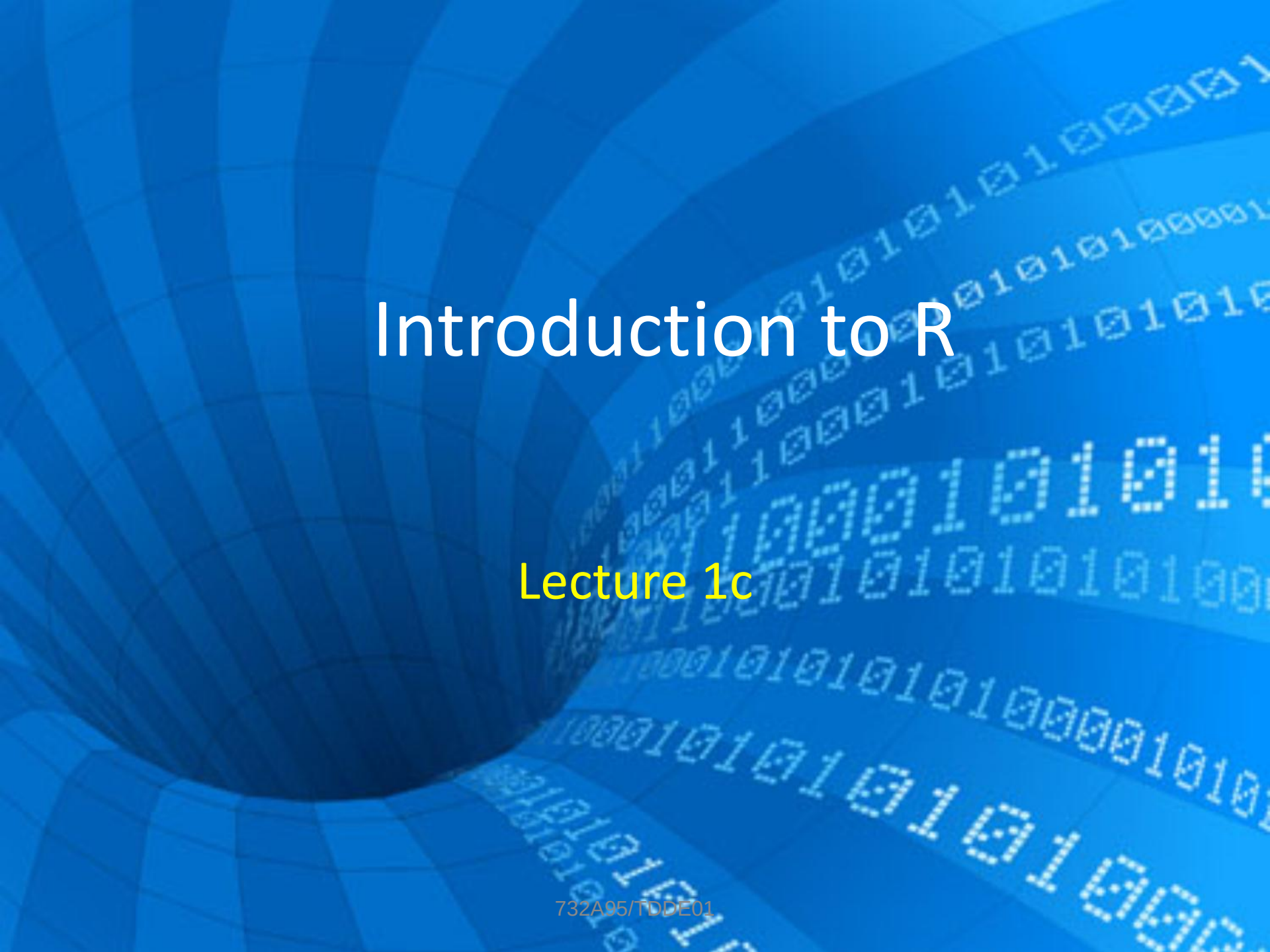
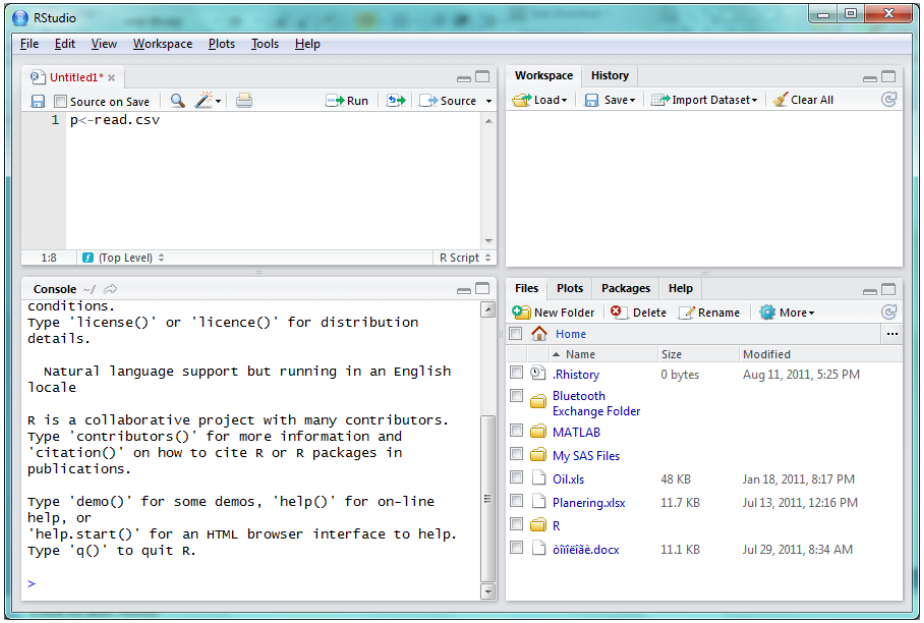


Introduction to R

Lecture 1c

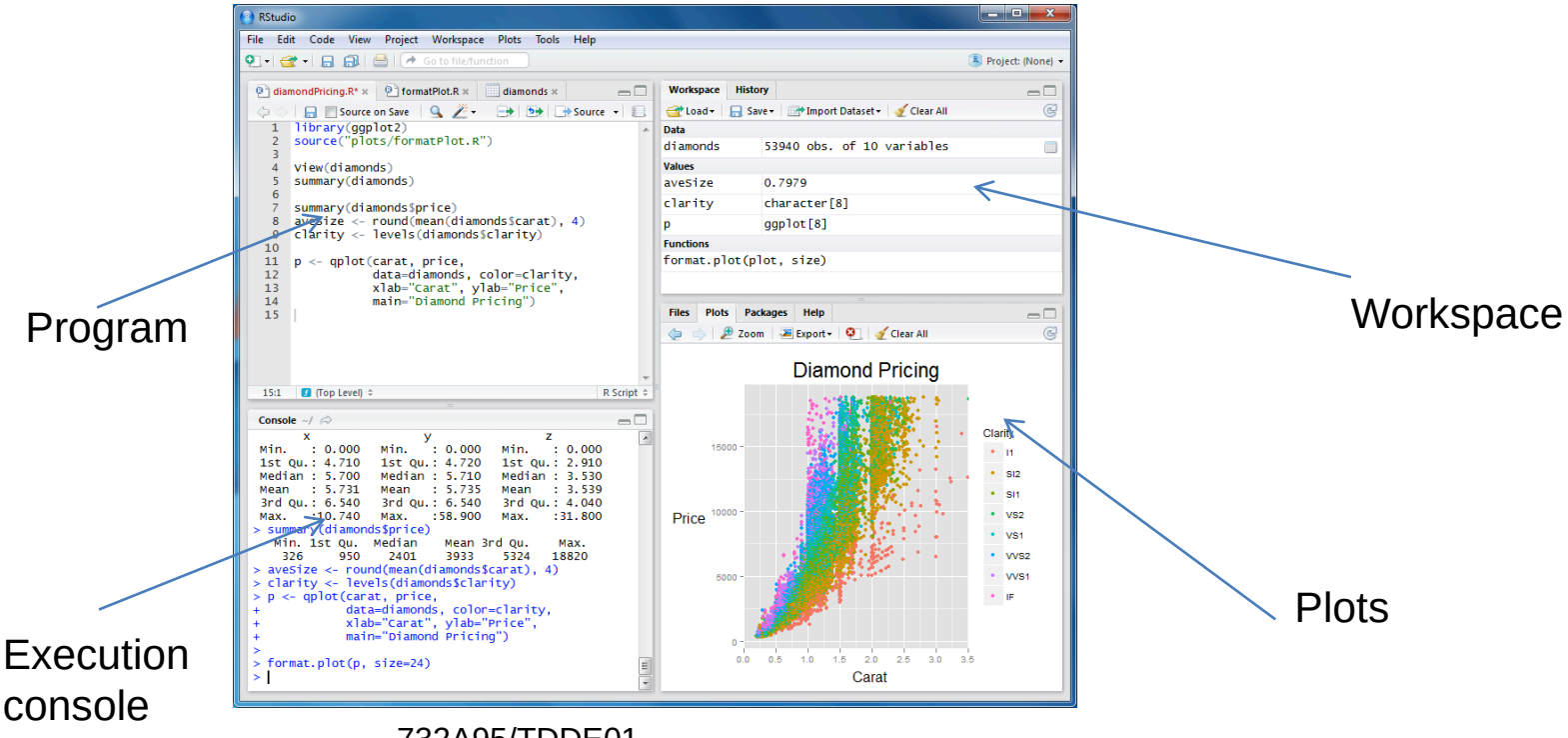
R

- R is a **programming language** of a higher-level
- Constantly increasing amount of packages (new research)
- Free of charge
- Website: http://www.r-project.org/
- Code Editor: http://rstudio.org/



Software: use RStudio

- Install R: http://www.r-project.org/
- Install RStudio: http://rstudio.org/



Basics in RStudio

Important to know:

- Create a new file and save it (File menu)
- Running one line or entire code (Edit menu)
- Running one line in console
- Workspace (Observe, Save, Clear)
- Setting current directory (Tools)
- Installing new package (Packages tabs)

Call help

- Specific function
  - `help(`*`function`*`)`
- Help browser
  - `help.start()`
- Search for something in help
  - `help.search(“`*`expression`*`”)`
- Quick reminder of function arguments:
  - `args(`*`function`*`)`
- Examples of how to use function:
  - `example(`*`function`*`)`
- If some method is not installed on the computer:
  - `RSiteSearch(”expression")`

Introduction

- R is case-sensitive (A and a)
- Each command on a new line
- Comment:

```
#R is a very cool language!
```

Initialize/set the variable

Use-> or <- or =

```
a<-3
a=3
3->b
```

Vectors

- Create a vector

  x<-c(1,3)

- See the result

  x

  print(x)

```
> x<-c(1,3)
> x
[1] 1 3
> print(x)
[1] 1 3
>
```

- Create an empty vector

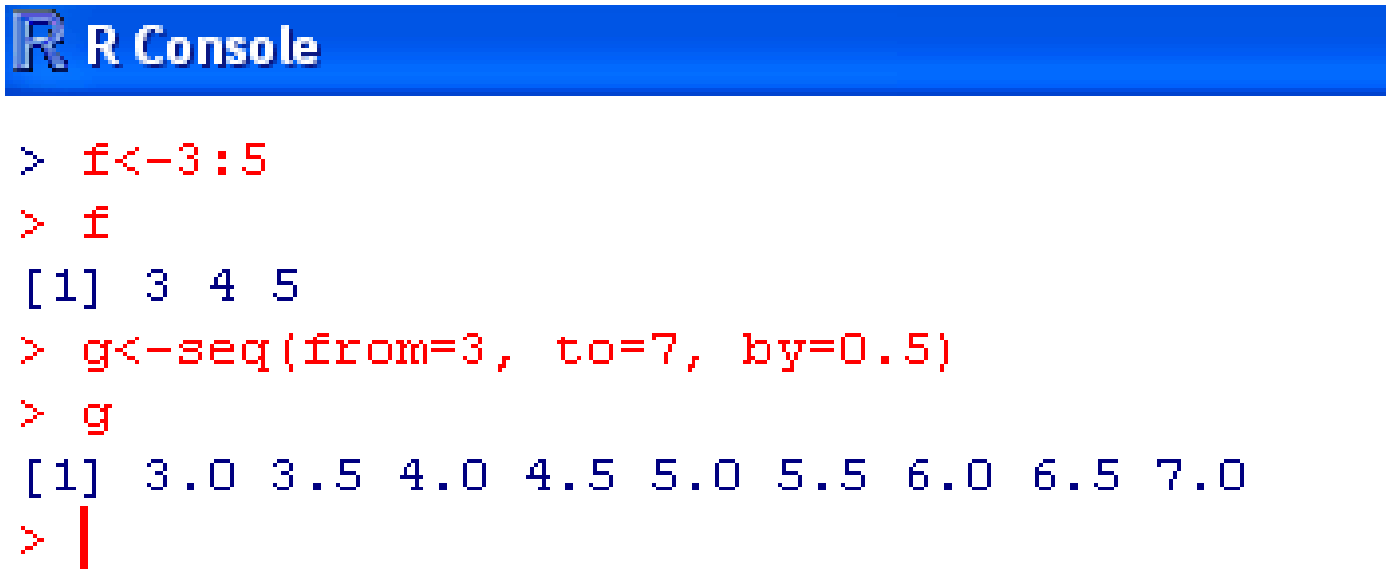
  Y=numeric(10)

  Y

```
> Y=numeric(10)
> Y
 [1] 0 0 0 0 0 0 0 0 0 0
```

Sequence

- Either‘: ‘ or seq()



Operation with vectors

- indexing
- Element-wise: +-*/^
- log exp sin cos
- length –number of elements
- sum - sum of all elements
- max min sort order
- which.min which.max

Logicals:

TRUE or FALSE:

A=TRUE;

== > >= < <= != & (and) | (or)

```
> a=1:5
> b=c(1,4,-1,3,0)
> a+b
[1] 2 6 2 7 5
> a*b
[1]  1  8 -3 12  0
> b+4
[1] 5 8 3 7 4
> length(a)
[1] 5
> sum(a^2)
[1] 55
> max(b)
[1] 4
> which.max(b)
[1] 2
> order(b)
[1] 3 5 1 4 2
> sort(b)
[1] -1  0  1  3  4
> b[1]
[1] 1
> b[2:4]
[1]  4 -1  3
> b[-2]
[1]  1 -1  3  0
```

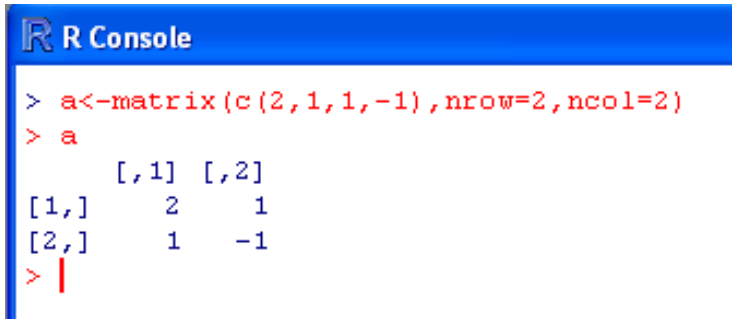
Matrices

Use `matrix()`

`a<-matrix(`*values*`,nrow=`*m*`,ncol=`*n*`)`

Values should be listed columnvise

`nrow=` and `ncol=` can be skipped



- Create empty matrix

```
> m=matrix(0,nrow=2,ncol=3)
> m
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
>
```

Matrix operations

```
> x<-c(1,2)
> a<-matrix(c(2,1,1,-1),2,2)
> b<-matrix(c(1,0,1,1),2,2)
> y=a%*%x
> y
     [,1]
[1,]    4
[2,]   -1
> c=a%*%b
> c
     [,1] [,2]
[1,]    2    3
[2,]    1    0
>
```

Usual vector operations can also be applied:

```
> m4=matrix(c(1,2,0,1), nrow=2)
> m5=matrix(c(2,2,5,1), nrow=2)
> m4
     [,1] [,2]
[1,]    1    0
[2,]    2    1
> m5
     [,1] [,2]
[1,]    2    5
[2,]    2    1
> m4*m5
     [,1] [,2]
[1,]    2    0
[2,]    4    1
```

Matrix operations

- Matrix operators/functions:
- transpose `b=t(a)`

  $b = a^T$
- Inverse $b = a^{-1}$

  `b=solve(a)`
- Solve $d=a^{-1}b$

  `d=solve(a,b)`

```
> a
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> t(a)
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> solve(a)
           [,1]       [,2]
[1,] 0.3333333  0.3333333
[2,] 0.3333333 -0.6666667
>
```

Indexing for matrices

- Positive index

```
x[1,6]      x[2:10,]
```

- Negative index

```
x[2,-(1:5)]
```
row 2 and all columns except 1:5

- Entire column or row

```
y=x[2,]  entire row 2
```

- Extraction

```
x[x>5]
```

```
> b
[1]  1  4 -1  3  0
> d=b[b>0]
> d
[1] 1 4 3
```

Replication

- Replication for vectors
  - rep(what, times)
- Replication for matrices
  - matrix()

```
> v1=rep(3,5)
> v1
[1] 3 3 3 3 3
> v2=rep(c(3,4),2)
> v2
[1] 3 4 3 4
> m1=matrix(1,nrow=2,ncol=2)
> m1
     [,1] [,2]
[1,]    1    1
[2,]    1    1
> m2=matrix(v2,nrow=4,ncol=2)
> m2
     [,1] [,2]
[1,]    3    3
[2,]    4    4
[3,]    3    3
[4,]    4    4
> m3=matrix(v2,nrow=2,ncol=4, byrow=T)
> m3
     [,1] [,2] [,3] [,4]
[1,]    3    4    3    4
[2,]    3    4    3    4
```

Matrix operations

- Dimension
  - dim(mat)
- Row/column stististics
  - colMeans, rowMeans, colSums,rowSums

```
> m4=matrix(c(1,3,5,2,4,6),nrow=3)
> m4
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> colMeans(m4)
[1] 3 4
> rowSums(m4)
[1]  3  7 11
```

```
> m2
     [,1] [,2]
[1,]    3    3
[2,]    4    4
[3,]    3    3
[4,]    4    4
> ns=dim(m2)
> ns[2]
[1] 2
```

- Apply a function over vector/matrix
  - Sapply()
  - Normally used when function works only element-wise

```
> sapply(v2,log)
[1] 1.098612 1.386294 1.098612 1.386294
> log(v2)
[1] 1.098612 1.386294 1.098612 1.386294
```

Vector/matrix operations

- Create confusion matrix (classification)
  - table(X,Y)
- Extract diagonal
  - Diag(X)

```
> X=c(1,3,1,1,2,3,1,2,2,2,1,1,3)
> Xfit=c(2,3,2,1,2,3,1,2,2,2,1,1,1)
> t1=table(Xfit,X)
> t1
    X
Xfit 1 2 3
   1 4 0 1
   2 2 4 0
   3 0 0 2
> t1[1,1]
[1] 4
> diag(t1)
1 2 3
4 4 2
>
```

Factors

- Text values

```
> f1=c("Man", "Woman")
> f1
[1] "Man"   "Woman"
> f2=c("Man", "Woman", "Man")
> table(f2)
f2
  Man Woman
    2     1
> f3=factor(c(1,0,1,1,0), levels=c(0,1), labels=c("Man", "Woman"))
> f3
[1] Woman Man   Woman Woman Man
Levels: Man Woman
>
```

Lists

- List is a collection of objects

```
> d<-15;
> a<-matrix(c(1,2,3,4),2,2);
> a
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> b<-list(first=d, second=a, x="mary")
> b
$first
[1] 15

$second
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$x
[1] "mary"
```

```
> b$x
[1] "mary"
> b[[3]]
[1] "mary"
```

Data frame

Vectors and matrices of the row length can be collected into a data frame

- Used to store the data of different types into a single table

Use `data.frame(`*object 1*, *object 2*, … , *object k*`)`

```
> x<-c(1,3)
> y<-c("M", "F")
> z<-data.frame(x,y)
> z
  x y
1 1 M
2 3 F
```

Data frame

- Any column in the data frame can be retrieved by

  dataframe$*object*

```
> z$x
[1] 1 3
> z[[1]]
[1] 1 3
> z$y
[1] M F
Levels: F M
```

- Any row in the data frame can be extracted by using matrix notation, for ex: **z[1,]**

Read data from Excel file

1. Save as ”comma-separated file”(csv)
2. Change current directory, Session→ Set Working Directory or setwd()
3. Use

```
Dataframe=read.csv2(file_name)
```

```
Dataframe=read.csv(file_name)
```

Conversion between types

- Data frame to matrix
- Matrix to data frame
- Numeric to factor
- Factor to numeric
- List to vector
- Vector to list

```
> df1=data.frame(X=c(1,2,3), Y=c(1,0,1))
> df1
  X Y
1 1 1
2 2 0
3 3 1
> m5=as.matrix(df1)
> m5
     X Y
[1,] 1 1
[2,] 2 0
[3,] 3 1
> df2=as.data.frame(m5)
> df2$X
[1] 1 2 3
```

```
> v6=c(1,4,2,2,1)
> f4=as.factor(v6)
> f4
[1] 1 4 2 2 1
Levels: 1 2 4
> f5=c("1", "0", "1", "1", "1")
> f5
[1] "1" "0" "1" "1" "1"
> as.list(f5)
[[1]]
[1] "1"

[[2]]
[1] "0"

[[3]]
[1] "1"

[[4]]
[1] "1"

[[5]]
[1] "1"

> l1=list(a=1, b=3)
> l1
$a
[1] 1

$b
[1] 3

> as.numeric(l1)
[1] 1 3
```

Loops

```
for (name  in expr1 )
{
…
}

while (condition)
{
…
}
```

```
> for (i in 1:5) {
+ y=seq(i,8)
+ print(y) }
[1] 1 2 3 4 5 6 7 8
[1] 2 3 4 5 6 7 8
[1] 3 4 5 6 7 8
[1] 4 5 6 7 8
[1] 5 6 7 8
>
```

Conditioning and loops

```
If (x==3) {
…
…
} else {
…
}
```

```
for (i in 2:99) {
…
}
```

```
while(x!=29) {
…
}
```

```
> m4=matrix(c(1,2,0,1), nrow=2)
> m4
     [,1] [,2]
[1,]    1    0
[2,]    2    1
> n=dim(m4)[1]
> I=numeric(n)
> for (i in 1:n){
+   if(max(m4[i,]>1)) I[i]=1
+ }
> I
[1] 0 1
```

Random number generation

- Random are not random
  - Use set.seed(12345) to get identical results
- A plenty of random number generators
  - Rnorm
  - Runif
  - …
- Use d for density p for CDF q for quantiles and r for simulation:

(ex: rnorm pnorm dnorm qnorm)

Distribution	R name	additional arguments
beta	beta	shape1, shape2, ncp
binomial	binom	size, prob
Cauchy	cauchy	location, scale
chi-squared	chisq	df, ncp
exponential	exp	rate
F	f	df1, df2, ncp
gamma	gamma	shape, scale
geometric	geom	prob
hypergeometric	hyper	m, n, k
log-normal	lnorm	meanlog, sdlog
logistic	logis	location, scale
negative binomial	nbinom	size, prob
normal	norm	mean, sd
Poisson	pois	lambda
Student's t	t	df, ncp
uniform	unif	min, max
Weibull	weibull	shape, scale
Wilcoxon	wilcox	m, n

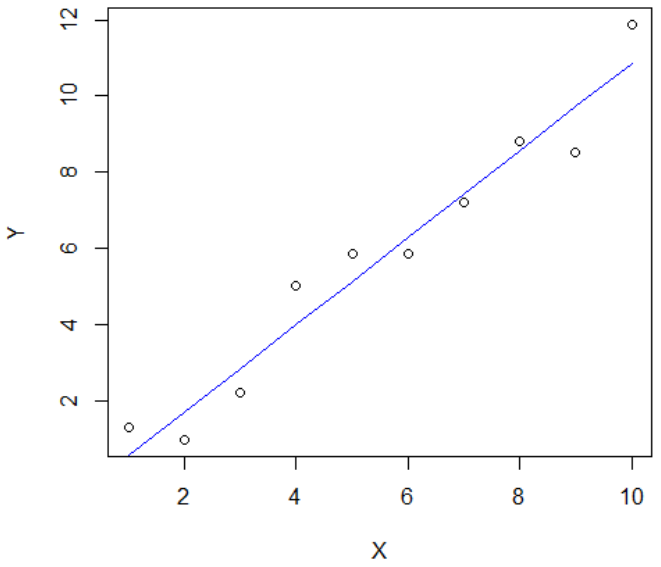
Using a function

- Use ?*name_of_function* to see function parameters
  - For ex. *?lm*
- There are some obligatory parameters and optional parameters
- The optional parameters can be specified in different order

```
X=1:10
Y=1:10+rnorm(10)
W=c(rep(1,5), rep(2,5))
mydata=data.frame(X,Y)

result=lm(Y~X,  weights=W,data=mydata)
?predict.lm
Fit=predict(result)

plot(X,Y)
points(X,Fit, type="l", col="blue")
```



Writing your own functions

- Function writing must always end with writing the value which should be returned!
- You may also use ”**return(value)**” to show what value the function should return

```
> myfun <-function(x=20, y, z)
+ {
+ if(z)
+ result=x+y
+ else
+ {
+ result=log(x)*y
+ }
+ result
+ }
> r<-myfun(1,2, TRUE)
> r
[1] 3
> r<-myfun(z=FALSE, y=0)
> r
[1] 0
>
```

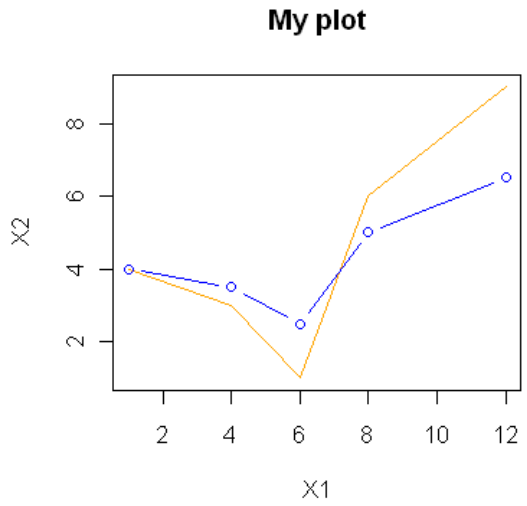
Graphical procedures

Some common procedures:

- plot(x,..) plots time series
- plot(x,y) scatter plot
- plot(x,y) followed by points(x,y) plots several scatterplots in one coordinate system
- hist(x,..) plots a hitogram
- persp(x,y,z,…) creates surface plots
- cloud(formula,data..) creates 3D scatter plot

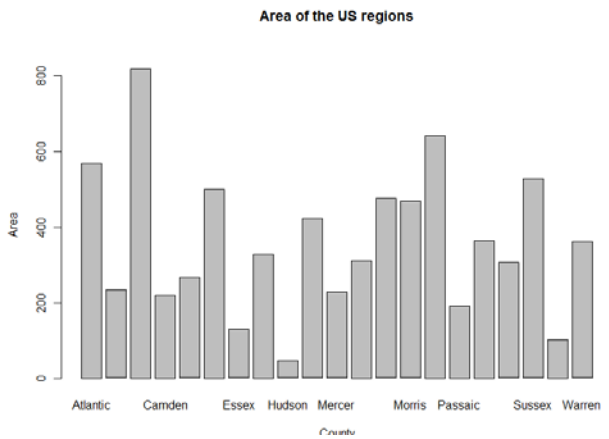
```
x<-c(1,4,7 8, 12);
y<-c(4,3,1,6,9);

plot(x,y, type="l", col="orange",
    main="My plot", xlab="X1", ylab="X2");
points(x, y/2+2, type="b", col="blue");
```



Graphical parameters

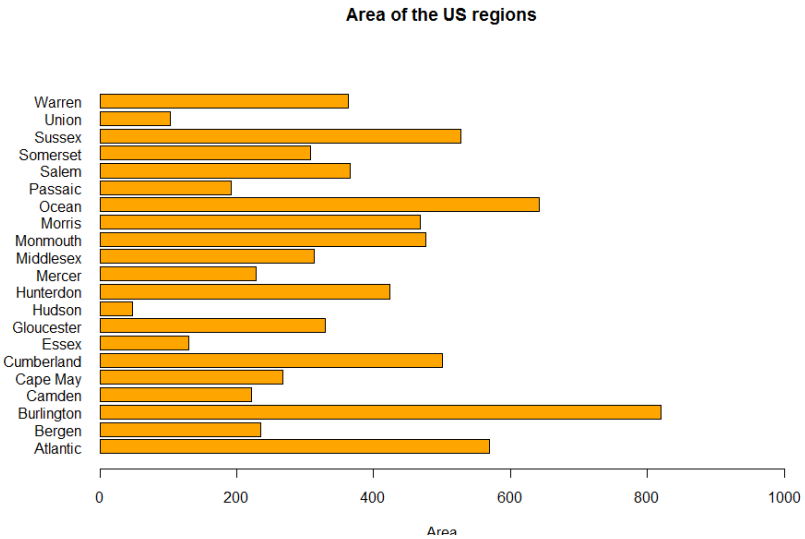
- Adjust color of a graphical object by specifying
  - col=
- Other typical parameters for graphical functions
  - main=”text” Title”text”
  - sub=”text” Footnote ”text”
  - xlab=”text” X-axis label
  - ylab=”text” Y-axis label




```
mydata<-read.csv2("Counties.csv");
barplot(mydata$Area, names.arg=mydata$County, main=”Area of the US regions”,
xlab=”County”, ylab=”Area”);
```

Graphical parametes

- Some parameters need to be specified either in the plotting function or inside par(…)
  - Pch=number – symbol that is plotted
  - Lty=number – linetype
  - Las=0 1 eller 2 Direction of axis values
  - mai=c(bottom, left, top, right) – margins (inch)
  - adj=between 0 and 1, horizontal justification




```
barplot(mydata$Area,
names.arg=mydata$County, horiz=TRUE, las=1,
xlim=c(0,1000), col="orange", main="Area of
the US regions", xlab="Area");
```

Some more examples

- Dividing training/test

```
data=data.frame(X=c(1,1,2,2,3), Y=c("M","F","M","M","F"))
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
```

```
> train
  X Y
4 2 M
5 3 F
> test
  X Y
1 1 M
2 1 F
3 2 M
```

- Computing misclassification rate

```
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
```

```
> X=c(1,3,1,1,2,3,1,2,2,2,1,1,3)
> Xfit=c(2,3,2,1,2,3,1,2,2,2,1,1,1)
> missclass(X,Xfit)
[1] 0.2307692
```