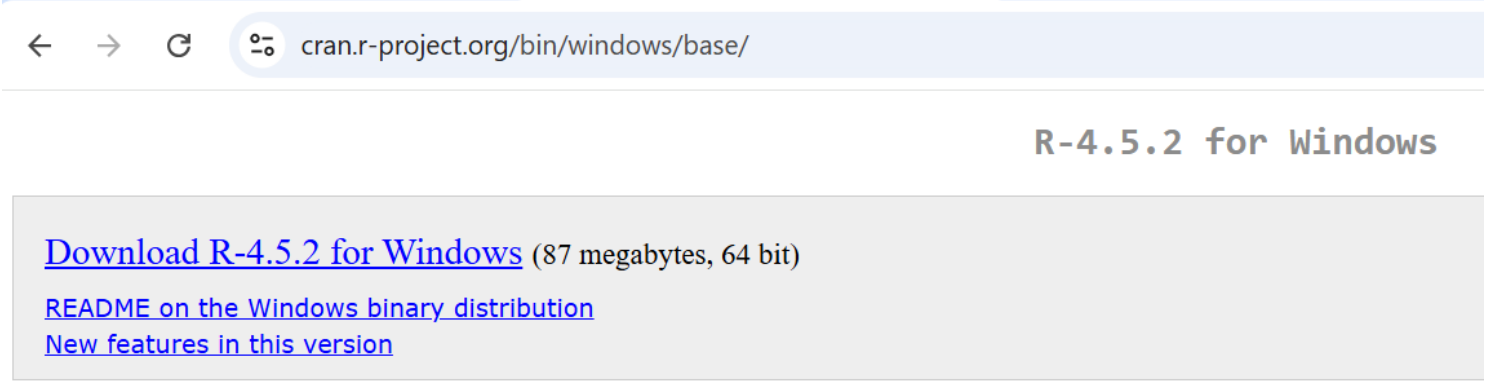
In this lecture, we will learn how to:

- Use R to analyze categorical data
- Interpret the results and effectively communicate a story from the data

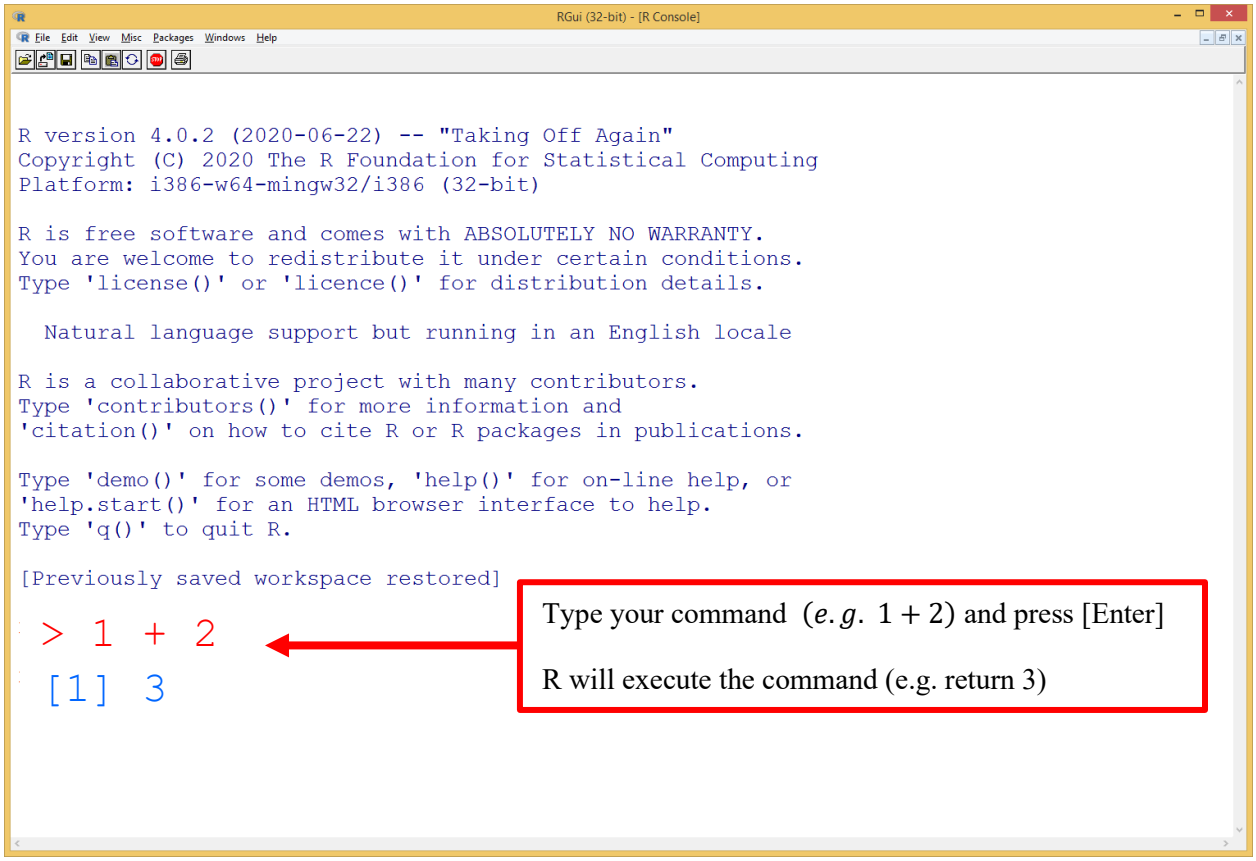
Table of Contents

What is R?

R is a programming language designed specifically for statistical computing.
Because R is a programming language, we need a program that translates R commands into instructions a computer can execute. You can download this program from the official website: https://cran.r-project.org/bin/windows/base/



After following the instructions to download and install R, you will see an application on your desktop named R. Opening R will bring up a console where you can type commands and immediately see the results.



Example - A random sample of students is selected from a large statistics class.
For each student, their Age, Major (Art, Business or Science) and Grade (pass, fail or withdraw) is recorded into a dataset named ***Students***.

Age	Major	Grade
23	Art	Pass
24	Science	Withdraw
30	Business	Pass
27	Business	Pass
19	Art	Fail
28	Business	Pass
20	Science	Pass
23	Business	Pass
18	Science	Pass
22	Science	Pass
⋮	⋮	⋮

The full dataset named '***Students***' can be downloaded from Brightspace.

Load a Dataset in R

To analyze the data, we need to load the dataset into R. How?

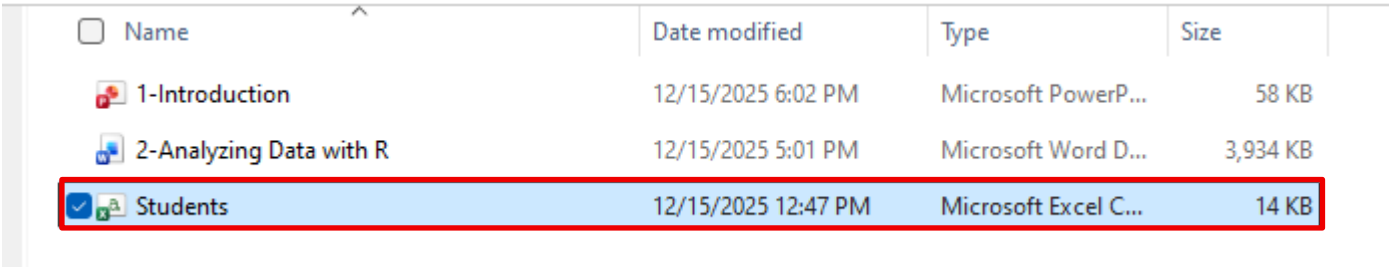
R is more friendly working with the dataset in a **CSV** file.

> A **Comma Separated Values** (**CSV**) file is a plain text file that contains a list of data.
>
> In the file, data are separated by the comma character (",").

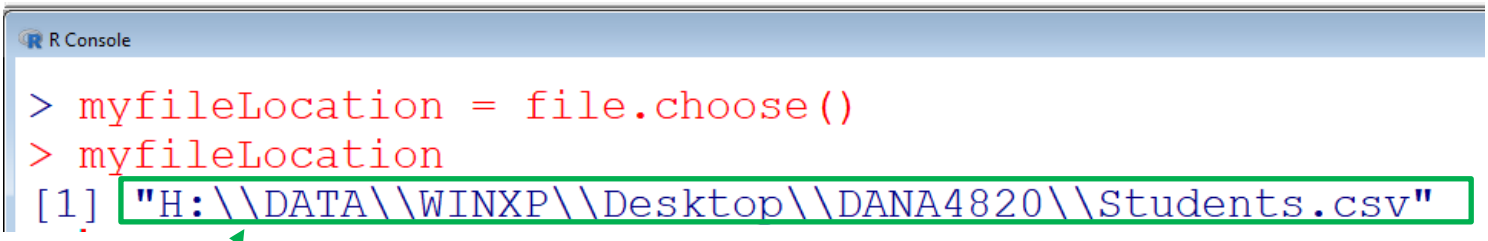
The first step is to locate the data file on your computer.
In R, you can do this using the `file.choose()` function, which allows you to browse and select the file.
The command is shown below.

```
#Use the file.choose() to find the file and
#save the location into a variable called myfileLocation
myfileLocation = file.choose()
```

After running this command, a window will show up that allows you to browse for the file.
Once you have located the file, click "Open" to select it.



Then the location of the file will be saved into the objected named, "myfileLocation".



This is the file location.

Next, we use the file location to read the csv file. To do that, we use the `read.csv(…)` function to read the data in the file. The following is the detailed syntax.

read.csv(file, header, sep)

where

1. file – the file location
2. header – header = TRUE if the file contains the name of the variables on the first line. header = FALSE if otherwise (the data are on the first line)
3. sep – Is the character that separates data on each file. By default, sep is set to a comma. (that's why this function is called "read.csv").

The R command is shown below.

```
#Read the file from the given file location
mydata = read.csv( myfileLocation,  header=TRUE,  sep=",")
```

In this command above, the file location is saved into the variable, "myfileLocation"

The input **header** is set to be TRUE because the file contains the name of the variables.

The input **separator** input is set to a comma (sep=",") that indicates the data are separated by a comma. Note that the separator input is optional (if data are separated by a comma).

The following is the results of the command above.

R Console

```
> myfileLocation = file.choose()
> myfileLocation
[1] "H:\\DATA\\WINXP\\Desktop\\DANA4820\\Students.csv"
> read.csv( myfileLocation, header=TRUE, sep=",")
    Age    Major    Grade
1    23      Art     pass
2    24  Science Withdraw
3    30 Business     Pass
4    27 Business     Pass
5    19      Art     Fail
6    28 Business     Pass
7    20  Science     Pass
8    23 Business     Pass
9    18  Science     Pass
10   22  Science     Pass
11   21  Science     Pass
12   25 Business     Pass
13   22  Science     Pass
14   20  Science     Pass
15   22  Science     Pass
16   20 Business     Fail
17   19 Business     Fail
18   23 Business     Pass
19   25      Art     Pass
20   21      Art Withdraw
```

The dataset is now displayed in the console. To analyze the data, we need to store it in an object.

Let's define an object named mydata to save the data. The command is shown below.

```
#Read the file from the given file location and save the data into "mydata"
mydata = read.csv( myfileLocation,  header=TRUE,  sep=",")
```

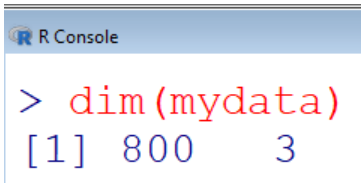
When you type and execute the command "**mydata**". You can see the dataset loaded in R.

```
> mydata = read.csv( myfileLocation, header=TRUE, sep=",")
> mydata
    Age    Major    Grade
1    23      Art     pass
2    24  Science Withdraw
3    30 Business     Pass
4    27 Business     Pass
5    19      Art     Fail
6    28 Business     Pass
7    20  Science     Pass
8    23 Business     Pass
9    18  Science     Pass
10   22  Science     Pass
11   21  Science     Pass
12   25 Business     Pass
13   22  Science     Pass
14   20  Science     Pass
15   22  Science     Pass
16   20 Business     Fail
17   19 Business     Fail
18   23 Business     Pass
19   25      Art     Pass
20   21      Art Withdraw
```

Manipulating data in R

After reading/saving the data into R, it is important to know how many rows and columns are in the dataset, we can use the `dim(…)` function to do that. The function requires an object that contains the data (e.g. `mydata`). The command is shown below.

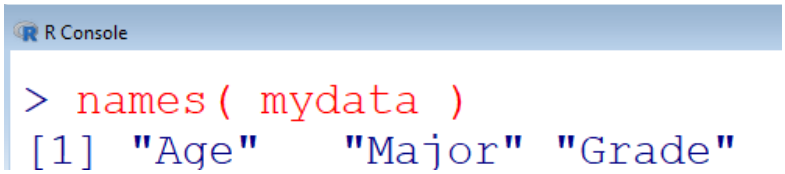
```
dim(mydata)
```



As we can see, the dataset contains 800 rows and 3 columns. In other words, the dataset contains 3 variables for 800 students. What are the variables?

We can use the `names(…)` function to get the names of all variables.
The command is shown below.

```
names(mydata)
```



As we can see, there are three variables,

1. Age
2. Major
3. Grade

To extract the data for each variable, we can use the **"$" operator**.

How? To extract the data for major, we execute the command below.

```
mydata$Major
```

R Console

```
> mydata$Major
 [1] "Art"      "Science"  "Business" "Business" "Art"      "Business"
 [7] "Science"  "Business" "Science"  "Science"  "Science"  "Business"
[13] "Science"  "Science"  "Science"  "Business" "Business" "Business"
[19] "Art"      "Art"      "Business" "Business" "Art"      "Art"
```

To extract the data for grade, we execute the command below.

```
mydata$Grade
```

R Console

```
> mydata$Grade
 [1] "pass"     "Withdraw" "Pass"     "Pass"     "Fail"     "Pass"
 [7] "Pass"     "Pass"     "Pass"     "Pass"     "Pass"     "Pass"
[13] "Pass"     "Pass"     "Pass"     "Fail"     "Fail"     "Pass"
[19] "Pass"     "Withdraw" "Pass"     "Pass"     "Pass"     "Pass"
```

Analyzing Data for a Categorical Variable

In this dataset, the variable "grade" is a categorical variable that consists of three categories: pass, fail and withdraw

To summarize data for a categorical variable, we create a frequency table.
A frequency table has two columns:

1. The first column lists all categories.
2. The second column lists the frequency of each category, which tells us how many times each category occurs.

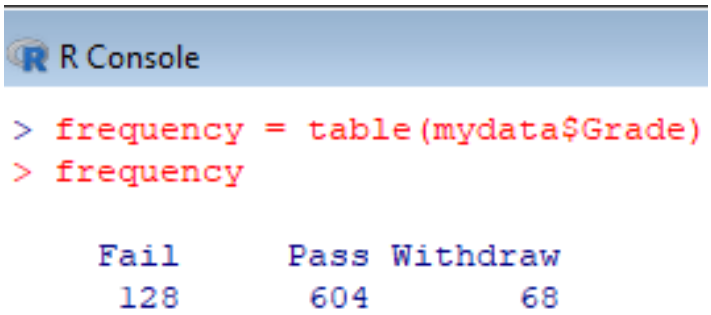
Grade	Frequency
Pass	
Fail	
Withdraw	

Making a frequency table in R

In R, we can use the `table(…)` function to create a frequency table for a single categorical variable. This function requires an object that contains the data (e.g. mydata$Major).
The command is shown below.

```
frequency = table(mydata$Grade)
frequency
```

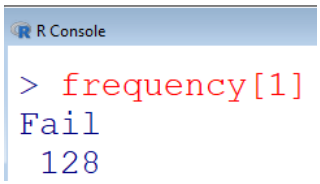
In the command above, the result of the `table(...)` function is stored in an object named **`frequency`**. When we type and execute **`frequency`**, the frequencies for all categories are displayed.



The returned object **frequency** is a vector. We can access each item of the vector using the square-bracket **[ ]** operator.

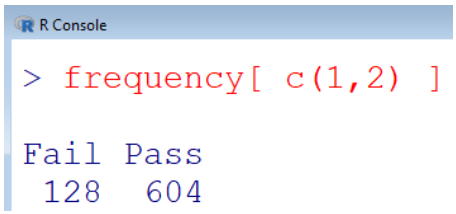
For example, to access the first item of **frequency** (e.g., the frequency of the *Failing* category), we execute the following command:

```
frequency[1]
```



To access the first and second items of the **frequency** (the frequency of the *Failing / Passing* category), we execute the following command:

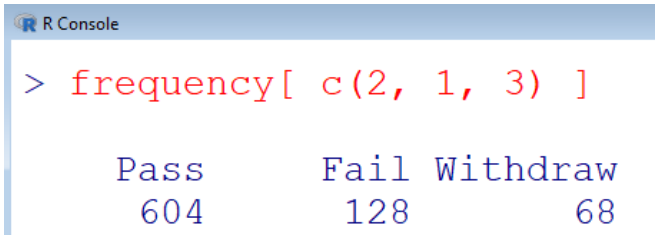
```
frequency[ c(1,2) ]
```



Note: In the command above, we use the `c(…)` function to combine two numbers

In addition, to reorder the categories as ***Pass, Fail, and Withdraw***, we can specify the desired order inside the square brackets, as shown in the command below.

```
frequency[ c("Pass", "Fail", "Withdraw") ]
```



Or we can execute the following command:

```
frequency[ c(2, 1, 3) ]
```

Why does the command work? In the original order, *Fail* is the first item, *Pass* is the second item, and *Withdraw* is the third item. We can rearrange the elements into the order we want by specifying the desired positions inside the square brackets.

Next, we calculate the **proportion** of each grade category

The proportion of a category is calculated by:

$$Proportion = \frac{Frequency\ of\ a\ category}{Sample\ Size}$$

Major	Frequency	Proportion
Pass	604	$\frac{604}{800} = 0.755$
Fail	128	$\frac{128}{800} = 0.160$
Withdraw	68	$\frac{68}{800} = 0.085$
Total	800	

In R, we can use the function `prop.table(…)` to calculate the proportion of each category.

The function requires the frequency table created from the `table(…)` function.

The command is shown below.

```
proportions = prop.table(frequency)
proportions
```

In the command above, the resulting proportions are stored in an object named **proportions**.
When we type and execute **proportions**, the proportions for all categories are displayed.

```
> proportions = prop.table(frequency)
> proportions

    Fail     Pass Withdraw
   0.160    0.755    0.085
```

To get the percentages, we can multiple the proportions by 100

```
proportions * 100
```

R Console

```
> proportions * 100

 Fail  Pass Withdraw
 16.0  75.5      8.5
```

Making a Bar Chart

To display categorical data, we can make a bar chart.

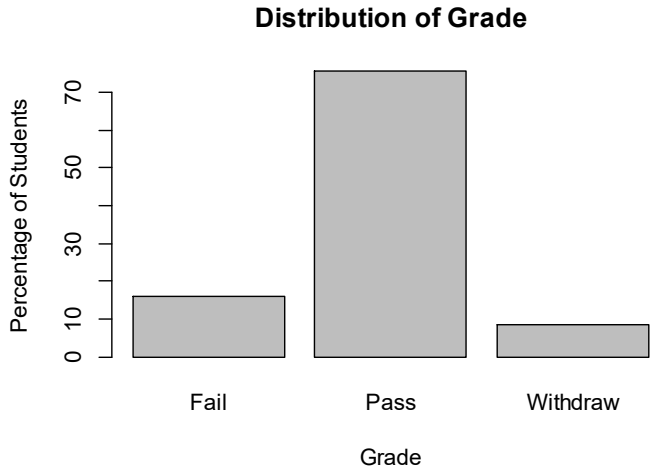
The function to create a bar chart is: `barplot`(**height**, **names.arg**, **main**, **xlab**, **ylab**, **col**)

- **height**: an object that contains frequency (or proportions) representing heights of the bars
- **names.arg** (optional): vector of names to appear under the bars
- **main** (optional): title for plot
- **xlab** (optional): title for x-axis (horizontal axis)
- **ylab** (optional): title for y-axis (vertical axis)
- **col** (optional): color of the bars

Here are the commands used to create a bar chart.

```
barplot(
  height = proportions * 100,
  main = "Distribution of Grade",
  xlab = "Grade",
  ylab = "Percentage of Students"
)
```

```
> barplot(
+   height = proportions * 100,
+   main = "Distribution of Grade",
+   xlab = "Grade",
+   ylab = "Percentage of Students"
+ )
```



Telling a Story about the data using Percentages

When telling a story about categorical data, we are usually interested in either

- identifying the most frequent and the least frequent categories or
- ordering the categories by their frequency.

In this dataset, we can see that most students (75.5%) pass the class, the fewest (8.5%) withdraw from the class.

Analyzing Data for Two Categorical Variables

This dataset contains two categorical variables: *Major* and *Grade*.
When analyzing two categorical variables (e.g., *Major* and *Grade*), we are usually interested in describing the relationship between them, if any.

To do this, we construct a **two-way frequency table**.

- The **row variable** is *Grade*, with three categories: *Pass*, *Fail*, and *Withdraw*.
- The **column variable** is *Major*, with three categories: *Art*, *Business*, and *Science*.

	Art	Business	Science	Total
Pass				
Fail				
Withdraw				
Total				

Making a Two-Way Frequency Table in R

Again, we can use the `table(…)` function to create the frequency table.

This time, function requires two objects that contain the data: Major and Grade

```
two.way.frequency = table( mydata$Grade,   mydata$Major)
two.way.frequency
```

Note: In the command above, the first object represents the row variable, and the second object represents the column variable.
The resulting two-way frequency table is stored in an object named **two_way_frequency**.

R Console

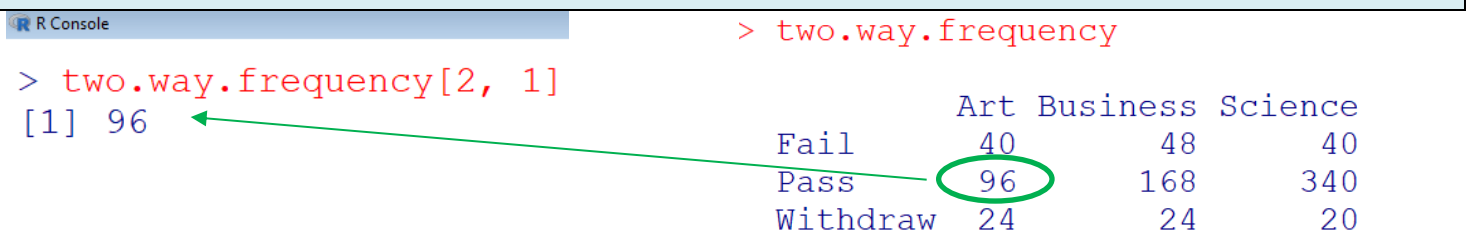
```
> two.way.frequency = table( mydata$Grade,   mydata$Major)
> two.way.frequency

          Art Business Science
  Fail     40       48      40
  Pass     96      168     340
  Withdraw 24       24      20
```

The returned object, `two.way.frequency`, is a matrix.

To access an individual entry in the matrix, we use the square-bracket ([]) operator.

For example, to obtain the value in the **second row and first column** (the number of *Art* students who *fail*), we run the following command:

```
two.way.frequency[2, 1]
```

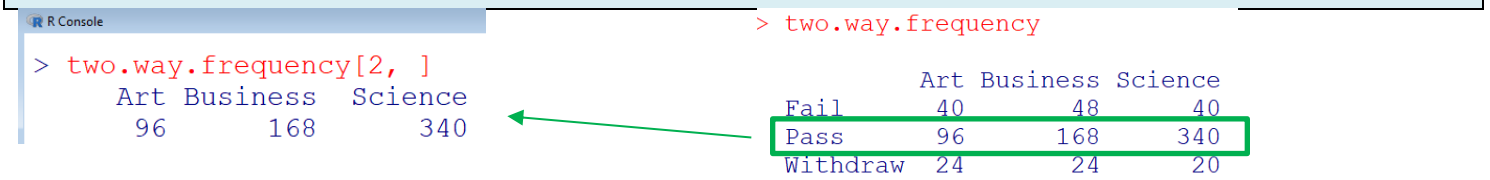


In the command above:

- The **first number** inside the square brackets represents the **row position** of the item.
- The **second number** represents the **column position** of the item.
- Be sure to use a **comma (,)** to separate the two position numbers.

To obtain the values in the **entire second row**, we run the following command:

```
two.way.frequency[2, ]
```

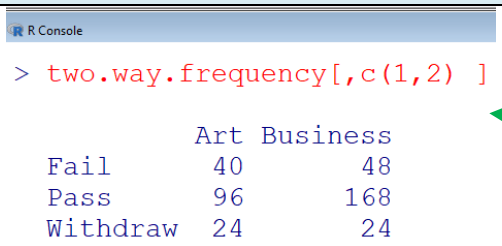


In the command above:

- The **first number** inside the square brackets represents the **row position**, and the **second position is left blank** to indicate that all columns are selected.
- Be sure to use a **comma (,)** after the row position.

To obtain the values in the **entire first/second columns**, we run the following command:

```
two.way.frequency[,c(1,2) ]
```




```
> two.way.frequency
          Art Business Science
Fail       40       48      40
Pass       96      168     340
Withdraw   24       24      20
```

In the command above:

- We leave the **row position blank** indicates that **all rows** are selected, and the **numbers inside the square brackets represent the column positions**.
- Be sure to include a **comma (,) before the column positions**,
- and use the **c(…) function** to combine multiple column positions.

To reorder the rows as *Pass*, *Fail*, and *Withdraw*, we can specify the desired order inside the square brackets, as shown in the command below.

```
two.way.frequency = two.way.frequency[ c("Pass", "Fail", "Withdraw"),  ]
two.way.frequency
```

```
> two.way.frequency = two.way.frequency[ c("Pass", "Fail", "Withdraw"),  ]
> two.way.frequency
          Art Business Science
Pass       96      168     340
Fail       40       48      40
Withdraw   24       24      20
```

After creating the two-way table of *Grade* and *Major*, we can begin to describe the relationship between the two variables.

	Art	Business	Science
Pass	90	168	340
Fail	40	48	40
Withdraw	24	24	20

Describing the Relationship between Two Categorical Variable

First, we divide the sample into three groups—Art, Business, and Science—based on students' majors. Within each group, we calculate the percentage of students who pass, fail, or withdraw from the class.

Art Majors

	Art	% of students
Pass	90	
Fail	40	
Withdraw	24	
Total		

Business Majors

	Business	% of students
Pass	168	
Fail	48	
Withdraw	24	
Total		

Science Majors

	Science	% of students
Pass	340	
Fail	40	
Withdraw	20	
Total		

Now, we can compare the percentage of students in a particular grade category (e.g., passing the class) across the three majors.

Question:

1. Which major(s) are most likely to have students pass the class?
2. Which major(s) are most likely to have students withdraw from the class?

In R, we can calculate the percentage of students in each grade category within each major.

Again, we can still use the `prop.tabe(…)` function to do that.

The following is the detailed syntax.

The function to create proportions is: `prop.table`(frequency table, margin)

- **frequency table**: the frequency table created with `table(…)`
- **margin** (optional): It takes the values of 1 or 2

  1 for row proportions (- the proportions are calculated based on the row totals)
  2 for column proportions (- the proportions are calculated based on the column totals)

If the value of margin is not specified, then it will calculate proportions based on the grand total

Let's look at the two-way frequency table again.

	Art	Business	Science
Pass	90	168	340
Fail	40	48	40
Withdraw	24	24	20

The percentage of each grade category should be calculated within each **column** (major), using the **column total** as the denominator. The following commands calculate the column proportions:

```
grade.proportion.by.major = prop.table( two.way.frequency, margin = 2 )
grade.proportion.by.major
```

```
> grade.proportion.by.major = prop.table( two.way.frequency, margin = 2 )
> grade.proportion.by.major

           Art Business Science
Pass      0.60     0.70    0.85
Fail      0.25     0.20    0.10
Withdraw  0.15     0.10    0.05
```

Finally, to obtain the percentages, multiply the proportions by 100.

```
grade.proportion.by.major * 100
```

R Console

```
> grade.proportion.by.major * 100

          Art Business Science
Pass       60       70      85
Fail       25       20      10
Withdraw   15       10       5
```

Making Side-by-Side Bar Charts

After calculating the percentages within each major, we can create a bar chart for each major, displayed side by side. This type of chart is called a ***side-by-side bar chart***.

The function to create a bar chart is: `barplot`(**height**, **names.arg**, **main**, **xlab**, **ylab**, **col**)

- **height**: an object that contains frequency (or proportions) representing heights of the bars Note: Column labels will appear on the x-axis, and row labels will appear in the legend.
- **beside** (optional): **TRUE for bars side-by-side** (FALSE for stacked bars)
- **legend.text** (optional): TRUE to use the row names of your table as legend labels
- **col** (optional): vector of colors
- **main** (optional): title for plot
- **xlab** (optional): title for x-axis (horizontal axis)
- **ylab** (optional): title for y-axis (vertical axis)

Let's create the side-by-side bar charts of column percentages.

```
barplot(height = grade.proportion.by.major * 100,
        beside = TRUE,
        legend.text = TRUE,
        col = c("red", "blue", "green"),
        xlab = "Gender",
        ylab = "Percentage of students",
        main = "Distribution of Major by Gender")
```