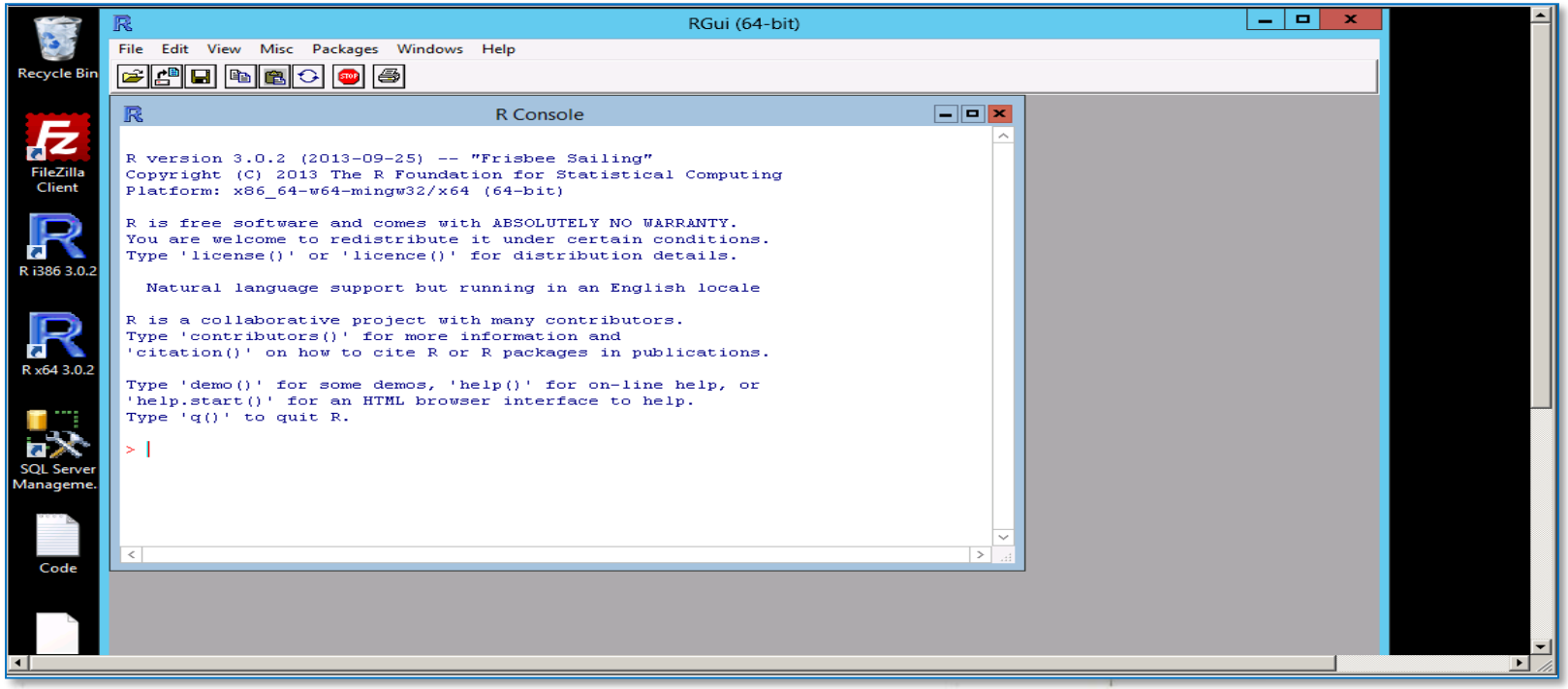
DataInquest

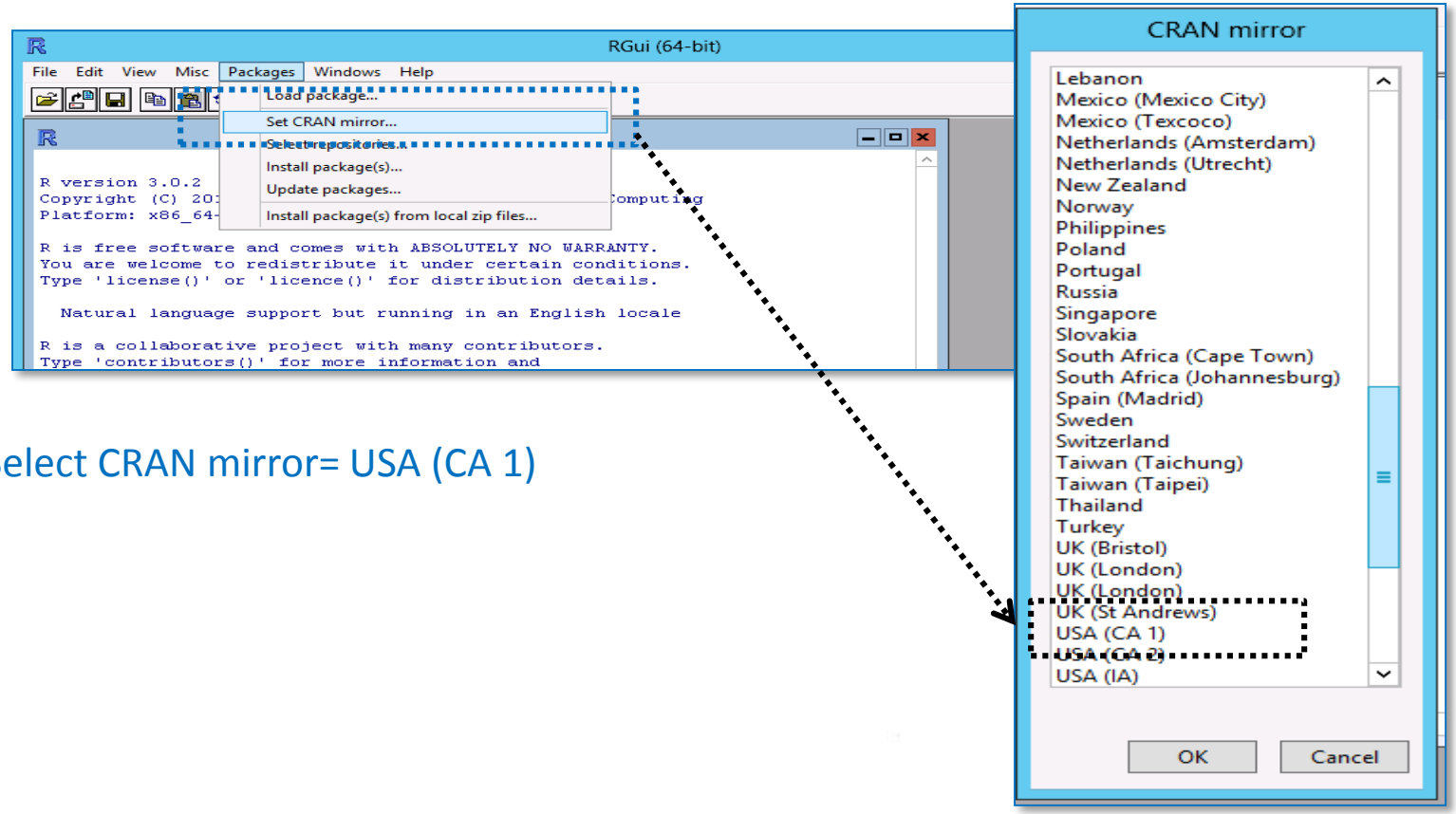
Big Data Technologies and Data Science training for all!!

Data Science

R software landing page

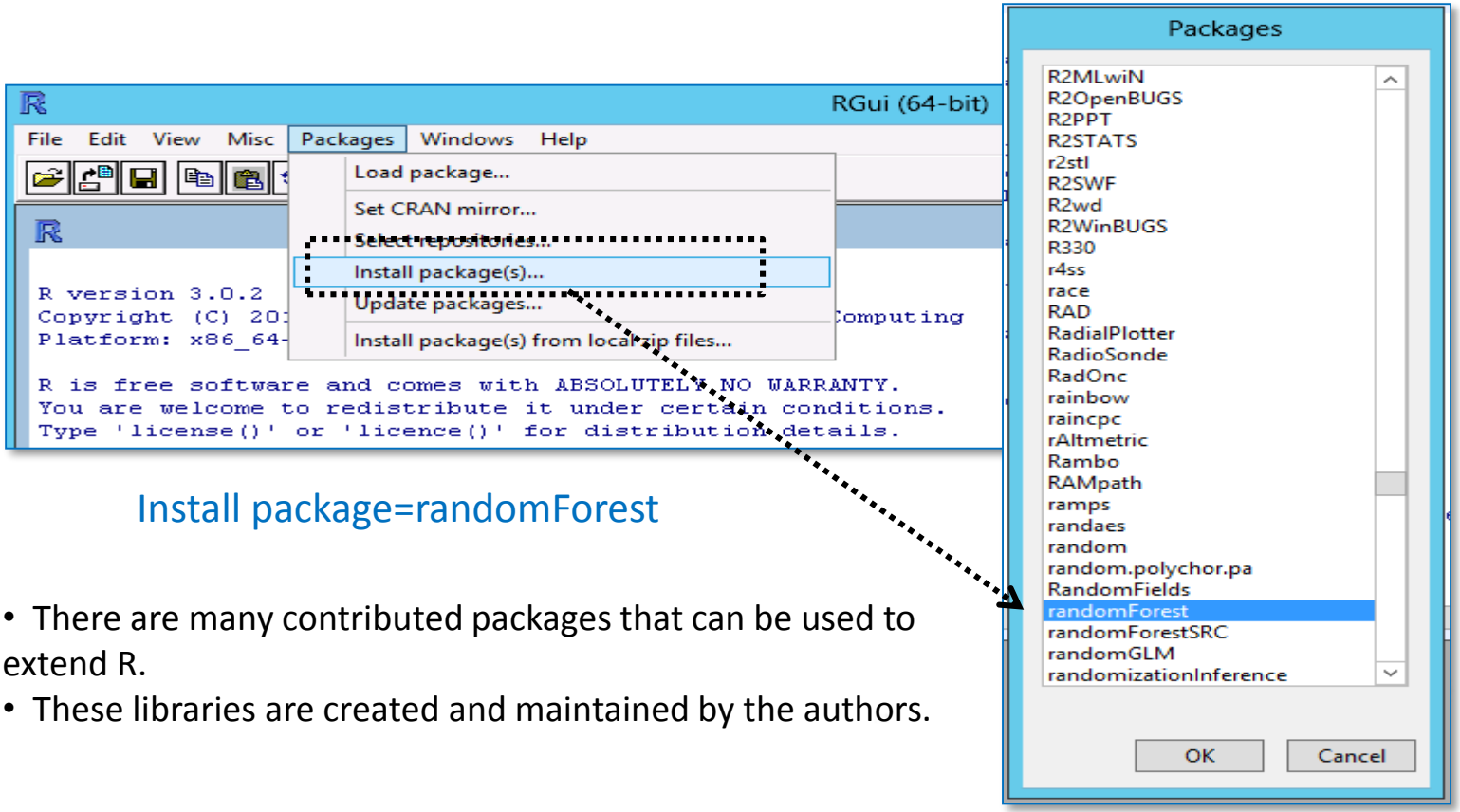


Set CRAN mirror to download packages



Select CRAN mirror= USA (CA 1)

Install package



Install package=randomForest

- There are many contributed packages that can be used to extend R.
- These libraries are created and maintained by the authors.

R as a calculator

- ✓ Calculator
  - +, -, /, *, ^, log, exp, …:

```
> (17*0.35)^(1/3)

> log(10)

> exp(1)

> 3^-1
```

✓ Variables are assigned using '<-':

```
> x<-12.6
> x
[1] 12.6
```

✓ Variables that contains many values (vectors), e.g. with the `concatenate` function:

```
> y<-c(3,7,9,11)
> y
[1]  3  7  9 11
```

✓ Type the numbers in at the keyboard using the `scan()` function:

```
> z<-scan()
1: 8
2: 4
3:
Read 3 items
> z
[1] 8 4
```

✓ Operator ':' means 'a series of integers between':

```
> x<-1:6
> x
[1] 1 2 3 4 5 6
```

✓ Series in non-integer steps (e.g. 0.1) using the `seq()` function :

```
> b<-seq(0.5,0,-0.1)          : negative values for decreasing series
> b
[1] 0.5 0.4 0.3 0.2 0.1 0.0
```

Generating Repeats

- ✓ The `rep` function replicates the first argument by the number of times specified in the second argument:

```
> rep("A",10)
[1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
```

- ✓ Repeated series:

```
> rep(1:6,2)
[1] 1 2 3 4 5 6 1 2 3 4 5 6
```

- ✓ Elements of a series to be repeated:

```
> rep(1:6,rep(3,6))     : vector of the same length (second argument)
[1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6
```

- ✓ To specify each repeat separately:

```
> rep(c(4,7,1,5),c(3,2,5,2))
[1] 4 4 4 7 7 1 1 1 1 1 5 5
```

Sorting

```
D <- data.frame(x=c(1,2,3,1), y=c(7,19,2,2))
```

✓ D # Sort on x

```
indexes <- order(D$x)
 D[indexes,]
```

✓ Print out sorted dataset, sorted in reverse by y

```
D[rev(order(D$y)),]
```

cbind Function

cbind () function combines vector, matrix or data frame by columns.

cbind (x1,x2,...)

x1, x2 : vector, matrix, data frames

```
> a<-c(1,2,3,4,5,6)
> b<-c(5,4,6,7,8,9)
> c<-cbind(a,b)
> c
     a b
[1,] 1 5
[2,] 2 4
[3,] 3 6
[4,] 4 7
[5,] 5 8
[6,] 6 9
```

rbind Function

rbind () function combines vector, matrix or data frame by rows.

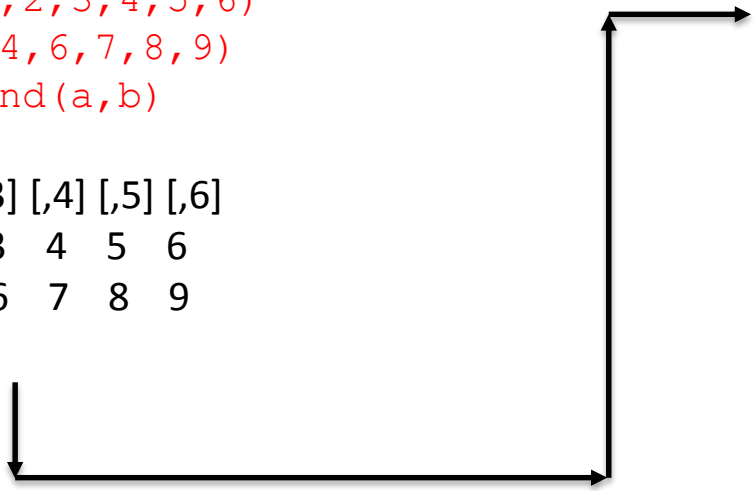
rbind (x1,x2,...)

x1,x2 : vector, matrix, data frames

```
> a<-c(1,2,3,4,5,6)
>b<-c(5,4,6,7,8,9)
> d<-rbind(a,b)
> d
  [,1] [,2] [,3] [,4] [,5] [,6]
a    1    2    3    4    5    6
b    5    4    6    7    8    9
>
```

Transpose Function

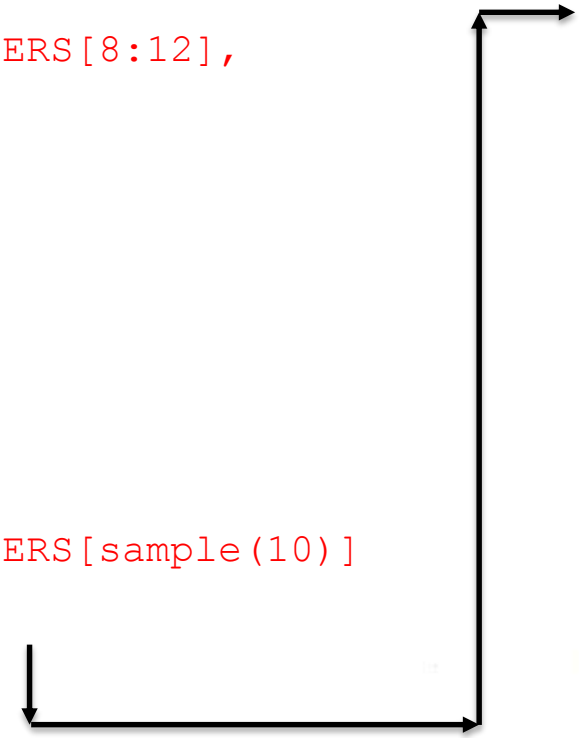
```
> a<-c(1,2,3,4,5,6)
>b<-c(5,4,6,7,8,9)
> d<-rbind(a,b)
> d
  [,1] [,2] [,3] [,4] [,5] [,6]
a    1    2    3    4    5    6
b    5    4    6    7    8    9
```



```
> t(d)
  a b
[1,] 1 5
[2,] 2 4
[3,] 3 6
[4,] 4 7
[5,] 5 8
[6,] 6 9
>
```

Merge Function

```
> A <-
data.frame(letter=LETTERS[8:12],
a=1:5)
> A
 letter a
1      H 1
2      I 2
3      J 3
4      K 4
5      L 5
> B <-
data.frame(letter=LETTERS[sample(10)]
, b=runif(10))
```



```
> B
letter         b
1       E 0.76990725
2       I 0.06591584
3       J 0.40945544
4       C 0.91913609
5       A 0.38310636
6       G 0.42976165
7       F 0.73346686
8       H 0.70133597
9       D 0.99432853
10      B 0.14944860
```

Merge Function… Cont.

```
> merge(A, B)
```

```
letter a        b
1      H 1 0.70133597
2      I 2 0.06591584
3      J 3 0.40945544
```

```
> merge(A,B,all=TRUE)
```

```
 letter  a        b
1       H  1 0.70133597
2       I  2 0.06591584
3       J  3 0.40945544
4       K  4        NA
5       L  5        NA
6       A NA 0.38310636
7       B NA 0.14944860
8       C NA 0.91913609
9       D NA 0.99432853
10      E NA 0.76990725
11      F NA 0.73346686
12      G NA 0.42976165
```

Merge Function… Cont.

```
> merge(A,B,all=FALSE,all.x=TRUE)
```

	letter	a	b
1	H	1	0.70133597
2	I	2	0.06591584
3	J	3	0.40945544
4	K	4	NA
5	L	5	NA

```
> merge(A,B,all=FALSE,all.y=TRUE)
```

	letter	a	b
1	H	1	0.70133597
2	I	2	0.06591584
3	J	3	0.40945544
4	A	NA	0.38310636
5	B	NA	0.14944860
6	C	NA	0.91913609
7	D	NA	0.99432853
8	E	NA	0.76990725
9	F	NA	0.73346686
10	G	NA	0.42976165

Subscripts: Obtaining Parts of Vectors

- ✓ Elements of vectors by subscripts in []:

```
> y[3]
```

- ✓ The third to the seventh elements of y:

```
> y[3:7]
```

- ✓ The third, fifth, sixth and ninth elements:

```
> y[c(3,5,6,7)]
```

- ✓ To drop an element from the array, use negative subscripts:

```
> y[-1]
```

- ✓ To drop the last element of the array without knowing its length:

```
> y[-length(y)]
```

Subscripts as Logical Variables

✓ Logical condition to find a subset of the values in a vector:

```
> y[y>6]
```

✓ To know the values for `z` for which `y>6`:

```
> z[y>6]
```

✓ Element of `y` not multiples of three:

```
> y[y%%3!=0]
```

Subscripts with Arrays (I)

- ✓ Three-dimensional array containing the numbers 1 to 30, with five rows and three columns in each two tables:

```
> A<-array(1:30,c(5,3,2))
> A
, , 1

     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
, , 2

     [,1] [,2] [,3]
[1,]   16   21   26
[2,]   17   22   27
[3,]   18   23   28
[4,]   19   24   29
[5,]   20   25   30
```

The numbers enter each table column-wise, from left to right (rows, then columns then tables)

Subscripts with Arrays (II)

✓ To select columns of A (e.g. second and third):

```
> A[,2:3,]
```

: Columns are the second (middle) subscript

```
, , 1

     [,1] [,2]
[1,]    6   11
[2,]    7   12
[3,]    8   13
[4,]    9   14
[5,]   10   15

, , 2

     [,1] [,2]
[1,]   21   26
[2,]   22   27
[3,]   23   28
[4,]   24   29
[5,]   25   30
```

Subscripts with Arrays (III)

- ✓ To select columns of A (e.g. second and third) and rows (e.g. two to four), of only the second table:

```
> A[2:4,2:3,2]
     [,1] [,2]
[1,]   22   27
[2,]   23   28
[3,]   24   29
```

: rows are the first, columns are the second, and table are the third subscript

Subscripts with Lists (I)

- ✓ Lists are subscribed like this `[[3]]`: list called “cars”, with three elements: “make”, “capacity” and “color”:

```
> cars<-list(c(“Toyota”,”Nissan”,”Honda”),
+        c(1500,1800,1750),c(“blue”,”red”,”black”,”silver”))

[[1]]
[1] "Toyota" "Nissan" "Honda"

[[2]]
[1] 1500 1800 1750

[[3]]
[1] "blue"   "red"    "black"  "silver"
```

Subscripts with Lists (I)

✓ Difference between ‘cars`[[3]]`’:

```
[1] "blue"   "red"    "black"  "silver"
```

✓ And ‘cars`[3]`’:

```
[[1]]
[1] "blue"   "red"    "black"  "silver"
```

Subscripts with Lists (II)

- ✓ Lists are subscribed like this `[[3]]`: list called “cars”, with three elements: “make”, “capacity” and “color”:

```
 > cars<-list(c(“Toyota”,”Nissan”,”Honda”),
 +              c(1500,1800,1750),c(“blue”,”red”,”black”,”silver”))
[[1]]
[1] "Toyota" "Nissan" "Honda"

[[2]]
[1] 1500 1800 1750

[[3]]
[1] "blue"   "red"    "black"  "silver"
```

- ✓ To extract one element of the sub-list:

```
> cars[[3]][2]
[1] "red"
```

Reading data from external file

- ✓ Change Working Directory

```
setwd("C:\\PERSONAL\\R Training\\Data")
```

- ✓ Read space delimited file

```
read.table("test.txt“,head=T)
```

- ✓ Read space delimited file

```
read.csv("mydata.csv“,head=T)
read.table("testcsv.csv",head=T,sep=',')
```

Part 1 Read.Table (some important parameters)

- ✓ **file** the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an *absolute* path, the file name is *relative* to the current working directory, getwd().
- ✓ **header** a logical value indicating whether the file contains the names of the variables as its first line.
- ✓ **sep** the field separator character. Values on each line of the file are separated by this character.
- ✓ **tringsAsFactors** logical: should character vectors be converted to factors?
- ✓ **nrows** number of rows to read
- ✓ **skip** number of rows to skip before reading
- ✓ **fill** logical if TRUE blank fields are added to rows of unequal length
- ✓ **strip.white** logical if TRUE leading and trailing white spaces are stripped from unquoted character strings
- ✓ **blank.lines.strip** logical" if TRUE blank lines are ignored
- ✓ **stringsAsFactors** logical if TRUE character values are input as factors

Generating Plots in R

Line Plots

```
# Define the cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

# Graph the cars vector with all defaults
plot(cars)
```

Line Plots

```
# Define the cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

# Graph cars using blue points overlayed by a line
plot(cars, type="o", col="blue")

# Create a title with a red, bold/italic font
title(main="Autos", col.main="red", font.main=4)
```

Line Plots

```
# Define 2 vectors
cars <- c(1, 3, 6, 4, 9)
trucks <- c(2, 5, 4, 5, 12)

# Graph cars using a y axis that ranges from 0 to 12
plot(cars, type="o", col="blue", ylim=c(0,12))

# Graph trucks with red dashed line and square points
 lines(trucks, type="o", pch=22, lty=2, col="red")

# Create a title with a red, bold/italic font
title(main="Autos", col.main="red", font.main=4)
```

Bar Plots

```
# Define the cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

# Graph cars
barplot(cars)
```

Bar Plots – Describing the plot configuration

```
> a<-c(1,2,5,4,3)
> b<-c(4,10,3,5,6)
> c<-c(34,23,12,10,5)
>

barplot(a,main="cars",xlab="Days",ylab="Total",names.arg=c("Mon","Tue","Wed","Thu
+ ","Fri"),border="blue",density=c(10,20,30,40,50))
>
```

Bar Plots – Changing colors

```
> a<-c(1,2,5,4,3)
> b<-c(4,10,3,5,6)
> c<-c(34,23,12,10,5)
>

barplot(a,main="cars",xlab="Days",ylab="Total",names.arg=c("Mon","Tue","Wed","Thu
+ ","Fri"),border="blue",density=c(10,20,30,40,50),col=rainbow(5))
>
```

Pie chart

```
# Define cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

 # Create a pie chart for cars
pie(cars)
```

Pie chart – Changing colors

```
# Define cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

 # Create a pie chart with defined heading and # custom colors and labels
pie(cars, main="Cars", col=rainbow(length(cars)),
labels=c("Mon","Tue","Wed","Thu","Fri"))
```

Pie chart – Defining other components

```
# Define cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

# Define some colors ideal for black & white print
colors <- c("white","grey70","grey90","grey50","black")

# Calculate the percentage for each day, rounded to one # decimal place
car_labels <- round(cars/sum(cars) * 100, 1)

# Concatenate a '%' char after each value
car_labels <- paste(car_labels, "%", sep="")

# Create a pie chart with defined heading and custom colors # and labels
pie(cars, main="Cars", col=colors, labels=car_labels, cex=0.8)
```