Dhananjay Bhavsar
Asst. Professor
DIMR

Business Intelligence

- Business intelligence (BI) is a broad category of applications, technologies, and processes for: gathering, storing, accessing, andanalyzing data to help business users make better decisions.
- This definition is broad.
- BI encompasses not only applications, but also technologies and processes.

Business Intelligence

- It includes not only “getting data out” (through tools and applications), but also getting “data in” (to a data mart or warehouse).
- It should be pointed out that some authors use the BI term to refer to “getting data out” and data warehousing as “getting data in.”
- And there are some authors who use the data warehousing term to refer to both “getting data in” and “getting data out,” much like we are using the BI term.
- The good news is that the differing terminology does not normally cause confusion because of the context in which the terms are used.

- Things Are Getting More Complex
- Organizations are finding business value in capturing, storing, and analyzing new kinds of data, such as social media, machine sensing, and clickstream.
- Many companies are performing new kinds of analytics (sentiment analysis), to better and more quickly understand and respond to what customers are saying about them and their products.
- The cloud, and appliances are being used as data stores
- Advanced analytics are growing in popularity and importance

- Organizations are finding business value in capturing, storing, and analyzing new kinds of data, such as socia media, machine sensing, and clickstream.
- three Vs -- volume, variety, and velocity – this kind of data is often called Big Data.
- Many companies are performing new kinds of analytics, such as sentiment analysis to better and more quickly understand and respond to what customers are saying about them and their products.
- There are changes in how this data is stored and analyzed. For example, some companies store and analyze data in the cloud.
- The open source software Hadoop and Map Reduce from Apache Software is being used to store and analyze massive amounts of multistructured data (as opposed to the more structured data maintained in RDMS).

What is Big Data?

- Meet customer needs
- Optimize operations
- Optimize infrastructure
- Find new sources of revenue
- Can reveal more patterns and anomalies
- IBM estimates that by million jobs will be created globally to support big data1.9 million of these jobs will be in the United States

What is Big Data?

- Information from multiple internal and external sources:
- Transactions
- Social media
- Enterprise content
- Sensors
- Mobile devices
- Companies leverage data to adapt products and services to:

- **A variable is a unit of data collection whose value can y.** When collecting or gathering data we collect data from individuals cases on particular variables.
- A variable is a unit of data collection whose value can vary.
- Variables can be defined into types according to the level of mathematical scaling that can be carried out on the data.
- There are four types of data or levels of measurement:
- When thinking about research we are looking at gathering knowledge through some form of observation.
- Research in the scientific sense

- we observe something we understand that observation in terms of its attributes for example, colour or texture, and we look at how that attribute varies across our observations.
- So, when we collect or gather data we collect information from individual cases (people, organisations, businesses and so on) about particular variables.
- Level of mathematical scaling?
- The level of mathematical precision with which the values of a variable can be expressed.

1. Nominal
2. Ordinal
3. Interval
4. Ratio

- Nominal dataNominal or categorical data is data that comprises of categories that cannot be ranked or ordered – each category is just different.
- The categories available cannot be placed in any order and no judgement can be made about the relative size or distance from one category to another.
- What does this mean?
- No mathematical operations can be performed on the data relative to each other.
- Therefore, nominal data reflect qualitative differences rather than quantitative ones.

- **Nominal data Examples:**
- **What is your gender? (please tick)**
- **Male Female**

  Did you enjoy the film? (please tick)YesNo

- Nominal data Systems for measuring nominal data must ensure that each category is mutually exclusive and the system of measurement needs to be exhaustive.
- Variables that have only two responses i.e. Yes or No, are known as dichotomies.
- Mutually exclusive: each observation (person, case, score) cannot fall into more than one category.
- Exhaustive: the system of categories system should have enough categories for all the observations.

- Ordinal data
- Ordinal data is data that comprises of categories that can be rank ordered.
- Similarly with nominal data the distance between each category cannot be calculated but the categories can be ranked above or below each other.
- What does this mean?
- Can make statistical judgements and perform limited maths.

- data works with the same assumptions as nominal data but is more complex in that here, whilst still comprising of categories, now the categories themselves can be rank-ordered i.e. one category has more or less of some underlying quality than being in another category.
- What does this mean?
- We can make statistical judgements about the relative position of one value to another and can perform limited mathematical calculations.

- Ordinal data
- Example:
- How satisfied are you with the level of service you have received? (please tick)
- Very satisfied
- Somewhat satisfied
- Neutral
- Somewhat dissatisfied
- Very dissatisfied

- **interval and ratio data**
  Both interval and ratio data are examples o
  scale data.
- Scale data:
- data is in numeric format ($50, $100, $150)data that can be measured on a continuous scale the distance between each can be observed and as a result measured the data can be placed in rank order..

- **interval and ratio data**
  Ordinal data can be ranked – similarly interval and ratio data also operate on a mechanism of scale.
- The key difference is that with interval and ratio data the data is numerical and has numerically equal distances between each category.
- For example,
- a 5cm difference in height between 150cm and 155cm is the same as a 5cm difference between 15cm and 20cm.
- Imagine a 30cm ruler – it shows equal distance between each mark i.e. 10 1mm markers between each cm.

- Interval data
- Interval data measured on a continuous scale and has no true zero point.
- Examples:
- Time – moves along a continuous measure or seconds, minutes and so on and is without a zero point of time.
- Temperature – moves along a continuous measure of degrees and is without a true zero.
- Temperature and true zero – whilst there is a zero degrees this is a measure of temperature!

- Ratio data
- Ratio data measured on a continuous scale and does have a true zero point.
- Examples:
- Age
- Weight
- Height

- **Descriptive Analytics**
  Descriptive analytics,
- such as reporting/OLAP,
- dashboards, and data visualization,
- have been widely used for some time.
- They are the core of traditional BI.
- Descriptive analytics, such as reporting/OLAP, dashboards, and data visualization,
- have been widely used for some time.
- They are the core of traditional BI.
- What has occurred?
- Descriptive analytics, such as data visualization, is important in helping users interpret the output from predictive and predictive analytics.

- **Predictive Analytics What will occur?** Algorithms for predictive analytics, such as regression analysis, machine learning, and neural networks, have also been around for some time.
- Prescriptive analytics are often referred to as advanced analytics.
- What will occur?
- Algorithms for predictive analytics, such as regression analysis, machine learning, and neural networks, have also been around for some time. Recently, however, software products have made them much easier to understand and use.

- **Predictive Analytics What will occur?**
  They have also been integrated into specific applications, such as for campaign management.
- Marketing is the target for many predictive analytics applications.
- Descriptive analytics, such as data visualization, is important in helping users interpret the output from predictive and predictive analytics. Prescriptive analytics are often referred to as advanced analytics.
- Marketing is the target for many predictive analytics applications.
- Descriptive analytics, such as data visualization, is important in helping users interpret the output from predictive and prescriptive analytics.

- **Prescriptive Analytics**
  Prescriptive analytics are often referred to as advanced analytics.
- Regression analysis, machine learning, and neural networks Often for the allocation of scarce resourcesWhat should occur?
- Prescriptive analytics provide an optimal solution, often for the allocation of scarce resources.
- They, too, have been in academia for a long time but are now finding wider use in practice.
- For example, the use of mathematical programming for revenue management is common for organizations that have “perishable” goods (e.g., rental cars, hotel rooms, airline seats).

- **Prescriptive Analytics**
  Harrah's has been using revenue managem
  for hotel room pricing for some time.
- For example, the use of mathematical programming for revenue management is common for organizations that have "perishable" goods (e.g., rental cars, hotel rooms, airline seats).
- Harrah's has been using revenue management for hotel room pricing for some time.

- **Organizational Transformation**
  Brought about by opportunity or necessity The firm adopts a new business model enabled by analytics
- Analytics are a competitive requirement
- This is the most interesting target. It occurs when a company either sees an opportunity or faces a problem that requires a new business model that can only be executed using analytics.
- To an extent, analytics is the business. Without analytics, the business model cannot be execute and the company cannot compete.

- For BI-based organizations, the use of BI/analytics is a requirement for successfully competing in the marketplace.In some industries, the use of advanced analytics has moved beyond a “nice to have” to being a requirement.

- 2013 Academic Research
- A 2011 TDWI report on Big Data Analytics found that 85% of respondents indicated that their firms would be using advanced analytics within three years
- A 2011 IBM/MIT Sloan Management Review research study found that top performing companies in their industry are much more likely to use analytics rather than intuition across the widest range of possible decisions.

- The relationship between the use of data and analytics in decision making and a variety of organizational performance measures is described in a 2011 study by Brynjolfsson, Hitt, and Kim in the Social Science Research Network (SSRN).
- Also, A 2010 IBM/MIT Sloan Management Review research study found that top performing companies in their industry are much more likely to use analytics rather than intuition across the widest range of possible decisions.
- A 2011 TDWI report on Big Data Analytics found that 85% of respondents indicated that their firms would be using advanced analytics within three years.
- A 2011 IDC study found that analytics is one of the top two IT priorities for this year.

- **Conditions that Lead to Analytics-based Organizations**
  The nature of the industrySeizing an opportunity Responding to a problem

- Complex Systems
- Tackle complex problems and provide individualized solutions Products and services are organized around the needs of individual customers Dollar value of interactions with each customer is high There is considerable interaction with each customer
- Examples: IBM, World Bank, Halliburton

- Volume Operations
- Serves high-volume markets through standardized products and services
- Each customer interaction has a low dollar value
- Customer interactions are generally conducted through technology rather than person-to-person
- Are likely to be analytics-based
- Examples: Amazon.com, eBay, Hertz
- Analytics are critical to high volume companies competing in the marketplace.

- The Nature of the Industry: Online Retailers
- BI Applications
- Analysis of clickstream data
- Customer profitability analysis
- Customer segmentation analysis
- Product recommendations
- Campaign management
- Pricing
- Forecasting

- Online retailers like Amazon.com and Overstock.com are great examples of high volume operations who rely on analytics to compete.
- As soon as you enter, their sites a cookie is placed on your PC and all clicks are recorded.
- Based on your clicks and any search terms, recommendation engines decide what products to display.
- After you purchase an item, they have additional information that is used in marketing campaigns.
- Customer segmentation analysis is used in deciding what promotions to send you.
- How profitable you are influences how the customer care center treats you.
- A pricing team helps set prices and decides what prices are needed to clear out merchandise.
- Forecasting models are used to decide how many items to order for inventory.
- Dashboards monitor all aspects of organizational performance.

- The Nature of the Industry
- Online retailers like Amazon.com and Overstock.com are high volume operations who rely on analytics to compete.
- When you enter their sites a cookie is placed on your PC and all clicks are recorded.
- Based on your clicks and any search terms, recommendation engines decide what products to display.
- After you purchase an item, they have additional information that is used in marketing campaigns.
- Customer segmentation analysis is used in deciding what promotions to send you..

- The Nature of the Industry
- How profitable you are influences how the customer care center treats you.
- A pricing team helps set prices and decides what prices are needed to clear out merchandise.
- Forecasting models are used to decide how many items to order for inventory.
- Dashboards monitor all aspects of organizational performance

- Online retailers like Amazon.com and Overstock.com are great examples of high volume operations who rely on analytics to compete.
- As soon as you enter, their sites a cookie is placed on your PC and all clicks are recorded.
- Based on your clicks and any search terms, recommendation engines decide what products to display.
- After you purchase an item, they have additional information that is used in marketing campaigns.
- Customer segmentation analysis is used in deciding what promotions to send you.
- How profitable you are influences how the customer care center treats you.
- A pricing team helps set prices and decides what prices are needed to clear out merchandise.
- Forecasting models are used to decide how many items to order for inventory.
- Dashboards monitor all aspects of organizational performance.

- Knowledge Requirements for Advanced Analytics
- Business Domain
- Video
- Data
- Modeling
- Choosing the right data to include in models is important.
- Predictive analytics should not be searching for a diamond in a coal mine.
- You will get too many spurious findings.
- Important to have some thoughts as to what variables might be related.
- Domain knowledge is necessary to understand how they can be used.
- Consider the story of the relationship between beer and diapers in the market basket of young males in convenience stores.
- You still have to decide (or experiment to discover) whether it is better to put them together or spread them across the store (in the hope that other things will be bought while walking the isles).

- Analytics requires an understanding of what data is available, how to access it, and how to manipulate it.
- Data is necessary for first building models and later testing them.
- Choosing the right data to include in models is important. Predictive analytics should not be searching for a diamond in a coal mine.
- You will get too many spurious findings. Rather, it is important have some thoughts as to what variables might be related.
- And once you have findings, domain knowledge is necessary to understand how they can be used.

- Consider the hoary story of the relationship between beer and diapers in the market basket of young males in convenience stores.
- You still have to decide (or experiment to discover) whether it is better to put them together or spread them across the store (in the hope that other things will be bought while walking the isles).
- Domain knowledge is important here. And finally, it is necessary to understand the models being used. At a minimum, this requires training in multivariate statistics.

- **Visualization: A Definition**
  Visualize: to form a mental model or mental image of something Visualization is a human cognitive activity, not something that a computer does

- Visualization
- Refers to the innovative use of images and interactive technology to explore large, high- density datasets
- Help users see patterns and relationships that would be difficult to see in text lists
- Rich graphs, charts
- Dashboards
- Maps
- Increasingly is being used to identify insights into both structured and unstructured data for such areas as
- operational efficiencies
- profitability
- strategic planning
- Video Tableau

- Acquisition of Insight
- Many people and institutions possess data that may ‘hide’ fundamental relations
- Realtors
- Bankers
- Air Traffic Controller
- Fraud investigators
- Engineers
- They want to be able to view some graphical representation of that data, maybe interact with it, and then be able to say…….ahha!

- Example: Fraud Detection
- The Serious Fraud Office (SFO) suspected mortgage fraud
- The SFO provided 12 filing cabinets of data
- After 12 person years a suspect was identified
- The suspect was arrested, tried and convicted

- Example: Fraud Detection continued
- The data was supplied in electronic form
- A visualization tool (Netmap) was used to examine the data
- After 4 person weeks the same suspect was identified
- A master criminal behind the fraud was also identified

- Is Information Visualization Useful?
- Drugs and Chips
- Texas Instruments
- Manufactures microprocessors on silicon wafers that are routed through 400 steps in many weeks. This process is monitored, gathering 140,000 pieces of information about each wafer. Somewhere in that heap of data can be warnings about things going wrong. Detect a bug early before bad chips are made. TI uses visualization tools to aid in this detection
- Eli Lilly
- Has 1500 scientists using an advanced information visualization tool (Spotfire) for decision making. “With its ability to represent multiple sources of information and interactively change your view, it’s helpful for homing in on specific molecules and deciding whether we should be doing further testing on them”
- Sheldon Ort of Eli Lilly, speaking to Fortune

- The cholera epidemic, London 1845
- Dr. John Snow, medical officer for London, investigated the cholera epidemic of 1845 in Soho. He noted that the deaths, indicated by points, tended to occur near the Broad Street pump. Closure of the pump coincided with a reduction in cholera.

- Challenger Disaster
- On 28th January 1986 the space shuttle Challenger exploded, and seven astronauts died, because two rubber O-Rings leaked.
- The previous day, engineers who designed the rocket opposed the launch, concerned that the O-Rings would not seal at the forecast temperature (25 to 29oF).
- After much discussion, the decision was taken to go ahead.
- Cause of the accident:
- An inability to assess the link between cool temperature and O-Ring damage on earlier flights.
- Many charts poorly presented

- Introduction to Information Visualization - Fall 2012
- Examples
- Geo data
- mapping
- Demo
- Introduction to Information Visualization - Fall 2012

- Introduction to Information Visualization - Fall 2012
- Examples
- Treemap
- Demo
- Introduction to Information Visualization - Fall 2012

- Introduction to Information Visualization - Fall 2012
- Examples
- Population
- “Trendalyzer”
- Demo
- Introduction to Information Visualization - Fall 2012

- Analytics Help the Cincinnati Zoo Know Its Customers
- What management, organization, and technology factors were behind the Cincinnati Zoo losing opportunities to increase revenue?
- Why was replacing legacy point-of-sale systems and implementing a data warehouse essential to an information system solution?
- How did the Cincinnati Zoo benefit from business intelligence? How did it enhance operational performance and decision making? What role was played by predictive analytics?
- Visit the IBM Cognos Web site and describe the business intelligence tools that would be the most useful for the Cincinnati Zoo.

Review: Getting started with R Solutions Exercise Sheet 2

Part I: Data types and structures in R

- What are types and structures Vectors and
- factors
- Lists
- Matrices
  Data frames

Part II: Accessing data

Data types and structures

Types

- logical
- numeric
- integer
- characte
- r
  complex

Structures

- vecto
- r
- factor
- list
- matri
  x
  data frame

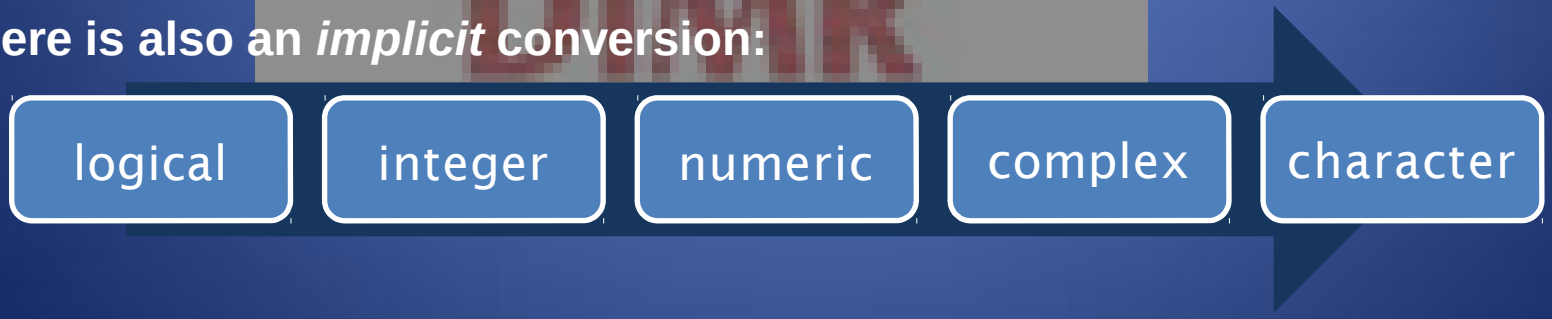
Data types

Data type	Description	Examples
logical	TRUE or FALSE	TRUE, FALSE
integer	whole numbers	-5L, 1L, 7L
numeric	integers and real numbers	5, -2, 3.1415, sqrt(2)
complex	complex numbers	2.1+3i, 5+0i
character	character string	“This is text”, “5”

Types can be *explicitly* converted:

`as.logical(), as.integer(), as.numeric(), as.complex(), as.character()`

There is also an *implicit* conversion:



Data types

Internal representation of TRUE and FALSE:
as.integer(TRUE); as.integer(FALSE)

You can check for a data type:
`is.logical(), is.integer(), is.numeric(), is.complex(), is.character()`

Find out the data type
`mode()`

Find out the class `class()`
`typeof()`

Try yourself:

```
x < 5
is.integer(x)
is.numeric(x)
x < 5L
is.integer(x)
is.numeric(x)
```

What is a vector?

- A vector is a collection of values that all have the same data
- type One-dimensional

Examples:
(-2, 3.4, 3.75, 5.2, 6)
(TRUE, FALSE, TRUE, TRUE, FALSE)
(“blue”, “green”, “red”, “red”)

You can create a vector with different functions:

`c()`	function to combine indvididual
`seq()`	values to create more complex
`rep()`	sequences
	to create replicates of values

Examples

```
### c() function
c(2, 7, 8, 12, 3, 25)
# you do not need the spaces
c(2,7,8,12,3,25)
# you can assign a vector to a variable
myVector < c(2, 7, 8, 12, 3, 25)
# you can indicate ranges with the : operator
c(2:5, 3:6)
# you can create a vector with named values
Students < c(master="Bob", bachelor="Lucy", phd="Jenny")

### seq() function
seq(from=1, to=10)
seq(1,10)
seq(from=1, to=10, by=2)
seq(1,10,2)
1:10
```

Examples

```
### rep() function
rep(1,4)
rep(4:6, 3)
rep(1:4, each=2)
rep(1:4, times=2, each=2)
```

Examples

```
### rep() function
rep(1,4)
rep(4:6, 3)
rep(1:4, each=2)
rep(1:4, times=2, each=2)
```

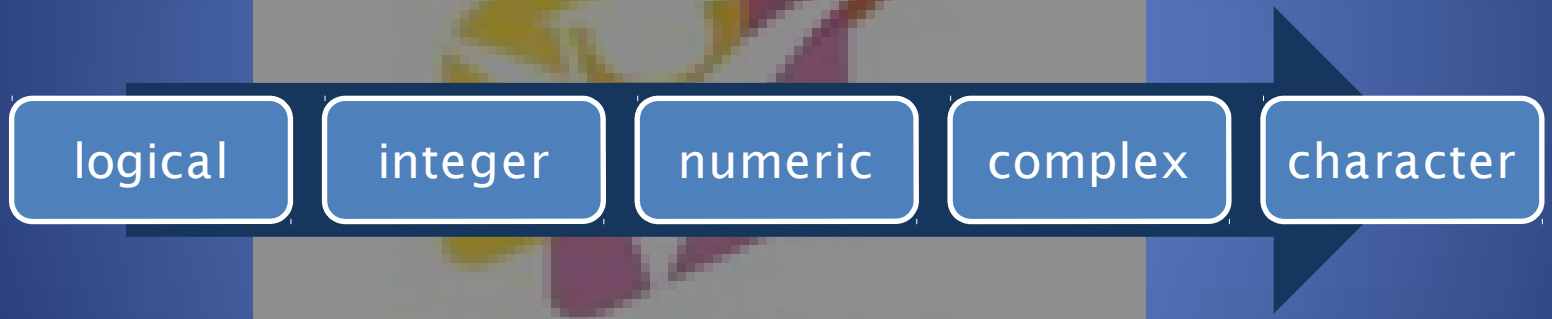
Try yourself:

```
x < c(5, "b")
y < c(FALSE, 3)
mode(x
)
mode(y
)
```

Data type conversion

- Vectors may only have one type
- When combining different types, R will coerce a vector into the most flexible type

Coercion rule (also: implicit type conversion)



Operations on vectors

You assess elements of a vector with the [ ]-operator:

```
x < c(12, 15, 13, 17, 11)
x[4]
x[3:5]
x[2]
x[(3:5)]
```

Standard operations on vectors are element-by-element:

```
c(2, 5, 3) + c(4, 2, 7)
2 + c(2, 5, 3)
c(2, 5, 3) ^ 2
```

Operations on vectors

```
sum(5:7)
prod(4:6
)    x <
1:5
x[3:5]
x[2]
x > 3
```

Some useful functions (x is an example vector)

`class(x)`	returns class/type of vector x
`length(x)`	returns the total number of elements
`x[length(x)]`	returns last value of vector x
`rev(x)`	returns reversed vector
`sort(x)`	returns sorted vector
`unique(x)`	returns vector without multiple elements

Factors

- A factor is used to store categorical
- data Can only contain predefined
- categories Can be ordered and unordered

Examples:

("yes", "no", "no", "yes", "yes")

("male", "female", "female", "male")

("small", "large", "small", "medium")

Factors

Factors can be created using **factor()**:

```
size < factor(c("small", "large", "small",
"medium"))

size

[1] small  large  small  medium
Levels: large medium small # unordered
factor
```

Note:
Quotes, such as "male" are not shown and levels are printed

The levels of a factor can be displayed using **levels()**

Lists

- A collection of data structures

- A list can encompass any data types, including lists
- Objects can have different lengths
- You can construct lists by using **list()**

- Almost all functions (e.g., t-test, linear regression, etc.) in R produce output that is stored in a list

Lists

Example of a list:

```
myList < list(1:3, c("a", "b"), c(TRUE,
FALSE, TRUE))
str(myList)

List of 3
 $ : int [1:3] 1 2 3
 $ : chr [1:2] "a" "b"
 $ : logi [1:3] TRUE FALSE TRUE
```

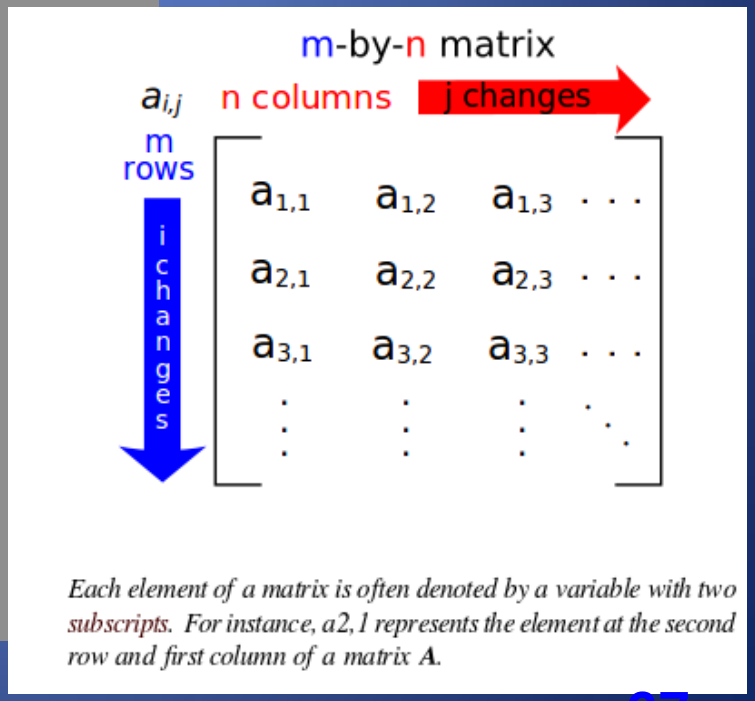
Matrix - Definition

Wikipedia:

In mathematics, a ***matrix*** (plural matrices) is a rectangular array of numbers, symbols, or expressions arranged in ***rows*** and ***columns***. The individual items in a matrix are called its ***elements*** or ***entries***. An example of a matrix with 2 rows and 3 columns is:

$$\begin{bmatrix} 1 & 9 & 13 \\ 20 & 5 & 6 \end{bmatrix}$$

2-by-3 matrix
2x3 matrix



Matrices - Basics

- You can create matrices by:

1. matrix()
2. converting a vector into a matrix
3. binding together vectors

matrix()	creates a matrix by specifying rows and colu
dim()	sets dimensions to a vector
cbind	combines columns
rbind	combines rows

- These functions are useful:

How does matrix() work?

Function of interest: matrix()

Which arguments can be used with this function?

```
?matrix()
```

matrix {base}

Matrices

Description

matrix creates a matrix from the given set of values.

as.matrix attempts to turn its argument into a matrix.

is.matrix tests if its argument is a (strict) matrix.

Usage

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

```
matrix(data=(1:8),nrow=2, ncol=3)
```

Matrices - Basics

Examples:

```
m < matrix(data = 1:8, nrow = 4, ncol = 2)
m
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
# indexing is rowbycolumn
m[2:3,1:2]
     [,1] [,2]
[1,]    2    6
[2,]    3    7
```

Try yourself:

```
m[3,2]
m[2,]
m[,2]
m[2:3,1:2]
```

How does matrix() work?

Function of interest: matrix()

Which arguments can be used with this function?

matrix(data=NA, nrow=1, ncol=1, byrow=FALSE, dimnames=NULL)

What does this mean? Let's try:

```
matrix(data=(1:6), nrow=2, ncol=3)
matrix(ncol=3, data=(1:6), nrow=2)
matrix(1:6, 2, 3)
matrix(1:6, 3, 2)
matrix(1:6, ncol=3, nrow=2)
matrix(1:6, nr=2, nc=3)
matrix(d=(1:6),nr=2, nc=3)
```

Error in matrix(d = (1:6), nr = 2, nc = 3) :

argument 1 matches multiple formal arguments

Matrices – Examples

```
z <   as.matrix(1:6)
z
      [,1]
[1,]     1
[2,]     2
[3,]     3
[4,]     4
[5,]     5
[6,]     6
```

Matrices – Examples

Examples:

```
cbind(1:3,5:7)
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
rbind(1:3,5:7,10:12)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    5    6    7
[3,]   10   11   12
```

More in the exercises!

Data frames

- A collection of vectors that are of equal length
- Two-dimensional, arranged in **rows** and **columns**
- Columns can contain vectors of different data types

  BUT: WITHIN a column, every cell must be the same type of data!
- Used to represent entire data sets

Data frames can be created using the **function**:

`data.frame()` creates a data frame object from a set of vectors

Data frames

Example of data.frame()

```
df <  data.frame(ID = 1:3, Sex = c("F", "F", "M"),
Mass        = c(17, 18,
                18))

df < data.frame(
          ID = 1:3,                  # data elements first column
          Sex = c("F",  "F", "M"), # data elements second column
          Mass = c(17,      18, 18)) data elements third column
                                     # column names
                                     #
```

Note:

Columns must be of same length

Remember: R uses the equal sign to specify named arguments

Data frames

Example of data.frame()

```
df < data.frame(ID = 1:3,
                Sex = c("F", "F", "M"),
                Mass = c(17, 18, 18)
               )
```

```
    ID Sex   Mass
  1  17 F
  2  2  F     18
  3  3  M     18
```

Data frames

Important: data.frame() automatically turns strings into factors

```
df_2 < data.frame(x = 1:3, y = c("a", "b", "c"))
str(df_2)        # str() displays internal structure of an R
                 # object

'data.frame':   3 obs. of   2 variables:
 $ x: int    1  3
 $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```

Argument stringsAsFactors = FALSE prevents this behaviour

Data frames

Important: data.frame() automatically turns strings into factors

```
df_2 < data.frame(x = 1:3, y = c("a", "b", "c"),
stringsAsFactors = FALSE)

str(df_2)

'data.frame':   3 obs. of  2 variables:
 $ x: int  1 2 3
 $ y: chr w/ 3 levels "a","b","c"
```

Data frames

- Creating a data frame by hand takes a lot of time
- Also, typing invites typos and errors
- You should avoid typing large data sets into R by hand

➡ Import entire data sets into R

Usually data frames are imported using the functions: **read.csv()** or **read.table()**

More tomorrow!

Indexing

Indexing by integer vector

You can use **x[ ]** to look up a single element or multiple elements in a vector

```
X <   (10:22)
[1]     10 11 13 14 15 1   1   1   1   20 21 22
           12            6   7   8   9

x[7                    x[1:4
]    16                ]      10 11 12 13
[1]                    [1]
x[c(1,4,9,12)]
[1] 10 13 18
21
```

Indexing

Indexing by integer vector

You can also use **negative integers** to return a vector consisting of all elements except the specified elements:

```
x[-2:-7]                    # excludes elements 2 to
[1] 10 17 18 19 20 21       7
22
```

Indexing

Indexing by integer vector

In multidimensional data structures (e.g. matrices and data frames) an element at the mth row, n^{th} column can be accessed by the expression **x[m, n]**

```
z< matrix(data=c(10:21), nrow = 3, ncol = 4)

z          [,1] [,2] [,3] [,4]
     [1,]   10   13   16   19
     [2,]   11   14   17   20
     [3,]   12   15   18   21

z[2, 3]                    # indexing is row by column
[1] 17
```

Indexing

Indexing by integer vector

The entire mth row can be extracted by the expression **x[m,]**

```
Z < matrix(data=c(10:21), nrow = 3, ncol = 4)

z[2  ]
,
[1]  1 14 17 20
```

The entire nth column can be extracted by the expression **x[, n]**

```
z < matrix(data=c(10:21), nrow = 3, ncol = 4)

z[ , 3]

[1] 16 17 18
```

Indexing

Indexing by integer vector

Multiple rows or columns can be extracted by:

```
z < matrix(data=c(10:21), nrow = 3, ncol = 4)

z[1:2, 1:2]

     [,1] [,2]
[1,]   10   13
[2,]   11   14

z[1:2, c(1, 3)]

     [,1] [,2]
[1,]   10   16
[2,]   11   17
```

Indexing

Indexing by name

You can index an element by name using the $ notation

```
df < data.frame(ID = 1:3, Sex = c("F", "F", "M"),
Mass = c(17, 18, 18))

str(df)

'data.frame':  3 obs. of  3 variables:
$ ID  : int  1 2 3
$ Sex : Factor w/ 2 levels "F","M": 1 1 2
$ Mass: num  17 18 18
```

Indexing

Indexing by name

You can index an element by name using the $ notation

```
df <  data.frame(ID = 1:3, Sex = c("F", "F", "M"),
Mass       = c(17, 18,
                18))

df$Mass

[1] 17 18
18
```

Indexing

Indexing by name

You can also use the single-bracket notation [ ] to index a set of elements by name

```
df < data.frame(ID = 1:3, Sex = c("F", "F", "M"),
Mass = c(17, 18, 18))

df[c("Sex",

    "Mass")]  Sex
1   F      17
2   M      18
3   M      18
```

Summary of data structures in R

	Single type	multiple types
1D	Vector	List
2D	Matrix	Data frame

Vectors

- A vector is an ordered collection of objects of the same type
- The function `c(...)` concatenates its arguments to form a vector
- To create a patterned vector
  - `:` Sequence of integers
  - `seq()` General sequence
  - `rep()` Vector of replicated elements

```
> v1 <- c(2.5, 4, 7.3, 0.1)
> v1
[1] 2.5 4.0 7.3 0.1

> v2 <- c("A", "B", "C", "D")
> v2
[1] "A" "B" "C" "D"

> v3 <- -3:3
> v3
[1] -3 -2 -1 0 1 2 3
```

```
> seq(0, 2, by=0.5)
[1] 0.0 0.5 1.0 1.5 2.0

> seq(0, 2, len=6)
[1] 0.0 0.4 0.8 1.2 1.6 2.0

>rep(1:5, each=2)   [1]
1 1 2 2 3 3 4 4 5 5

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

Reference Elements of a Vector

- Use [ ] with a vector/scalar of positions to reference elements of a vector
- Include a minus sign before the vector/scalar to remove elements

```
> x <- c(4, 7, 2, 10, 1, 0)
> x[4]
[1] 10
>x[1:3]
[1] 4 7 2
>x[c(2,5,6)]
[1] 7 1 0
> x[-3]
[1] 4 7 10 1 0
>x[-c(4,5)]
[1] 4 7 2 0
> x[x>4]
[1] 7 10
>x[3] <- 99
>x
[1] 4 7 99 10 1 0
```

which() and match()

- Additional functions that will return the indices of a vector
  - `which()` Indices of a logical vector where the condition is TRUE
  - `which.max()` Location of the (first) maximum element of a numeric vector
  - `which.min()` Location of the (first) minimum element of a numeric vector
  - `match()` First position of an element in a vector

```
> x <- c(4, 7, 2, 10, 1, 0)
> x>=4
[1] TRUE TRUE FALSE TRUE FALSE FALSE
>which(x>=4)
[1] 1 2 4
> which.max(x)
[1] 4
>x[which.max(x)]
[1] 10
>max(x)
[1] 10
```

```
> y <- rep(1:5, times=5:1)
> y
[1] 1 1 1 1 1 2 2 2 2 3 3 3 4 4 5
>match(1:5, y)
[1] 1 6 10 13 15
>match(unique(y), y)
[1] 1 6 10 13 15
```

Vector Operations

- When vectors are used in math expressions the operations are performed element by element

```
> x <- c(4,7,2,10,1,0)
> y <- x^2 + 1
> y
[1]   17   50    5  101    2    1
> x*y
[1]   68  350   10 1010    2    0
```

Useful Vector Functions

sum(x)	prod(x)	Sum/product of the elements of x
cumsum(x)	cumprod(x)	Cumulative sum/product of the elements of x
min(x)	max(x)	Minimum/Maximum element of x
mean(x)	median(x)	Mean/median of x
var(x)	sd(x)	Variance/standard deviation of x
cov(x,y)	cor(x,y)	Covariance/correlation of x and y
range(x)		Range of x
quantile(x)		Quantiles of x for the given probabilities
fivenum(x)		Five number summary of x
length(x)		Number of elements in x
unique(x)		Unique elements of x
rev(x)		Reverse the elements of x
sort(x)		Sort the elements of x
which()		Indices of TRUEs in a logical vector
which.max(x)	which.min(x)	Index of the max/min element of x
match()		First position of an element in a vector
union(x, y)		Union of x and y
intersect(x, y)		Intersection of x and y
setdiff(x, y)		Elements of x that are not in y
setequal(x, y)		Do x and y contain the same elements?

Matrices

- A matrix is just a two-dimensional generalization of a vector
- To create a matrix,

```
matrix(data=NA, nrow=1, ncol=1, byrow = FALSE, dimnames = NULL)
```

`data` a vector that gives data to fill the matrix; if `data` does not have enough elements to fill the matrix, then the elements are recycled.

`nrow` desired number of rows

`ncol` desired number of columns

`byrow` if `FALSE` (default) matrix is filled by columns, otherwise by rows

`dimnames` (optional) list of length 2 giving the row and column names respectively, list names will be used as names for the dimensions

```
> x <- matrix(c(5,0,6,1,3,5,9,5,7,1,5,3), nrow=3, ncol=4, byrow=TRUE,
+             dimnames=list(rows=c("r.1", "r.2", "r.3"),
+                           cols=c("c.1", "c.2", "c.3", "c.4")))
> x
     cols
rows c.1 c.2 c.3 c.4
  r.1   5   0   6   1
  r.2   3   5   9   5
  r.3   7   1   5   3
```

Reference Elements of a Matrix

- Reference matrix elements using the [ ] just like with vectors, but now with 2-dimensions

```
> x <- matrix(c(5,0,6,1,3,5,9,5,7,1,5,3), nrow=3, ncol=4, byrow=TRUE)
> x
     [,1] [,2] [,3] [,4]
[1,]    5    0    6    1
[2,]    3    5    9    5
[3,]    7    1    5    3
> x[2,3]                    # Row 2, Column 3
[1] 9
> x[1,]                     # Row 1
[1] 5 0 6 1
>x[,2]                      # Column 2
[1] 0 5 1
> x[c(1,3),]                # Rows 1 and 3, all Columns
     [,1] [,2] [,3] [,4]
[1,]    5    0    6    1
[2,]    7    1    5    3
```

Reference Elements of a Matrix

- We can also reference parts of a matrix by using the row or column names
- Sometimes it is better to reference a row/column by its name rather than by the numeric index. For example, if a program adds or permutes the columns of a matrix then the numeric index of the columns may change. As a result you might reference the wrong column of the new matrix if you use the old index number. However the name of each column will not change.
- Reference matrix elements using the [ ] but now use the column or row name, with quotations, inplace of the index number
- You don't have to specify the names when you create a matrix. To get or set the column, row, or both dimension names of `A`:

```
colnames(A)
rownames(A)
dimnames(A)
```

- Can also name the elements of a vector, `c("name.1"=1, "name.2"=2)`. Use the function `names()` to get or set the names of vector elements.

Reference Elements of a Matrix

```
> N<- matrix(c(5,8,3,0,4,1), nrow=2, ncol=3, byrow=TRUE)
> colnames(N) <- c("c.1", "c.2", "c.3")
> N
     c.1 c.2 c.3
[1,]   5   8   3
[2,]   0   4   1
>N[,"c.2"]                    # Column named "c.2"
[1] 8 4
> colnames(N)
[1] "c.1" "c.2" "c.3"
> M <- diag(2)
> (MN<- cbind(M, N))            # Placing the expression in parentheses
        c.1 c.2 c.3             # will print the result
[1,]   1   0   5   8   3
[2,]   0   1   0   4   1
>MN[,2]                       # Column 2
[1] 0 1
>MN[,"c.2"]                   # Column named "c.2"
[1] 8 4
```

Matrix Operations

- When matrices are used in math expressions the operations are performed element by element.
- For matrix multiplication use the `%*%` operator
- If a vector is used in matrix multiplication, it will be coerced to either a row or column matrix to make the arguments conformable. Using %*% on two vectors will return the inner product (%o% for outer product) *as a matrix* and not a scalar. Use either `c()` or `as.vector()` to convert to a scalar.

```
> A<- matrix(1:4, nrow=2)
> B<- matrix(1, nrow=2, ncol=2)
> A*B
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> A%*%B
     [,1] [,2]
[1,]    4    4
[2,]    6    6
```

```
> y <- 1:3
> y%*%y
     [,1]
[1,]   14
> A/(y%*%y)
Error in A/(y%*%y):non-conformable arrays
> A/c(y%*%y)
           [,1]      [,2]
[1,] 0.07142857 0.2142857
[2,] 0.14285714 0.2857143
```

Useful Matrix Functions

`t(A)`	Transpose of `A`
`det(A)`	Determinate of `A`
`solve(A, b)`	Solves the equation `Ax=b` for `x`
`solve(A)`	Matrix inverse of `A`
`MASS::ginv(A)`	Generalized inverse of `A`(MASS package)
`eigen(A)`	Eigenvalues and eigenvectors of `A` Choleski
`chol(A)`	factorization of `A`
`diag(n)`	Create a `n`×`n` identity matrix
`diag(A)`	Returns the diagonal elements of a matrix `A`
`diag(x)`	Create a diagonal matrix from a vector `x`
`lower.tri(A),upper.tri(A)`	Matrix of logicals indicating lower/upper triangular matrix
`apply()`	Apply a function to the margins of a matrix
`rbind(...)`	Combines arguments by rows
`cbind(...)`	Combines arguments by columns and
`dim(A)`	Dimensions of `A`
`nrow(A), ncol(A)`	Number of rows/columns of `A`
`colnames(A), rownames(A)`	Get or set the column/row names of `A`
`dimnames(A)`	Get or set the dimension names of `A`

apply()

- The `apply()` function is used for applying functions to the margins of a matrix, array, or dataframes.

```
apply(X, MARGIN, FUN, ...)
```

X A matrix, array or dataframe

MARGIN Vector of subscripts indicating which margins to apply the function to
1=rows, 2=columns, c(1,2)=rows and columns

FUN Function to be applied

... Optional arguments for FUN

- You can also use your own function (more on this later)

```
> (x <- matrix(1:12, nrow=3, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
>apply(x, 1, sum)        # Row totals
[1] 22 26 30
>apply(x, 2, mean)       # Column means
[1] 2 5 8 11
```

Example - Simulating Survival Data

- The `apply()` function is great for simulating survival data. Suppose we want to simulate $n = 10$ observations where, the event times T follow an exponential distribution with mean $\lambda = 0.25$ and the censoring times C are distributed uniformly from 0 to 1. Then the observed data is, $X = \min(T, C)$ and $\delta = I(T < C)$.

```
# Sample size
n = 10

# Generate event and censor times (look at the documentation to see #
how Rparameterizes the exponential distribution)
event <- rexp(n, 4)
censor <- runif(n)

# Select the minimum time and create an indicator variable
time <- apply(cbind(censor, event), 1, min)
index <- apply(cbind(censor, event), 1, which.min)-1

cbind(event, censor, time, index)          # Verify
data <- cbind(time, index)                 # Simulated dataset
```

Arrays

- An array is a multi-dimensional generalization of a vector
- To create an array,

```
array(data = NA, dim = length(data), dimnames = NULL)
```

`data` A vector that gives data to fill the array; if `data` does not have enough elements to fill the matrix, then the elements are recycled.

`dim` Dimension of the array, a vector of length one or more giving the maximum indices in each dimension

`dimnames` Name of the dimensions, list with one component for each dimension, either NULL or a character vector of the length given by dim for that dimension. The list can be named, and the list names will be used as names for the dimensions.

- Values are entered by columns
- Like with vectors and matrices, when arrays are used in math expressions the operations are performed element by element.
- Also like vectors and matrices, the elements of an array must all be of the same type (numeric, character, logical, etc.)

Arrays

- Sample 2 × 3 × 2 array,

```
> w<- array(1:12, dim=c(2,3,2),
            dimnames=list(c("A","B"), c("X","Y","Z"), c("N","M")))
> w
, , N

  XYZ
A 1 3 5
B 2 4 6

, , M

  X Y Z
A 7 9 11
B 8 10 12
```

Reference Elements of an Array

- Reference array elements using the [ ] just like with vectors and matrices, but now with more dimensions

```
> w<- array(1:12, dim=c(2,3,2),
            dimnames=list(c("A","B"), c("X","Y","Z"), c("N","M")))
> w[2,3,1]        # Row 2, Column 3, Matrix 1
[1] 6
> w[,"Y",]        # Column named "Y"
  N M
A 3 9
B 4 10
> w[1,,]          # Row 1
  N M
X 1 7
Y 3 9
Z 5 11
> w[1:2,,"M"]     # Rows 1 and 2, Matrix "M"
  X Y Z
A 7 9 11
B 8 10 12
```

Useful Array Functions

`apply()`	Apply a function to the margins of an array
`aperm()`	Transpose an array by permuting its dimensions
`dim(x)`	Dimensions of `x`
`dimnames(x)`	Get or set the dimension names of `x`

apply()

- We can use the `apply()` function for more then one dimension
- For a 3-dimensional array there are now three margins to apply the function to: 1=rows, 2=columns, and 3=matrices.

```
 # Column sums
>apply(w, 2, sum)
 XYZ
18  26 34

# Row and matrix sums
> apply(w, c(1,3), sum)
   NM
A   9 27
B 12  30
```

Lists

- A list is a general form of a vector, where the elements don't need to be of the same type or dimension.
- The function `list(...)` creates a list of the arguments
- Arguments have the form *name=value*. Arguments can be specified with and without names.

```
> x <- list(num=c(1,2,3), "Nick", identity=diag(2))
> x
$num
[1] 1 2 3

[[2]]
[1] "Nick"

$identity
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Reference Elements of a List

- Elements of a list can be referenced using [ ] as well as [[ ]] or $.

```
> x <- list(num=c(1,2,3), "Nick", identity=diag(2))

> x[[2]]                    # Second element of x
[1] "Nick"
>x[["num"]]                 # Element named "num"
[1] 1 2 3
> x$identity                # Element named "identity"
     [,1] [,2]
[1,]    1    0
[2,]    0    1
>x[[3]][1,]                 # First row of the third element
[1] 1 0
> x[1:2]                    # Create a sublist of the first two elements
$num
[1] 1 2 3

[[2]]
[1] "Nick"
```

Useful List Functions

`lapply()`	Apply a function to each element of a list, returns a list
`sapply()`	Same as `lapply()`, but returns a vector or matrix by default
`vapply()`	Similiar to `sapply()`, but has a pre-specified type of return value
`replicate()`	Repeated evaluation of an expression, useful for replicating lists
`unlist(x)`	Produce a vector of all the components that occur in `x`
`length(x)`	Number of objects in `x`
`names(x)`	Names of the objects in `x`

Example - lapply(), sapply(), vapply()

- First generate a list L of seven different vectors. Then calculate the five number summary of each vector in L using lapply(), sapply() and vapply().

```
# List of seven different vectors
L<- lapply(3:9, seq)

# Calculate the five number summary for each vector in L
lapply(L, fivenum)
sapply(L, fivenum)
vapply(L, fivenum, c(Min.=0, "1st Quart"=0, Median=0, "3rd Qu"=0, Max.=0))
```

- Since 3:9 is not a list, R calls as.list(3:9) which coerces the vector 3:9 to a list of length 7 where each number is an element of L. Also note that seq(n) is the same as 1:n.

Dataframes

- R refers to datasets as dataframes
- A dataframe is a matrix-like structure, where the columns can be of different types. You can also think of a dataframe as a list. Each column is an element of the list and each element has the same length.
- A dataframe is the fundamental data structure used by R 's statistical modeling functions
- Lecture 8 will be completely devoted to the managment and use of dataframes

	Population	Income	Illiteracy	Life.Exp	state.region
Alabama	3615	3624	2.1	69.05	South
Alaska	365	6315	1.5	69.31	West
Arizona	2212	4530	1.8	70.55	West
Arkansas	2110	3378	1.9	70.66	South
California	21198	5114	1.1	71.71	West
Colorado	2541	4884	0.7	72.06	West
Connecticut	3100	5348	1.1	72.48	Northeast

Numeric

- Technically, numeric data in `R` can be either `double` or `integer`, but in practice numeric data is almost always `double` (type `double` refers to real numbers). See `?integer` and `?double`.
- `.Machine` outputs numeric characteristics of the machine running `R`, such as the largest integer or the machine's precision
- `format()` formats an object for pretty printing. `format()` is a generic function that is used with other types of objects. See `?format()` for additional arguments.

```
> # trim - If FALSE right justified with common width
>format(c(1,10,100,1000), trim = FALSE)
[1] "   1" " 10" " 100" "1000"
>format(c(1,10,100,1000), trim = TRUE)
[1] "1"    "10"   "100" "1000"
> # nsmall - Minimum number of digits to the right of the decimal point
>format(13.7, nsmall = 3)
[1] "13.700"
> # scientific - Use scientific notation
>format(2^16, scientific = TRUE)
[1] "6.5536e+04"
```

Logical

- Logical values are represented by the reserved words `TRUE` and `FALSE` caps or simply `T` and `F`.

`!x`	NOT `x`
`x &y x`	`x` AND `y` elementwise, returns a vector
`&&y x`	`x` AND `y`, returns a single value
`| y x`	`x` OR `y` elementwise, returns a vector
`|| y`	`x` OR `y`, returns a single value
`xor(x,y)`	Exclusive OR of `x` and `y`, elementwise
`x %in% y`	`x` IN `y`
`x < y`	`x < y`
`x > y`	`x > y`
`x <= y`	`x ≤ y`
`x >= y`	`x ≥ y`
`x == y`	`x = y`
`x != y`	`x ≠ y`
`isTRUE(x)`	`TRUE` if `x` is `TRUE`
`all(...)`	`TRUE` if all arguments are `TRUE`
`any(...)`	`TRUE` if at least one argument is `TRUE`
`identical(x,y)`	Safe and reliable way to test two objects for being *exactly* equal
`all.equal(x,y)`	Test if two objects are *nearly* equal

Example - Logical Operations

```
> x <- 1:10

> (x %% 2==0) | (x > 5)        # What elements of x are even or greater than 5?
 1     FALSE TRUE FALSE TRUE FALSE TRUE TRUE TRUE TRUE TRUE
> x[(x %% 2==0) | (x > 5)]
[1] 2 4 6 7 8 9 10

> y <- 5:15                    # What elements of x are in y?
> x %in% y
 1     FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE
> x[x %in% y]
[1] 5 6 7 8 9 10

>any(x>5)                       # Are any elements of x greater then 5?
[1] TRUE
>all(x>5)                       # Are all the elements of x greater then 5?
[1] FALSE
```

Isn't that equal?

- In general, logical operators may not produce a single value and may return an `NA` if an element is `NA` or `NaN`.
- If you must get a single `TRUE` or `FALSE`, such as with `if` expressions, you should *NOT* use `==` or `!=`. Unless you are absolutely sure that nothing unusual can happen, you should use the `identical()` function instead.
- `identical()` only returns a single logical value, `TRUE` or `FALSE`, never `NA`

```
> name <- "Nick"
> if(name=="Nick") TRUE else FALSE
[1] TRUE

> # But what if name is never set to "Nick"?
> name <- NA
> if(name=="Nick") TRUE else FALSE
Error in if (name == "Nick") TRUE else FALSE:
  missing value where TRUE/FALSE needed
> if(identical(name, "Nick")) TRUE else FALSE
[1] FALSE
```

Isn't that equal?

- With `all.equal()` objects are treated as equal if the only difference is probably the result of inexact floating-point calculations. Returns `TRUE` if the mean relative difference is less then the specified tolerance.
- `all.equal()` either returns `TRUE` or a character string that describes the difference. Therefore, do not use `all.equal()` directly in `if` expressions, instead use with `isTRUE()` or `identical()`.

```
>(x <- sqrt(2))
[1] 1.414214
> x^2
[1] 2
> x^2==2
[1] FALSE
> all.equal(x^2, 2)
[1] TRUE
> all.equal(x^2, 1)
[1] "Mean relative difference: 0.5"
> isTRUE(all.equal(x^2, 1))
[1] FALSE
```

Character

- Character strings are defined by quotation marks, single ' ' or doul

`cat()`	Concatenate objects and print to console (\n for newline)
`paste()`	Concatenate objects and return a string
`print()`	Print an object
`substr()`	Extract or replace substrings in a character vector
`strtrim()`	Trim character vectors to specified display widths
`strsplit()`	Split elements of a character vector according to a substring
`grep()`	Search for matches to a pattern within a character vector, returns a vector of the indices that matched
`grepl()`	Like `grep()`, but returns a logical vector
`agrep()`	Similar to `grep()`, but searches for approximate matches
`regexpr()`	Similar to `grep()`, but returns the position of the first instance of a pattern *within* a string
`gsub()`	Replace all occurrences of a pattern with a character vector
`sub()`	Like `gsub()`, but only replaces the first occurrence
`tolower()`, `toupper()`	Convert to all lower/upper case
`noquote()`	Print a character vector without quotations
`nchar()`	Number of characters
`letters`, `LETTERS`	Built-in vector of lower and upper case letters

Example - Character Functions

```
animals <- c("bird", "horse","fish")
home    <- c("tree", "barn", "lake")

length(animals)     # Number of strings
nchar(animals)      # Number of characters in each string

cat("Animals:", animals)          # Need \n to move cursor to a newline #
cat(animals, home, "\n")          Joins one vector after the other

paste(animals, collapse=" ")      # Create one long string of animals
a.h=paste(animals, home, sep=".") # Pairwise joining of animals and home

# Split strings at ".", fixed=TRUE since "." is used for pattern matching
unlist(strsplit(a.h, ".", fixed=TRUE))

substr(animals, 2, 4) # Get characters 2-4 of each animal
strtrim(animals, 3)   # Print the first three characters

toupper(animals)      # Print animals in all upper case
```

Example - Pattern Matching

- A regular expression is a pattern that describes a set of strings.
  - ^ Start of character string
  - $ End of character string
  - . Any character
  - .{*n*} Any *n* characters
  - [· – ·] Range of letters, i.e. [a-c] is a, b, c
- See `?regexp` for more options

```
# colors() is a character vector of all the built-in color names
colors()[grep("red", colors())]      # All colors that contain "red"
colors()[grep("^red", colors())]     # Colors that start with "red"
colors()[grep("red$", colors())]     # Colors that end with "red"
colors()[grep("red.", colors())]     # Colors with one character after "red"
colors()[grep("^[r-t]", colors())] # Colors that begin with r, s, or t

places <- c("home", "zoo", "school", "work", "park")
gsub("o", "O", places)       # Replace all "o" with "O"
sub("o", "O", places)        # Replace the first "o" with "O"

# Capitalize the first letter, uses Perl-like regular expressions
gsub("(\\w)(\\w*)", "\\U\\1\\L\\2", places, perl=TRUE)
```

Factor

- A factor is a categorical variable with a defined number of ordered or unordered levels. Use the function `factor` to create a factor variable.

```
> factor(rep(1:2, 4), labels=c("trt.1", "trt.2"))
[1] trt.1 trt.2 trt.1 trt.2 trt.1 trt.2 trt.1 trt.2
Levels: trt.1 trt.2

> factor(rep(1:3, 4), labels=c("low", "med", "high"), ordered=TRUE)
1 low med high low med high low med high low med high  Levels: low <
med < high
```

`levels(x)`	Retrieve or set the levels of `x`
`nlevels(x)`	Return the number of levels of `x`
`relevel(x, ref)`	Levels of `x` are reordered so that the level specified by `ref` is first
`reorder()`	Reorders levels based on the values of a second variable
`gl()`	Generate factors by specifying the pattern of their levels
`cut(x, breaks)`	Divides the range of `x` into intervals (factors) determined by `breaks`

Example - Factor Functions

```
# Generate factor levels for 3 treatments and 2 cases per treatment  f
<- gl(3, 2, labels=paste("trt",1:3, sep="."))
levels(f)
nlevels(f)
relevel(f, "trt.2")


x <- runif(10)
cut(x, 3)                    # Cut x into three intervals
cut(x, c(0,.25,.5,.75,1))    # Cut x at the given cut points
```

Dates and Times

- R has objects that are dates only and objects that are dates and times. We will just focus on dates. Look at ?DateTimeClasses for information about how to handles dates and times.
- An R date object has the format: *Year -Month-Day*
- Operations with dates,
  - Days can be added or subtracted to a date
  - Dates can be subtracted
  - Dates can be compared using logical operators

Sys.Date()	Current date
as.Date()	Convert a character string to a date object
format.Date()	Change the format of a date object
seq.Date()	Generate sequence of dates
cut.Date()	Cut dates into intervals
weekdays, months, quarters	Extract parts of a date object
julian	Number of days since a given origin

.Date suffix is optional for calling format.Date(), seq.Date() and cut.Date(), but is necessary for viewing the appropriate documentation

Convert Strings to Date Objects

- Converting a string to a date object requires specifying a format string that defines the date format
- Any character in the format string other then the % symbol is interpreted literally.
- Common conversion specifications (see `?strptime` for a complete list),

%a	Abbreviated weekday name
%A	Full weekday name
%d	Day of the month
%B	Full month name
%b	Abbreviated month name
%m	Numeric month (01-12)
%y	Year without century (be very careful)
%Y	Year with century

```
> dates.1 <- c("5jan2008", "19aug2008", "2feb2009", "29sep2009")
> as.Date(dates.1, format="%d %b %Y")
[1] "2008-01-05" "2008-08-19" "2009-02-02" "2009-09-29"
> dates.2 <- c("5-1-2008", "19-8-2008", "2-2-2009", "29-9-2009")
> as.Date(dates.2, format="%d-%m-%Y")
[1] "2008-01-05" "2008-08-19" "2009-02-02" "2009-09-29"
```

Sequence of Dates

- To create a sequence of dates,

```
seq.Date(from, to, by, length.out =
NULL)
```

`from`, `to` Start and ending date objects

`by` A character string, containing one of "day", "week", "month" or "year". Can optionally be preceded by a (positive or negative) integer and a space, or followed by a "s".

`length.out` Integer, desired length of the sequence

```
> seq.Date(as.Date("2011/1/1"), as.Date("2011/1/31"), by="week")
[1] "2011-01-01" "2011-01-08" "2011-01-15" "2011-01-22" "2011-01-29"
>seq.Date(as.Date("2011/1/1"), as.Date("2011/1/31"), by="3 days") [1]
 "2011-01-01" "2011-01-04" "2011-01-07" "2011-01-10" "2011-01-13"
 [6] "2011-01-16" "2011-01-19" "2011-01-22" "2011-01-25" "2011-01-28"
[11] "2011-01-31"
> seq.Date(as.Date("2011/1/1"), by="week",length.out=10)
 [1] "2011-01-01" "2011-01-08" "2011-01-15" "2011-01-22" "2011-01-29"
 [6] "2011-02-05" "2011-02-12" "2011-02-19" "2011-02-26" "2011-03-05"
```

Cutting Dates

- To divide a sequence of dates in to levels,

```
cut.Date(x, breaks, start.on.monday = TRUE)
```

```
> jan <- seq.Date(as.Date("2011/1/1"), as.Date("2011/1/31"), by="days")
> cut(jan, breaks="weeks")
 [1] 2010-12-27 2010-12-27 2011-01-03 2011-01-03 2011-01-03 2011-01-03
 [7] 2011-01-03 2011-01-03 2011-01-03 2011-01-10 2011-01-10 2011-01-10
[13] 2011-01-10 2011-01-10 2011-01-10 2011-01-10 2011-01-17 2011-01-17
[19] 2011-01-17 2011-01-17 2011-01-17 2011-01-17 2011-01-17 2011-01-24
[25] 2011-01-24 2011-01-24 2011-01-24 2011-01-24 2011-01-24 2011-01-24
[31] 2011-01-31
6 Levels: 2010-12-27 2011-01-03 2011-01-10 2011-01-17 ... 2011-01-31
```

January 2011

Sun	Mon	Tue	Wed	Thr	Fri	Sat
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Operations with Dates

- Operations with dates,
  - Q Days can be added or subtracted to a date
  - Q Dates can be subtracted
  - Q Dates can be compared using logical operators

```
> jan1 <- as.Date("2011/1/1")
>(jan8 <- jan1 + 7)          # Add 7 days to 2011/1/1
[1] "2011-01-08"
> jan1 - 14                  # Subtract 2 weeks from 2011/1/8
[1] "2010-12-18"
> jan8 - jan1                # Number of days between 2011/1/1 and 2011/1/8
Time difference of 7 days
> jan8 > jan1                # Compare dates
[1] TRUE

> # Use format to extract parts of a date object or change the appearance
>format.Date(jan8, "%Y")
[1] "2011"
>format.Date(jan8, "%b-%d")
[1] "Jan-01"
```