VIDEO: “LOGICAL ARRAYS AND INDEXING” (7:52)

(0:07)
This video introduces logical arrays and vector indexing. We will show how to calculate logical arrays to count the number of items in the subgroup. We will also use them to extract rows and columns for subgroups and to make comparisons using relational operators. Finally, we will ask question involving strings using the string compare function.

(0:32)
We will begin by describing logical arrays. A logical array is just an array that contains 1’s and 0’s- 1 means yes or true and 2 means no or false. Logical arrays are used to express answers to questions about other arrays. The logical array should be the same size of the array the question pertains to. We use logical array to count entries to see if the question is true or to extract rows and columns to the arrays corresponding to a true answer. Let’s look at an example – suppose we have done the study of several sections of a course and we want to know how many questions are in section 3. We have a vector called section which contains the same number as the corresponding students. Let’s think about how we will do this by hand. By hand we will go entry by entry and mark those corresponding to section 3. Here entry 1, 3, 4, and 5 are all students in section 3. Once we have gone through the array, we tally our marks and count the students in section 3. Now suppose we have a vector containing those marks. There is a 1 corresponding to the marked entries. We will call this vector sec3. Once we have calculated sec3 we simply apply a sum to find the total number of students. The question however is how do we calculate sect3? The answer is we use the relational operator “==” which allows us to test each entry in an array. Our questions “how many students in section 3?” becomes “==3” this is how we produce the logical vector sect3. The == goes element by element through section and compares each element the value 3. The parentheses on the expression is not needed we just put them in for liability.

(2:35)
Let’s look at another example. Suppose we have vector age containing the age of the same students in our study. If we wanted to calculate the same students in section 3, we would have to go entry and pick out students from the right section. We call from the previous example that we have a logical vector sec3 marking the students from section 3. We will have to put the third, fourth, and fifth items of age in order to calculate the average of section 3. We can use vector indexing to do this. Instead of putting different values for the row numbers we use sec3 as the index. MATLAB pulls out the entries corresponding to the ones in sec3. Writing age(sect3) will pull out entries 1,3,4 and 5 of age. Let’s assign it the variable “sect3ages.” Once we have it we simply take the mean to calculate the average age of the students in section 3. The bottom line is that we can use a logical vector to extract the rows and columns in section 3. The only catch is that the vector must have the same number of elements as the corresponding number of rows or columns. In this example age had 6 rows and we wanted to extract row a locgical vector had to have exactly 6 elements.

(3:58)

Let's look at an example of extracting rows and columns. Suppose we wanted to know on average, how long it took for students in section 3 to fall asleep. We recorded the data for 3 days in a "toSleepMinutes" array the three rows correspond to the 3 days and the 6 columns to the 6 students. The variable sect3 marks the students in section3. Students in section 3 correspond 1,2,4, and 5. We can extract those columns and compute the averages.
We use sect3 as a column's specifier. We extract those columns in sect3 then we take the average. Again, we can use a logical vector to extract rows and columns from any array provided that the vector is the right size.

(4:56)
We can ask any other types of questions based on this strategy. Suppose we want to know how many students are at least 19 years of age. Let's look at the age array again. If we were going to do this by hand, we would mark the values greater than or equal to 19 entries 1,2,4, and 5. If we had a logical vector with these entries marked, we can simply sum it to find the total. To calculate this logical vector, we use the great than or equal to relational operator. We write age >= 19 and lets' assign it to the variable "atLeast19" then we sum. There are 6 relational operator all together. Equal to, not equal to, greater than or equal to, less than, greater than or equal to or less than or equal to- these can be used to make numerical comparisons.

(5:52)
Unfortunately, the relational operators do not work nonnumeric data. Suppose we want to know how many students are women. We have an array gender designating weather the corresponding student in male or female. The first, fourth, ad fifth student are female. We want create an index vector women with 1's corresponding to females. Gender contains the words "female" and "male" nonnumerical values. Unfortunately, the obvious solution gender equals female does not work because relational operators only work with numeric data. Instead we have to use string compare. String compare (strcmp) is a special MATLAB function for comparing the equality of strings. The statement string compare gender female compares each element of gender with the work female. String compare returns a logical vector with 1's corresponding to the positions where the values match. Let's assign the result to women then we just sum to find the total number of women.

(6:58)
There is a second version of string compare. String compare I which ignores the different between upper and level case and testing for equality. The point to remember here is when you have string values you can use == you have to use string compare.

(7:15)
Let's summarize- we can formulate the answers to yes or no questions as logical arrays. Once we computed a logical array, we can extract the number of rows and columns for a logical array. If the questions involve numerical comparisons, we can use numerical operators. There is six of them. If questions involve strings, we have to use string compare or string compare i. Extracting and analyzing subgroups is often an essential step to understanding the data.