Lesson 0 Math Review and Intro to R

Outline

- Descriptive Statistics: Measures of Central Tendency and Variation
- Probability
  - Basic Review
  - Independent Events
  - Conditional Probability
- Introduction to R
  - Operators
  - Data Types
  - Data Structures (Vectors, Factors, Matrices, Data Frames)
  - Control structures
  - Functions

Measures of Central Tendency

- The descriptive measures that indicate the center of the data set or the most typical value of the data set are known as the *measures of central tendency*. The common measures are *mean*, *median*, and *mode*.
  - Mean: The mean is the numeric sum of the values divided by the number of values, it is also commonly referred to as the "average".

$$\bar{x} = \frac{\sum x_i}{n}$$

  - Median: The number that splits the top 50% and bottom 50% of the data.
  - Mode: The value(s) that occur(s) most frequently in the dataset.

Measures of Central Tendency

- Example: The following data set shows the scores for students in an exam:

  75, 72, 78, 70, 78, 78, 88, 75, 78, 72

- The mean of the data is 764/10=76.4
- For calculating the median, arrange the data in increasing order:

  70, 72, 72, 75, 75, 78, 78, 78, 78, 88

  Median = (75+78)/2 = 76.5.
- Mode = 78

Measures of Variation

- While the measures of central tendency provide one way for us to describe data, they are limited in that they don't provide a way for us to describe the spread (or variability) of the data.
- Consider the two data sets below:

  data set 1: 71, 72, 75, 75, 77

  data set 2: 26, 31, 75, 75, 163

The two data sets have same measures of center (same mean 74, median 75 and mode 75) but differ significantly - the spread (or variability) in the data set 2 is wider than in the data set 1.

Measures of Variation

- The most used measures of variability are the *range*, the *standard deviation* and the *interquartile range*.
  - The range is difference between the largest value (maximum) and the smallest value (minimum).
  - The standard deviation s is computed using the measure of the deviation from the mean. The variance of a sample (s^2) is defined as follows, where n is the number of values in the sample and $\bar{x}$ is the mean of the sample. (Example in next slide)

$$s^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

  - The more variation there is in the data set, the larger the standard deviation.

Measures of Variation

- To compute the sample standard deviation.
  - Subtract each value from the mean (66.8)
  - Square the difference
  - Sum the square of the differences
  - Divide the sum of the squares by the sample size (n-1) and take the square root

$$s^2 = \frac{\sum(x - \bar{x})^2}{n-1} = \frac{2323.6}{9} = 258.1778$$

$$\sqrt{s^2} = \sqrt{258.1778} = 16.07$$

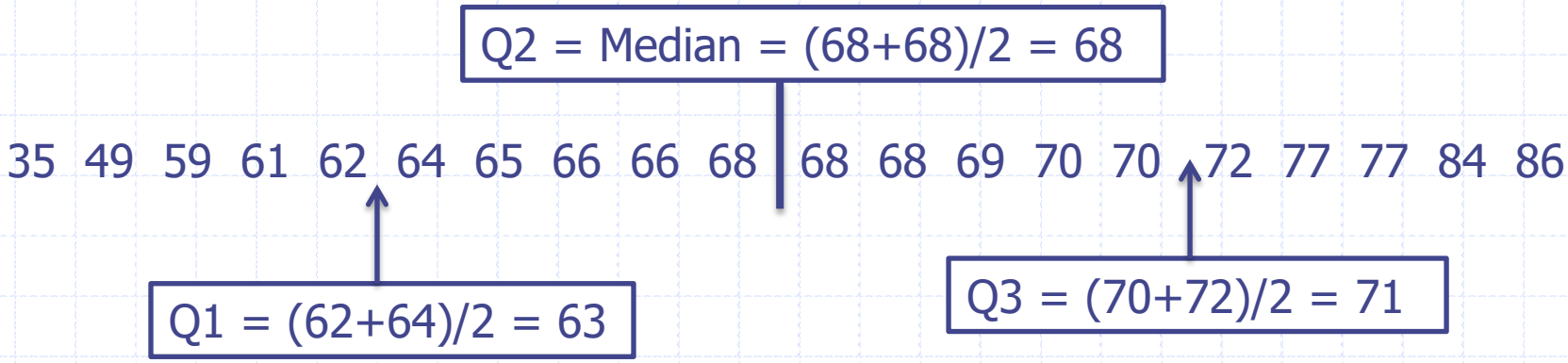
x_i	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
35	-31.8	1011.24
72	5.2	27.04
68	1.2	1.44
59	-7.8	60.84
61	-5.8	33.64
84	17.2	295.84
86	19.2	368.64
77	10.2	104.04
49	-17.8	316.84
77	10.2	104.04
	Sum	2323.6

Measures of Variation

- The mean (measure of center) and standard deviation (measure of variation) are sensitive to the extreme values. Measures based on percentiles are not as affected by extreme values.

Measures of Variation

- The quartiles are denoted by the three values Q1, Q2 and Q3. The first quartile, Q1, divides the bottom 25% of the data from the top 75%. The second quartile, Q2, is the median that divides the bottom 50% from the top 50%. The third quartile, Q3, divides the bottom 75% from the top 25%.
- Example (20 data points):



- The interquartile range (IQR) = Q3 - Q1

Population vs. Sample

- So far we have been discussing descriptive statistics of the sample (i.e. $\bar{x}$ denotes sample mean and s denotes sample standard deviation). We use these descriptive statistics when we are referring to a sample of data.
- The population describes all observations in a given population (i.e. all patients in the U.S. with diabetes). To denote the population mean we use the symbol μ ("mu"), to denote the population standard deviation we use the symbol σ^2 ("sigma" squared).

$$\mu = \frac{\sum x_i}{N} \qquad \sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

Optional: Why n-1?

The *sample variance* is a way to estimate population variance by treating a relatively small sample as representative of the whole population. We take a sample of n values y_1, ..., y_n from the population, where $n < N$ (where N is number of individuals in whole population) and estimate the variance on the basis of this sample. Directly taking the variance of the sample data gives the average of the squared deviations:

$$\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

This estimate of population variance tends, on average, to be slightly too small – it is said to be *downward biased.* This is shown by the calculation of the expected value of σ_y^2 (see https://en.wikipedia.org/wiki/Variance)

$$E[\sigma_y^2] = \frac{n-1}{n}\sigma^2$$

Optional: Why n-1?

Since the y_i are selected randomly, both $\bar{y}$ and σ_y^2 are random variables. Their expected values can be evaluated by averaging over the ensemble of all possible samples $\{y_i\}$ of size n from the population. For σ_y^2 this gives:

$$
\begin{aligned}
E[\sigma_y^2] &= E\left[\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \frac{1}{n}\sum_{j=1}^{n} y_j\right)^2\right] \\
&= \frac{1}{n}\sum_{i=1}^{n} E\left[y_i^2 - \frac{2}{n} y_i \sum_{j=1}^{n} y_j + \frac{1}{n^2}\sum_{j=1}^{n} y_j \sum_{k=1}^{n} y_k\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\left[\frac{n-2}{n}E[y_i^2] - \frac{2}{n}\sum_{j\neq i} E[y_i y_j] + \frac{1}{n^2}\sum_{j=1}^{n}\sum_{k\neq j} E[y_j y_k] + \frac{1}{n^2}\sum_{j=1}^{n} E[y_j^2]\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\left[\frac{n-2}{n}(\sigma^2+\mu^2) - \frac{2}{n}(n-1)\mu^2 + \frac{1}{n^2}n(n-1)\mu^2 + \frac{1}{n}(\sigma^2+\mu^2)\right] \\
&= \frac{n-1}{n}\sigma^2.
\end{aligned}
$$

Hence σ_y^2 gives an estimate of the population variance that is biased by a factor of $\frac{n-1}{n}$. For this reason, σ_y^2 is referred to as the *biased sample variance*. Correcting for this bias yields the *unbiased sample variance*:

$$
s^2 = \frac{n}{n-1}\sigma_y^2 = \frac{n}{n-1}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i-\bar{y})^2\right) = \frac{1}{n-1}\sum_{i=1}^{n}(y_i-\bar{y})^2
$$

Either estimator may be simply referred to as the *sample variance* when the version can be determined by context. The same proof is also applicable

Probability

- The probability is a numeric measure that represents the chance or likelihood that a particular event will occur. The value ranges 0 from (impossible event) to 1 (certain event).
- The collection of all possible outcomes for an experiment is known as the *sample space*. In the equal likelihood model, all outcomes have the same chance of occurring and the probability of each outcome is the reciprocal of the size of the sample space.

$$probability\ of\ event = \frac{\#\ ways\ event\ can\ occur}{total\ \#\ possible\ outcomes}$$

- An event is a collection of outcomes for an experiment.

Probability

- Consider the experiment of rolling a pair of balanced dice. The sample space consists of equally likely outcomes as shown below. Since all the outcomes are equally likely, the probability for any one of these outcomes is 1/36.

Die 2		1	2	3	4	5	6
Die 1	1	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
	2	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
	3	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)
	4	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)
	5	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)
	6	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)

Exercise

- Fill out the following table.

Event	Outcomes	Probability of the event
Sum of dice is 1		
Sum of dice is 6		
Two dice are equal		

Exercise - Solution

- Fill out the following table.

Event	Outcomes	Probability of the event
Sum of dice is 1	None	0
Sum of dice is 6	(1, 5) (2, 4) (3, 3) (4, 2) (5, 1)	5/36
Two dice are equal	(1,1) (2,2) (3,3) (4,4) (5,5) (6,6)	6/36

Probability

- P(E) represents the probability of an event E.
- If F and E are events, then the *probability of E given F* – notation: P(E|F) – is the probability that E occurs given that F occurs. In the equal likelihood model, assuming P(F) > 0, P(E|F) is computed in the following way – here, F is seen as the *restricted sample space*

$$P(E|F) = \frac{\#\ ways\ both\ E\ and\ F\ can\ occur}{\#\ ways\ F\ can\ occur}$$

- In general, when P(F)>0, conditional probability can be computed in the following way:

P(E|F) = (probability both E and F) / prob F
= P(E ∩ F) / P(F)

Probability

- Example: Consider the roll of a single die with the six possible outcomes {1, 2, 3, 4, 5, 6}. Let A be the event that a 4 is rolled, and let B be the event that the die comes up even.
  - P(A) = 1/6 and P(B) = 3/6
  - P(A|B) = 1/3 (Given the die comes up even, the possible outcomes are only {2, 4, 6})
  - $P(A \cap B) = 1/6$ ($A \cap B$ is the event "both even and 4")

Probability

- Events A and B are *mutually exclusive* if it is impossible for both events to occur at the same time – in other words, $A \cap B$ is empty.
  - If A and B are mutually exclusive, then $P(A \text{ or } B) = P(A \cup B) = P(A) + P(B)$
  - In general, $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- Events A and B are *independent* if the occurrence of one of the events does not affect the likelihood of the other event occurring. Formally, A and B are defined to be independent if
  - $P(A \text{ and } B) = P(A \cap B) = P(A) * P(B)$
  - In general $P(A \cap B) = P(B) * P(A \mid B)$.
- In example on previous slide, we have:

$$P(A|B) = P(A \cap B) / P(B) = (1/6)/(3/6) = 1/3$$

Probability

Exercise. Suppose A and B are events. Let -A be the event that A does not happen. Notice A and –A are mutually exclusive. Show:

1. $B = (A \cap B) \cup (-A \cap B)$
2. $P(B) = P(A \cap B) + P(-A \cap B)$
3. $P(B) = P(A) * P(B \mid A) + P(-A) * P(B \mid -A)$

   (We will use Exercise 3 in lesson 6)

Introduction to R and RStudio

- R is a programming language and free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.
- Download and install R from

  https://www.r-project.org/
- RStudio is a free and open-source integrated development environment (IDE) for R
- Download and install RStudio from

  https://www.rstudio.com/

Operators

- Assignment Operators:
  - <-, =, <<-, ->, -->
  - By convention, we use '<-' in this course.
- Arithmetic Operators:
  - +, -, *, /, ^(power), %%(module)
- Relational Operators:
  - >, <, !=, >=, <=,…
- Logical Operators:
  - &, |, !
- Miscellaneous Operators:
  - : (creates a series of numbers)
  - %in% (identify if an element belongs to a vector)

```
> v <- 2:8
> v
[1] 2 3 4 5 6 7 8
> y<-5
> print(y %in% v)
[1] TRUE
```

Data Types

- The following data types are frequently used in R:
  - Numeric
  - Integer
  - Logical
  - Character
  - Complex (a+bi)

Numeric Data

- The default data type for numbers is `numeric`.
- The "mode" function will tell you the storage type of your data.

```
> x<-5/2
> x
[1] 2.5
> mode(x)
[1] "numeric"
```

- For creating integer objects, the "as.integer" function is used to convert the data to the integer type.

```
> as.integer(5/2)
[1] 2
```

Character Data

- Any value written within a pair of single quote or double quotes in R is treated as a string. Internally R stores every string within double quotes, even when you create them with single quote.
- String values are represented as character objects.
- String objects can be concatenated using the `paste` function. The function takes a variable number of arguments of any type and returns a string using space as the default separator. The separator may also be explicitly specified using the `sep` argument.

```
> firstname <- 'Donald'
> lastname <- 'Trump'
> fullname <- paste(firstname, lastname)
> fullname
[1] "Donald Trump"
> fullname <- paste(firstname, lastname, sep = ", ")
> fullname
[1] "Donald, Trump"
```

Data Type Conversions

- The functions `as.numeric`, `as.character`, `as.logical`, `as.integer`, and `as.complex` are used for converting the given data into the required type. If the data cannot be converted, the value `NA` is returned.
- For numeric values, the value 0 is converted to FALSE and any non-zero value is converted to TRUE using `as.logical`.

```
> a<-0
> b<-9
> as.logical(a)
[1] FALSE
> as.logical(b)
[1] TRUE
```

```
> x<- TRUE
> y<-FALSE
> as.numeric(x)
[1] 1
> as.numeric(y)
[1] 0
```

Data Structures

- The following data structures are commonly used in R:
  - Vector: a collection of values of the same type
  - Factor: A collection of values from a fixed set of possible values
  - Matrix: A 2 dimensional collection of values of the same type
  - Array: any dimensional collection of values of the same type
  - List: A collection of any types
  - Data frame: A collection of vectors all of the same length

Vectors

- A vector is a collection of values that all have the same type.
- The `c()` function can be used to combine any number of existing vectors into a new vector.

```
> names <- c('joe','andy','bob')
> ages <- c(25, 45, 34)
> namesAndages <- c(names, ages)
> namesAndages
[1] "joe"  "andy" "bob"  "25"   "45"   "34"
> mode(namesAndages)
[1] "character"
```

- When vectors of different types are combined, the resulting vector will be converted to a common type. The type hierarchy is shown: `logical < integer < numeric < double < complex < character`

Vectors

- For numeric vectors the `length`() function returns the number of values in the given vector and the `sum`() function adds all the values.

```
> ages<-c(32,27,31)
> length(ages)
[1] 3
> sum(ages)
[1] 90
> ages.2017<-ages+1
> ages.2017
[1] 33 28 32
```

- Comparison operators can be used with vectors.

```
> ages>30
[1]  TRUE FALSE  TRUE
```

Vectors

- Arithmetic operators can be performed to create new vectors. If the length of the vectors are not equal, then the elements in the vector of the shorter vector are recycled to fill the gap.

```
> x <- c(10, 20, 30, 40)
> y <- c(2, 4)
>
> x*y
[1]  20  80  60 160
> x+y
[1] 12 24 32 44
```

Indexing Vectors

- The values of a vector can be indexed one at a time using integer values from 1 to the length of the vector.

```
> age.diagnosis<-c(35,72,68,59,61,84,86,77,49,77)
> age.diagnosis[1]
[1] 35
```

- Multiple values of a vector can be index in one step using a sequence of integers or using a vector of explicit indices.

```
> age.diagnosis[1:4]
[1] 35 72 68 59
> age.diagnosis[c(1,5,6:8)]
[1] 35 61 84 86 77
```

Sequences

- As seen previously, a sequence of values forming a vector of consecutive numbers can be generated the : operator (from:to). If the first value is less than the second, the sequences is generate in increasing order. Otherwise, the sequence is generated in decreasing order.

```
> 25:20
[1] 25 24 23 22 21 20
> 20:25
[1] 20 21 22 23 24 25
> LETTERS[4:1]
[1] "D" "C" "B" "A"

> age.diagnosis<-c(35,72,68,59,61,84,86,77,49,77)
> n<-length(age.diagnosis)
> age.diagnosis[5:n]
[1] 61 84 86 77 49 77
```

Sequences

- The generic function for generating sequences is the `seq()` function.
- The `by` argument and `length` argument can also be specified when generating sequences.

```
> age.diagnosis<-c(35,72,68,59,61,84,86,77,49,77)
> n<-length(age.diagnosis)
> age.diagnosis[seq(1:n)]
 [1] 35 72 68 59 61 84 86 77 49 77
> age.diagnosis[seq(1,n,by=2)]
[1] 35 68 61 86 49
> age.diagnosis[seq(1,by=2,length=5)]
[1] 35 68 61 86 49
```

Factors

- Factors are variables that have a limited number of different values (i.e. year in high school). The data can be stored in a numeric or character vector.
- The `factor` function creates levels for each of the distinct values of the factor variable.

```
> hs.year<-c("Senior","Freshman","Junior","Senior","Sophmore","Freshman")
> h<-factor(hs.year)
> h
[1] Senior   Freshman Junior   Senior   Sophmore Freshman
Levels: Freshman Junior Senior Sophmore
```

Factors

- The `summary()` function shows the frequencies of each of the distinct values.

```
> summary(h)
Freshman   Junior   Senior Sophmore
       2        1        2        1
```

- In cases where ordering exists among the different levels, the `ordered()` function factors the input data with the levels order by alphabetical order as default.

```
> hs.year<-c("Senior","Freshman","Junior","Senior","Sophmore","Freshman")
> m<-ordered(hs.year)
> m
[1] Senior   Freshman Junior   Senior   Sophmore Freshman
Levels: Freshman < Junior < Senior < Sophmore
```

- This ordering can be changed.

```
> hs.year<-c("Senior","Freshman","Junior","Senior","Sophmore","Freshman")
> m1<-ordered(hs.year,levels=c("Freshman","Sophmore","Junior","Senior"))
> m1
[1] Senior   Freshman Junior   Senior   Sophmore Freshman
Levels: Freshman < Sophmore < Junior < Senior
```

Matrices

- Suppose we have the exam grades of 5 students in 3 classes as shown below:

English	Math	Science
90	95	92
78	46	51
77	98	90
85	78	61
100	89	87

- To bring this data into R, we can create a numeric matrix.

```
> data<-c(90,95,92,78,46,51,77,98,90,85,78,61,100,89,87)
> grades<-matrix(data,nrow=5,ncol=3,byrow=TRUE)
> grades
     [,1] [,2] [,3]
[1,]   90   95   92
[2,]   78   46   51
[3,]   77   98   90
[4,]   85   78   61
[5,]  100   89   87
```

Matrices

- The values in the matrix can be assess using indices.

- Matrices are indexed by matrix[row,column]. Multiple columns and rows can be indexed.

```
> grades
     [,1] [,2] [,3]
[1,]   90   95   92
[2,]   78   46   51
[3,]   77   98   90
[4,]   85   78   61
[5,]  100   89   87
> grades[1,2]
[1] 95
> grades[1,]
[1] 90 95 92
> grades[,2]
[1] 95 46 98 78 89
```

```
> grades[c(1,4,5),c(2,3)]
     [,1] [,2]
[1,]   95   92
[2,]   78   61
[3,]   89   87
```

Matrices

- Oftentimes, it is useful to name the columns and rows of a matrix. This can be done using the `dimnames()` function.

```
> dimnames(grades)<-list(c("Janet","Eric","Paul","George","Matt"),
+ c("English","Math","Science"))
> grades
       English Math Science
Janet       90   95      92
Eric        78   46      51
Paul        77   98      90
George      85   78      61
Matt       100   89      87
```

- Once the rows and columns have been named, the values can be specified using the corresponding row and/or column name.

```
> grades["George",]
English    Math Science
     85      78      61
```

Matrices

- The `dim()` function return the dimensions of the matrix and the `nrow()` and `ncol()` functions return the number of rows and columns of the matrix.

```
> dim(grades)
[1] 5 3
> nrow(grades)
[1] 5
> ncol(grades)
[1] 3
```

- Modifying matrix entries is done similar to modifying vector entries.

```
> grades[1,2]<-94
> grades
       English Math Science
Janet       90   94      92
Eric        78   46      51
Paul        77   98      90
George      85   78      61
Matt       100   89      87
```

Lists

- Lists are a combination of data (they do not have to all be of the same type).

```
> sex<-c("m","f","f")
> age<-c(43,67,20)
> info<-list("Start",sex,age,"End")
> info
[[1]]
[1] "Start"

[[2]]
[1] "m" "f" "f"

[[3]]
[1] 43 67 20

[[4]]
[1] "End"
```

- Components of a list are accessed using the [] notation.

```
> info[3]
[[1]]
[1] 43 67 20
```

Data Frames

- Suppose we have the following data:

Name	Sex	Age.Diagnosis	Years.Diagnosis
Jennifer	Female	49	10
Alex	Male	57	5
Wes	Male	69	2
Ryan	Male	75	12

- We can construct a data frame as follows:

```
> name<-c("Jennifer","Alex","Wes","Ryan")
> sex<-c("Female","Male","Male","Male")
> age.diagnosis<-c(49,57,69,75)
> years.diagnosis<-c(10,5,2,12)
> hosp.info<-data.frame(name,sex,age.diagnosis,years.diagnosis)
```

- By default, the names of the vectors are used as the column names of the data frame.

Accessing Data Frame Data

- The data in a data frame can be access in a way similar to accessing data in a matrix.

```
> hosp.info[,1]
[1] Jennifer Alex     Wes      Ryan
Levels: Alex Jennifer Ryan Wes
```

- The `summary()` function produces a summary of the column data. The output of the `summary()` function varies depending on the data type.

```
> summary(hosp.info$sex)
Female   Male
     1      3
> summary(hosp.info[,3])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   49.0    55.0    63.0    62.5    70.5    75.0
```

Slicing Data Frame Rows

- The rows of the data may be sliced to produce a new data frame.

```
> jenny<-hosp.info[1,]
> jenny
     name    sex age.diagnosis years.diagnosis
1 Jennifer Female            49              10
```

- Subsets of a data frame can be created using the `subset()` function.

```
> males<-subset(hosp.info,sex=="Male")
> males
  name  sex age.diagnosis years.diagnosis
2 Alex Male            57               5
3  Wes Male            69               2
4 Ryan Male            75              12
```

Modifying a Data Frame

- A new column can be added to a data frame using a vector of explicit values or through creation of a new variable.

```
> hosp.info$new.age<-hosp.info$age.diagnosis+hosp.info$years.diagnosis
> hosp.info
      name    sex age.diagnosis years.diagnosis new.age
1 Jennifer Female            49               10      59
2     Alex   Male            57                5      62
3      Wes   Male            69                2      71
4     Ryan   Male            75               12      87
```

- The `new.age` variable computes the current age of the patients.
- A column can be removed entirely from the data frame by assigning the value NULL for the entire column.

```
> hosp.info$name<-NULL
> hosp.info
     sex age.diagnosis years.diagnosis new.age
1 Female            49               10      59
2   Male            57                5      62
3   Male            69                2      71
4   Male            75               12      87
```

Control Structures

- if, if-else
- switch statement
- for loop
- while loop
- repeat loop

For Loop

- The `for` statement can be used to loop over a series of elements stored in a list, vector, matrix or data frame.

```
> a<-c(100,81,64,49,36,25,16,9,4,1)
> for (i in a){
+   print(paste("Square root of",i, "is",sqrt(i)))
+    }
[1] "Square root of 100 is 10"
[1] "Square root of 81 is 9"
[1] "Square root of 64 is 8"
[1] "Square root of 49 is 7"
[1] "Square root of 36 is 6"
[1] "Square root of 25 is 5"
[1] "Square root of 16 is 4"
[1] "Square root of 9 is 3"
[1] "Square root of 4 is 2"
[1] "Square root of 1 is 1"
```

Repeat Loop

- The `repeat` loop executes the same code again and again until a stop condition is met.

```
data <- c(90,95,92,78,46,51,77,98,90,85,78,61,100,89,87)
i <- 1
repeat{
        if(data[i] > 80) {
                print(data[i])
                i <- i+1 #no i++ in R
        } else
          break
}
```

Functions and Function Arguments

- A function in R takes in zero or more arguments and returns a value. Below we show an example of a function called "dec" which takes an argument, x, and returns a value of x-1.

```
> dec<-function(x){
+   return(x-1)
+ }
```

- To call the function `dec` all we need to do is input a value for the argument x.

```
> dec(6)
[1] 5
```