CHAPTER 2

BASIC TASKS IN R

This chapter introduces a number of basic skills you need to work in R. We start by discussing how to assign and use objects—the foundation of object-oriented programming. Then, we cover assigning vectors and creating, exporting, and importing data frames. Finally, we address converting variables into a different data type (e.g., numeric to factor).

CODING IN R: OBJECT-ORIENTED PROGRAMMING

R uses object-oriented programming. Essentially, you can assign data, values, output, and functions to an object: a data structure that holds this information for later use. This assignment uses the symbols < and - together as <-. In RStudio, this symbol can be created using `alt` and `-` together.

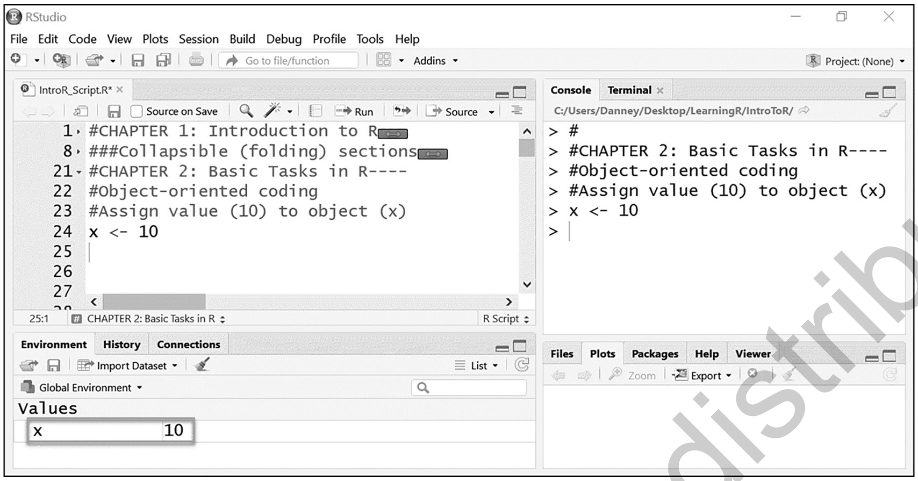
We can use this symbol to assign a value on the right side of the symbol to an object on the left side of the symbol. For example, we can assign the value 10 to the object x.

```
x <- 10
```

Remember: You will use `ctrl` + `enter` to run the line of code. You may also want to use a comment (#) to write yourself a reminder.

After you successfully run the code, you can see the newly created object at the bottom left of the screen in the *`Environment`* pane. This pane is helpful as you can refer to it when recalling object names. Alternatively, you can request a **l**i**s**t of objects: *`ls()`*. This function returns all of the objects currently in the *`Environment`*.

Note: Throughout this book, periods are occasionally added to lines of code because they are part of a sentence; however, the **code will not work with the period.**

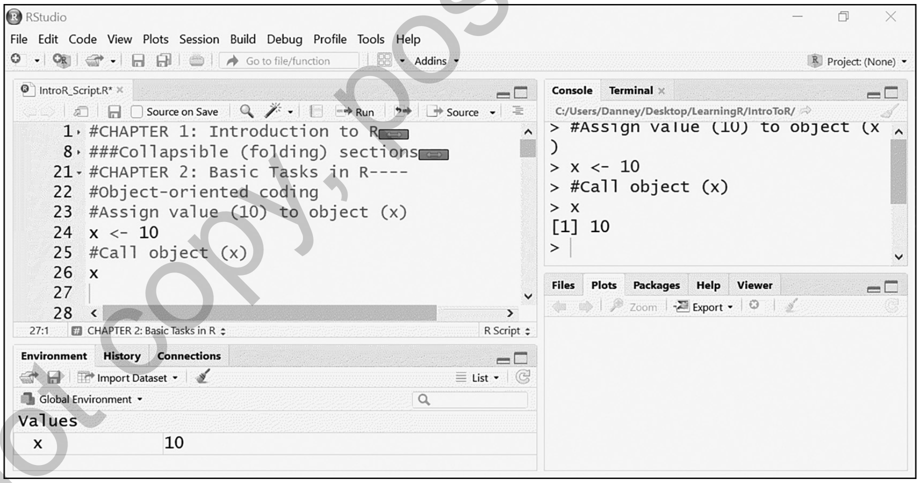


```
#CHAPTER 1: Introduction to R
###Collapsible (folding) sections
#CHAPTER 2: Basic Tasks in R----
#Object-oriented coding
#Assign value (10) to object (x)
x <- 10
```

```
> #
> #CHAPTER 2: Basic Tasks in R----
> #Object-oriented coding
> #Assign value (10) to object (x)
> x <- 10
>
```

```
Values
x               10
```

Once you create an object, you can use or call that object, or you can remove (*rm*) the object if you no longer need it. For example, we can call the object *x*, which should return the value 10. We simply type *x* and hit *ctrl* + *enter* to run the line of code. You should see the new line of code and the value associated with the object in the *Console* pane.



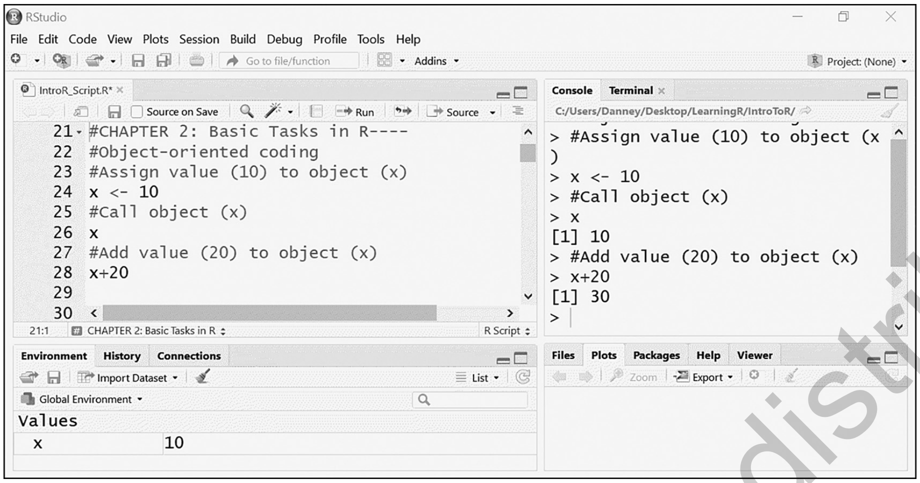
```
#CHAPTER 1: Introduction to R
###Collapsible (folding) sections
#CHAPTER 2: Basic Tasks in R----
#Object-oriented coding
#Assign value (10) to object (x)
x <- 10
#Call object (x)
x
```

```
> #Assign value (10) to object (x
)
> x <- 10
> #Call object (x)
> x
[1] 10
>
```

```
Values
x               10
```

We also can use the object. A simple example is to add the value 20 to the existing object using this code:

```
x+20
```



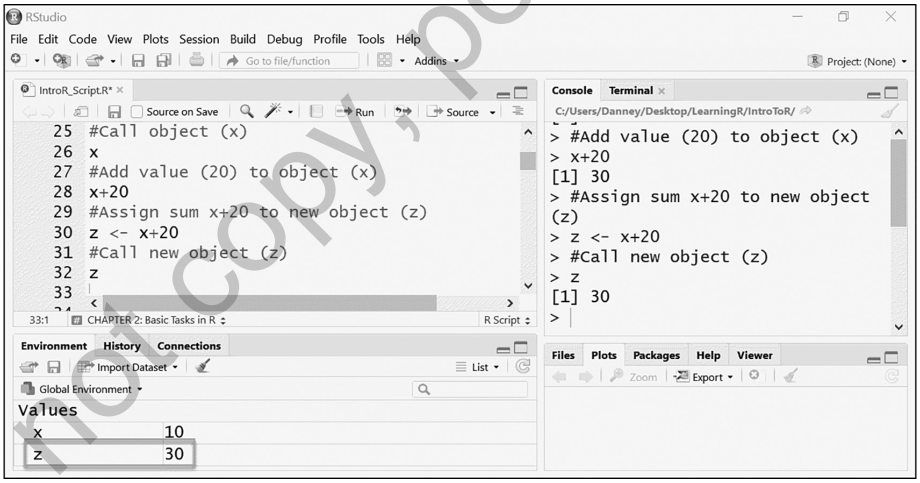
```
#CHAPTER 2: Basic Tasks in R----
#Object-oriented coding
#Assign value (10) to object (x)
x <- 10
#Call object (x)
x
#Add value (20) to object (x)
x+20
```

```
> #Assign value (10) to object (x
)
> x <- 10
> #Call object (x)
> x
[1] 10
> #Add value (20) to object (x)
> x+20
[1] 30
>
```

```
Values
 x              10
```

We could assign this new value (i.e., *x* + 20) to a new object (*z*). We then need to call object *z* to see the sum. Alternatively, you can see the value in the *Environment* pane at the bottom right of the screen. Remember: The <- symbol is made using *alt* and - together.

```
z <- x+20
z
```



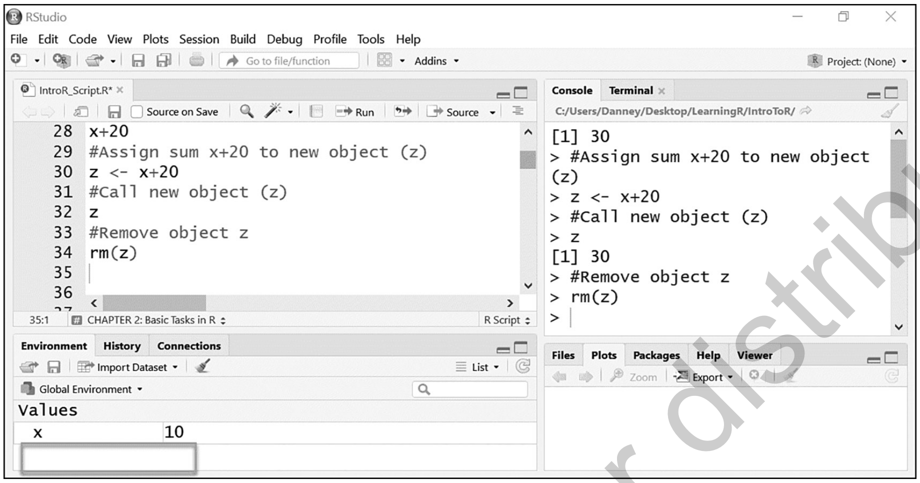
```
#Call object (x)
x
#Add value (20) to object (x)
x+20
#Assign sum x+20 to new object (z)
z <- x+20
#Call new object (z)
z
```

```
> #Add value (20) to object (x)
> x+20
[1] 30
> #Assign sum x+20 to new object
(z)
> z <- x+20
> #Call new object (z)
> z
[1] 30
>
```

```
Values
 x              10
 z              30
```

If you want to remove an object, you can simply type *rm(*INSERT OBJECT*)*. For example, if we did not need the object *z*, we can remove it from the *Environment*. Once you complete this action, you should see the object *z* is no longer listed in the *Environment* pane: *rm(z)*.

Note: If you want to clear the *Console*, you can hit *ctrl* + *l* at the same time.



```
x+20
#Assign sum x+20 to new object (z)
z <- x+20
#Call new object (z)
z
#Remove object z
rm(z)
```

```
[1] 30
> #Assign sum x+20 to new object
(z)
> z <- x+20
> #Call new object (z)
> z
[1] 30
> #Remove object z
> rm(z)
>
```

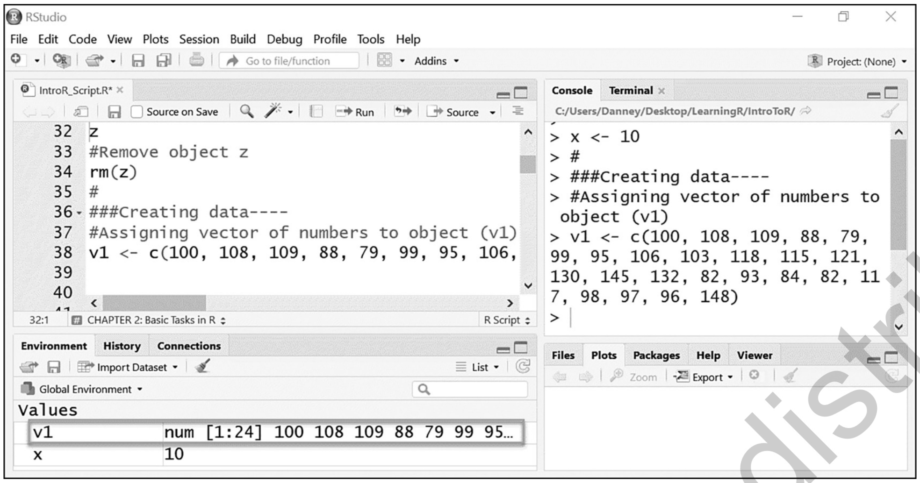
```
Values
 x              10
```

CREATING DATA

Data come from a number of sources (e.g., Microsoft Excel®, IBM SPSS®) in several formats (e.g., CSV, DTA, SAV). You can easily import these files using the *foreign* package, or you can use the *Import Dataset* button in the *Environment* pane, which uses a built-in package. For the moment, we are going to focus on using data when there is not an existing dataset already. As an example, if we had 24 individuals take an intelligence test and wanted to enter their scores, we could directly enter the scores and save them to an object (*v1*). This object then holds the list of numbers (called a vector) for later use.

```
v1 <- c(100, 108, 109, 88, 79, 99, 95, 106, 103, 118, 115,
    121, 130, 145, 132, 82, 93, 84, 82, 117, 98, 97, 96, 148)
v1
```

Note: Once you run the code, it appears in the *Console* pane, but there is no additional output because the numbers were assigned to the object *v1*. We can see the new object in the *Environment* pane along with the numbers assigned to the vector, though. We could call the object (*v1*), which would result in the values appearing in the *Console* pane.



```
z
#Remove object z
rm(z)
#
###Creating data----
#Assigning vector of numbers to object (v1)
v1 <- c(100, 108, 109, 88, 79, 99, 95, 106,
```

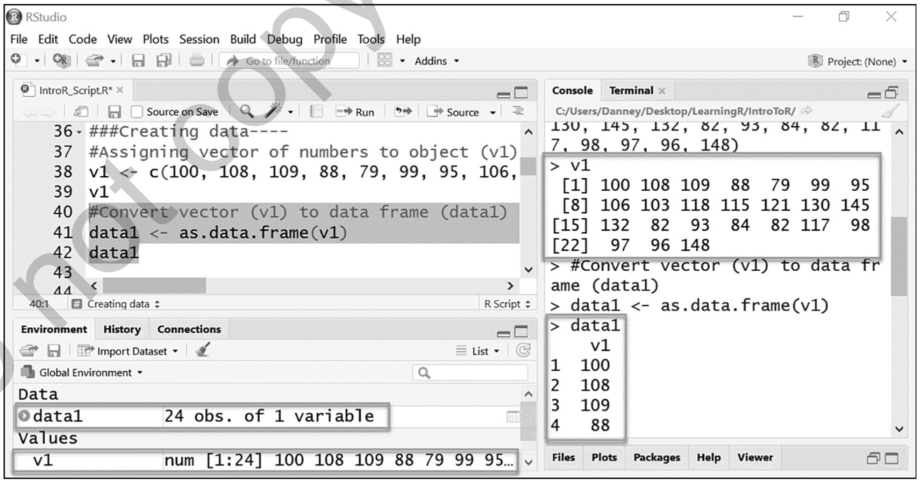
```
> x <- 10
> #
> ###Creating data----
> #Assigning vector of numbers to
 object (v1)
> v1 <- c(100, 108, 109, 88, 79,
99, 95, 106, 103, 118, 115, 121,
130, 145, 132, 82, 93, 84, 82, 11
7, 98, 97, 96, 148)
> |
```

```
Values
 v1            num [1:24] 100 108 109 88 79 99 95…
 x             10
```

Once created, this vector of scores can be converted into a data frame (*data1*) using the function *as.data.frame*. In the *Console* pane, *v1* has a different orientation than *data1*, which coincides with the shift from a vector of scores (*v1*) to a variable or column in a data frame (*data1*). Similarly, in the *Environment* pane, the data frame focuses on the number of observations and variables, and the vector presents individual scores.

```
data1 <- as.data.frame(v1)
data1
```

Note: You can highlight and run multiple lines of code to create the object and call the object at the same time.



```
###Creating data----
#Assigning vector of numbers to object (v1)
v1 <- c(100, 108, 109, 88, 79, 99, 95, 106,
v1
#Convert vector (v1) to data frame (data1)
data1 <- as.data.frame(v1)
data1
```

```
130, 145, 132, 82, 93, 84, 82, 11
7, 98, 97, 96, 148)
> v1
 [1] 100 108 109  88  79  99  95
 [8] 106 103 118 115 121 130 145
[15] 132  82  93  84  82 117  98
[22]  97  96 148
> #Convert vector (v1) to data fr
ame (data1)
> data1 <- as.data.frame(v1)
> data1
   v1
1  100
2  108
3  109
4   88
```

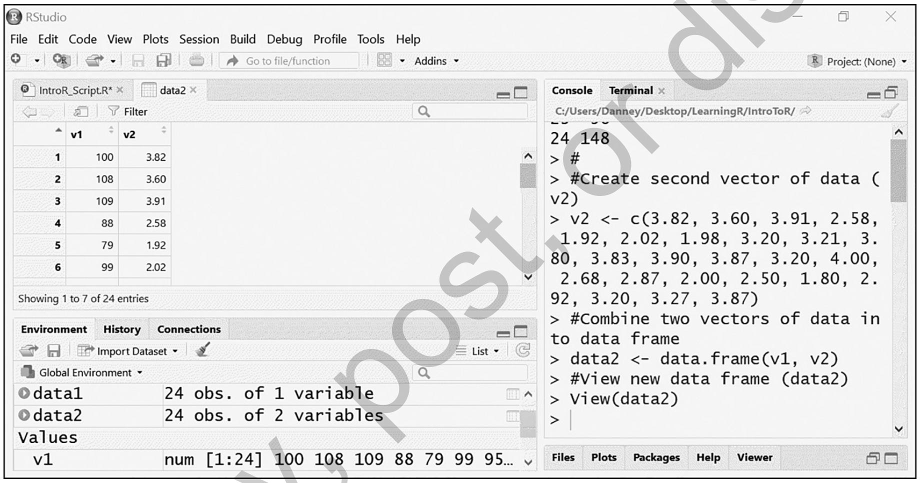
```
Data
 data1         24 obs. of 1 variable
Values
 v1            num [1:24] 100 108 109 88 79 99 95…
```

We rarely work with a single variable. Create another vector (*v2*) with grade point average (GPA) scores for 24 individuals. Then, combine the two vectors into a data frame (*data2*).

```
v2 <- c(3.82, 3.60, 3.91, 2.58, 1.92, 2.02, 1.98, 3.20, 3.21,
    3.80, 3.83, 3.90, 3.87, 3.20, 4.00, 2.68, 2.87, 2.00,
    2.50, 1.80, 2.92, 3.20, 3.27, 3.87)
data2 <- data.frame(v1, v2)
```

You can `View` the new data frame in RStudio using the `View` function. This function opens a new tab in the top left `Source` pane with `data2` visible.

```
View(data2)
```



	v1	v2
1	100	3.82
2	108	3.60
3	109	3.91
4	88	2.58
5	79	1.92
6	99	2.02

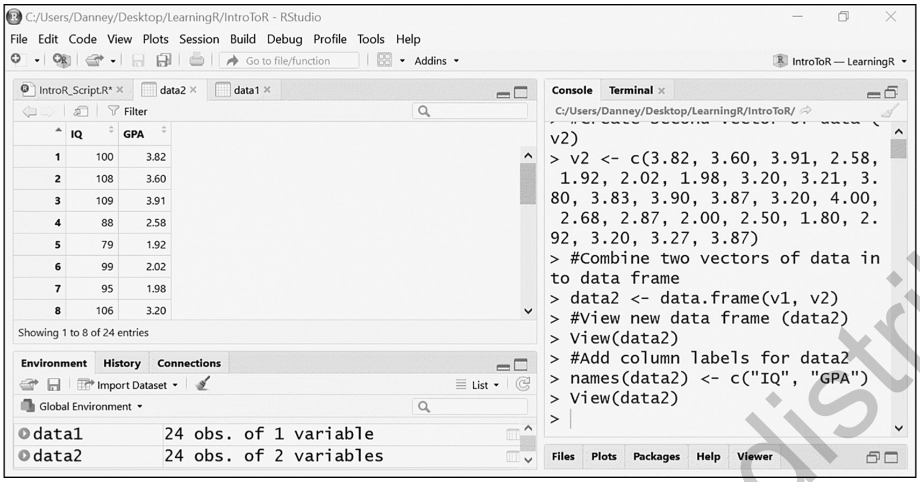
Showing 1 to 7 of 24 entries

```
data1          24 obs. of 1 variable
data2          24 obs. of 2 variables
Values
  v1           num [1:24] 100 108 109 88 79 99 95…
```

```
24 148
> #
> #Create second vector of data (
v2)
> v2 <- c(3.82, 3.60, 3.91, 2.58,
 1.92, 2.02, 1.98, 3.20, 3.21, 3.
80, 3.83, 3.90, 3.87, 3.20, 4.00,
 2.68, 2.87, 2.00, 2.50, 1.80, 2.
92, 3.20, 3.27, 3.87)
> #Combine two vectors of data in
to data frame
> data2 <- data.frame(v1, v2)
> #View new data frame (data2)
> View(data2)
> 
```

These variables are not labeled (i.e., named), which could cause problems if we look at these data a few weeks from now or send the data to someone else. We can use the script to add column labels (IQ, GPA) with the `names` function. These labels are informative and short, which will be useful when we look at the file at a later time or call the variable later because the label is descriptive and quick to type. Alternatively, the labels can be added as you create the new data frame with the `data.frame` function.

```
names(data2) <- c("IQ", "GPA")
data2_Alt <- data.frame(IQ=v1, GPA=v2)
```


C:/Users/Danney/Desktop/LearningR/IntroToR - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
IntroToR — LearningR
IntroR_Script.R*
data2
data1
Filter
Showing 1 to 8 of 24 entries
Environment History Connections
Import Dataset
List
Global Environment
Console Terminal
C:/Users/Danney/Desktop/LearningR/IntroToR/
Files Plots Packages Help Viewer

	IQ	GPA
1	100	3.82
2	108	3.60
3	109	3.91
4	88	2.58
5	79	1.92
6	99	2.02
7	95	1.98
8	106	3.20

```
data1          24 obs. of 1 variable
data2          24 obs. of 2 variables
```

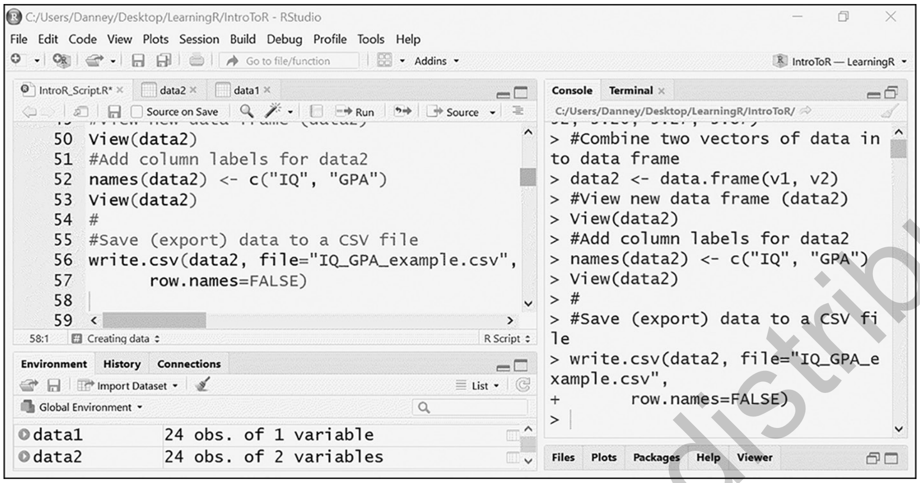
```
v2)
> v2 <- c(3.82, 3.60, 3.91, 2.58,
 1.92, 2.02, 1.98, 3.20, 3.21, 3.
80, 3.83, 3.90, 3.87, 3.20, 4.00,
 2.68, 2.87, 2.00, 2.50, 1.80, 2.
92, 3.20, 3.27, 3.87)
> #Combine two vectors of data in
to data frame
> data2 <- data.frame(v1, v2)
> #View new data frame (data2)
> View(data2)
> #Add column labels for data2
> names(data2) <- c("IQ", "GPA")
> View(data2)
>
```

EXPORTING DATA

You probably want to save the data once you create the data frame and label the variables. You can export the data in several formats. One possibility is to write the data frame to a comma-separated values (CSV) file using the *write.csv* function. Once you run this code, you should see a new file named *IQ_GPA_example.csv* in your project folder.

```
write.csv(data2, file="IQ_GPA_example.csv", row.names=FALSE)
```

Note: If you did not create a project, you will need to type out the file location: *write.csv(data2, file="C:/Users/MrAwesome/Desktop/StatsPurgatory/RforNewbies/IQ_GPA_example.csv")*. If you created a project, R assumes you want to save the file in your project folder. You can easily see from this example how starting a project folder saves time in the long run, and if you want to save the file somewhere else, manually writing the new destination (i.e., write out the full location) will override the assumption you want to save to the project folder.


C:/Users/Danney/Desktop/LearningR/IntroToR - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
IntroR_Script.R* data2 data1
Console Terminal
Environment History Connections
Import Dataset
Global Environment
Files Plots Packages Help Viewer

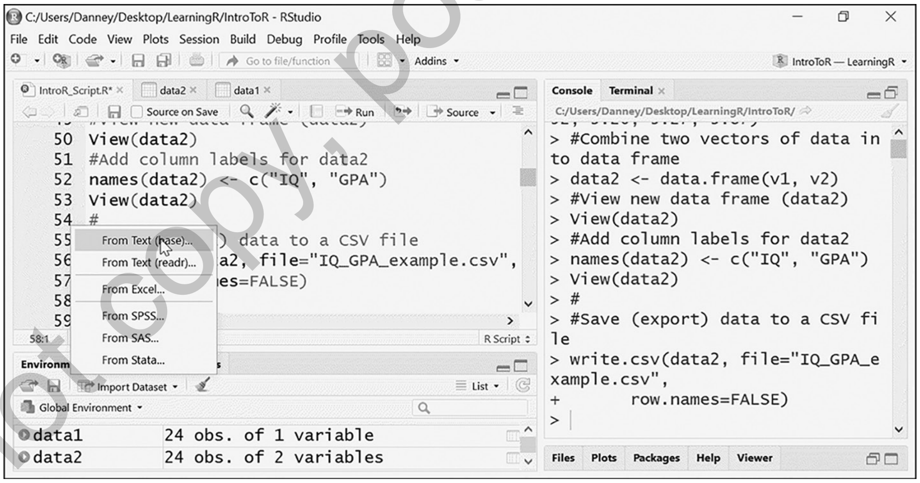
```
View(data2)
#Add column labels for data2
names(data2) <- c("IQ", "GPA")
View(data2)
#
#Save (export) data to a CSV file
write.csv(data2, file="IQ_GPA_example.csv",
      row.names=FALSE)
```

```
> #Combine two vectors of data in
to data frame
> data2 <- data.frame(v1, v2)
> #View new data frame (data2)
> View(data2)
> #Add column labels for data2
> names(data2) <- c("IQ", "GPA")
> View(data2)
> #
> #Save (export) data to a CSV fi
le
> write.csv(data2, file="IQ_GPA_e
xample.csv",
+       row.names=FALSE)
>
```

```
data1       24 obs. of 1 variable
data2       24 obs. of 2 variables
```

IMPORTING DATA

If we want to import the data at a later point, we simply go to the *Environment* pane, click *Import Dataset,* select *From Text (base)…*, and select *Browse….*

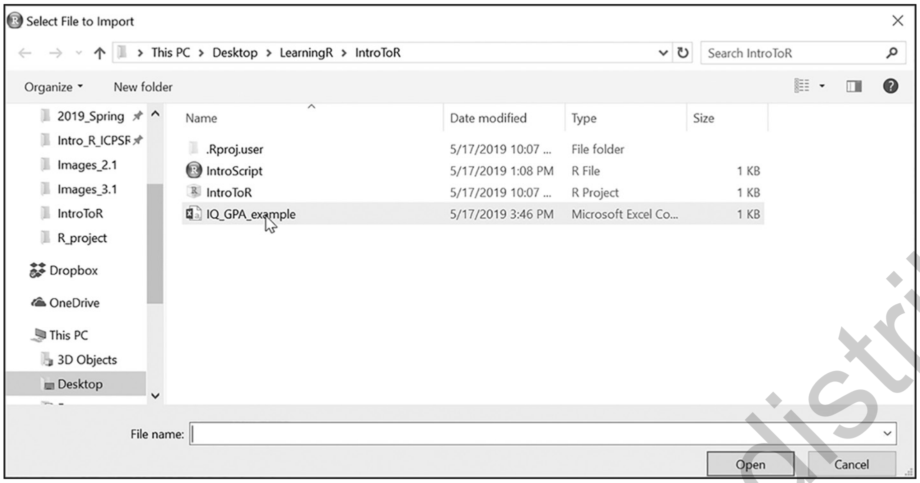


```
View(data2)
#Add column labels for data2
names(data2) <- c("IQ", "GPA")
View(data2)
#
```

```
> #Combine two vectors of data in
to data frame
> data2 <- data.frame(v1, v2)
> #View new data frame (data2)
> View(data2)
> #Add column labels for data2
> names(data2) <- c("IQ", "GPA")
> View(data2)
> #
> #Save (export) data to a CSV fi
le
> write.csv(data2, file="IQ_GPA_e
xample.csv",
+       row.names=FALSE)
>
```

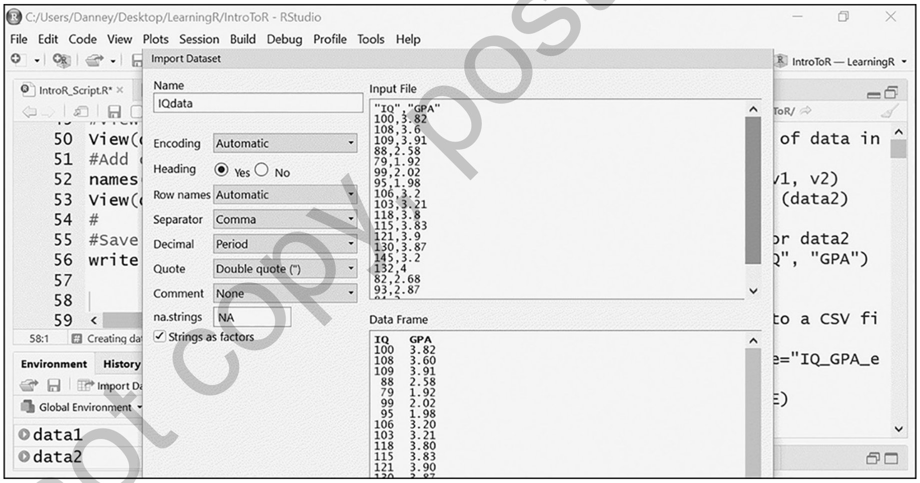
```
data1       24 obs. of 1 variable
data2       24 obs. of 2 variables
```

Because we set a project file, the browser opens in the current project file, although we could easily locate the file using the browser. Select the desired file (i.e., *IQ_GPA_example.csv*) and click *Open*.



Name	Date modified	Type	Size
.Rproj.user	5/17/2019 10:07 ...	File folder	
IntroScript	5/17/2019 1:08 PM	R File	1 KB
IntroToR	5/17/2019 10:07 ...	R Project	1 KB
IQ_GPA_example	5/17/2019 3:46 PM	Microsoft Excel Co...	1 KB

The data can be imported using this *`Import Dataset`* pane. Check the settings. In this case, the defaults should work. I changed the *`Name`* to *`IQdata;`* a shorter name makes it faster to reference the data frame later in your script. When you are ready, select *`Import`*.


C:/Users/Danney/Desktop/LearningR/IntroToR - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Import Dataset
IntroToR — LearningR
IntroR_Script.R*
Environment History
Global Environment
data1
data2

Name: IQdata
Encoding: Automatic
Heading: (● Yes) ○ No
Row names: Automatic
Separator: Comma
Decimal: Period
Quote: Double quote (")
Comment: None
na.strings: NA
✓ Strings as factors

Input File

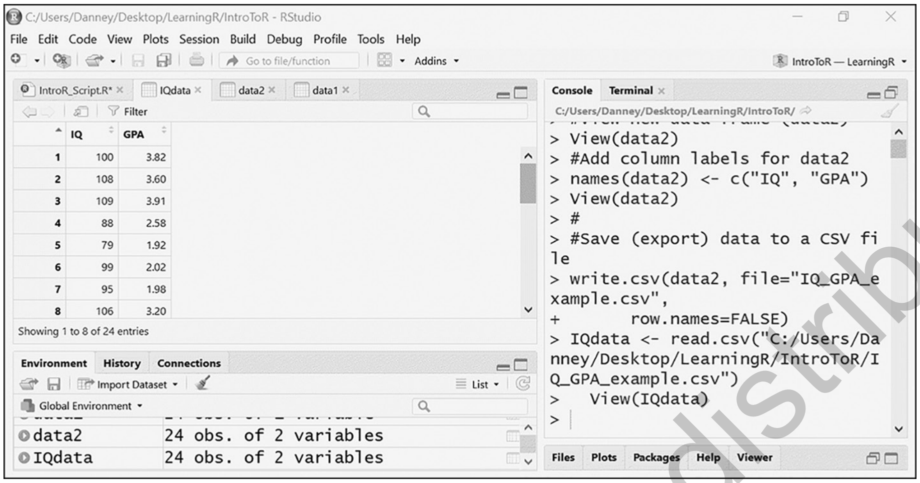
```
"IQ","GPA"
100,3.82
108,3.6
109,3.91
88,2.58
79,1.92
99,2.02
95,1.98
106,3.2
103,3.21
118,3.8
115,3.83
121,3.9
130,3.87
145,3.2
132,4
82,2.68
93,2.87
```

Data Frame

```
IQ   GPA
100  3.82
108  3.60
109  3.91
 88  2.58
 79  1.92
 99  2.02
 95  1.98
106  3.20
103  3.21
118  3.80
115  3.83
121  3.90
```

RStudio writes a code using the import settings and sends it to the *`Console`* pane. This code creates the object *`IQdata`* and opens a viewer, seen on the left. We could manually write the code with the *`read.csv`* function and make it shorter.

```
IQdata <- read.csv("IQ_GPA_example.csv")
```



	IQ	GPA
1	100	3.82
2	108	3.60
3	109	3.91
4	88	2.58
5	79	1.92
6	99	2.02
7	95	1.98
8	106	3.20

Showing 1 to 8 of 24 entries

```
data2     24 obs. of 2 variables
IQdata    24 obs. of 2 variables
```

```
> View(data2)
> #Add column labels for data2
> names(data2) <- c("IQ", "GPA")
> View(data2)
> #
> #Save (export) data to a CSV fi
le
> write.csv(data2, file="IQ_GPA_e
xample.csv",
+         row.names=FALSE)
> IQdata <- read.csv("C:/Users/Da
nney/Desktop/LearningR/IntroToR/I
Q_GPA_example.csv")
>   View(IQdata)
> 
```

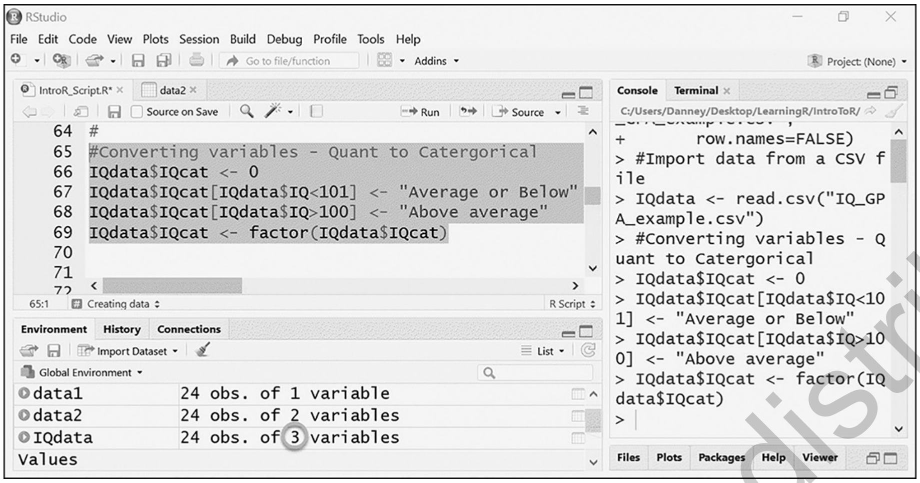
CONVERTING VARIABLES

We can convert the quantitative IQ scores into a categorical variable. One option would be to convert them into a dichotomous variable with two categories: (1) above 100 and (2) equal to or below 100. First, add a new variable (*IQcat*) to the data frame (*IQdata*) and fill the variable with zeros. Adding this variable should result in the *IQdata* object showing 24 observations of three variables in the *Environment* pane.

```
IQdata$IQcat <- 0
```

Score ranges on the *IQ* variable (e.g., < 101) in the *IQdata* data frame can be used to assign category labels (e.g., *"Average or Below"*) to the *IQcat* variable. Once the code for the first category is complete, it can be modified to create the second category (i.e., *"Above average"*). Then, we can specify that the variable is a categorical variable using the *factor* function. This step associates a numerical value with each label. This function also can be used to denote a variable is categorical and not quantitative, which will allow functions like *summary* to present the data using the appropriate statistics.

```
IQdata$IQcat[IQdata$IQ<101] <- "Average or Below"
IQdata$IQcat[IQdata$IQ>100] <- "Above average"
IQdata$IQcat <- factor(IQdata$IQcat)
```



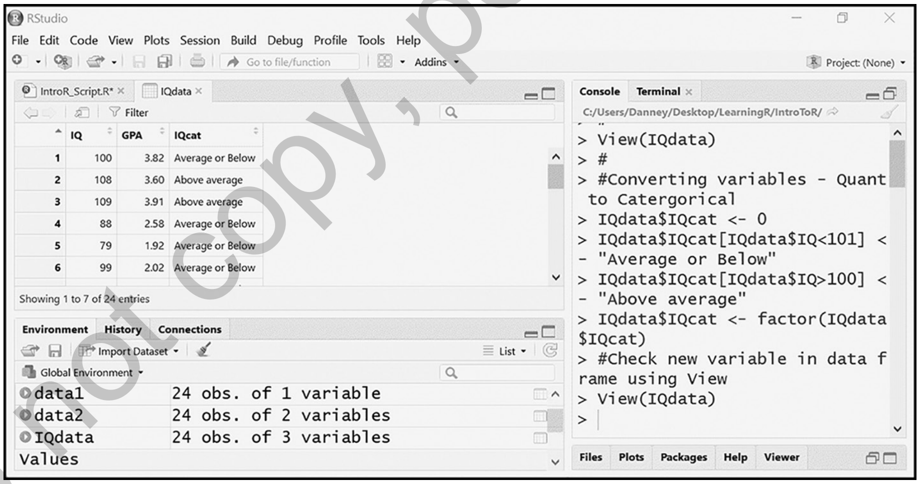
```
#
#Converting variables - Quant to Catergorical
IQdata$IQcat <- 0
IQdata$IQcat[IQdata$IQ<101] <- "Average or Below"
IQdata$IQcat[IQdata$IQ>100] <- "Above average"
IQdata$IQcat <- factor(IQdata$IQcat)
```

```
+        row.names=FALSE)
> #Import data from a CSV f
ile
> IQdata <- read.csv("IQ_GP
A_example.csv")
> #Converting variables - Q
uant to Catergorical
> IQdata$IQcat <- 0
> IQdata$IQcat[IQdata$IQ<10
1] <- "Average or Below"
> IQdata$IQcat[IQdata$IQ>10
0] <- "Above average"
> IQdata$IQcat <- factor(IQ
data$IQcat)
>
```

```
Global Environment
data1     24 obs. of 1 variable
data2     24 obs. of 2 variables
IQdata    24 obs. of 3 variables
Values
```

We can check that the categorical variable was created properly using the *View* function. Once you are sure the data are correct, save the finished data frame to a file (*IQ_GPA_example2.csv*) for future use with the *write.csv* function.

```
View(IQdata)
write.csv(IQdata, file="IQ_GPA_example2.csv", row.names=FALSE)
```



	IQ	GPA	IQcat
1	100	3.82	Average or Below
2	108	3.60	Above average
3	109	3.91	Above average
4	88	2.58	Average or Below
5	79	1.92	Average or Below
6	99	2.02	Average or Below

Showing 1 to 7 of 24 entries

```
> View(IQdata)
> #
> #Converting variables - Quant
 to Catergorical
> IQdata$IQcat <- 0
> IQdata$IQcat[IQdata$IQ<101] <
- "Average or Below"
> IQdata$IQcat[IQdata$IQ>100] <
- "Above average"
> IQdata$IQcat <- factor(IQdata
$IQcat)
> #Check new variable in data f
rame using View
> View(IQdata)
>
```

```
Global Environment
data1     24 obs. of 1 variable
data2     24 obs. of 2 variables
IQdata    24 obs. of 3 variables
Values
```

Occasionally, variables are incorrectly identified as quantitative (i.e., integer or numeric) when they are categorical (i.e., factor) or vice versa. If you want to check how a variable is classified, you can use the function *class*( ) and add the variable of interest in parentheses. If this check shows the variable is classified incorrectly, you can specify the level of measurement. For example, you can specify a variable as categorical using the *factor* function, and you can coerce a variable to be quantitative using the *as.numeric* and *as.character* functions.

```
class(IQdata$IQ)
as.numeric(as.character(INSERT FACTOR VARIABLE))
```

Chapter 2 Summary of Key Functions (AKA: Function Cheat Sheet)

Function Call	Package	Description
ls	Included in base	Lists objects in Environment
rm	Included in base	Removes an object from Environment
c	Included in base	Concatenates (i.e., combines) elements
as.data.frame	Included in base	Coerces object to data frame
data.frame	Included in base	Creates a new data frame
View	Included in utils	Opens tab with object (e.g., data frame)
names	Included in base	Adds names to data frame or returns names
write.csv	Included in utils	Saves CSV file with object in project folder
read.csv	Included in utils	Assigns dataset from CSV file to data frame
factor	Included in base	Coerces object to factor class
class	Included in base	Returns class for object
as.numeric	Included in base	Coerces object to numeric class
as.character	Included in base	Converts object to a character vector