Logical Operators and Control Structures

Ben Augustine

August 26, 2019

Logical Operations

Logical operators are used to test whether certain conditions hold. They are useful for many tasks, including subsetting data structures.

First, let's look at boolean data types, which can only take the values *TRUE* and *FALSE*.

```
val=TRUE #assign that "val" is true
val #look at val
```

```
## [1] TRUE
```

```
class(val) #query the type of object val is
```

```
## [1] "logical"
```

```
is.logical(val) #ask if val is a "logical" data type
```

```
## [1] TRUE
```

```
T #can also just use a capital T or F (unless you assign over them--try not to do that)
```

```
## [1] TRUE
```

We can use *!* to negate a logical. A logical NOT.

```
val #look at val
```

```
## [1] TRUE
```

```
!val #what is the opposite of val? "Not val"
```

```
## [1] FALSE
```

```
!TRUE #opposite of TRUE is?
```

```
## [1] FALSE
```

Often, we want to test for equality.

```
val=5 #set val to 5
val==5 #test if val is equal to 5
```

```
## [1] TRUE
```

```
val!=5 #test if val is not equal to 5
```

```
## [1] FALSE
```

We can test for greater or less than using >, <, >=, and <=.

```
val<5 #is val less than 5?
```

```
## [1] FALSE
```

```
val>5 #is val greater than 5?
```

```
## [1] FALSE
```

```
val>=5 #is val less than or equal to 5?
```

```
## [1] TRUE
```

```
val<=5 #is val greater than or equal to 5?
```

```
## [1] TRUE
```

We can combine test conditions using & and / (and, or).

```
val_1=5
val_2=10
val_1>4&val_2<15 #is val_1 greater than 4 (TRUE) and is val_2 less than 15 (TRUE)
```

```
## [1] TRUE
```

```
(val_1>4)&(val_2<15) #better to use parentheses (follows typical order of operations)
```

```
## [1] TRUE
```

```
(val_1<4)&(val_2<15) #is val_1 less than 4 (FALSE) and is val_2 less than 15 (TRUE)
```

```
## [1] FALSE
```

```
(val_1<4)|(val_2<15) #is val_1 less than 4 (FALSE) and is val_2 less than 15 (TRUE)
```

```
## [1] TRUE
```

```
#these types of statements are equivalent to asking
TRUE&TRUE
```

```
## [1] TRUE
```

```
TRUE&FALSE
```

```
## [1] FALSE
```

```
TRUE|FALSE
```

```
## [1] TRUE
```

```
FALSE|FALSE
```

```
## [1] FALSE
```

We can test for missing values. Missing values in R are indicated with *NA*.

```
data=c(1,3,NA,2,5) #specify a vector with 1 missing value
is.na(data) #is each element of "data" coded as NA?
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

Often, we want to identify which elements of a data structure meet certain conditions.

```
which(is.na(data)) #which elements of "data" are coded as NA?
```

```
## [1] 3
```

```
which(data>1) #which elements of "data" are greater than 1?
```

```
## [1] 2 4 5
```

We often want to use logical tests to subset data structures

```
idx=which(is.na(data)) #assign idx to be the elements of "data" coded as NA
data[-idx] #display data with idx removed
```

```
## [1] 1 3 2 5
```

```
data[-which(is.na(data))] #alternatively, do this in 1 line
```

```
## [1] 1 3 2 5
```

```
data[idx] #display only the element stored in idx
```

```
## [1] NA
```

```
data=data[-idx] #permanently remove "idx" from "data" by reassigning "data"
data
```

```
## [1] 1 3 2 5
```

Let's try using logical tests to subset a data frame with rows and columns

```
data(iris) #load iris data set
str(iris) #look at its structure
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
iris$Species=="setosa" #test which elements of the Species column of the iris data frame are "setosa"
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
which(iris$Species=="setosa") #which elements meet this condition?
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
## [47] 47 48 49 50
```

```
iris[iris$Species=="setosa",] #subset the iris data frame to only the setosa species
```

```
##    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
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
## 11          5.4         3.7          1.5         0.2  setosa
## 12          4.8         3.4          1.6         0.2  setosa
## 13          4.8         3.0          1.4         0.1  setosa
## 14          4.3         3.0          1.1         0.1  setosa
## 15          5.8         4.0          1.2         0.2  setosa
## 16          5.7         4.4          1.5         0.4  setosa
## 17          5.4         3.9          1.3         0.4  setosa
## 18          5.1         3.5          1.4         0.3  setosa
## 19          5.7         3.8          1.7         0.3  setosa
## 20          5.1         3.8          1.5         0.3  setosa
## 21          5.4         3.4          1.7         0.2  setosa
## 22          5.1         3.7          1.5         0.4  setosa
## 23          4.6         3.6          1.0         0.2  setosa
## 24          5.1         3.3          1.7         0.5  setosa
## 25          4.8         3.4          1.9         0.2  setosa
## 26          5.0         3.0          1.6         0.2  setosa
## 27          5.0         3.4          1.6         0.4  setosa
## 28          5.2         3.5          1.5         0.2  setosa
## 29          5.2         3.4          1.4         0.2  setosa
## 30          4.7         3.2          1.6         0.2  setosa
## 31          4.8         3.1          1.6         0.2  setosa
## 32          5.4         3.4          1.5         0.4  setosa
## 33          5.2         4.1          1.5         0.1  setosa
## 34          5.5         4.2          1.4         0.2  setosa
## 35          4.9         3.1          1.5         0.2  setosa
## 36          5.0         3.2          1.2         0.2  setosa
## 37          5.5         3.5          1.3         0.2  setosa
## 38          4.9         3.6          1.4         0.1  setosa
## 39          4.4         3.0          1.3         0.2  setosa
## 40          5.1         3.4          1.5         0.2  setosa
## 41          5.0         3.5          1.3         0.3  setosa
## 42          4.5         2.3          1.3         0.3  setosa
## 43          4.4         3.2          1.3         0.2  setosa
## 44          5.0         3.5          1.6         0.6  setosa
## 45          5.1         3.8          1.9         0.4  setosa
## 46          4.8         3.0          1.4         0.3  setosa
## 47          5.1         3.8          1.6         0.2  setosa
## 48          4.6         3.2          1.4         0.2  setosa
## 49          5.3         3.7          1.5         0.2  setosa
## 50          5.0         3.3          1.4         0.2  setosa
```

```
iris[-which(iris$Species=="setosa"),] #subset to exclude the setosa species
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 51           7.0         3.2          4.7         1.4 versicolor
## 52           6.4         3.2          4.5         1.5 versicolor
```

```
## 53          6.9         3.1          4.9         1.5 versicolor
## 54          5.5         2.3          4.0         1.3 versicolor
## 55          6.5         2.8          4.6         1.5 versicolor
## 56          5.7         2.8          4.5         1.3 versicolor
## 57          6.3         3.3          4.7         1.6 versicolor
## 58          4.9         2.4          3.3         1.0 versicolor
## 59          6.6         2.9          4.6         1.3 versicolor
## 60          5.2         2.7          3.9         1.4 versicolor
## 61          5.0         2.0          3.5         1.0 versicolor
## 62          5.9         3.0          4.2         1.5 versicolor
## 63          6.0         2.2          4.0         1.0 versicolor
## 64          6.1         2.9          4.7         1.4 versicolor
## 65          5.6         2.9          3.6         1.3 versicolor
## 66          6.7         3.1          4.4         1.4 versicolor
## 67          5.6         3.0          4.5         1.5 versicolor
## 68          5.8         2.7          4.1         1.0 versicolor
## 69          6.2         2.2          4.5         1.5 versicolor
## 70          5.6         2.5          3.9         1.1 versicolor
## 71          5.9         3.2          4.8         1.8 versicolor
## 72          6.1         2.8          4.0         1.3 versicolor
## 73          6.3         2.5          4.9         1.5 versicolor
## 74          6.1         2.8          4.7         1.2 versicolor
## 75          6.4         2.9          4.3         1.3 versicolor
## 76          6.6         3.0          4.4         1.4 versicolor
## 77          6.8         2.8          4.8         1.4 versicolor
## 78          6.7         3.0          5.0         1.7 versicolor
## 79          6.0         2.9          4.5         1.5 versicolor
## 80          5.7         2.6          3.5         1.0 versicolor
## 81          5.5         2.4          3.8         1.1 versicolor
## 82          5.5         2.4          3.7         1.0 versicolor
## 83          5.8         2.7          3.9         1.2 versicolor
## 84          6.0         2.7          5.1         1.6 versicolor
## 85          5.4         3.0          4.5         1.5 versicolor
## 86          6.0         3.4          4.5         1.6 versicolor
## 87          6.7         3.1          4.7         1.5 versicolor
## 88          6.3         2.3          4.4         1.3 versicolor
## 89          5.6         3.0          4.1         1.3 versicolor
## 90          5.5         2.5          4.0         1.3 versicolor
## 91          5.5         2.6          4.4         1.2 versicolor
## 92          6.1         3.0          4.6         1.4 versicolor
## 93          5.8         2.6          4.0         1.2 versicolor
## 94          5.0         2.3          3.3         1.0 versicolor
## 95          5.6         2.7          4.2         1.3 versicolor
## 96          5.7         3.0          4.2         1.2 versicolor
## 97          5.7         2.9          4.2         1.3 versicolor
## 98          6.2         2.9          4.3         1.3 versicolor
## 99          5.1         2.5          3.0         1.1 versicolor
## 100         5.7         2.8          4.1         1.3 versicolor
## 101         6.3         3.3          6.0         2.5  virginica
## 102         5.8         2.7          5.1         1.9  virginica
## 103         7.1         3.0          5.9         2.1  virginica
## 104         6.3         2.9          5.6         1.8  virginica
## 105         6.5         3.0          5.8         2.2  virginica
## 106         7.6         3.0          6.6         2.1  virginica
```

```
## 107          4.9         2.5          4.5         1.7  virginica
## 108          7.3         2.9          6.3         1.8  virginica
## 109          6.7         2.5          5.8         1.8  virginica
## 110          7.2         3.6          6.1         2.5  virginica
## 111          6.5         3.2          5.1         2.0  virginica
## 112          6.4         2.7          5.3         1.9  virginica
## 113          6.8         3.0          5.5         2.1  virginica
## 114          5.7         2.5          5.0         2.0  virginica
## 115          5.8         2.8          5.1         2.4  virginica
## 116          6.4         3.2          5.3         2.3  virginica
## 117          6.5         3.0          5.5         1.8  virginica
## 118          7.7         3.8          6.7         2.2  virginica
## 119          7.7         2.6          6.9         2.3  virginica
## 120          6.0         2.2          5.0         1.5  virginica
## 121          6.9         3.2          5.7         2.3  virginica
## 122          5.6         2.8          4.9         2.0  virginica
## 123          7.7         2.8          6.7         2.0  virginica
## 124          6.3         2.7          4.9         1.8  virginica
## 125          6.7         3.3          5.7         2.1  virginica
## 126          7.2         3.2          6.0         1.8  virginica
## 127          6.2         2.8          4.8         1.8  virginica
## 128          6.1         3.0          4.9         1.8  virginica
## 129          6.4         2.8          5.6         2.1  virginica
## 130          7.2         3.0          5.8         1.6  virginica
## 131          7.4         2.8          6.1         1.9  virginica
## 132          7.9         3.8          6.4         2.0  virginica
## 133          6.4         2.8          5.6         2.2  virginica
## 134          6.3         2.8          5.1         1.5  virginica
## 135          6.1         2.6          5.6         1.4  virginica
## 136          7.7         3.0          6.1         2.3  virginica
## 137          6.3         3.4          5.6         2.4  virginica
## 138          6.4         3.1          5.5         1.8  virginica
## 139          6.0         3.0          4.8         1.8  virginica
## 140          6.9         3.1          5.4         2.1  virginica
## 141          6.7         3.1          5.6         2.4  virginica
## 142          6.9         3.1          5.1         2.3  virginica
## 143          5.8         2.7          5.1         1.9  virginica
## 144          6.8         3.2          5.9         2.3  virginica
## 145          6.7         3.3          5.7         2.5  virginica
## 146          6.7         3.0          5.2         2.3  virginica
## 147          6.3         2.5          5.0         1.9  virginica
## 148          6.5         3.0          5.2         2.0  virginica
## 149          6.2         3.4          5.4         2.3  virginica
## 150          5.9         3.0          5.1         1.8  virginica
```

```
iris2=iris[-which(iris$Species=="setosa"),] #we need to assign the subsetted data to a data object to s
```

Now, let's use logical tests to modify specific data values. What if we want to set the values of "sepal length" less than 5 as missing data (don't ask why!)?

```
idx=which(iris$Sepal.Length<5) #find the sepal lengths less than 5
iris$Sepal.Length[idx]=NA
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2             NA         3.0          1.4         0.2     setosa
## 3             NA         3.2          1.3         0.2     setosa
## 4             NA         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
## 7             NA         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9             NA         2.9          1.4         0.2     setosa
## 10            NA         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12            NA         3.4          1.6         0.2     setosa
## 13            NA         3.0          1.4         0.1     setosa
## 14            NA         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     setosa
## 23            NA         3.6          1.0         0.2     setosa
## 24           5.1         3.3          1.7         0.5     setosa
## 25            NA         3.4          1.9         0.2     setosa
## 26           5.0         3.0          1.6         0.2     setosa
## 27           5.0         3.4          1.6         0.4     setosa
## 28           5.2         3.5          1.5         0.2     setosa
## 29           5.2         3.4          1.4         0.2     setosa
## 30            NA         3.2          1.6         0.2     setosa
## 31            NA         3.1          1.6         0.2     setosa
## 32           5.4         3.4          1.5         0.4     setosa
## 33           5.2         4.1          1.5         0.1     setosa
## 34           5.5         4.2          1.4         0.2     setosa
## 35            NA         3.1          1.5         0.2     setosa
## 36           5.0         3.2          1.2         0.2     setosa
## 37           5.5         3.5          1.3         0.2     setosa
## 38            NA         3.6          1.4         0.1     setosa
## 39            NA         3.0          1.3         0.2     setosa
## 40           5.1         3.4          1.5         0.2     setosa
## 41           5.0         3.5          1.3         0.3     setosa
## 42            NA         2.3          1.3         0.3     setosa
## 43            NA         3.2          1.3         0.2     setosa
## 44           5.0         3.5          1.6         0.6     setosa
## 45           5.1         3.8          1.9         0.4     setosa
## 46            NA         3.0          1.4         0.3     setosa
## 47           5.1         3.8          1.6         0.2     setosa
## 48            NA         3.2          1.4         0.2     setosa
## 49           5.3         3.7          1.5         0.2     setosa
## 50           5.0         3.3          1.4         0.2     setosa
## 51           7.0         3.2          4.7         1.4 versicolor
## 52           6.4         3.2          4.5         1.5 versicolor
## 53           6.9         3.1          4.9         1.5 versicolor
```

```
## 54          5.5         2.3          4.0         1.3 versicolor
## 55          6.5         2.8          4.6         1.5 versicolor
## 56          5.7         2.8          4.5         1.3 versicolor
## 57          6.3         3.3          4.7         1.6 versicolor
## 58           NA         2.4          3.3         1.0 versicolor
## 59          6.6         2.9          4.6         1.3 versicolor
## 60          5.2         2.7          3.9         1.4 versicolor
## 61          5.0         2.0          3.5         1.0 versicolor
## 62          5.9         3.0          4.2         1.5 versicolor
## 63          6.0         2.2          4.0         1.0 versicolor
## 64          6.1         2.9          4.7         1.4 versicolor
## 65          5.6         2.9          3.6         1.3 versicolor
## 66          6.7         3.1          4.4         1.4 versicolor
## 67          5.6         3.0          4.5         1.5 versicolor
## 68          5.8         2.7          4.1         1.0 versicolor
## 69          6.2         2.2          4.5         1.5 versicolor
## 70          5.6         2.5          3.9         1.1 versicolor
## 71          5.9         3.2          4.8         1.8 versicolor
## 72          6.1         2.8          4.0         1.3 versicolor
## 73          6.3         2.5          4.9         1.5 versicolor
## 74          6.1         2.8          4.7         1.2 versicolor
## 75          6.4         2.9          4.3         1.3 versicolor
## 76          6.6         3.0          4.4         1.4 versicolor
## 77          6.8         2.8          4.8         1.4 versicolor
## 78          6.7         3.0          5.0         1.7 versicolor
## 79          6.0         2.9          4.5         1.5 versicolor
## 80          5.7         2.6          3.5         1.0 versicolor
## 81          5.5         2.4          3.8         1.1 versicolor
## 82          5.5         2.4          3.7         1.0 versicolor
## 83          5.8         2.7          3.9         1.2 versicolor
## 84          6.0         2.7          5.1         1.6 versicolor
## 85          5.4         3.0          4.5         1.5 versicolor
## 86          6.0         3.4          4.5         1.6 versicolor
## 87          6.7         3.1          4.7         1.5 versicolor
## 88          6.3         2.3          4.4         1.3 versicolor
## 89          5.6         3.0          4.1         1.3 versicolor
## 90          5.5         2.5          4.0         1.3 versicolor
## 91          5.5         2.6          4.4         1.2 versicolor
## 92          6.1         3.0          4.6         1.4 versicolor
## 93          5.8         2.6          4.0         1.2 versicolor
## 94          5.0         2.3          3.3         1.0 versicolor
## 95          5.6         2.7          4.2         1.3 versicolor
## 96          5.7         3.0          4.2         1.2 versicolor
## 97          5.7         2.9          4.2         1.3 versicolor
## 98          6.2         2.9          4.3         1.3 versicolor
## 99          5.1         2.5          3.0         1.1 versicolor
## 100         5.7         2.8          4.1         1.3 versicolor
## 101         6.3         3.3          6.0         2.5  virginica
## 102         5.8         2.7          5.1         1.9  virginica
## 103         7.1         3.0          5.9         2.1  virginica
## 104         6.3         2.9          5.6         1.8  virginica
## 105         6.5         3.0          5.8         2.2  virginica
## 106         7.6         3.0          6.6         2.1  virginica
## 107          NA         2.5          4.5         1.7  virginica
```

```
## 108          7.3         2.9          6.3         1.8  virginica
## 109          6.7         2.5          5.8         1.8  virginica
## 110          7.2         3.6          6.1         2.5  virginica
## 111          6.5         3.2          5.1         2.0  virginica
## 112          6.4         2.7          5.3         1.9  virginica
## 113          6.8         3.0          5.5         2.1  virginica
## 114          5.7         2.5          5.0         2.0  virginica
## 115          5.8         2.8          5.1         2.4  virginica
## 116          6.4         3.2          5.3         2.3  virginica
## 117          6.5         3.0          5.5         1.8  virginica
## 118          7.7         3.8          6.7         2.2  virginica
## 119          7.7         2.6          6.9         2.3  virginica
## 120          6.0         2.2          5.0         1.5  virginica
## 121          6.9         3.2          5.7         2.3  virginica
## 122          5.6         2.8          4.9         2.0  virginica
## 123          7.7         2.8          6.7         2.0  virginica
## 124          6.3         2.7          4.9         1.8  virginica
## 125          6.7         3.3          5.7         2.1  virginica
## 126          7.2         3.2          6.0         1.8  virginica
## 127          6.2         2.8          4.8         1.8  virginica
## 128          6.1         3.0          4.9         1.8  virginica
## 129          6.4         2.8          5.6         2.1  virginica
## 130          7.2         3.0          5.8         1.6  virginica
## 131          7.4         2.8          6.1         1.9  virginica
## 132          7.9         3.8          6.4         2.0  virginica
## 133          6.4         2.8          5.6         2.2  virginica
## 134          6.3         2.8          5.1         1.5  virginica
## 135          6.1         2.6          5.6         1.4  virginica
## 136          7.7         3.0          6.1         2.3  virginica
## 137          6.3         3.4          5.6         2.4  virginica
## 138          6.4         3.1          5.5         1.8  virginica
## 139          6.0         3.0          4.8         1.8  virginica
## 140          6.9         3.1          5.4         2.1  virginica
## 141          6.7         3.1          5.6         2.4  virginica
## 142          6.9         3.1          5.1         2.3  virginica
## 143          5.8         2.7          5.1         1.9  virginica
## 144          6.8         3.2          5.9         2.3  virginica
## 145          6.7         3.3          5.7         2.5  virginica
## 146          6.7         3.0          5.2         2.3  virginica
## 147          6.3         2.5          5.0         1.9  virginica
## 148          6.5         3.0          5.2         2.0  virginica
## 149          6.2         3.4          5.4         2.3  virginica
## 150          5.9         3.0          5.1         1.8  virginica
```

any and *all* are useful functions for logical operators

```
any(is.na(data)) #are any elements of "data" coded as NA?
```

```
## [1] FALSE
```

```
all(is.na(data)) #are all elements of "data" coded as NA?
```

```
## [1] FALSE
```

Finally, we can do math on logical data structures

```
sum(iris$Sepal.Width<3) #add the elements of "sepal width" that are less than 3
```

```
## [1] 57
```

```
sum(iris$Sepal.Width>3) #add the elements of "sepal width" that are more than 3
```

```
## [1] 67
```

```
#looking at the inputs to sum(), we can see that TRUE is counted as 1 and FALSE as 0
iris$Sepal.Width<3
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
##  [56]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
##  [67] FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
##  [78] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE
##  [89] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
## [100]  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE
## [111] FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE
## [122]  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
## [133]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [144] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

```
iris$Sepal.Width>3
```

```
##   [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
##  [12]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [23]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [34]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
##  [45]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
##  [56] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [67] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [111]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE
## [122] FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE
## [133] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
## [144]  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE
```

```
sum(iris$Sepal.Width>3) #add the elements of "sepal width" that are more than 3
```

```
## [1] 67
```

```r
#Let's do the same for Sepal Length
sum(iris$Sepal.Length<6)
```

```
## [1] NA
```

```r
sum(iris$Sepal.Length>6)
```

```
## [1] NA
```

```r
#Why are we getting an NA?
sum(iris$Sepal.Length<6,na.rm=TRUE)
```

```
## [1] 61
```

Control Structures

Control structures allow you to control the flow of execution of a series of R expressions. They are useful for automating repetitive tasks and performing particular tasks depending upon the values of the inputs. We will look at the *for*, *while*, and *if* structures.

for loops generally take the form:

```r
for( variable in sequence ){
  expression
  expression
  expression
}
```

They allow us do things like this:

```r
a=0 #initialize a variable with value 0
for(i in 1:10){ #we increment i from 1 to 10
  a=a+1 #for each iteration, i, we add 1 to a
  print(a) #prints the value of a on each iteration
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

One thing to understand here is what the sequence of numbers after *in* is doing. In the loop above, 1:10 specifies a *sequence of numbers* which *i* will cycle through.

```r
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
seq(from=1,to=10,by=1) #equivalent
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

We can specify any sequence of numbers (well, integers) we want. For example:

```r
seq(from=10,to=1,by=-1) #specify a sequence starting at 10, ending at 1, incremented by -1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```r
for(i in seq(10,1,-1)){ #put this sequence in the for loop
  print(i)
}
```

```
## [1] 10
## [1] 9
## [1] 8
## [1] 7
## [1] 6
## [1] 5
## [1] 4
## [1] 3
## [1] 2
## [1] 1
```

Another example:

```r
idx=c(1,2,200,4000) #define idx to be a sequence of arbitrary integers
for(i in idx){ #iterate through the sequence stored in idx
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 200
## [1] 4000
```

Often, one *for* loop is not sufficient for the task at hand. You may need to nest multiple for loops inside one another like this:

```r
for(i in 1:3){ #increment i from 1 to 3
  for(j in 1:3){ #increment j from 1 to 3
    print(c(i,j)) #prints the value of i and j on each iteration
  }
}
```

```
## [1] 1 1
## [1] 1 2
## [1] 1 3
## [1] 2 1
## [1] 2 2
## [1] 2 3
## [1] 3 1
## [1] 3 2
## [1] 3 3
```

Here is an example where we set "val" to 0, add 1 on each i iteration, and add 10 on each j iteration

```
val=0 #initialize a variable with value 0
for(i in 1:3){ #we increment i from 1 to 3
  for(j in 1:3){
    val=val+1 #for each j iteration we add 1 to val
    print(val) #prints the value of val on each iteration
  }
  val=val+10 #for each i iteration, we add 10
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 14
## [1] 15
## [1] 16
## [1] 27
## [1] 28
## [1] 29
```

In the previous examples, i is incremented from 1 to 10, but not used directly. Often, we want to utilize the value of i, or whichever variable we use to represent the itertion number inside the loop. Here, we will store the values of a on each iteration of the *for* loop.

```
store=rep(NA,10) #preallocate a vector of length 10 with missing values
a=0 #initialize a variable with value 0
for(i in 1:10){ #we increment i from 1 to 10
  a=a+1 #for each iteration, i, we add 1 to a
  store[i]=a #on each iteration, we store the current value of a into the ith index of store
}
store #evaluate store to see the values we recorded
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

This example introduces the concept of *preallocation*. Say we want to store the output of a loop in a vector. One way to do this would be to increase the size of the vector on each iteration like:

```
vec=c() #specify an empty vector
for(i in 1:25){
  vec=c(vec,i) #append the loop index to "vec"
}
vec
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25
```

This approach "grows" the vector one element at a time. *Preallocation* is where we specify the size of the vector (or other data object) ahead of time and fill in the elements one by one. For example:

```
vec=rep(NA,25) #Preallocate an empty vector
for(i in 1:25){
  vec[i]=i #assign "i" to the ith element of "vec"
}
vec
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25
```

Preallocation is much faster because the memory for the entire vector or data object is specified ahead of time rather than rewriting it each time the object changes. We can use Sys.time() to see this.

```
vec=c() #specify an empty vector
a=Sys.time()
for(i in 1:25000){
  vec=c(vec,i) #append the loop index to "vec"
}
b=Sys.time()
b-a
```

```
## Time difference of 0.7776999 secs
```

```
vec=c() #specify an empty vector
vec=rep(NA,25000) #Preallocate an empty vector
a=Sys.time()
for(i in 1:25){
  vec[i]=i #assign "i" to the ith element of "vec"
}
b=Sys.time()
b-a
```

```
## Time difference of 0.003915071 secs
```

Now, we will move on to *while* loops. These are typically less commonly used, but sometimes necessary for particular tasks. *while* loops generally take the form:

```
while( condition ){
  expression
  expression
  expression
}
```

while loops allow us do do things like:

```
x=0
while(x<15){ #logical operator testing condition that x is less than 15
  x=x+2
  print(x)
}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 8
## [1] 10
## [1] 12
## [1] 14
## [1] 16
```

Here, we added 2 to x until it was no longer less than 15. One thing to watch out for with *while* loops is an infinite loop. For example:

```
x=0
while(x<15){
  x=x*2
  print(x)
}
```

If you run this loop, it will never satisify the condition required to stop. If you find yourself in this situation, you can press *escape* to stop the script from running (or press the *stop* button in the console).

The final set of control statements we will look at are *if* and *else*. These generally take the form:

```
if( condition ){
  expression
} else {
  expression
}
```

Although, the *else* does not always need to be specified if you do not want to do anything if *condition* is not met.

```
x=0
if(x==0){
  print("x is equal to 0")
}
```

```
## [1] "x is equal to 0"
```

```
x=1
if(x==0){
  print("x is equal to 0")
}else{
  print("x is not equal to 0")
}
```

```
## [1] "x is not equal to 0"
```

Often, there are more than 2 options. We can then use *else if*

```
x=1
if(x==0){
  print("x is equal to 0")
}else if(x<0){
  print("x is less than 0")
}else{
  print("x is greater than 0")
}
```

```
## [1] "x is greater than 0"
```

if statements are often nested within *for* loops to automate tasks that depend on the outcome of a logical operator. Here, we will specify a matrix with some missing values, loop over the rows with i and over the columns with j and count the missing values.

```
mat=matrix(1:9,nrow=3,ncol=3) #specify a 3 x 3 matrix
mat[1,3]=NA #set two elements to NA
mat[3,2]=NA
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    4   NA
## [2,]    2    5    8
## [3,]    3   NA    9
```

```
#how can we use a for loop to count the missing values in the matrix a?
count=0 #create an object to store the count of NAs
#go through each element of a, test for NA, and increment the count
for(i in 1:nrow(mat)){ #iterate across the rows of mat
  for(j in 1:ncol(mat)){ #iterate across the columns of mat
    if(is.na(mat[i,j])){ #is this element NA?
      count=count+1 #if yes, increment "count"
    }
  }
}
count
```

```
## [1] 2
```

```
#an alternative solution without using a for loop
sum(is.na(a))
```

```
## [1] 0
```

Exercises

1. Let's load the built-in "iris" data set.

```r
data("iris")
head(iris) #show the first 6 rows of the iris data frame
```

```
##   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
```

```r
head(iris,20) #look at first 20 rows
```

```
##    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
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
## 11          5.4         3.7          1.5         0.2  setosa
## 12          4.8         3.4          1.6         0.2  setosa
## 13          4.8         3.0          1.4         0.1  setosa
## 14          4.3         3.0          1.1         0.1  setosa
## 15          5.8         4.0          1.2         0.2  setosa
## 16          5.7         4.4          1.5         0.4  setosa
## 17          5.4         3.9          1.3         0.4  setosa
## 18          5.1         3.5          1.4         0.3  setosa
## 19          5.7         3.8          1.7         0.3  setosa
## 20          5.1         3.8          1.5         0.3  setosa
```

This data set contains various measurements for three iris species.

```r
unique(iris$Species) #what are the unique values of the "Species" column of the "iris" object?
```

```
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica
```

```r
table(iris$Species) #how many entries are there for each species?
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

1a. Count the number of entries for each species (i.e., reproduce the results from table() above) using logical subsetting. Hint: iris$Species, ?sum

1b. How many entries for species "setosa" have a petal width of less than 1? Hint: &

1c. How many entries have a petal width of less than 1 or a sepal length less than or equal to 5?

1c. How many entries have a petal width of less than 1 or a sepal length less than or equal to 5 and are of species "setosa"?

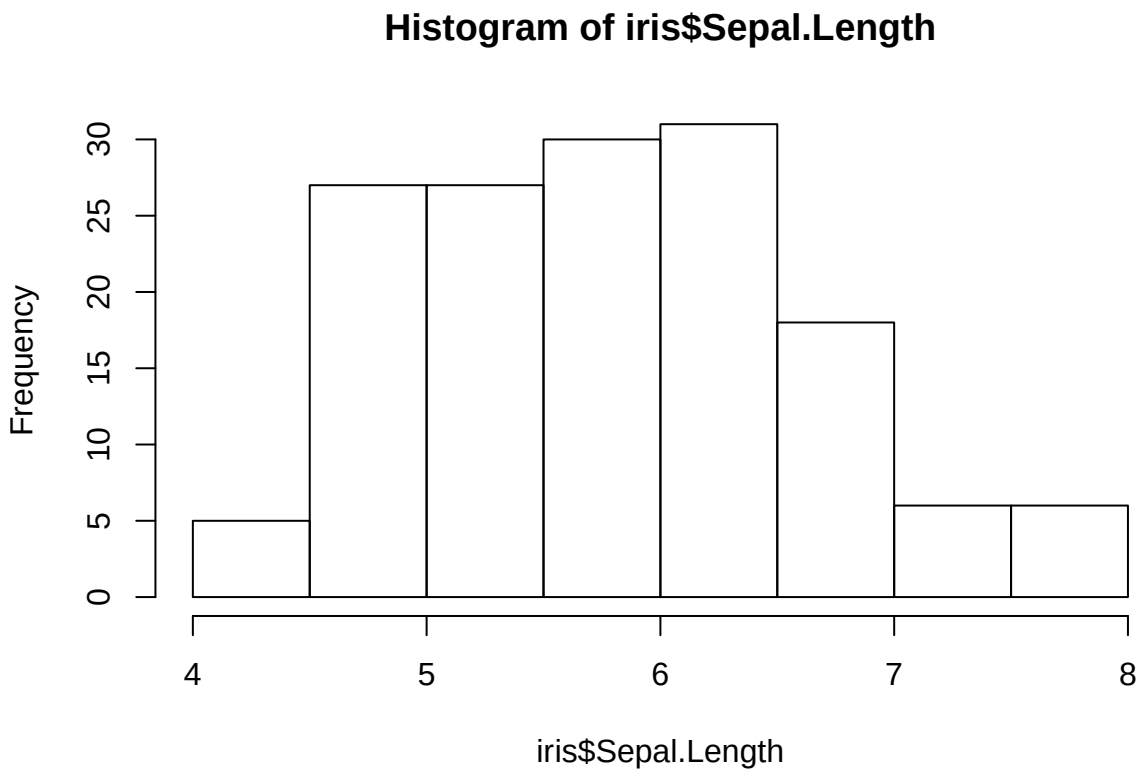
1d. Are any sepal lengths less than 8? Are they all less than 8?

1e. Use one or more *for* loops to count the number of entries for each species. Hint:

```
counts=rep(0,3) #create a vector of length 3 to store the counts
for(i in ?){
  #find species for row i
  counts[i]=counts[i]+1#increment appropriate count
}
```

1f. Let's look at a histogram of the values for sepal length in the iris data set.

```
hist(iris$Sepal.Length)
```



Create a new variable "SepalCat" in the iris data frame, which will will use to break the sepal length measurements into sepal length categories, "small", "medium", and "large". Use a *for* loop to assign these category levels. You can assign them using whatever criteria you like or you can use the quantiles of the data. For example, you could assign "small" to sepal lengths less than the 25th percentile, "medium" to sepal lengths in the middle 50th percentile, and "large" to sepal lengths greater than the 75th percentile. Hint:

```
iris$SepalCat=NA #create a new data frame column SepalCat, filled with missing data
head(iris) #make sure it worked
for(i in ?){
  #is iris$Sepal.Length small? then assign small
  iris$SepalCat[i]="small"
  #is it medium? Then assign medium
  iris$SepalCat[i]="medium"
  #else assign large
  iris$SepalCat[i]="large"
}
iris$SepalCat
```

2. Below is code modified from an earlier example. The *for* loop contains an error. Why does it not run? Can you fix it? Try not to look back at the original code.

```
mat=matrix(1:9,nrow=3,ncol=3) #specify a 3 x 3 matrix
mat[1,3]=NA #set two elements to NA
mat[3,2]=NA
mat
#how can we use a for loop to count the missing values in the matrix a?
count=0 #create an object to store the count of NAs
#go through each element of a, test for NA, and increment the count
for(i in 1:nrow(mat)){ #iterate across the rows of mat
  for(j in 1:ncol(mat)){ #iterate across the columns of mat
    if(is.na(mat[i,j+1])){ #is this element NA?
      count=count+1 #if yes, increment "count"
    }
  }
}
count
```

3. An equation for exponential population growth in discrete time is $N_t = \lambda N_{t-1}$. Use a *for* loop to calculate the population size over 10 years, starting with N(1)=100. Let's say lambda=1.1. What are N_2 through N_{11}? Can this be done without a for loop? Hint:

```
lambda=1.1 #set lambda
N=rep(NA,11) #specify a vector to hold N for 11 years
N[1]=100 #fill in the first year
N #which indices do we need to fill in?
for(t in ?){ #loop over these indices
  N[t]=? #How do we plug in the equation above?
}
```

4. Let's look at the lynx data set. Load and plot the data set. What are we looking at?

```
data(lynx)
lynx
plot(lynx)
?lynx
```

Use a for loop to calculate the change in population size between years. Can you do this without a for loop? If so, how might you do it (advanced question!)?

5. A very inefficient way to calculate C=A/B (A divided by B) is to subtract B from A until A is less than B to get the whole number and then use the remainder to get the remaining fraction. Say A is 100 and B is 11. How many times can we subtract 11 from A until A is less than 11? This is the whole number of times that A can be divided by B. Then we are left with the remainder. Use a while loop to calculate A/B. Hint:

```
value=100
divisor=11
count=1 #number of times we have subtracted the divisor
remainder=100-11
while(divisor<remainder){
  count=? #increment count
  remainder=? #subtract divisor from remainder
}
count+remainder/divisor #the answer for C
```