# Variables - Reprise

**Discrete Attribute:**

Has only a finite or countably infinite set of values
Examples: zip codes, genotypes, gender, colors in red light, smoker
Often represented as integer variables.
Note: binary attributes are a special case of discrete attributes

**Continuous Attribute:**

Has real numbers as attribute values
Examples: temperature, height, or weight.
Practically, real values can only be measured and represented using a finite number of digits.
Continuous attributes are typically represented as floating-point variables.

Tan, Steinbeck, Kumar

# Variables - Reprise

**Categorical:**

Nominal

**Description**: The values are different names and provide enough information to distinguish one from another

**Examples:** ID numbers, colors, eye color, zip codes

**Operations**: mode, contingency correlation, Chi-square test

Ordinal

**Description**: The values provide enough information to order objects

**Examples:** rankings (e.g., taste of potato chips on a scale from 1-5), grades, height in {tall, medium, short}

**Operations**: median, percentiles, rank correlations, run tests, sign tests

Tan, Steinbeck, Kumar

# Variables - Reprise

**Numeric:**

Interval

**Description**: The differences between values are meaningful
**Examples:** Calendar dates, time, PH, temperature in Celsius or Farenheit
Difference between 5pm and 4pm is the same as 4am and 3am
**Operations**: mean, standard deviation, Pearson's correlation, t and F tests

Ratio

**Description**: Both differences and ratios make sense
**Examples:** temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current
**Operations**: geometric mean, harmonic mean, percent variations

Be careful - "color" is usually categorical - nominal but what about in a phyics experiment ? Its part of a spectrum and might be measured in terms of some continuous quantity ?

Tan, Steinbeck, Kumar

# Variables - Reprise

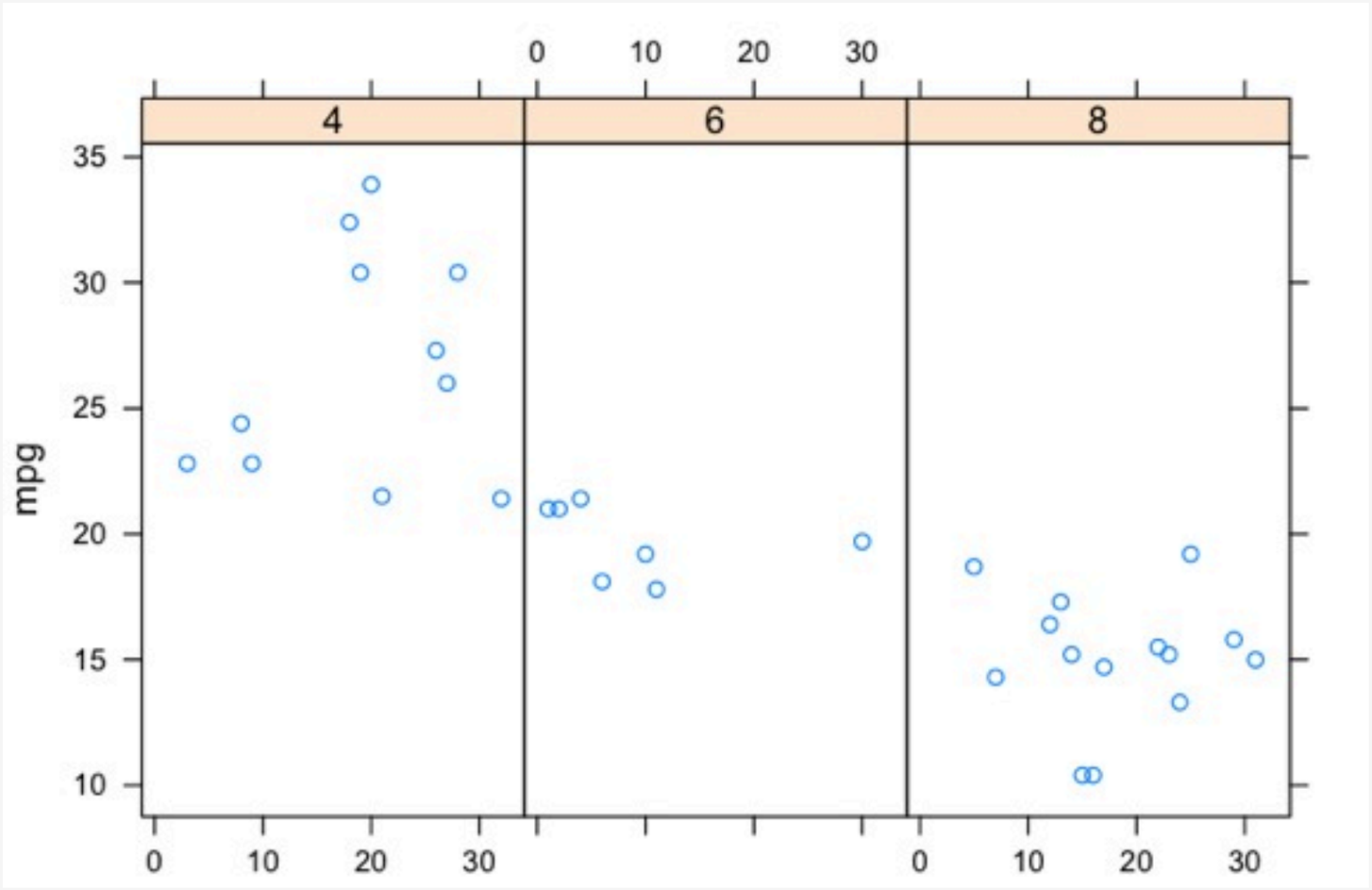
```
> head(mtcars,15)
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood 10.4  8 472.0 205 2.93 5.250 17.98  0  0    3    4
```

# Variables - Reprise

Which variables are
Continuous ? Discrete ? Categorical ?

# Variables - Reprise

```
xyplot(mpg~1:nrow(mtcars)|factor(cyl),data=mtcars)
```




```
aggregate(mpg~cyl,data=mtcars,mean)
  cyl      mpg
1   4 26.66364
2   6 19.74286
3   8 15.10000
```

# Variables - Objects

Everything, (vector, factor, matrix, array, list , data.frame), is an **object**, which also has a **type** and belongs to a **class**:

```
3+5
[1] 8

typeof(3)
[1] "double"

class(3) # “class” and “mode” can be used interchangeably
[1] "numeric"

typeof(`+`)
[1] "builtin"

Use the "str" function() - It is a good summary command

str(3)
num 3
```

# Variables - Objects - Numeric

The four primary variable classes are: **numeric**, character, factor, and dates. Its important to know how to manipulate these and , if necessary, convert between them.

Don't rush through these basic concepts as you will almost always have to change the type of a given variable to apply a function or statistical procedure.

```
var1 = 3
var1
[1] 3

sqrt(var1)
[1] 1.732051

var1 = 33.3

str(var1)
 num 33.3
```

# Variables - Objects - Numeric

The four primary variable classes are: **numeric**, character, factor, and dates. Its important to know how to manipulate these and , if necessary, convert between them.

Don’t rush through these basic concepts as you will almost always have to change the type of a given variable to apply a function or statistical procedure.

```
myvar = 5

myvar + myvar # Addition
[1] 6

myvar - myvar      # Subtraction
[1] 0

myvar * myvar # Multiplication
[1] 9

myvar / myvar # Division
[1] 1

myvar ^ myvar # myvar raised to the power of myvar
[1] 3125
```

# Variables - Objects - Numeric

Its worth pointing out that there is a distinction between integers and real values. Don't worry too much about this now but keep it in mind. If you really want an integer then you have to "ask" for it:

```
aa = 5

class(aa)
[1] "numeric"

str(aa)
 num 5

aa = as.integer(aa)  # We use a "coercion" function here

class(aa)
[1] "integer"

aa = 5.67

as.integer(aa)   # Truncates the value - note it doesn't round.
[1] 5
```

# Variables - Objects - Character

Character strings are possible also. This variable type is for when you wish to store informative labels about something. Generally speaking string variables are “descriptive” whereas numeric variables are quantitative.

```
var.one = "Hello there ! My name is Steve."
var.two = "How do you do ?"

var.one
[1] "Hello there ! My name is Steve."

nchar(var.one)     # Number of characters present
[1] 31

toupper(var.one)
[1] "HELLO THERE ! MY NAME IS STEVE."

mydna = c("A","G","T","C","A")

# See BioConductor http://www.bioconductor.org/

str(mydna)
 chr [1:5] "A" "G" "T" "C" "A"

mydna
[1] "A" "G" "T" "C" "A"
```

# Variables - Objects - Character

Character strings are possible also. This variable type is for when you wish to store informative labels about something. Generally speaking string variables are “descriptive” whereas numeric variables are quantitative.

```
paste(var.one,var.two)
[1] "Hello there ! My name is Steve. How do you do ?"

paste(var.one,var.two,sep=":")
[1] "Hello there ! My name is Steve.:How do you do ?"

strsplit(var.one," ")
[[1]]
[1] "Hello"  "there"  "!"      "My"     "name"   "is"     "Steve."


patientid = "ID:011472:M:C"  # Encodes Birthday, Gender, and Race

strsplit(patientid,":")
[[1]]
[1] "ID"     "011472" "M"      "C"

bday = strsplit(patientid,":")[[1]][2]  # Get just the birthday
```

# Variables - Objects - Dates

Dates are an important data type in R. In many cases dates are treated as characters when printing. However, dates can be operated upon arithmetically. If you work a lot with dates then consider working with the “zoo” package which handles Time Series data quite well.

```
Sys.Date()
[1] "2011-08-01"

Sys.Date()+1
[1] "2011-08-02"

class(Sys.Date())
[1] "Date"

string = "2011-04-27”
class(string)
[1] "character"

as.Date(string)
[1] "2011-04-27"
```

# Variables - Objects - Dates

If your input dates are not in the standard format, a format string can be composed using the elements shown in Table . The following examples show some ways that this can be used:

```
as.Date("03/17/1996")
Error in charToDate(x) :
  character string is not in a standard unambiguous format

as.Date("03/17/1996",format="%m/%d/%Y")
[1] "1996-03-17"

as.Date('1/15/2001',format='%m/%d/%Y')
[1] "2001-01-15"

as.Date('April 26, 2001',format='%B %d, %Y')
[1] "2001-04-26"

as.Date("2012-10-27")
[1] "2012-10-27"
```

| Code | Value |
|---|---|
| %d | Day of the month (decimal number) |
| %m | Month (decimal number) |
| %b | Month (abbreviated) |
| %B | Month (full name) |
| %y | Year (2 digit) |
| %Y | Year (4 digit) |

# Variables - Objects - Dates

Once you get your date formatted you can access parts of it easily:

```
my.date = as.Date("2012-10-27")
my.date - 1
[1] "2012-10-26"

format(my.date,"%Y")   # Note all of these are character strings
[1] "2012"

format(my.date,"%b")
[1] "Oct"

format(my.date,"%y")
[1] "12"

format(my.date,"%b %d")
[1] "Oct 27"

format(my.date,"%b %d, %Y")
[1] "Oct 27, 2012"
```

| Code | Value |
|---|---|
| %d | Day of the month (decimal number) |
| %m | Month (decimal number) |
| %b | Month (abbreviated) |
| %B | Month (full name) |
| %y | Year (2 digit) |
| %Y | Year (4 digit) |

# Variables - Objects - Dates

The difftime function let's us pass character strings to it.

```
difftime("2005-10-21 08:32:58","2003-8-15 09:18:05")

Time difference of 797.9687 days
```

Here is one way to deal with date strings say from an Excel file. Let's say that the dates are in month/day/year format:

```
strptime(c("03/27/2003","03/27/2003","04/14/2008"),format="%m/%d/%Y")
"2003-03-27" "2003-03-27" "2008-04-14"
```

# Variables - Objects - Dates

```
rdates
   Release       Date
1      1.0 2000-02-29
2      1.1 2000-06-15
3      1.2 2000-12-15
4      1.3 2001-06-22
5      1.4 2001-12-19
6      1.5 2002-04-29
7      1.6 2002-10-10
8      1.7 2003-04-16
9      1.8 2003-10-08
10     1.9 2004-04-12
11     2.0 2004-10-04

mean(rdates$Date)
[1] "2002-05-20"

range(rdates$Date)
[1] "2000-02-29" "2004-10-04"

rdates$Date[11] - rdates$Date[1]
Time difference of 1679 days


Data Manipulation with R - Phil Spector pp 64
```

# Variables - Objects - Logical

Logical variables are those that take on a TRUE or FALSE value. Either by direct assignment or as the result of some comparison:

```
some.variable = TRUE

some.variable = (4 < 5)
```

This is an important type because it will eventually allow us to construct expressions to be used in if statements for programming

```
if (some_logical_condition) {
   do something
}

if ( 4 < 5 ) {
  print(“Four is less than Five”)
}
```

# Variables - Objects - Logical

```
if ( 4 < 5 ) {
  print("Four is less than Five")
}

my.var = ( 4 < 5)

if (my.var) {
  print("four is less than five")
}

if (! my.var ) {
   print("four is greater than five")
}

my.var = (4 < 5) & ( 4 < 6 )  # Logical AND operator
```

Both expressions must be TRUE in order for the combined expression to return TRUE.

```
my.var
[1] TRUE

my.var = (4 < 5) | ( 4 < 6 ) # Logical OR operator.
my.var = TRUE
```

Only one of these expressions needs to be TRUE for the entire expression to be TRUE

# Variables - Objects - Interrogating

It is a common operation to want to interrogate variables to see what they are (or aren't). There is an entire family of "is" functions to accomplish this type of activity. It is also common to "coerce" variables into another type. There is an entire family of "as" functions to accomplish this.

| **Interrogation** | **Coercion** | **Type of Operand** |
|---|---|---|
| `is.array()` | `as.array()` | Arrays |
| `is.character()` | `as.character()` | Character |
| `is.data.frame()` | `as.data.frame()` | Data Frames |
| `is.factor()` | `as.factor()` | Factors |
| `is.list()` | `as.list()` | Lists |
| `is.logical()` | `as.logical()` | Logical |
| `is.matrix()` | `as.matrix()` | Matrix |
| `is.numeric()` | `as.numeric()` | Numeric |
| `is.vector()` | `as.vector()` | Vector |

# Variables - Objects - Interrogating

It is a common operation to want to interrogate variables to see what they are (or aren't).

```
my_int = as.integer(5)

is.integer(my_int)  # These are good for use in programming loops
[1] TRUE

is.numeric(my_int)
[1] TRUE

is.character(my_int)
[1] FALSE

is.logical(my_int)
[1] FALSE
```

# Variables - Objects - Interrogating

It is also common to “coerce” variables into another type. There is an entire family of “as” functions to accomplish this.

```
my_int = as.integer(5)

as.character(my_int)
[1] "5"

as.integer(as.character(my_int))
[1] 5

my_number = 12.345

as.character(my_number)
[1] "12.345"

as.logical(1)
[1] TRUE

as.character(as.logical(1))
[1] "TRUE"
```

# Variables - Objects - Interrogating

These types of functions show up frequently when users write their own functions. The "is" functions assist with checking the arguments for the correct type.

```
my.func = function(x) {

        if (!is.numeric(x) ) {
           stop("Hey. I need a numeric vector here")
        } else {
           return(mean(x))
        }
}
```