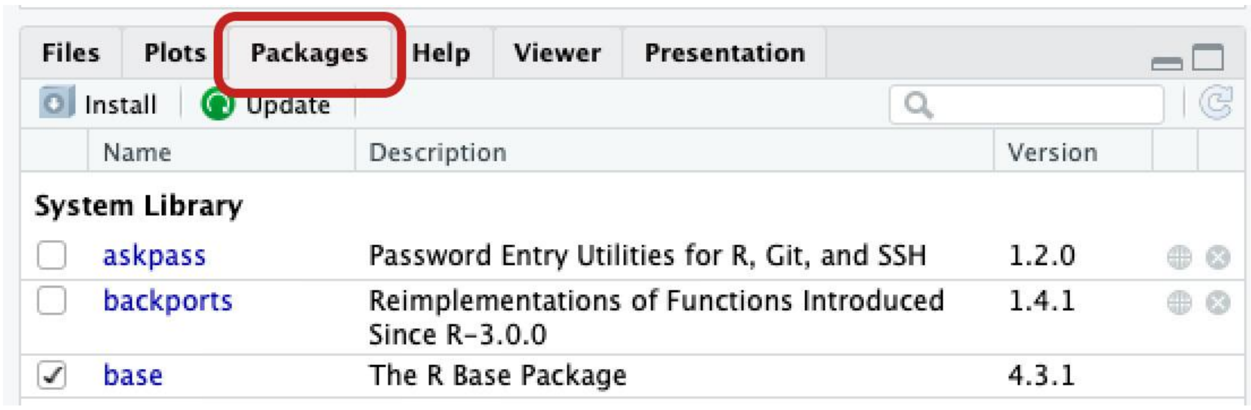
Earlier in this course and program, you learned that a logical **operator** allows you to combine logical statements and return a logical value, **`TRUE`** or **`FALSE`**. In this reading, you'll explore how the main types of logical operators can be used with datasets. In an optional section, you can also explore how logical operators create conditional statements that allow you to work with datasets in R.

In this scenario, you're a researcher working with the `airquality` dataset available in R. It contains data about daily air quality measurements in New York from May to September of 1973.

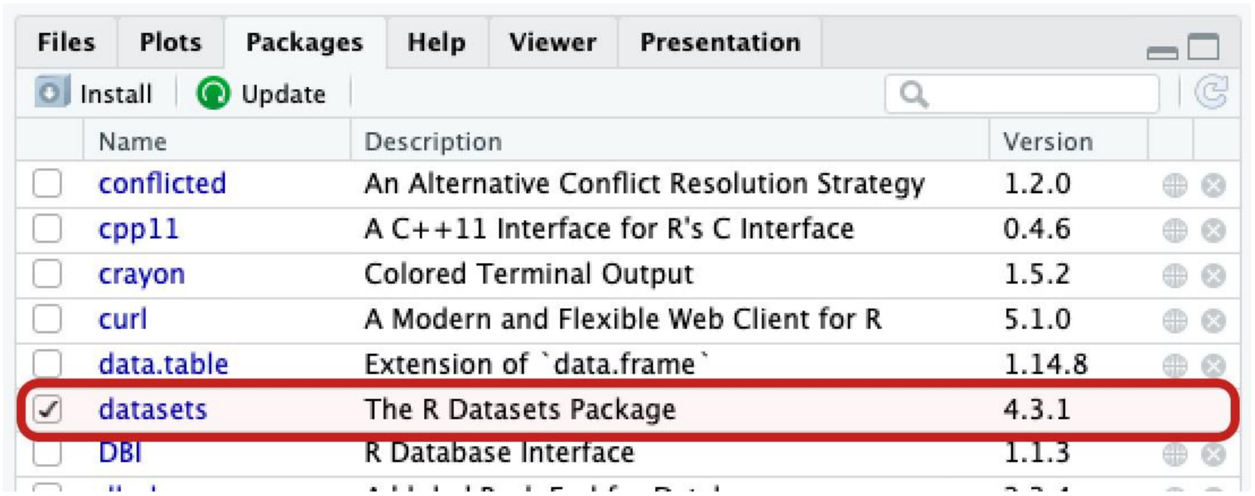
To access the preloaded `airquality` dataset in RStudio:

1. Log in to your [Posit Cloud]() account and open an existing RStudio project or create a new project.
2. Select the **Packages** tab in the Output pane.



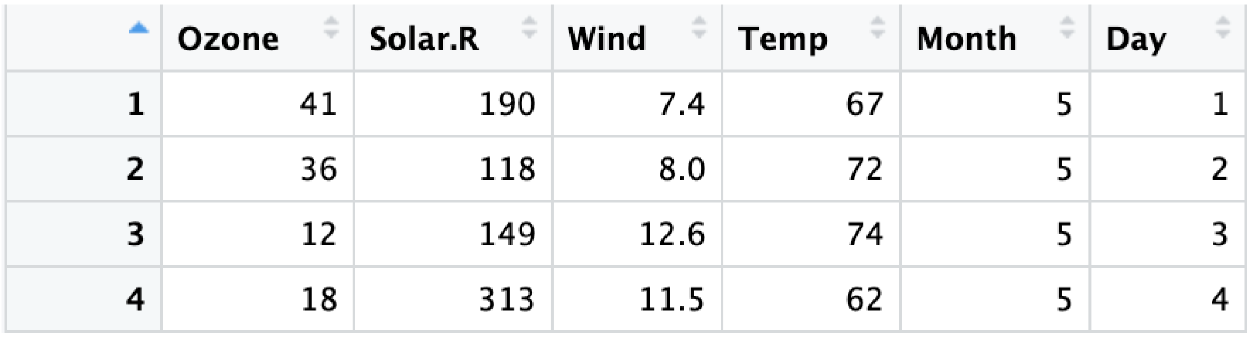
	Name	Description	Version
	System Library		
☐	askpass	Password Entry Utilities for R, Git, and SSH	1.2.0
☐	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.4.1
✓	base	The R Base Package	4.3.1

3. Select the **datasets** box in the System Library list of packages.


	Name	Description	Version
☐	conflicted	An Alternative Conflict Resolution Strategy	1.2.0
☐	cpp11	A C++11 Interface for R's C Interface	0.4.6
☐	crayon	Colored Terminal Output	1.5.2
☐	curl	A Modern and Flexible Web Client for R	5.1.0
☐	data.table	Extension of `data.frame`	1.14.8
✓	datasets	The R Datasets Package	4.3.1
☐	DBI	R Database Interface	1.1.3

4. Run the following command **`data("airquality")`** in the console to load the **`airquality`** dataset.
5. Then, run the command **`View(airquality)`** to open the dataset in the Script Editor.

This dataset is a data frame with six columns: **`Ozone`** (the ozone measurement), **`Solar.R`** (the solar measurement), **`Wind`** (the wind measurement), **`Temp`** (the temperature in Fahrenheit), and the **`Month`** and **`Day`** these measurements were taken. Each row represents a specific month and day combination.


	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8.0	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4

Next, explore how the **`AND`**, **`OR`**, and **`NOT`** operators might be helpful in this situation.

AND example

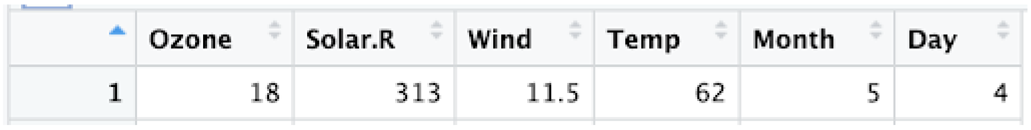
You want to find observations (rows) in which conditions are both extremely sunny and windy. You define this as observations that have a Solar measurement of over 150 and a Wind measurement of over 10.

In R, you can express this logical statement as:

```
airquality[, "Solar.R"] > 150 & airquality[, "Wind"] > 10
```

This code specifies that R should return a value of **TRUE** for rows in which the **airquality** dataset's **Solar.R** value is greater than 150 and its **Wind** value is greater than 10, and a value of **FALSE** otherwise.

Only the rows where both of these conditions are true fulfill the criteria, such as the following row:



	Ozone	Solar.R	Wind	Temp	Month	Day
1	18	313	11.5	62	5	4

OR example

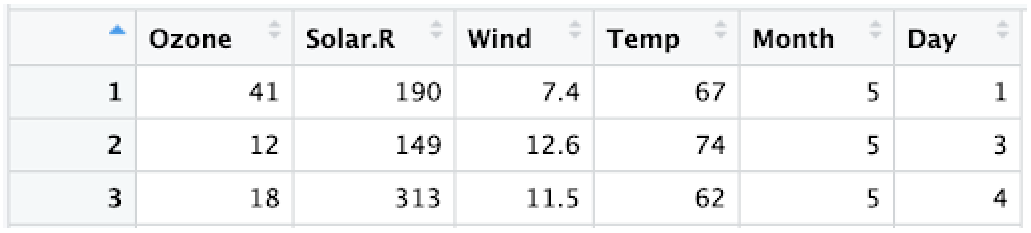
Next, you want to specify rows where it's extremely sunny or it's extremely windy, which you define as having a **Solar** measurement of over 150 or a **Wind** measurement of over 10.

In R, you can express this logical statement as :

```
airquality[, "Solar.R"] > 150 | airquality[, "Wind"] > 10
```

This code specifies that R should return a value of **TRUE** when either the **airquality** dataset's **Solar.R** value is greater than 150 or its `Wind` value is greater than 10. Otherwise, R will return a value of **FALSE**.

Rows where either of these conditions are true fulfill the criteria, such as the following rows:



	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	12	149	12.6	74	5	3
3	18	313	11.5	62	5	4

NOT example

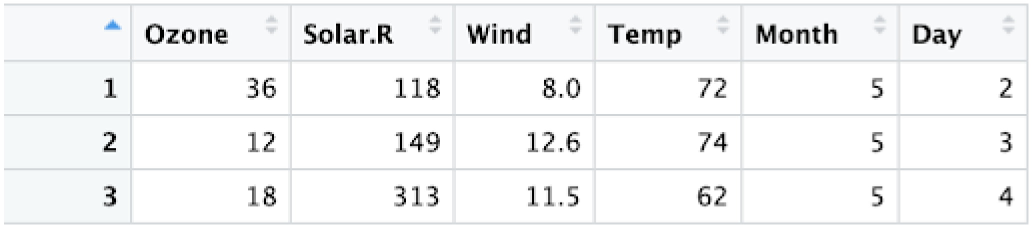
If you just want to focus on the weather measurements for days that aren't the first day of the month, use the NOT condition.

In R, this logical statement is expressed:

```
airquality[, "Day"] != 1
```

This code specifies that R should return a value of **`TRUE`** when the **`airquality`** dataset's **`Day`** value is not 1 and a value of **`FALSE`** when the **`Day`** value is 1.

Rows where this condition is true fulfill the criteria, such as the following rows:



	Ozone	Solar.R	Wind	Temp	Month	Day
1	36	118	8.0	72	5	2
2	12	149	12.6	74	5	3
3	18	313	11.5	62	5	4

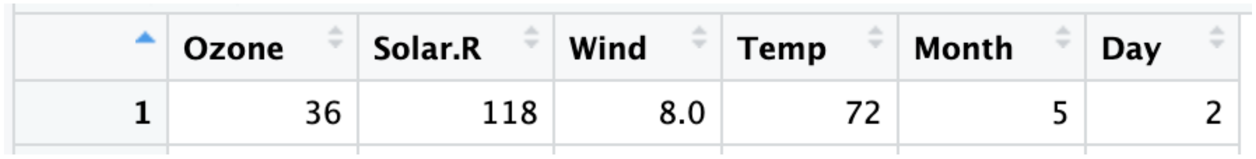
Finally, you want to focus on scenarios that are neither extremely sunny nor extremely windy, based on your previous definitions of extremely sunny and extremely windy. In other words, the following statement should not be true: either a **`Solar`** measurement greater than 150 or a **`Wind`** measurement greater than 10.

Notice that this statement is the opposite of the OR statement used above. To express this statement in R, you can put an exclamation point (**`!`**) in front of the previous OR statement:

```
!(airquality[, "Solar.R"] > 150 | airquality[, "Wind"] > 10)
```

R will apply the NOT operator to everything within the parentheses.

Rows where this condition is true fulfill the criteria, such as the following row:



	Ozone	Solar.R	Wind	Temp	Month	Day
1	36	118	8.0	72	5	2

Optional: Conditional statements

A **conditional statement** is a declaration that, if a certain condition holds, then a certain event must take place. For example, "If the temperature is above freezing, then I will go outside for a walk." If the first condition is true (the temperature is above freezing), then the second condition will occur (I will go for a walk). Conditional statements in R code have a similar logic.

In this section, you'll explore how to create conditional statements in R using three related statements:

- **`if()`**
- **`else()`**
- **`else if()`**

`if` statement

The **`if`** statement sets a condition, and if the condition evaluates to TRUE, the R code associated with the if statement is executed.

In R, you place the code for the condition inside the parentheses of the **`if`** statement. The code to be executed if the condition is **`TRUE`** follows in curly braces (**`expr`**). Note that in this case, the second curly brace is placed on its own line of code and identifies the end of the code that you want to execute.

Unset
```
if (condition) {

  expr

}
```

For example, create a variable `x` equal to 4.

`x <- 4`

Next, create a conditional statement: if `x` is greater than 0, then R will print out the string **`"x is a positive number"`**.

Unset
```
if (x > 0) {

  print("x is a positive number")

}
```

As x is equal to 4, the condition is true (because 4 is greater than 0). Therefore, when you run the code, R prints out the string **"x is a positive number".**

But, if you change `x` to a negative number, such as -4, then the condition will be **FALSE** because -4 is not greater than 0. If you run the code, R will not execute the print statement. Instead, a blank line will appear as the result.

Here's an example using the **airquality** data:

```
if(airquality[1, "Temp"] < 80) {print("It's not a hot day.")}
```

Running this code will check the value in the first row of the **airquality** dataset's **Temp** column. If that value is less than 80 degrees, it will print **"It's not a hot day."** Otherwise, it won't print anything.

`else` statement

The **else** statement is used in combination with an **if** statement. This is how the code is structured in R:

Unset
```
if (condition) {

  expr1

} else {

  expr2

}
```

The code associated with the **else** statement is executed only when the condition in the **if** statement is not **TRUE**. In other words, if the condition is **TRUE**, then R will execute the code in the **if** statement (**expr1**); if the condition is not **TRUE**, then R will execute the code in the else statement (**expr2**).

Explore the following example. First, create a variable x equal to 7.

```
x <- 7
```

Next, set up the following conditions:

- If x is greater than 0, R will print **"x is a positive number".**
- If x is less than or equal to 0, R will print **"x is either a negative number or zero".**

In the code, the first condition (x > 0) is part of the **`if`** statement. The second condition of x less than or equal to 0 is implied in the **`else`** statement. If x > 0, then R will print **"x is a positive number"**. Otherwise, R will print **"x is either a negative number or zero".**

Unset

```
x <- 7

if (x > 0) {

  print ("x is a positive number")

} else {

  print ("x is either a negative number or zero")

}
```

As 7 is greater than 0, the condition of the if statement is true. So, when you run the code, R prints out **"x is a positive number".**

But, if you set x equal to -7, the condition of the if statement is *not* true because -7 is not greater than 0. Therefore, R will execute the code in the else statement. When you run the following code, R prints out **"x is either a negative number or zero".**

Unset

```
x <- -7

if (x > 0) {

  print("x is a positive number")

} else {
  print ("x is either a negative number or zero")

}
```

Here's an example using the **airquality** data:

Unset
```
if (airquality$Temp[1] < 80) {
  print("It's not a hot day!")
} else {
  print("It's a hot day.")
}
```

Similar to the first example, the **if** statement checks the **Temp** value for the first row in **airquality**. This time, if the **if** statement's condition is **FALSE** because **Temp** is greater than 80, the **else** statement is printed.

else if statement

In some cases, you might want to customize your conditional statement even further by adding the **else if** statement. The **else if** statement comes in between the **if** statement and the **else** statement. This is the code structure:

Unset
```
if (condition1) {

  expr1

} else if (condition2) {

  expr2

} else {
  expr3

}
```

If the **`if`** condition (**`condition1`**) is met, then R executes the code in the first expression (**`expr1`**). If the **`if`** condition is not met and the **`else if`** condition (**`condition2`**) is met, then R executes the code in the second expression (**`expr2`**). If neither of the two conditions are met, R executes the code in the third expression (**`expr3`**).

In the previous example, using only the **`if`** and **`else`** statements, R can only print "**`x is either a negative number or zero`**" if x equals 0 or x is less than zero. Imagine you want R to print the string "**`x is zero`**" if x equals 0. You need to add another condition using the **`else if`** statement.

Try an example. First, create a variable `x` equal to negative 1. Run the code to save the variable to memory.

`x <- -1`

Now, set up the following conditions:

- If x is less than 0, print "**`x is a negative number`**"
- If x equals 0, print "**`x is zero`**"
- Otherwise, print "**`x is a positive number`**"

In the code, the first condition will be part of the if statement, the second condition will be part of the else if statement, and the third condition will be part of the else statement. If x is less than 0, then R will print "**`x is a negative number`**". If `x` is equal to 0, then R will print "**`x is zero`**". Otherwise, R will print "**`x is a positive number`**".

Unset

```
x <- -1

if (x < 0) {

  print("x is a negative number")
} else if (x == 0) {

  print("x is zero")

} else {

  print("x is a positive number")

}
```

When you run the code, the condition for the if statement evaluates to **`TRUE`** because -1 is less than 0. So, R prints **`"x is a negative number"`**.

```
[1] "x is a negative number"
```

If you make `x` equal 0, R will first check the if condition **`(x < 0)`** and determine that it is **`FALSE`**. Then, R will evaluate the else if condition. This condition, **`x==0,`** is **`TRUE`**. So, in this case, R prints **`"x is zero"`**.

If you make x equal to 1, both the if condition and the else if condition evaluate to **`FALSE`**. So, R will execute the else statement and print **`"x is a positive number"`**.

As soon as R discovers a condition that evaluates to **`TRUE`**, R executes the corresponding code and ignores the rest.

Here's an example using the **`airquality`** data:

Unset

```
ozone_level <- airquality[1,"Ozone"]

if(is.na(ozone_level)){

  print("Ozone reading is missing for the first day.")

} else if(ozone_level < 30){

  print("Low ozone level.")
} else if(ozone_level < 100){

  print("Moderate ozone level.")

} else{

  print("High ozone level.")

}
```

In this example, the code first assigns the ozone measurement of the first day in the **`airquality`** dataset to a variable **`ozone_level`**.

It then checks if this value is missing with the **`is.na()`** function. If the value is missing, R prints that the ozone reading is missing.

If the **`ozone_level`** value is not missing, R moves to the next condition, checking if it's less than 30 to determine if the ozone level is low. If it's not less than 30 but still less than 100, it prints that it's a moderate ozone level. If none of the previous conditions are met, meaning the ozone level is 100 or more, it prints that it's a high ozone level.

This sequence of **`if`**, **`else if`**, and **`else`** statements allows for multiple, mutually exclusive conditions to be checked in order.

Key takeaways

In R, the logical operators for AND (**`&`**), OR (**`|`**), and NOT(**`!`**) can be used to check a condition and return a logical data type. In R, logical data is presented as **`T`** or **`TRUE`** when a condition is met, and **`F`** or **`FALSE`** when it is not. Additionally, you can use **`if`**, **`else`**, and **`else if`** statements to complete a task depending on whether one or more conditions are met. Using logical operators and conditional statements with your data can be a powerful analytical tool when you need to find observations that match specific criteria.

Resources for more information

To learn more about logical operators and conditional statements, check out DataCamp's tutorial Conditionals and Control Flow in R. DataCamp is a popular resource for people learning about computer programming. The tutorial is filled with useful examples of coding applications for logical operators and conditional statements (and relational operators), and offers a helpful overview of each topic and the connections between them.

You can save this reading for future reference. Feel free to download a .pdf version of this reading below:

Download .pdf file
