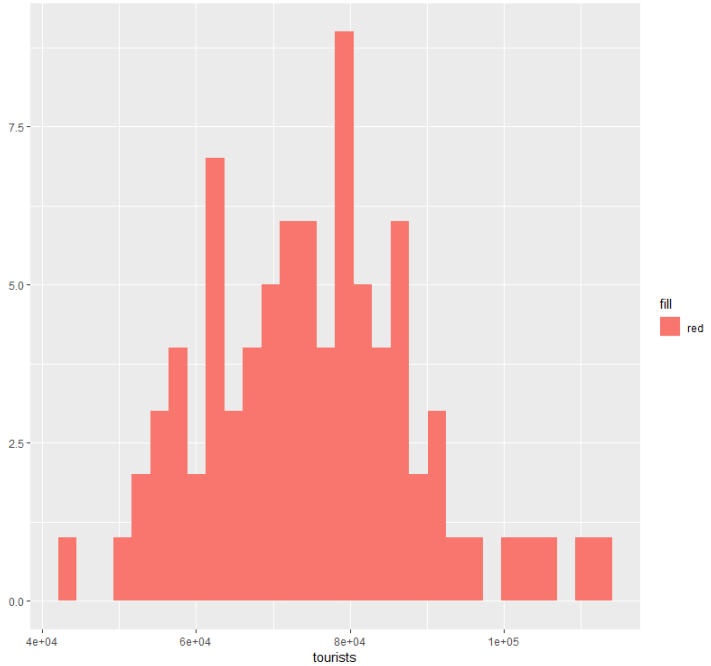
- Boxplot with qplot() function

```
qplot(Region,Tourists_1000s,data=Thai_tourist, geom = "boxplot",fill="red")
```



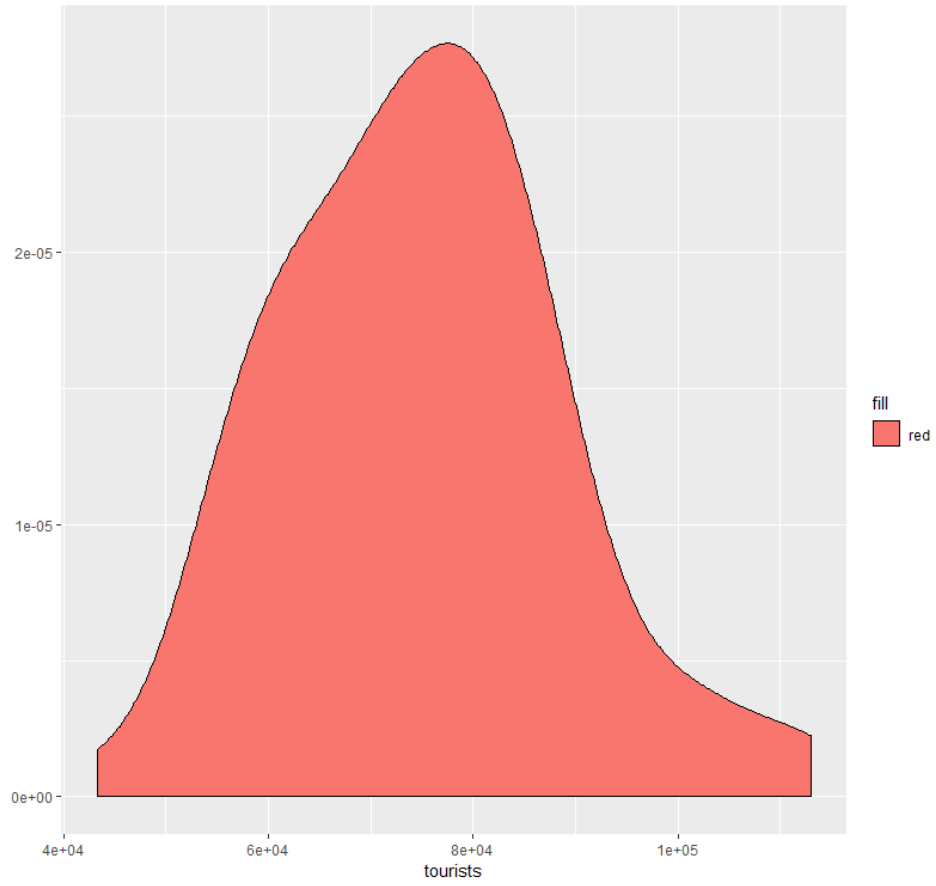
- Barplot with qplot() function

```
qplot(tourists,data=Thai_UK, geom = "histogram",fill="red")
```



- Density plot with qplot() function

```
qplot(tourists,data=Thai_UK, geom = "density",fill="red")
```



Exercise: Use qplot() function to visualise various individual attribute and a pair of attributes from `occupancy_data` data frame you created in Part 1 of this workshop.

Eg:

```
qplot(occupancy_data$Temperature)
qplot(occupancy_data$Temperature, occupancy_data$Humidity)
```