CT5102: Programming for Data Analytics

Lecture 4: Matrices and Data Frames

Dr. Jim Duggan,

School of Engineering & Informatics

National University of Ireland Galway.

https://github.com/JimDuggan/PDAR

https://twitter.com/_jimduggan

Matrices

	Homogenous	Heterogenous
1d	Atomic Vector	List
2d	**Matrix**	Data Frame
nd	Array	

- A matrix can be initialized from a vector, where the numbers of rows and columns are specified.
- R stores matrices by column-major order, and by default matrices are filled in this manner.

Declaring a matrix

```
>
> a <- matrix(1:6, ncol=3, nrow=2)
>
> a
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
> dim(a)
[1] 2 3
```

Adding rows and columns

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

Naming rows and columns

```
> rownames(a) <- c("A","B")
>
> a
  [,1] [,2] [,3]
A    1    3    5
B    2    4    6
>
> colnames(a) <- c("a","b","c")
>
> a
  a b c
A 1 3 5
B 2 4 6
```

Subsetting Matrices

- The most common way of subsetting 2d matrix is a simple generalisation of 1d subsetting
- Supply a 1d index for each dimension, separated by a comma
- Blank subsetting is useful, as it lets you keep all rows or all columns

Using row index…

```
> b <- matrix(1:9, nrow=3)
>
> colnames(b) <- c("A","B","C")
>
> b
     A B C
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
>
> b[1:2,]
     A B C
[1,] 1 4 7
[2,] 2 5 8
```

```
> b[c(T,F),]
     A B C
[1,] 1 4 7
[2,] 3 6 9
>
>
> b[-3,]
     A B C
[1,] 1 4 7
[2,] 2 5 8
```

Using column index…

```
> b
     A B C
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
>
> b[,1:2]
     A B
[1,] 1 4
[2,] 2 5
[3,] 3 6
```

```
> b[,c(T,F)]
     A C
[1,] 1 7
[2,] 2 8
[3,] 3 9
>
> b[,c("A","C")]
     A C
[1,] 1 7
[2,] 2 8
[3,] 3 9
```

Sample Matrix Operations

Operator or Function	Description
A * B	Element-wise multiplication
A / B	Element-wise division
A %*% B	Matrix multiplication
t(A)	Transpose of A
e<-eigen(A)	List of eigenvalues and eigenvectors for matrix A

apply() function

- The **apply()** function can be used to process rows and columns for a matrix, and the general form of this function (Matloff 2009) is **apply(m, dimcode, f, fargs)**, where:
  - **m** is the target matrix
  - **dimcode** identifies whether it's a row or column target. The number 1 applies to rows, whereas 2 applies to columns
  - **f** is the function to be called
  - **fargs** are the optional set of arguments that can be applied to the function **f**.

Examples…

```
> b
     A B C
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
>
> apply(b,1,sum)
[1] 12 15 18
```

```
> b
     A B C
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
>
> apply(b,2,sum)
 A  B  C
 6 15 24
```

Challenge 4.1

- Create a 4x4 matrix, with values from 1:16, where the matrix is filled by row order
- Add a column called “SumRow” which contains the sum of each row in the matrix
- Add a row (to the original matrix) called “SumCol” which contains the sum of each column in the matrix
- Use the apply() function

Data Frames

- The most common way of storing data in R
- Under the hood, a data frame is a list of equal-length vectors
- A two-dimensional structure, it shares properties of both a list and a matrix

	Homogenous	Heterogenous
1d	Atomic Vector	List
2d	Matrix	**Data Frame**
nd	Array	

Creating a data frame…

```
> df <- data.frame(x=1:5,y=LETTERS[1:5],stringsAsFactors=F)
>
> str(df)
'data.frame':   5 obs. of  2 variables:
 $ x: int  1 2 3 4 5
 $ y: chr  "A" "B" "C" "D" ...
```

```
> df
  x y
1 1 A
2 2 B
3 3 C
4 4 D
5 5 E
```

Logical subsetting

- Common technique for extracting rows out of a data frame

```
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Examples

```
> mtcars[mtcars$gear == 5,]
                mpg cyl  disp  hp drat    wt qsec vs am gear carb
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
>
> mtcars[mtcars$gear == 5 & mtcars$cyl == 8,]
                mpg cyl disp  hp drat   wt qsec vs am gear carb
Ford Pantera L 15.8   8  351 264 4.22 3.17 14.5  0  1    5    4
Maserati Bora  15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
```

Sampling from a data frame…

- Selecting n random observations from a data frame

```
> mtcars[sample(nrow(mtcars),2),]
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
>
> mtcars[sample(nrow(mtcars),2),]
                mpg cyl  disp hp drat    wt qsec vs am gear carb
Merc 240D      24.4   4 146.7 62 3.69 3.190 20.0  1  0    4    2
Toyota Corolla 33.9   4  71.1 65 4.22 1.835 19.9  1  1    4    1
```

Adding new columns

```
> cities <- data.frame(Name=c("Dublin", "London", "Paris", "Madrid"),
+                      Population=c(553165,8673713,2244000,3141991))
>
> cities
    Name Population
1 Dublin     553165
2 London    8673713
3  Paris    2244000
4 Madrid    3141991
>
> cities$Type <- ifelse(cities$Population > 3000000,"LARGE","MEDIUM")
>
> cities
    Name Population   Type
1 Dublin     553165 MEDIUM
2 London    8673713  LARGE
3  Paris    2244000 MEDIUM
4 Madrid    3141991  LARGE
```

Reading from a Spreadsheet

Year	Leinster	Munster	Connacht	Ulster
1841	1973731	2396161	1418859	740048
1851	1672738	1857736	1010031	571052
1861	1457635	1513558	913135	517783
1871	1339451	1393485	846213	474038
1881	1278989	1331115	821657	438259
1891	1187760	1172402	724774	383758
1901	1152829	1076188	646932	345874
1911	1162044	1035495	610984	331165
1926	1149092	969902	552907	300091
1936	1220411	942272	525468	280269
1946	1281117	917306	492797	263887
1951	1336576	898870	471895	253252
1956	1338942	877238	446221	235863

```
library(gdata)

pop <- read.xls("R code/04 Matrices & DF/CensusData.xlsx")
```

pop – a data frame in R

```
> head(pop)
  Year Leinster Munster Connacht Ulster
1 1841  1973731 2396161  1418859 740048
2 1851  1672738 1857736  1010031 571052
3 1861  1457635 1513558   913135 517783
4 1871  1339451 1393485   846213 474038
5 1881  1278989 1331115   821657 438259
6 1891  1187760 1172402   724774 383758
```

Getting the total population

```
> pop$Total <- pop$Leinster+pop$Munster+pop$Connacht+pop$Ulster
>
> head(pop)
  Year Leinster Munster Connacht Ulster   Total
1 1841  1973731 2396161  1418859 740048 6528799
2 1851  1672738 1857736  1010031 571052 5111557
3 1861  1457635 1513558   913135 517783 4402111
4 1871  1339451 1393485   846213 474038 4053187
5 1881  1278989 1331115   821657 438259 3870020
6 1891  1187760 1172402   724774 383758 3468694
```

Writing to an Excel File

```
pop$Total <- pop$Leinster+pop$Munster+pop$Connacht+pop$Ulster

library(xlsx)

write.xlsx(x = pop, file = "./R code/04 Matrices & DF/Output.xlsx",
           sheetName = "Processesed Data", row.names = FALSE)
```

	A	B	C	D	E	F
1	Year	Leinster	Munster	Connacht	Ulster	Total
2	1841	1973731	2396161	1418859	740048	6528799
3	1851	1672738	1857736	1010031	571052	5111557
4	1861	1457635	1513558	913135	517783	4402111
5	1871	1339451	1393485	846213	474038	4053187
6	1881	1278989	1331115	821657	438259	3870020
7	1891	1187760	1172402	724774	383758	3468694
8	1901	1152829	1076188	646932	345874	3221823
9	1911	1162044	1035495	610984	331165	3139688
10	1926	1149092	969902	552907	300091	2971992
11	1936	1220411	942272	525468	280269	2968420
12	1946	1281117	917306	492797	263887	2955107

Checking for complete cases in a data

```
>
> subpop<-pop[1:6,]
>
> subpop
  Year Leinster Munster Connacht Ulster   Total
1   NA  1973731 2396161  1418859 740048 6528799
2 1851  1672738 1857736       NA 571052 5111557
3 1861  1457635 1513558   913135 517783 4402111
4 1871  1339451 1393485   846213 474038 4053187
5 1881  1278989 1331115   821657 438259 3870020
6 1891  1187760 1172402   724774 383758 3468694
>
> complete.cases(subpop)
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
>
> subpop[complete.cases(subpop),]
  Year Leinster Munster Connacht Ulster   Total
3 1861  1457635 1513558   913135 517783 4402111
4 1871  1339451 1393485   846213 474038 4053187
5 1881  1278989 1331115   821657 438259 3870020
6 1891  1187760 1172402   724774 383758 3468694
```

Find rows containing missing data

```
> head(pop)
  Year Leinster Munster Connacht Ulster   Total
1   NA  1973731 2396161  1418859 740048 6528799
2 1851  1672738 1857736  1010031 571052 5111557
3 1861  1457635 1513558   913135 517783 4402111
4 1871  1339451 1393485   846213 474038 4053187
5 1881  1278989 1331115   821657 438259 3870020
6 1891  1187760 1172402   724774 383758 3468694
>
> which(!complete.cases(pop))
[1] 1
>
> pop[which(!complete.cases(pop)),]
  Year Leinster Munster Connacht Ulster   Total
1   NA  1973731 2396161  1418859 740048 6528799
```

Checking for invalid values (e.g. negative values)

```
> df
  Year Leinster Munster Connacht Ulster
1 1841  1973731 2396161  1418859 740048
2 1851     -122 1857736  1010031 571052
>
> data.frame(sapply(df,function(x)ifelse(x<0,NA,x)))
  Year Leinster Munster Connacht Ulster
1 1841  1973731 2396161  1418859 740048
2 1851       NA 1857736  1010031 571052
```

Merge Function

```
> s
      ID FirstName Surname Age Discount
1 1234567      Jane   Smith  21     0.25
2 1234568      Matt Johnson  25     0.10
> res
      ID Subject Grade
1 1234567   CT111    80
2 1234568   CT111    80
> new<-merge(s,res,by="ID")
> new
      ID FirstName Surname Age Discount Subject Grade
1 1234567      Jane   Smith  21     0.25   CT111    80
2 1234568      Matt Johnson  25     0.10   CT111    80
```

Available Data Sets in R

AirPassengers, BJsales, BOD, CO2, ChickWeight, DNase, EuStockMarkets, Formaldehyde, HairEyeColor, Harman23.cor, Harman74.cor, Indometh, InsectSprays, JohnsonJohnson, LakeHuron, LifeCycleSavings, Loblolly Nile, Orange, OrchardSprays, PlantGrowth, Puromycin…

Iris flower data set

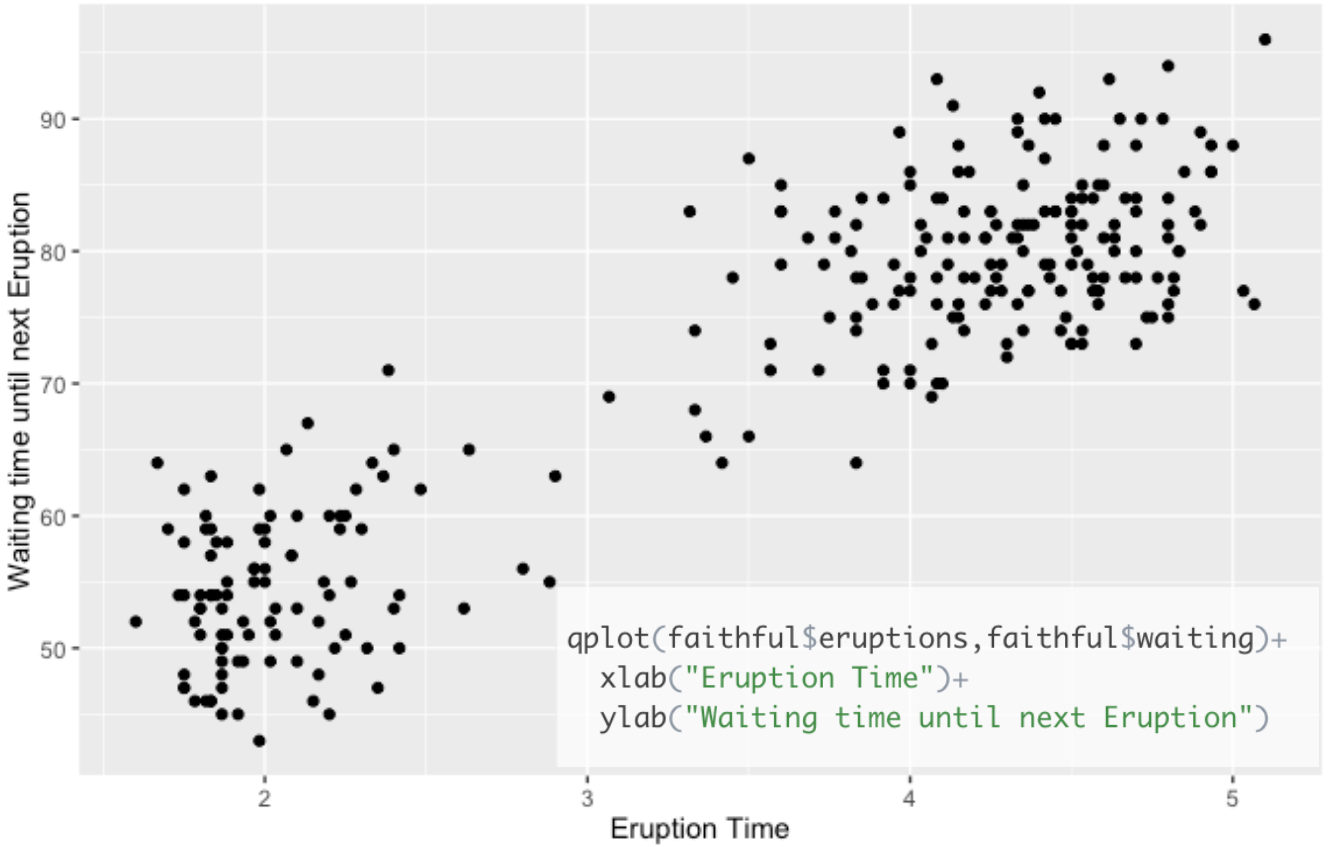
From Wikipedia, the free encyclopedia

The ***Iris* flower data set** or **Fisher's *Iris* data set** is a multivariate data set introduced by Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems* as an example of linear discriminant analysis.[1] It is sometimes called **Anderson's *Iris* data set** because Edgar Anderson collected the data to quantify the morphologic variation of *Iris* flowers of three related species.[2] Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus".[3]

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Old Faithful

Type	Cone geyser
Eruption height	106 feet (32 m) to 185 feet (56 m)
Frequency	45 to 125 minutes
Duration	1.5 to 5 minutes



Challenge 4.2

- Given that a data frame can be manipulated using matrix notation, find another way to calculate the total population (*hint: use the apply function*)

```
> head(pop)
  Year Leinster Munster Connacht Ulster
1 1841  1973731 2396161  1418859 740048
2 1851  1672738 1857736  1010031 571052
3 1861  1457635 1513558   913135 517783
4 1871  1339451 1393485   846213 474038
5 1881  1278989 1331115   821657 438259
6 1891  1187760 1172402   724774 383758
```