Analytics Programming – R

Week 6

Leadership

Naveen Kumar

ChooseMIS
Major in the Future

Agenda

- Week 5 Review
- Data Structures
- Data Merging
- Loops and Business Applications
- User Defined Functions
- Hands-on with ggplot
- Summary and Conclusion

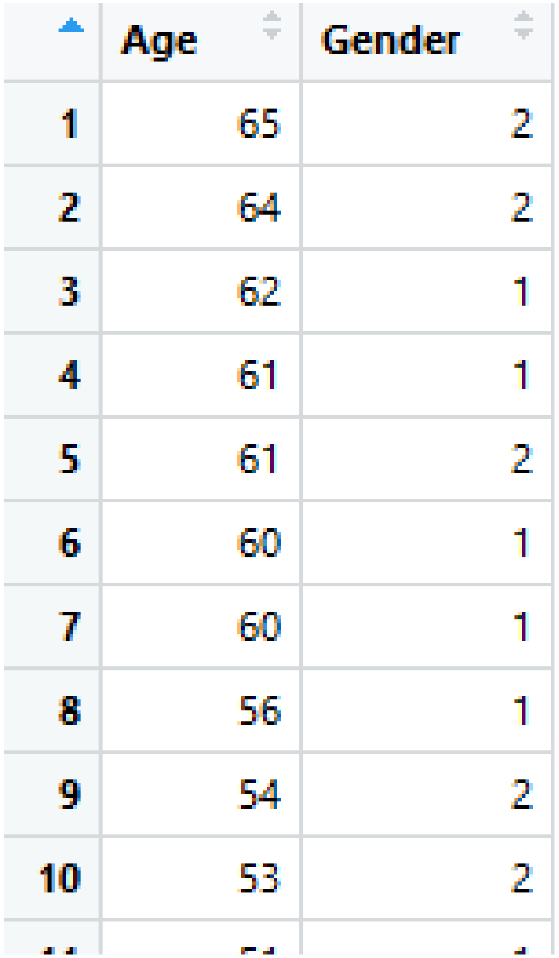
Data Structures & Data Types

ChooseMIS
Major in the Future

cbind()

- cbind() function is used to combine different columns into a dataframe.
- In the example we are creating a dataframe with age and gender columns from med.data using cbind() and naming them as "Age" and "Gender"

age.data <- as.data.frame(cbind(Age = med.data$AgeInYears, Gender = med.data$Gender))



	Age	Gender
1	65	2
2	64	2
3	62	1
4	61	1
5	61	2
6	60	1
7	60	1
8	56	1
9	54	2
10	53	2

ChooseMIS
Major in the Future

What is Data Structure & Why Learn About It?

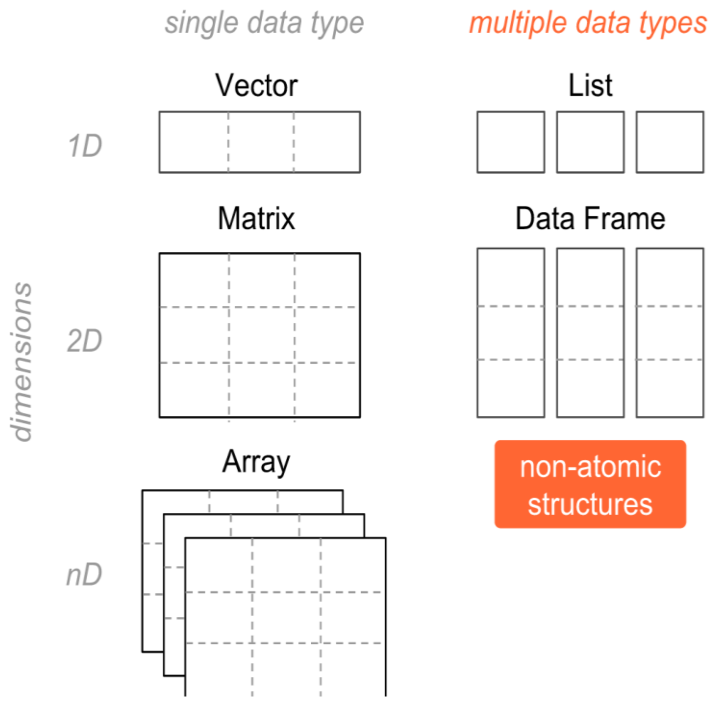
- **What?**
  - Data containers that hold different objects we create in programming

    Example: Matrix, Array, Data Frame etc.
- **Why?**
  - Data structure allows us to process data in a way that R can understand

R – Data Structures

- Data Structures are data containers or objects to hold and handle data
- R provides a handful of very flexible data objects

	Homogeneous Data	**Heterogeneous Data**
One Dimension	Vector	
Two Dimensions	Matrix	Data Frame
N Dimensions	Array	List



Vector

- Only one-dimensional objects
- Vector can be of any length (in one-dimension though)
  - Scalar is a one element vector
- All elements of a vector should be of the same data type

Example:

```
> x <- c(1,2,3,4,5)
> print(x)
[1] 1 2 3 4 5
> y <- 1:5     (creates a vector with values 1 to 5)
> print(y)
[1] 1 2 3 4 5
```

Vector Examples

- **Integer Vector** with age of students in a group

  age <- c(23L, 29L, 37L, 22L, 23L)

  Though an Integer Vector can be defined as a Double Vector as well; it is a lot more memory efficient to define Integer values explicitly as Integers.

- **Numeric (Double) Vector** with GPA of students in a group

  GPA <- c(3.22, 2.99, 4.0, 3.10, 2.89)

- **Character Vector** of First Names of students in a group

  fs.name <- c("Anne", "Dave", "Matt", "Amy", "Paul")

- **Logical Vector** indicating if a student is "in state" or not.

  in.state <- c(TRUE, TRUE, FALSE, FALSE, TRUE)

Matrix and Array

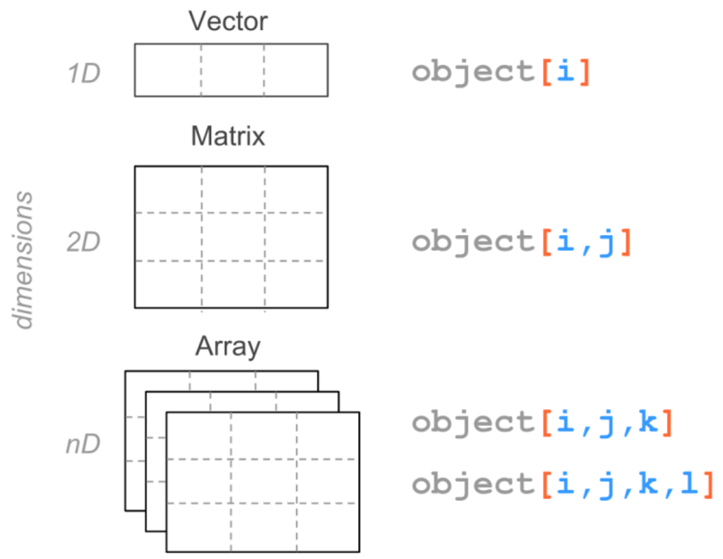
Dimensions

- Matrix is two dimensional objects (has rows and columns)
- Array is multi-dimensional objects i.e. it can store data in more than two dimensions.

Homogeneous

- All elements of a matrix and array should be of the same data type.

Single Data Type / Homogeneous



If you mix different data types into a matrix or array; R will implicitly coerce the elements into the most flexible data type.

ChooseMIS Major in the Future

Matrix Example

```
> input.matrx <- matrix(c(2,2.5,3,4,5,7),2,3)
> input.matrx
     [,1] [,2] [,3]
[1,]  2.0    3    5
[2,]  2.5    4    7

> class(input.matrx)
[1] "matrix" "array"

> typeof(input.matrx)
[1] "double"

> dim(input.matrx)
[1] 2 3
```

```
> input.matrx <- matrix(c(2,2.5,3,4,5,7), 2,3,
byrow = TRUE)
> input.matrx
     [,1] [,2] [,3]
[1,]    2  2.5    3
[2,]    4  5.0    7
```

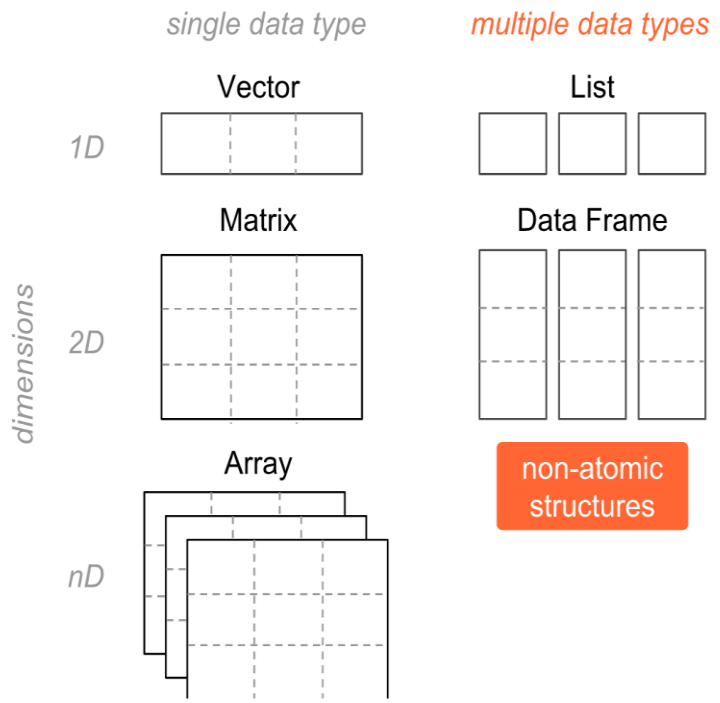
ChooseMIS Major in the Future

Data Frame

- A data.frame is the primary data structure that R provides for handling tabular data sets

- Columns are typically homogeneous

- Data frame can hold different types of columns



Data Frame Example

```
beer.data <- read.csv(file = "BeerDataExample.csv", sep =
"," , header = TRUE)
```

```
> class(beer.data)
[1] "data.frame"
```

Data Types

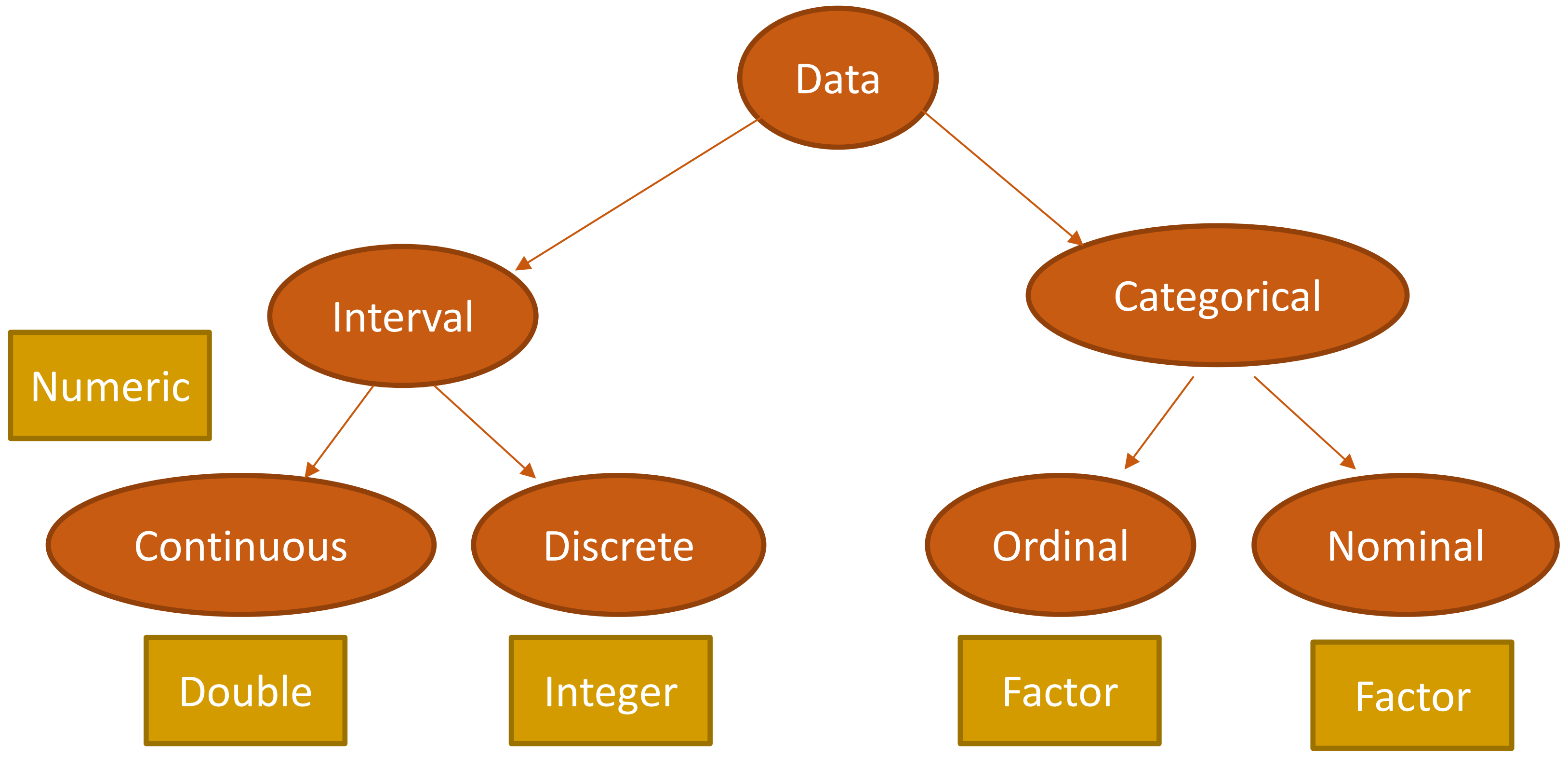
- **What is a Data Type?**

  The data type of a value (or vector) is an attribute/property that defines what kind of data that value/vector can have.

- **Why do we need to know data types?**

  Data types are important in programming languages so that memory can be allocated to store the values of a particular data type in an efficient manner.

Mapping of Programming Data Types to Analytical Data Types



Some Useful Commands

class()

- Returns the type of an object that is used to store data
- Example: factor, matrix, dataframe
- Example:

```
> employee.names <- c("Joe", "Adam", "William")
> class(employee.names)
[1] "character"
```

Data Type: Character

Character is used to represent string/text data in R

```
>company.name <- c("Google", "IBM", "HP")
    > class(company.name)
    [1] "character"
```

The R function **is.character()** is used to determine if an object is a character or not. It returns a Boolean value i.e., TRUE/FALSE

```
>is.character(company.name)
[1] TRUE
```

length() function returns the length of vectors and any R objects

```
> length(company.name)
[1] 3
```

Data Type: Double

Data type double includes integers, rational numbers (fractions and decimals)

Example: 2.45, 78.022, 1.01

```
>company.innovation.index <- c(0.9, 0.6, 0.5)
>class(company.innovation.index)
[1] "numeric"
```

The R function **is.double()** is used to determine if an object is stored as "double" data type or not. It returns a Boolean value i.e., TRUE/FALSE

```
>is.double(company.innovation.index)
[1] TRUE
```

Data Type: Integer

Integers are the numbers which do not have any fractional/ decimal component.
Example: 2, 0, and 45

```
>company.employees <- c(40000, 25000, 10000)
> class(company.employees)
[1] "numeric"
> typeof(company.employees)
[1] "double"
```

In the R-example the notation 40000L explicitly stores the data as an integer.

```
> company.employees <- c(40000L, 25000L, 10000L)
> class(company.employees)
[1] "integer"
> length(company.employees)
[1] 3
```

The R function **is.integer()** is used to determine if an object is stored as "integer" data type or not. It returns a Boolean value i.e., TRUE/FALSE

```
> is.integer(company.employees)
[1] TRUE
```

Data Type: Logical

Logical data type represents Boolean data

Example: TRUE/FALSE, 0 or 1

```
>company.mobile.interest <- c(TRUE, FALSE, TRUE)

>class(company.mobile.interest) [1] "logical"

>typeof(company.mobile.interest) [1] "logical"

> str(company.mobile.interest)
logi [1:3] TRUE FALSE TRUE
```

The R function **is.logical()** is used to determine if an object is stored as “logical” data type or not. It again returns a Boolean value i.e., TRUE/FALSE

```
>is.logical(company.mobile.interest)
[1] TRUE
```

Factor Data

Factor is designed to handle categorical data. It is a vector in which the distinct values are stored as levels

To create a factor, typically you pass a vector to the function factor()

```
>size.chr <- c ('S', 'L', 'M', 'M', 'S')
> size <- factor(size.chr)
> class(size)
[1] "factor"
```

Factors are internally stored as vector of integers. This makes factors excellent to work categorical (especially ordinal) data.

```
> typeof(size)
[1] "integer"
> str(size)
Factor w/ 3 levels "L","M","S": 3 1 2 2 3
```

The R function **is.factor()** is used to determine if an object is stored as “factor” data type or not. It again returns a Boolean value i.e., TRUE/FALSE

```
> is.factor(size)
[1] TRUE
```

Data Merging

What is it?

Why is it useful?

Data Merging

What is Data Merging?

It is a process of combining two or more data sets into one single data set.

Why is it useful?

In real world, all the data that is necessary for our analysis may not be present in a single worksheet/file.

Hence it is important to merge data from different sources and tables so that it can be analyzed all in one go.

Left Outer Join

- Use datasets Billingdata and Vitaldata for merging
- Read the two csv files using the below code

```
billing.info <- read.csv('Billingdata.csv', sep = ",", header = TRUE)
vital.info <- read.csv('Vitaldata.csv', sep = ",", header = TRUE)
```

- The columns that are common in both these datasets are “FirstName” and “LastName”
- As there can be more patient with the same firstname or lastname, use both columns as the merging conditions

Left Outer Join

- merge() function is used to perform joins in R

Syntax:

- merge(x , y , by = …, all. …..)

  "billing.info" is the x

  "vital.info" is the y

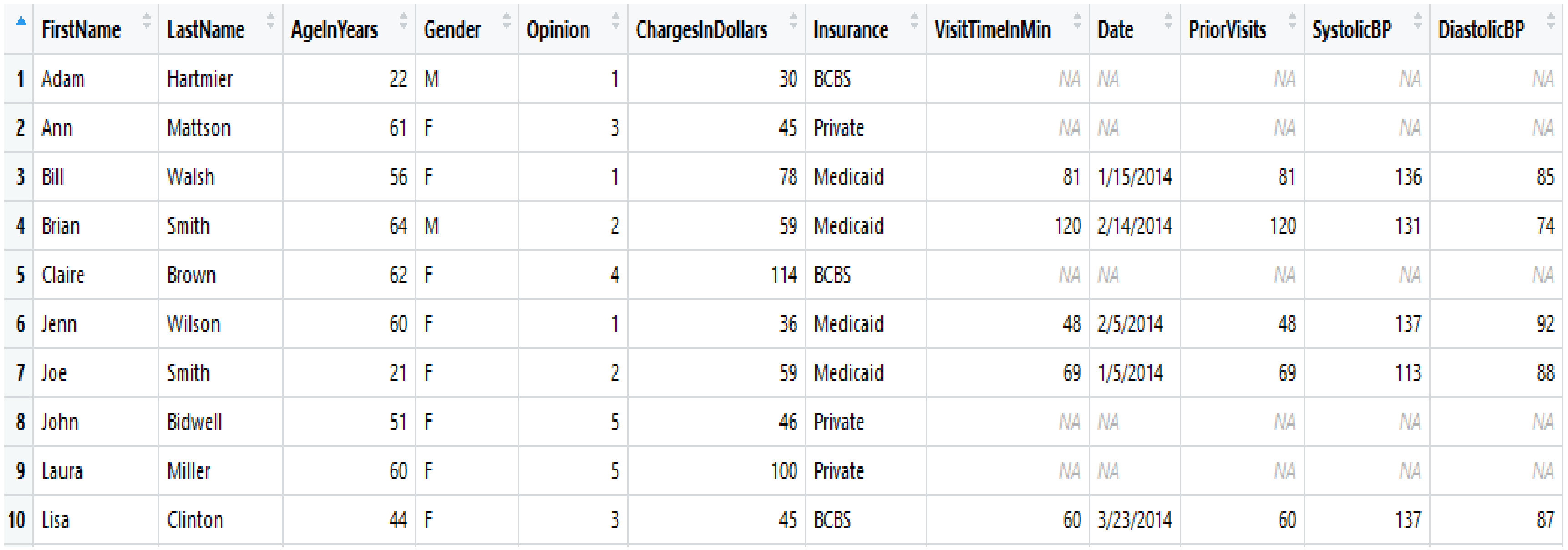
  "by= " is the condition by which the records are merged

  "all. " is the argument that is used to select left (all.x) or right (all.y) outer join

Left Outer Join

- **leftjoin.data <- merge(billing.info, vital.info, by = c('FirstName', 'LastName'), all.x = TRUE)**

Here we are setting all.x = TRUE, because we want all the records in table 1 (billing.info) and only matched records in table 2(vital.info)



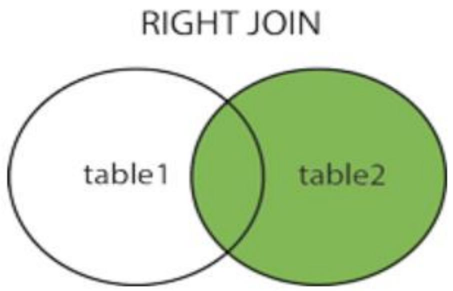
	FirstName	LastName	AgeInYears	Gender	Opinion	ChargesInDollars	Insurance	VisitTimeInMin	Date	PriorVisits	SystolicBP	DiastolicBP
1	Adam	Hartmier	22	M	1	30	BCBS	NA	NA	NA	NA	NA
2	Ann	Mattson	61	F	3	45	Private	NA	NA	NA	NA	NA
3	Bill	Walsh	56	F	1	78	Medicaid	81	1/15/2014	81	136	85
4	Brian	Smith	64	M	2	59	Medicaid	120	2/14/2014	120	131	74
5	Claire	Brown	62	F	4	114	BCBS	NA	NA	NA	NA	NA
6	Jenn	Wilson	60	F	1	36	Medicaid	48	2/5/2014	48	137	92
7	Joe	Smith	21	F	2	59	Medicaid	69	1/5/2014	69	113	88
8	John	Bidwell	51	F	5	46	Private	NA	NA	NA	NA	NA
9	Laura	Miller	60	F	5	100	Private	NA	NA	NA	NA	NA
10	Lisa	Clinton	44	F	3	45	BCBS	60	3/23/2014	60	137	87

ChooseMIS
Major in the Future

Right Outer Join

- Right outer join returns all the records from table 2 and only matched records from table 1 (matching based on the condition/criteria mentioned)

```
rightjoin.data <- merge(billing.info, vital.info, by = c('FirstName',
'LastName'), all.y = TRUE)
```

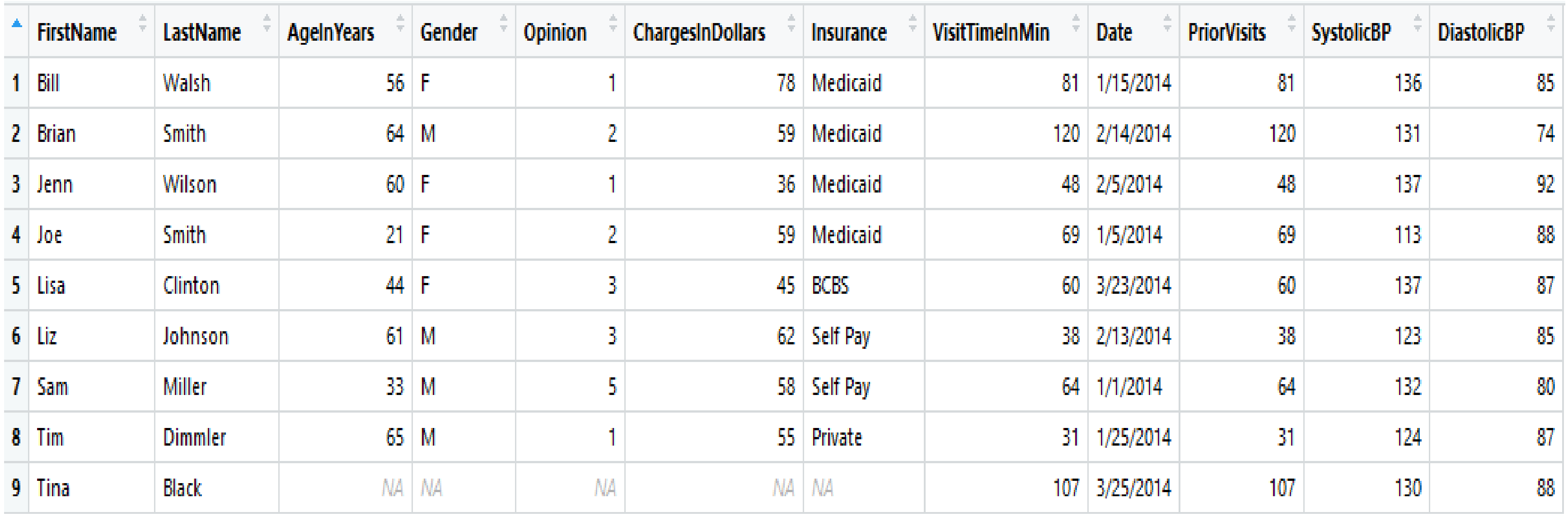


- The only difference between left and right join syntax is in case of right join we enter all.y = TRUE because we only want all the records from table 2(i.e., vital.info) and only matched data from table 1.

ChooseMIS Major in the Future

Right Outer Join

- **rightjoin.data <- merge(billing.info, vital.info, by = c('FirstName', 'LastName'), all.y = TRUE)**

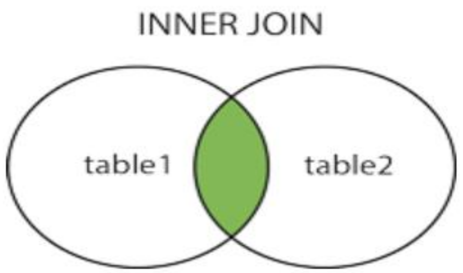


	FirstName	LastName	AgeInYears	Gender	Opinion	ChargesInDollars	Insurance	VisitTimeInMin	Date	PriorVisits	SystolicBP	DiastolicBP
1	Bill	Walsh	56	F	1	78	Medicaid	81	1/15/2014	81	136	85
2	Brian	Smith	64	M	2	59	Medicaid	120	2/14/2014	120	131	74
3	Jenn	Wilson	60	F	1	36	Medicaid	48	2/5/2014	48	137	92
4	Joe	Smith	21	F	2	59	Medicaid	69	1/5/2014	69	113	88
5	Lisa	Clinton	44	F	3	45	BCBS	60	3/23/2014	60	137	87
6	Liz	Johnson	61	M	3	62	Self Pay	38	2/13/2014	38	123	85
7	Sam	Miller	33	M	5	58	Self Pay	64	1/1/2014	64	132	80
8	Tim	Dimmler	65	M	1	55	Private	31	1/25/2014	31	124	87
9	Tina	Black	NA	NA	NA	NA	NA	107	3/25/2014	107	130	88

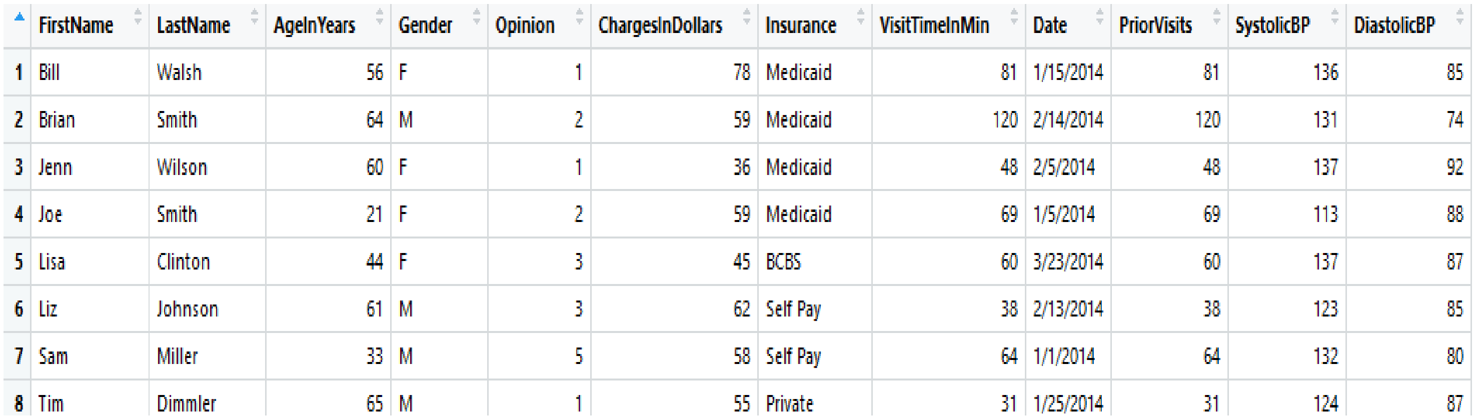
ChooseMIS
Major in the Future

Inner Join

- Inner join merges tables 1 and 2 records that satisfy a specific condition
- Inner join returns the records that are common in both tables 1 and 2



innerjoin.data <- merge(billing.info, vital.info, by = c('FirstName', 'LastName'), all= FALSE)



	FirstName	LastName	AgeInYears	Gender	Opinion	ChargesInDollars	Insurance	VisitTimeInMin	Date	PriorVisits	SystolicBP	DiastolicBP
1	Bill	Walsh	56	F	1	78	Medicaid	81	1/15/2014	81	136	85
2	Brian	Smith	64	M	2	59	Medicaid	120	2/14/2014	120	131	74
3	Jenn	Wilson	60	F	1	36	Medicaid	48	2/5/2014	48	137	92
4	Joe	Smith	21	F	2	59	Medicaid	69	1/5/2014	69	113	88
5	Lisa	Clinton	44	F	3	45	BCBS	60	3/23/2014	60	137	87
6	Liz	Johnson	61	M	3	62	Self Pay	38	2/13/2014	38	123	85
7	Sam	Miller	33	M	5	58	Self Pay	64	1/1/2014	64	132	80
8	Tim	Dimmler	65	M	1	55	Private	31	1/25/2014	31	124	87

ChooseMIS Major in the Future

Decision making using Flow Control (Conditional statements)

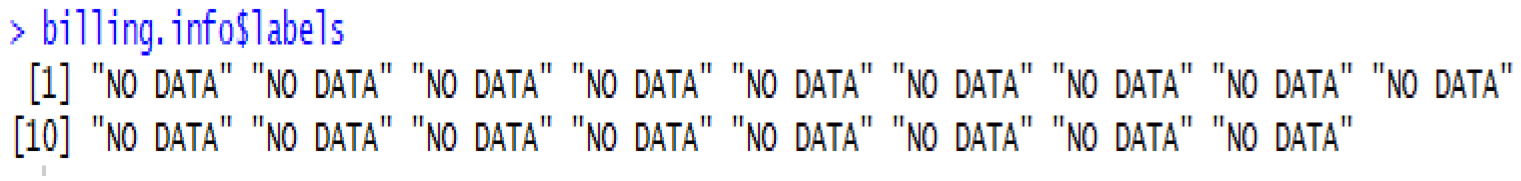
ChooseMIS
Major in the Future

Classify Patient As High/Low Maintenance Based On Charges

- Create new variable “labels” in billing.info dataset with value “NO DATA”

billing.info$labels <- rep("NO DATA", nrow(billing.info))

- rep() function replicates the specified value.
- In the above example, we are replicating “NO DATA” as many times as the number of rows in billing.info dataset



```
> billing.info$labels
 [1] "NO DATA" "NO DATA" "NO DATA" "NO DATA" "NO DATA" "NO DATA" "NO DATA" "NO DATA" "NO DATA"
[10] "NO DATA" "NO DATA" "NO DATA" "NO DATA" "NO DATA" "NO DATA" "NO DATA" "NO DATA"
```

ChooseMIS
Major in the Future

Classify Patient As High/Low Maintenance Based On Charges

Use of if-else()

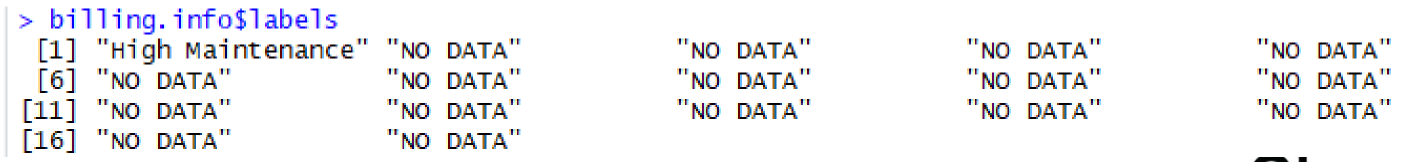
- A statement or an expression where a decision is made to execute some part of code based on True/False condition.
- 'if-else’ is more commonly used conditional expression
- **if** (needs an explicit condition) and **else** (does not need an explicit condition and so it has to be combined with an if statement)

ChooseMIS
Major in the Future

Classify Patient As High/Low Maintenance Based On Charges

- In the below example, the conditional statement is “if the first value of ChargesInDollars in billing.info is greater than 50” then we add the string “High Maintenance” to the first row labels column.
- Else we enter “Low Maintenance”

```
if(billing.info$ChargesInDollars[1]>50)
{
  billing.info$labels[1] <- "High Maintenance"
} else {
  billing.info$labels[1] <- "Low Maintenance"
}
```



```
> billing.info$labels
 [1] "High Maintenance" "NO DATA"          "NO DATA"          "NO DATA"          "NO DATA"
 [6] "NO DATA"          "NO DATA"          "NO DATA"          "NO DATA"          "NO DATA"
[11] "NO DATA"          "NO DATA"          "NO DATA"          "NO DATA"          "NO DATA"
[16] "NO DATA"          "NO DATA"
```

ChooseMIS
Major in the Future

Introduction to Loops – for Loop

- Loops are used in R to run a specific block of code until it is no longer true.
- for loop can be used not only for vectors but also for lists, matrices, and dataframes.

Syntax:

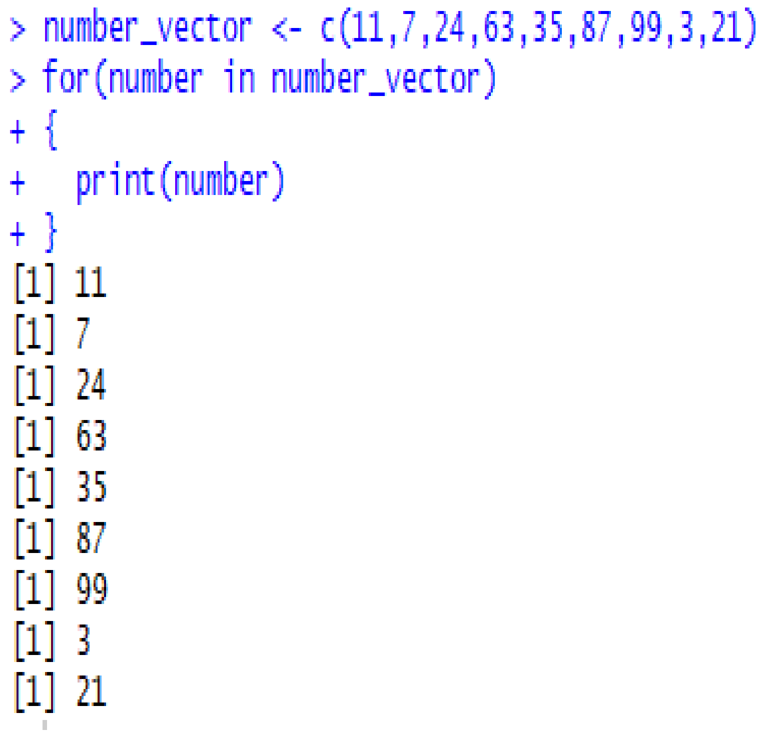
```
for (element in vector/matrix/list) {
 expression}
```

Example:

```
number_vector <- c(11,7,24,63,35,87,99,3,21)
for(number in number_vector)
{
print(number)
}
```

Introduction to Loops – for Loop

- The example is a simple for loop in which every number in the number_vector is passes through the loop one after the other and the number is printed using print() function



```
> number_vector <- c(11,7,24,63,35,87,99,3,21)
> for(number in number_vector)
+ {
+   print(number)
+ }
[1] 11
[1] 7
[1] 24
[1] 63
[1] 35
[1] 87
[1] 99
[1] 3
[1] 21
```

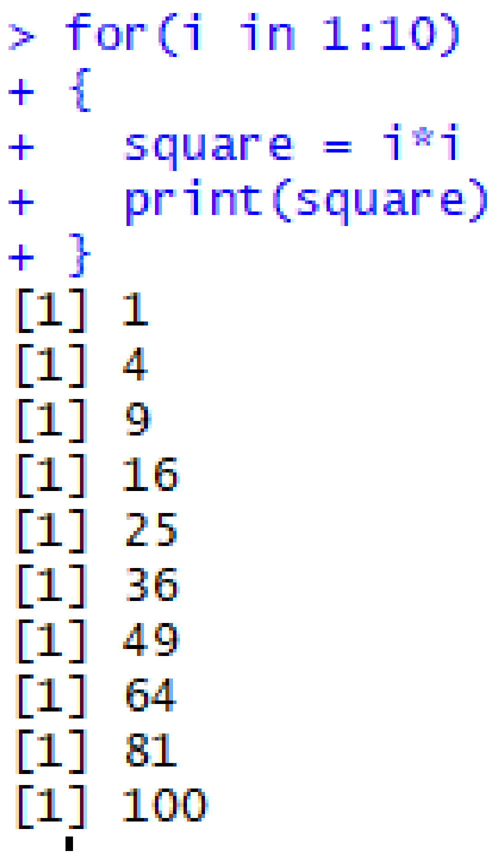
Introduction to Loops – for Loop

- for() loop runs in three different steps

Initialization: In this step, we initialize a variable (**i** in this case)and test whether i within our specified range 1:10

Assignment: Next we open the loop and assign the product i*i to a variable "square"

Iteration: The variable i first takes value 1 from 1:10, enters the loop, performs 1*1 and prints result as 1. Next it takes 2 and prints 4, and so on..



```
> for(i in 1:10)
+ {
+   square = i*i
+   print(square)
+ }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
```

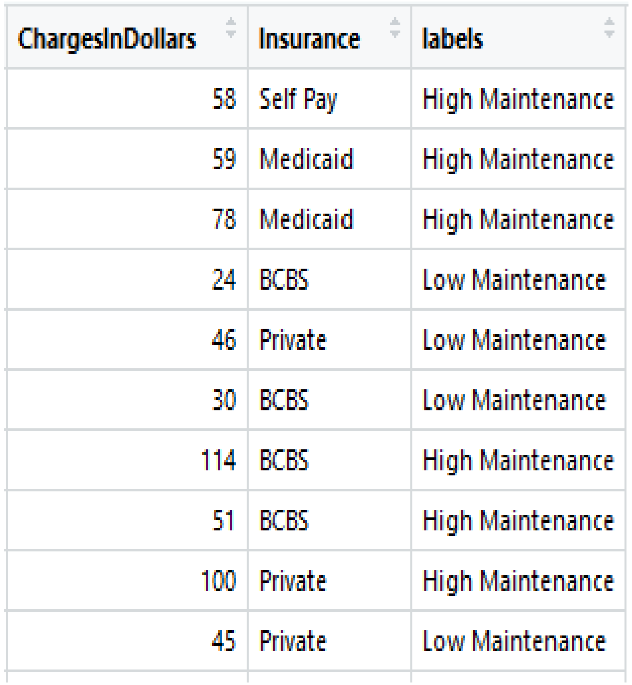
ChooseMIS
Major in the Future

Introduction to Loops

```
for(i in 1:nrow(billing.info)){
 if(billing.info$ChargesInDollars[i] > 50){
  billing.info$labels[i] <- "High Maintenance"
 } else {
  billing.info$labels[i] <- "Low Maintenance"
 }
}
```

- In the above example, we are iterating through each row value of the column “ChargesInDollars” in billing.info dataset

- The Conditional statement is defined in the if-else() statement and the values High/Low Maintenance are updated based on the result from the conditional statement (TRUE/FALSE)



ChargesInDollars	Insurance	labels
58	Self Pay	High Maintenance
59	Medicaid	High Maintenance
78	Medicaid	High Maintenance
24	BCBS	Low Maintenance
46	Private	Low Maintenance
30	BCBS	Low Maintenance
114	BCBS	High Maintenance
51	BCBS	High Maintenance
100	Private	High Maintenance
45	Private	Low Maintenance

User Defined Functions

- Functions groups together a set of statements so that they can be run more than once and allows us to specify what goes into the function
- In R, there are several built-in functions
- R also accepts user created functions
- **Syntax**

```
myfunction <- function(input1, input2) {
  #execute code to return something
}
```

ChooseMIS
Major in the Future

User Defined Functions

- function() command is used to define a user-created function in R

- In the example below, we are defining a function **sum.numbers** using the arguments num1 and num2

- Once the function is defined, sum will be calculated for any numbers passed in to **sum.numbers** function

- Example:

```
sum.numbers <- function(num1, num2) {
  print(num1+num2)
}
sum.numbers(45,82)
```

Hands-on with ggplot

Summary and Questions

ChooseMIS
Major in the Future