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 list 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.
 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.

    RScript

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

    x+20
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.

\[
z <- x+20
\]

\( z \)

If you want to remove an object, you can simply type \texttt{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: \texttt{rm(z)}.
Note: If you want to clear the console, you can hit `ctrl + l` at the same time.

## 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.

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

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.
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.

```r
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.
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 \texttt{View} the new data frame in RStudio using the \texttt{View} function. This function opens a new tab in the top left \texttt{Source} pane with \texttt{data2} visible.

\texttt{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 \texttt{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 \texttt{data.frame} function.

\[
\text{names(data2)} < - c(“IQ”, “GPA”) \\
data2\_Alt < - data.frame(IQ=v1, GPA=v2)
\]
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.

```r
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:

```r
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.
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….

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.
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.

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.

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

\[
\text{IQdata}\$\text{IQcat} \leftarrow 0
\]

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.

\[
\begin{align*}
\text{IQdata}\$\text{IQcat}[\text{IQdata}\$\text{IQ}<101] & \leftarrow \text{“Average or Below”} \\
\text{IQdata}\$\text{IQcat}[\text{IQdata}\$\text{IQ}>100] & \leftarrow \text{“Above average”} \\
\text{IQdata}\$\text{IQcat} & \leftarrow \text{factor(IQdata}\$\text{IQcat)}
\end{align*}
\]
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.

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

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.

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

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

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