

Crosstabs: Counts, Proportions, and More

from Doing LVC with R*

Matt Hunt Gardner

2022-09-27

Table of contents

Token Counts	1
Summary Statistics for Continous Variables	11
Dealing with Decimals	12
More Summary Statistics for Continous Variables	15
Position functions with <code>summarize()</code>	16
Count functions with <code>summarize()</code>	17
Logical functions	18
Proportions	18

It took me two years to figure out how to do cross-tabs in R the way that *Goldvarb* does cross-tabs. Below I show you how to build cross-tabs from scratch.

Token Counts

A good starting point is the function `table()`. This function returns token numbers.

💡 Get the data first

If you don't have the `td` data loaded in R, go back to Doing it all again, but `tidy`^a and run the code.

^ahttps://lingmethodshub.github.io/content/R/lvc_r/050_lvc_r.html

```
# Get the number of tokens by level of Dep.Var
table(td$Dep.Var)
```

```
Deletion Realized
386      803
```

This tells you that there are 386 *Deletion* tokens and 803 not deleted, or *Realized* tokens. If you add another factor group like *Age.Group*, you get the number of tokens for each level of *Dep.Var* for each level of that additional factor group. These two factor groups are returned as the rows and then columns in the table.

```
# Get the number of tokens by level of Dep.Var
# and Age.Group
table(td$Dep.Var, td$Age.Group)
```

^{*}https://lingmethodshub.github.io/content/R/lvc_r/

	Old	Middle	Young
Deletion	67	125	194
Realized	134	235	434

If you add one more factor group, `Sex`, it divides the data in what R calls “pages”. The first page is the number of tokens for each level of `Dep.Var` by each level of `Age.Group` for female data (`Sex = F`), and then the same for the male data (`Sex = M`).

```
# Get the number of tokens by Dep.Var, Sex, and
# Age.Group
table(td$Dep.Var, td$Age.Group, td$Sex)

, , = F
```

	Old	Middle	Young
Deletion	43	73	72
Realized	107	165	199

```
, , = M
```

	Old	Middle	Young
Deletion	24	52	122
Realized	27	70	235

You can add the option `deparse.level = 2` to include the names of the columns in the table.

```
# Get the number of tokens by Dep.Var, Sex, and
# Age.Group
table(td$Dep.Var, td$Age.Group, td$Sex, deparse.level = 2)

, , td$Sex = F

      td$Age.Group
td$Dep.Var Old Middle Young
Deletion   43     73     72
Realized  107    165    199

, , td$Sex = M
```

	Old	Middle	Young
Deletion	24	52	122
Realized	27	70	235

If you wrap the `table()` function in the `addmargins()` function you get the sums of each row and column, and another page for both the male and the female data together.

```
# Get the number of tokens by Dep.Var, Sex, and
# Age.Group, with column, row and page totals
addmargins(table(td$Dep.Var, td$Age.Group, td$Sex,
  deparse.level = 2))

, , td$Sex = F
```

	td\$Age.Group			
td\$Dep.Var	Old	Middle	Young	Sum
Deletion	43	73	72	188
Realized	107	165	199	471
Sum	150	238	271	659

, , td\$Sex = M

	td\$Age.Group			
td\$Dep.Var	Old	Middle	Young	Sum
Deletion	24	52	122	198
Realized	27	70	235	332
Sum	51	122	357	530

, , td\$Sex = Sum

	td\$Age.Group			
td\$Dep.Var	Old	Middle	Young	Sum
Deletion	67	125	194	386
Realized	134	235	434	803
Sum	201	360	628	1189

If you change the order of factor groups you include in the `table()` function you can change which factors are rows, which are columns, and which are pages. You can also keep adding factors as additional pages. The order is always: rows, columns, page 1, page 2, etc.

```
# Get the number of tokens by Age.Group,  
# Education, Sex, and Dep.Var, with row, column,  
# and page totals  
addmargins(table(td$Age.Group, td$Education, td$Sex,  
  td$Dep.Var, deparse.level = 2))
```

, , td\$Sex = F, td\$Dep.Var = Deletion

	td\$Education				
td\$Age.Group	Educated	Not Educated	Student	Sum	
Old	2	41	0	43	
Middle	68	5	0	73	
Young	20	0	52	72	
Sum	90	46	52	188	

, , td\$Sex = M, td\$Dep.Var = Deletion

	td\$Education				
td\$Age.Group	Educated	Not Educated	Student	Sum	
Old	0	24	0	24	
Middle	16	36	0	52	
Young	48	24	50	122	
Sum	64	84	50	198	

, , td\$Sex = Sum, td\$Dep.Var = Deletion

td\$Education

td\$Age.Group	Educated	Not Educated	Student	Sum
Old	2	65	0	67
Middle	84	41	0	125
Young	68	24	102	194
Sum	154	130	102	386

, , td\$Sex = F, td\$Dep.Var = Realized

	td\$Education			
td\$Age.Group	Educated	Not Educated	Student	Sum
Old	30	77	0	107
Middle	153	12	0	165
Young	52	0	147	199
Sum	235	89	147	471

, , td\$Sex = M, td\$Dep.Var = Realized

	td\$Education			
td\$Age.Group	Educated	Not Educated	Student	Sum
Old	0	27	0	27
Middle	30	40	0	70
Young	77	31	127	235
Sum	107	98	127	332

, , td\$Sex = Sum, td\$Dep.Var = Realized

	td\$Education			
td\$Age.Group	Educated	Not Educated	Student	Sum
Old	30	104	0	134
Middle	183	52	0	235
Young	129	31	274	434
Sum	342	187	274	803

, , td\$Sex = F, td\$Dep.Var = Sum

	td\$Education			
td\$Age.Group	Educated	Not Educated	Student	Sum
Old	32	118	0	150
Middle	221	17	0	238
Young	72	0	199	271
Sum	325	135	199	659

, , td\$Sex = M, td\$Dep.Var = Sum

	td\$Education			
td\$Age.Group	Educated	Not Educated	Student	Sum
Old	0	51	0	51
Middle	46	76	0	122
Young	125	55	177	357
Sum	171	182	177	530

, , td\$Sex = Sum, td\$Dep.Var = Sum

td\$Education

td\$Age.Group	Educated	Not Educated	Student	Sum
Old	32	169	0	201
Middle	267	93	0	360
Young	197	55	376	628
Sum	496	317	376	1189

The above function produces 9 “pages”, one for each combination of *Sex* (two levels) and *Dep.Var* (two levels), plus the sum of each (one additional level each), and the sum for both. With more than three factor groups like this it is very useful to have the column names included in the output. Scroll to the sixth page, for example (the one that begins `, , td$Sex = Sum, td$Dep.Var = Realized`). It shows the number of tokens by *Age.Group* and *Education* (the first two factor groups in the function), when *Sex* equals *Sum* (e.g., *M* and *F* combined) and *Dep.Var* equals *Realized*.

One advantage of doing cross-tabs in *R*, rather than *Goldvarb*, is that you can simultaneously cross more than two factor groups at once. But, the presentation of these factors in pages may not be the most useful. The function `fTable()` in the package *vcd* presents the cross-tab in a more condensed format. The last factor group in the `table()` function will be the variable for the columns in `fTable()`, so you always want to make that the dependent variable. Below is the `fTable()` for the cross-tab of *Age.Group*, *Education*, *Sex*, and *Dep.Var*. You can see, for example, that there are 52 *Deletion* tokens from young, student, female speakers and that there are no tokens from old, educated men.

```
# Get the number of tokens by Age.Group,
# Education, Sex, and Dep.Var, with row, column
# and page totals, presented in a flattened table
library(vcd)
fTable(table(td$Age.Group, td$Education, td$Sex, td$Dep.Var))
```

Deletion Realized				
Old	Educated	F	2	30
		M	0	0
	Not Educated	F	41	77
		M	24	27
	Student	F	0	0
		M	0	0
Middle	Educated	F	68	153
		M	16	30
	Not Educated	F	5	12
		M	36	40
	Student	F	0	0
		M	0	0
Young	Educated	F	20	52
		M	48	77
	Not Educated	F	0	0
		M	24	31
	Student	F	52	147
		M	50	127

```
# Do the same but include the margin values
fTable(addmargins(table(td$Age.Group, td$Education,
  td$Sex, td$Dep.Var)))
```

Deletion Realized					Sum
Old	Educated	F	2	30	32

		M	0	0	0
		Sum	2	30	32
	Not Educated	F	41	77	118
		M	24	27	51
		Sum	65	104	169
	Student	F	0	0	0
		M	0	0	0
		Sum	0	0	0
	Sum	F	43	107	150
		M	24	27	51
		Sum	67	134	201
	Middle Educated	F	68	153	221
		M	16	30	46
		Sum	84	183	267
	Not Educated	F	5	12	17
		M	36	40	76
		Sum	41	52	93
	Student	F	0	0	0
		M	0	0	0
		Sum	0	0	0
	Sum	F	73	165	238
		M	52	70	122
		Sum	125	235	360
	Young Educated	F	20	52	72
		M	48	77	125
		Sum	68	129	197
	Not Educated	F	0	0	0
		M	24	31	55
		Sum	24	31	55
	Student	F	52	147	199
		M	50	127	177
		Sum	102	274	376
	Sum	F	72	199	271
		M	122	235	357
		Sum	194	434	628
	Sum Educated	F	90	235	325
		M	64	107	171
		Sum	154	342	496
	Not Educated	F	46	89	135
		M	84	98	182
		Sum	130	187	317
	Student	F	52	147	199
		M	50	127	177
		Sum	102	274	376
	Sum	F	188	471	659
		M	198	332	530
		Sum	386	803	1189

Of course we can use the pipe `%>%` to make things a bit easier

```
# Get the number of tokens by Age.Group,
# Education, Sex, and Dep.Var, with row, column
# and page totals, presented in a flattened table
```

```
table(td$Age.Group, td$Education, td$Sex, td$Dep.Var) %>%
  addmargins() %>%
  ftable()
```

			Deletion	Realized	Sum
Old	Educated	F	2	30	32
		M	0	0	0
		Sum	2	30	32
	Not Educated	F	41	77	118
		M	24	27	51
		Sum	65	104	169
	Student	F	0	0	0
		M	0	0	0
		Sum	0	0	0
	Sum	F	43	107	150
		M	24	27	51
		Sum	67	134	201
Middle	Educated	F	68	153	221
		M	16	30	46
		Sum	84	183	267
	Not Educated	F	5	12	17
		M	36	40	76
		Sum	41	52	93
	Student	F	0	0	0
		M	0	0	0
		Sum	0	0	0
	Sum	F	73	165	238
		M	52	70	122
		Sum	125	235	360
Young	Educated	F	20	52	72
		M	48	77	125
		Sum	68	129	197
	Not Educated	F	0	0	0
		M	24	31	55
		Sum	24	31	55
	Student	F	52	147	199
		M	50	127	177
		Sum	102	274	376
	Sum	F	72	199	271
		M	122	235	357
		Sum	194	434	628
Sum	Educated	F	90	235	325
		M	64	107	171
		Sum	154	342	496
	Not Educated	F	46	89	135
		M	84	98	182
		Sum	130	187	317
	Student	F	52	147	199
		M	50	127	177
		Sum	102	274	376
	Sum	F	188	471	659
		M	198	332	530

Sum 386 803 1189

Another **tidy** way to find out the number of tokens by the different levels of a factor group is using the **group_by()** and **tally()** functions. First, we specify how to group the data, i.e., what combination of factors we want to investigate. In this case, we want the number of tokens for every combination of **Age.Group**, **Education**, **Sex** and **Dep.Var**. Next we use the **tally()** function to provide the token counts for each of those combinations. The results are very similar to those produced by **fTable(table())**.

```
# Group data by Age, Education, and Sex then
# tally each group
td %>%
  group_by(Age.Group, Education, Sex, Dep.Var) %>%
  tally()
```

```
# A tibble: 24 x 5
# Groups:   Age.Group, Education, Sex [12]
  Age.Group Education Sex Dep.Var n
  <fct>      <fct>    <fct> <fct> <int>
1 Old      Educated    F      Deletion 2
2 Old      Educated    F      Realized 30
3 Old      Not Educated F      Deletion 41
4 Old      Not Educated F      Realized 77
5 Old      Not Educated M      Deletion 24
6 Old      Not Educated M      Realized 27
7 Middle   Educated    F      Deletion 68
8 Middle   Educated    F      Realized 153
9 Middle   Educated    M      Deletion 16
10 Middle  Educated    M      Realized 30
# ... with 14 more rows
# i Use `print(n = ...)` to see more rows
```

As the results of **tally()** is a *tibble*, only the first 10 rows will be printed. To print all the rows add **print(n=Inf)** at the end.

```
# Group data by Age, Education, and Sex, tally
# each group, then print all rows
td %>%
  group_by(Age.Group, Education, Sex, Dep.Var) %>%
  tally() %>%
  print(n = Inf)
```

```
# A tibble: 24 x 5
# Groups:   Age.Group, Education, Sex [12]
  Age.Group Education Sex Dep.Var n
  <fct>      <fct>    <fct> <fct> <int>
1 Old      Educated    F      Deletion 2
2 Old      Educated    F      Realized 30
3 Old      Not Educated F      Deletion 41
4 Old      Not Educated F      Realized 77
5 Old      Not Educated M      Deletion 24
6 Old      Not Educated M      Realized 27
7 Middle   Educated    F      Deletion 68
8 Middle   Educated    F      Realized 153
9 Middle   Educated    M      Deletion 16
10 Middle  Educated    M      Realized 30
```


11	Middle	Not Educated	F	Deletion	5
12	Middle	Not Educated	F	Realized	12
13	Middle	Not Educated	M	Deletion	36
14	Middle	Not Educated	M	Realized	40
15	Young	Educated	F	Deletion	20
16	Young	Educated	F	Realized	52
17	Young	Educated	M	Deletion	48
18	Young	Educated	M	Realized	77
19	Young	Not Educated	M	Deletion	24
20	Young	Not Educated	M	Realized	31
21	Young	Student	F	Deletion	52
22	Young	Student	F	Realized	147
23	Young	Student	M	Deletion	50
24	Young	Student	M	Realized	127

The above code gives us the number of **Realized** and **Deletion** tokens for each combination of **Age.Group**, **Education**, and **Sex**. What if we want the total number of tokens for each combination, rather than the number of each level of **Dep.Var**. In this case, you can just drop **Dep.Var** from the **group_by()** function.

```
# Get total number of tokens per group by
# removing Dep.Var
td %>%
  group_by(Age.Group, Education, Sex) %>%
  tally() %>%
  print(n = Inf)
```

```
# A tibble: 12 x 4
# Groups:   Age.Group, Education [7]
  Age.Group Education    Sex      n
  <fct>      <fct>      <fct> <int>
1 Old       Educated      F       32
2 Old       Not Educated F      118
3 Old       Not Educated M       51
4 Middle    Educated      F      221
5 Middle    Educated      M       46
6 Middle    Not Educated F       17
7 Middle    Not Educated M       76
8 Young     Educated      F       72
9 Young     Educated      M      125
10 Young    Not Educated M       55
11 Young    Student       F      199
12 Young    Student       M      177
```

We know now that there are 32 tokens from **Old**, **Educated**, **F** (female) speakers. The previous **tally()** shows us that 2 of the tokens are **Deletion** and 30 are of **Realized**.

An alternative to **tally()** is the much more flexible **summarize()** function.¹ With this function you can apply a summary statistic function to each combination of the grouping variables. If no summary statistic function is created, the a tibble of the combination of the groups is produced.

```
# Create a tibble of all combinations of
# Age.Group, Education, and Sex (for which there
# are rows of data)
```

¹ **summarise()** and **summarize()** are synonyms.

```
td %>%
  group_by(Age.Group, Education, Sex) %>%
  summarize()
```

```
# A tibble: 12 x 3
# Groups:   Age.Group, Education [7]
  Age.Group Education Sex
  <fct>      <fct>    <fct>
1 Old       Educated   F
2 Old       Not Educated F
3 Old       Not Educated M
4 Middle    Educated   F
5 Middle    Educated   M
6 Middle    Not Educated F
7 Middle    Not Educated M
8 Young     Educated   F
9 Young     Educated   M
10 Young    Not Educated M
11 Young    Student    F
12 Young    Student    M
```

To get the count, or number of rows, of each combination, we create a new column in the tibble that is the output of `summarize()` and assign to it the value of the count function `n()`

```
# Create a tibble of grouping variables, then add
# a new column 'Tokens' with the value of the
# count function
td %>%
  group_by(Age.Group, Education, Sex, Dep.Var) %>%
  summarize(Tokens = n()) %>%
  print(n = Inf)
```

```
# A tibble: 24 x 5
# Groups:   Age.Group, Education, Sex [12]
  Age.Group Education Sex Dep.Var Tokens
  <fct>      <fct>    <fct> <fct>    <int>
1 Old       Educated   F Deletion    2
2 Old       Educated   F Realized   30
3 Old       Not Educated F Deletion   41
4 Old       Not Educated F Realized   77
5 Old       Not Educated M Deletion   24
6 Old       Not Educated M Realized   27
7 Middle    Educated   F Deletion   68
8 Middle    Educated   F Realized  153
9 Middle    Educated   M Deletion   16
10 Middle    Educated   M Realized   30
11 Middle    Not Educated F Deletion    5
12 Middle    Not Educated F Realized   12
13 Middle    Not Educated M Deletion   36
14 Middle    Not Educated M Realized   40
15 Young     Educated   F Deletion   20
16 Young     Educated   F Realized   52
17 Young     Educated   M Deletion   48
18 Young     Educated   M Realized   77
```

19	Young	Not Educated	M	Deletion	24
20	Young	Not Educated	M	Realized	31
21	Young	Student	F	Deletion	52
22	Young	Student	F	Realized	147
23	Young	Student	M	Deletion	50
24	Young	Student	M	Realized	127

The `summarize()` function can be used with a number of summary statistic functions, including, but not limited to, the following:

Type	Some Useful Functions
Center	<code>mean()</code> , <code>median()</code>
Spread	<code>sd()</code> , <code>IQR()</code>
Range	<code>min()</code> , <code>max()</code>
Position	<code>first()</code> , <code>last()</code> , <code>nth()</code>
Count	<code>n()</code> , <code>n_distinct()</code>
Logical	<code>any()</code> , <code>all()</code>

Summary Statistics for Continuous Variables

This seems like an appropriate place to describe how to summarize values that are continuous, like `YOB`. Normally in variationist sociolinguistics we are very concerned with frequency and proportion of usage, and we will explore how to generate those statistics in the following section. Here, however, let's explore the functions available to use inside `summarize()`. These functions can be used on their own, also. For example, the first two, `mean()` and `median()` provide the arithmetic mean (basically the average) of a set of numbers while the `median()` provides the exact middle number of a set of values organized from smallest to largest (if there are an even number of values, `median()` returns the halfway point between the two middle numbers).

```
# Get mean year of birth
mean(td$YOB)
```

```
[1] 1969.447
```

```
# Get median year of birth
median(td$YOB)
```

```
[1] 1984
```

We already know that the mean year of birth for the `td` data set is 1969.447. You can also see that the middle number of all years of birth organized from oldest to youngest is 1984. If we wanted to find the mean or median year of birth for either just male or just female speakers, we have two options. We can use the base filter technique, or we can use the `tidy` method to group the data and summarize it.

```
# Get mean year of birth of just female speakers
mean(td$YOB[td$Sex == "F"])
```

```
[1] 1963.487
```

```
# Get mean year of birth of just male speaker
mean(td$YOB[td$Sex == "M"])
```

```
[1] 1976.857
```

```
# Get mean year of birth for each level of Sex
td %>%
  group_by(Sex) %>%
  summarize(Mean.YOB = mean(YOB))

# A tibble: 2 x 2
  Sex    Mean.YOB
<fct>    <dbl>
1 F      1963.
2 M      1977.
```

Dealing with Decimals

Tibbles are intended to be succinct and concise, so they provide very few values after the decimal place by default. If you require more decimal values, the easiest (trust me) thing to do is to convert the tibble into a *data frame*.

```
# Get mean year of birth by Sex, converted to
# data frame
td %>%
  group_by(Sex) %>%
  summarize(Mean.YOB = mean(YOB)) %>%
  as.data.frame()

Sex Mean.YOB
1  F 1963.487
2  M 1976.857
```

data frames will display whole numbers, and numbers with decimals up to the total number of digits set by `options()` function. Keep in mind, though, that changing this value changes the global options for R. An alternative is to use the `format()` function.

```
# Change number of significant digits displayed
# to 6
options(digits = 6)
# Get mean year of birth by sex, converted to
# data frame
td %>%
  group_by(Sex) %>%
  summarize(Mean.YOB = mean(YOB)) %>%
  as.data.frame()

Sex Mean.YOB
1  F 1963.49
2  M 1976.86
```

```
# Change number of significant digits displayed
# to 10
options(digits = 10)
# Get mean year of birth by sex, converted to
# data frame
td %>%
  group_by(Sex) %>%
```

```
summarize(Mean.YOB = mean(YOB)) %>%  
as.data.frame()
```

```
Sex    Mean.YOB  
1  F 1963.487102  
2  M 1976.856604
```

```
# Change number of significant digits displayed  
# to 3  
options(digits = 3)  
# Get mean year of birth by sex, converted to  
# data frame  
td %>%  
  group_by(Sex) %>%  
  summarize(Mean.YOB = mean(YOB)) %>%  
  as.data.frame()
```

```
Sex Mean.YOB  
1  F      1963  
2  M      1977
```

```
# Change number of significant digits displayed  
# to 3  
options(digits = 3)  
# Get mean year of birth by sex, converted to  
# data frame but showing 10 significant digits  
td %>%  
  group_by(Sex) %>%  
  summarize(Mean.YOB = mean(YOB)) %>%  
  as.data.frame() %>%  
  format(digits = 10)
```

```
Sex    Mean.YOB  
1  F 1963.487102  
2  M 1976.856604
```

For very large numbers R will often display values in exponential notation. We can alter this by setting the value of `scipen` inside the `option()` function. Again, though, remember that this is a global change for your whole R session. For `scipen` positive values increase the likelihood of using real numbers, negative values increase the likelihood of using exponential notation. To ensure printouts are always real numbers, set `scipen` to 9999 (this is the default). To ensure printouts are always exponential notation, set `scipen` to -9999. To demonstrate, below we multiply mean `YOB` by 10000.

```
# Change number of significant digits displayed  
# to 6, alter the likelihood of use of real  
# number rather than scientific notation by 0  
options(digits = 6, scipen = 0)  
# Get mean year of birth by sex multiplied by  
# 100000, converted to data frame  
td %>%  
  group_by(Sex) %>%  
  summarize(Mean.YOB = mean(YOB) * 1e+05) %>%  
  as.data.frame()
```

```
Sex Mean.YOB
1  F 196348710
2  M 197685660
```

With `scipen` set to 0, we still get real numbers as the values `Mean.YOB` are not too big. To ensure we have real numbers, though, we change the `scipen` value.

```
# Change number of significant digits displayed
# to 6, alter the likelihood of use of real
# number rather than scientific notation by 9999
options(digits = 6, scipen = 9999)
# Get mean year of birth by sex multiplied by
# 100000, converted to data frame
td %>%
  group_by(Sex) %>%
  summarize(Mean.YOB = mean(YOB) * 10000) %>%
  as.data.frame()
```

```
Sex Mean.YOB
1  F 19634871
2  M 19768566
```

If, instead we prefer exponential notation, we use the maximum negative `scipen` value, `-9999/`

```
# Change number of significant digits displayed
# to 6, alter the likelihood of use of real
# number rather than scientific notation by -9999
options(digits = 6, scipen = -9999)
# Get mean year of birth by sex multiplied by
# 100000, converted to data frame
td %>%
  group_by(Sex) %>%
  summarize(Mean.YOB = mean(YOB) * 10000) %>%
  as.data.frame()
```

```
Sex Mean.YOB
1  F 1.96349e+07
2  M 1.97686e+07
```

Above, the value `1.96349e+07` means 1.96349×10^7 . The easiest way to calculate this is to simply move the decimal places 7 spaces to the right (as the exponent is positive), which gives `19634900`. Notice some precision is lost because our number of `digits` is only 6.

```
# Change number of significant digits displayed
# to 10, alter the likelihood of use of real
# number rather than scientific notation by -9999
options(digits = 1e+01, scipen = -9.999e+03)
# Get mean year of birth by sex multiplied by
# 100000, converted to data frame
td %>%
  group_by(Sex) %>%
  summarize(Mean.YOB = mean(YOB) * 1e+04) %>%
  as.data.frame()
```

```
Sex Mean.YOB
```

```
1 F 1.963487102e+07
2 M 1.976856604e+07
```

Now, with more **digits** we have more precision; $1.963487102 \times 10^7 = 19634671.02$. If the exponential values are negative, move the decimal place to the left. For example, $1.963487102 \times 10^{-7} = 0.0000001963467102$.

Similarly, we can set whether or not we want scientific notation using the `format()` function. The **scientific** option can be either **TRUE** or **FALSE**, or a value like **scipen**.

```
# Change number of significant digits displayed
# to 3, alter the likelihood of use of real
# number rather than scientific notation by 9999
options(digits = 3e+00, scipen = 9.999e+03)
# Get mean year of birth by sex multiplied by
# 100000, converted to data frame, digits
# formatted to 10 significant digits, and
# exponential notation
td %>%
  group_by(Sex) %>%
  summarize(Mean.YOB = mean(YOB) * 1e+04) %>%
  as.data.frame() %>%
  format(digits = 1e+01, scientific = TRUE)
```

```
Sex      Mean.YOB
1 F 1.963487102e+07
2 M 1.976856604e+07
```

More Summary Statistics for Continuous Variables

The other summary statistics for continuous variables include spread functions and the range functions. Some spread functions are `sd()`, which returns the standard deviation; and `IQR()` which returns the interquartile range.² Some range functions include: `min()`, which returns the lowest value; `max()`, which returns the highest value. To find the maximum spread (from highest to lowest), we can either subtract the `min()` value from the `max()` value, or employ the `diff()` function plus the `range()` function (which produces a vector containing the minimum and maximum values).

We can include these functions inside the same `summarize()` function as we used above.

```
# Get mean, standard deviation, interquartile
# range, minimum value, maximum value, and range
# of values (twice) for year of birth
td %>%
  group_by(Sex) %>%
  summarize(Mean.YOB = mean(YOB), SD.YOB = sd(YOB),
            IQR.YOB = IQR(YOB), Min.YOB = min(YOB), Max.YOB = max(YOB),
            Range = max(YOB) - min(YOB), Range2 = diff(range(YOB)))
```

```
# A tibble: 2 x 8
Sex      Mean.YOB SD.YOB IQR.YOB Min.YOB Max.YOB Range Range2
<fct>    <dbl> <dbl> <dbl> <int> <int> <int> <int>
```

²If we order the data from lowest to highest values, 50% of the data will be less than the mean, and 50% of the data will be higher than the mean. The mean is also called the 2nd quartile. The first quartile is halfway between the mean and the lowest value in the data. The third quartile is halfway between the mean and the highest value in the data. The interquartile range is the difference between the 3rd quartile and the 1st quartile and represents the spread of the middle 50% of the data.

1 F	1963.	26.5	45	1915	1999	84	84
2 M	1977.	19.6	33	1921	1994	73	73

Based on these values, we can make the following statements:

- Among females in the (t, d) data, the average or mean year of birth is 1963 ± 26.5 years.
- The oldest female speakers was born in 1915, and the youngest female speaker was born in 1999.
- Fifty-percent of women were born in the 45 years centered around 1963.
- The female data represents 84 years of apparent time³.

Position functions with `summarize()`

The position functions `first()`, `last()`, and `nth()` also work on the data created by `group_by()` and `summarize()`. `first()` returns the first value, `last()` returns the last value, and `nth()` returns the value after a specific number of rows.

```
# Get first six rows of just Sex and Dep.Var
# columns of td
td %>%
  select(Sex, Dep.Var) %>%
  head()
```

```
Sex Dep.Var
1  F Realized
2  F Deletion
3  F Deletion
4  F Deletion
5  M Realized
6  M Deletion
```

```
# Get last six rows of just Sex and Dep.Var
# columns of td
td %>%
  select(Sex, Dep.Var) %>%
  tail()
```

```
Sex Dep.Var
1184 F Realized
1185 F Realized
1186 F Realized
1187 M Realized
1188 M Deletion
1189 M Realized
```

Above we use the `select()` function to choose just the `Sex` and `Dep.Var` columns and run the `head()` and `tail()` functions in order to see the first and last six values for both in the data. We do this just for comparisons sake. Now, lets use the position functions an compare them to our results.

```
# Get first, last, second, and second to last
# value of Dep.Var by Sex
td %>%
```

³https://en.wikipedia.org/wiki/Apparent-time_hypothesis


```
group_by(Sex) %>%
  summarize(First = first(Dep.Var), Last = last(Dep.Var),
            Second = nth(Dep.Var, 2), Second.Last = nth(Dep.Var,
-2))
```

```
# A tibble: 2 x 5
  Sex   First    Last    Second Second.Last
<fct> <fct>    <fct>    <fct>    <fct>
1 F     Realized Realized Deletion Realized
2 M     Realized Realized Deletion Deletion
```

Compare the male values with those from the `head()` and `tail()` functions above. The first (row 5) is **Realized**, the last (row 1198) is **Realized**. The second (row 6) is **Deletion**, and the second to last (row 1188) is also **Deletion**.

Count functions with `summarize()`

We've already looked at `n()` above, but there is also the `n_distinct()` function, which reports the number of distinct values. We can use this, for example, to find the number of speakers in each social category. To do this using base R filtering is a lot more complicated to code (so much so its not even worth doing). One example is shown below. It would need to be repeated for every combination of sex, education, and age group.

```
# Example using base R filtering, finding the
# number of unique speakers who are female,
# educated, and middle aged

n_distinct(td$Speaker[td$Sex == "F" & td$Education ==
  "Educated" & td$Age.Group == "Middle"])
```

```
[1] 12
```

```
# Much easier way to find number of unique
# speakers for every combination of Sex,
# Education, and Age. Group

td %>%
  group_by(Sex, Education, Age.Group) %>%
  summarize(Speaker.Count = n_distinct(Speaker)) %>%
  print(n = Inf)
```

```
# A tibble: 12 x 4
# Groups:   Sex, Education [6]
  Sex   Education    Age.Group Speaker.Count
<fct> <fct>        <fct>         <int>
1 F     Educated     Old             1
2 F     Educated     Middle          12
3 F     Educated     Young           3
4 F     Not Educated Old             6
5 F     Not Educated Middle          1
6 F     Student     Young          11
7 M     Educated     Middle           3
8 M     Educated     Young           6
9 M     Not Educated Old             5
```

10 M	Not Educated Middle	7
11 M	Not Educated Young	3
12 M	Student Young	8

You'll notice that there is no value for older educated males. This is because there are no speakers in the data from this group.

Logical functions

The two logical functions only work on data that is logical (i.e., is **TRUE** or **FALSE**). **any()** returns the answer to the question "Are any values **TRUE**?" and **all()** returns the answer to the question "Are all values **TRUE**?". There are no logical values in the **td** data set, so let's make some as an example.

```
# Create a new column in which all values are
# FALSE
td$Logical.Test <- FALSE
# Modify the new column so for any tokens from
# young female speakers are coded as TRUE instead
# of FALSE
td$Logical.Test[td$Sex == "F" & td$Age.Group == "Young"] <- TRUE

# Get logical value (TRUE or FALSE) of whether
# any tokens and all tokens of Logical.Test are
# TRUE, by Sex
td %>%
  group_by(Sex) %>%
  summarize(Any.True = any(Logical.Test), All.True = all(Logical.Test))

# A tibble: 2 x 3
  Sex   Any.True All.True
<fct> <lgl>    <lgl>
1 F     TRUE     FALSE
2 M     FALSE     FALSE
```

Above we created a logical column in which only tokens from young females are set to **TRUE**. The **any()** function returns **TRUE** for **F** but not for **M** because there is at least one **TRUE** value in the female data. Conversely, the **all()** function returns **FALSE** for **F** because not all of the female values are **TRUE**.

Proportions

Finding out the proportion of a variant is just like finding out the number of tokens. Using the base *R* methods, you simply wrap the **table()** function in a **prop.table()** function.

```
# Proportion of each level of Dep.Var
prop.table(table(td$Dep.Var))
```

```
Deletion Realized
0.325    0.675
```

Usually proportions are expressed as hundredths. To force *R* to express numbers in hundredths, you can use the **options()** function to set the number of significant digits displayed to two.

```
# Display values rounded to nearest hundredth.
options(digits = 2)
```

```
# Proportion of each level of Dep.Var
prop.table(table(td$Dep.Var))
```

```
Deletion Realized
0.32      0.68
```

In the example above there is only one dimension: `Dep.Var`. The `prop.table()` outer function takes the `table()` inner function and divides the number of tokens in each cell by some total (e.g. denominator). The default denominator is the total number of tokens in the whole table. Because, in the example above, the total number of tokens in the one dimension table is the same as the total number of `Dep.Var` tokens, you don't need to specify anything further. In the example below, however, there are two dimensions: `Dep.Var` and `Age.Group`. If you do not specify which total to use as a denominator, the proportions expressed use the total number of tokens in the table as the denominator.⁴ If you want to know the percentage of deletion tokens that come from `Young`, `Middle` and `Old` speakers, you set `margin = 1`, meaning that you want the total (e.g., denominator) to be the sum of the tokens for the first variable in the function, (e.g., rows total). If instead you want to know the percentage of `Young` tokens (or `Middle` tokens, or `Old` tokens) that are `Deletion`, and the percentage that are `Realized`, you set `margin = 2`, or rather set the denominator to the sum of the second factor group in the function (e.g., column total). This follows R's global pattern of rows, columns, page 1, page 2, etc. You can verify this by adding up the proportions in each table below. In the first table all of the proportions add up to 1. In the second table, on the other hand, the proportions add up to 1 going across the rows. In the third table they add up to 1 going down the columns.

```
# Proportion of each level of Dep.Var and
# Age.Group (all values sum to 1)
prop.table(table(td$Dep.Var, td$Age.Group))
```

```
      Old Middle Young
Deletion 0.056  0.105 0.163
Realized 0.113  0.198 0.365
```

```
# Proportion of each level of Age.Group for each
# level of Dep.Var (each row sums to 1)
prop.table(table(td$Dep.Var, td$Age.Group), margin = 1)
```

```
      Old Middle Young
Deletion 0.17   0.32  0.50
Realized 0.17   0.29  0.54
```

```
# Proportion of each level of Dep.Var for each
# level of Age.Group (each column sums to 1)
prop.table(table(td$Dep.Var, td$Age.Group), margin = 2)
```

```
      Old Middle Young
Deletion 0.33   0.35  0.31
Realized 0.67   0.65  0.69
```

In order to achieve the three-dimension cross-tabs you get from *Goldvarb*, with one dependent variable and two independent variables, you must set up the `prop.table(table())` function with your variables in the following order: *independent variable 1*, *independent variable 2*, *dependent variable*. You must also specify a particular `margin`, e.g., denominator. In a *Goldvarb*-style cross-tab each cell is the number of tokens for one level of the dependent variable (e.g., the application or non-application value) divided by the total

⁴You'll notice that the values in this table are expressed in thousandths instead of hundredths. This is because the proportion for `Deletion` and `Old` tokens requires three decimal places to have two meaningful digits.

number of tokens for that cell. In an R proportion table the total number of tokens per cell is the number of tokens for the value of the row and the column at the same time — not the row total, or the column total. To specify that you want the denominator to be the cell total you set `margin = c(1,2)`, where the `c()` concatenating function specifies both row (1) and column (2). The result is a separate page for proportions of each level of `Dep.Var`. The proportions for the corresponding cells in each page add up to 1.

```
# Proportion of each level of Dep.Var for each
# level of Age.Group and Sex (all corresponding
# cells sum to 1)
prop.table(table(td$Age.Group, td$Sex, td$Dep.Var),
  margin = c(1, 2))
```

, , = Deletion

	F	M
Old	0.29	0.47
Middle	0.31	0.43
Young	0.27	0.34

, , = Realized

	F	M
Old	0.71	0.53
Middle	0.69	0.57
Young	0.73	0.66

You can keep adding factor groups to your proportion table, but you must do two things. You must keep the dependent variable, `Dep.Var`, as the rightmost variable in the function, and you must include all the other variables in the margin specification. For example, below you add `Education` as the third variable, and add 3 to the margin specification. There will be a separate page for each combination of the levels of `Education` and `Dep.Var`.

```
# Proportion of each level of Dep.Var for each
# level of Age.Group, Sex and Education
prop.table(table(td$Age.Group, td$Sex, td$Education,
  td$Dep.Var), margin = c(1, 2, 3))
```

, , = Educated, = Deletion

	F	M
Old	0.062	
Middle	0.308	0.348
Young	0.278	0.384

, , = Not Educated, = Deletion

	F	M
Old	0.347	0.471
Middle	0.294	0.474
Young		0.436

```
, , = Student, = Deletion
```

```
      F      M
Old
Middle
Young 0.261 0.282
```

```
, , = Educated, = Realized
```

```
      F      M
Old   0.938
Middle 0.692 0.652
Young 0.722 0.616
```

```
, , = Not Educated, = Realized
```

```
      F      M
Old   0.653 0.529
Middle 0.706 0.526
Young           0.564
```

```
, , = Student, = Realized
```

```
      F      M
Old
Middle
Young 0.739 0.718
```

Again, you can make these larger tables easier to read by flattening the pages using `fTable()`. Here the **NaN** means there is no data in the cell.

```
# Proportion of each level of Dep.Var for each
# level of Age.Group, Sex and Education,
# presented as a flattened table. Here the `NaN`
# just means there is no data in the cell.
library(vcd)
fTable(prop.table(table(td$Age.Group, td$Sex, td$Education,
  td$Dep.Var), margin = c(1, 2, 3)))
```

		Deletion	Realized
Old	F Educated	0.062	0.938
	Not Educated	0.347	0.653
	Student	NaN	NaN
Middle	M Educated	NaN	NaN
	Not Educated	0.471	0.529
	Student	NaN	NaN
Young	F Educated	0.308	0.692
	Not Educated	0.294	0.706

	Student	NaN	NaN
M	Educated	0.348	0.652
	Not Educated	0.474	0.526
	Student	NaN	NaN
Young F	Educated	0.278	0.722
	Not Educated	NaN	NaN
	Student	0.261	0.739
M	Educated	0.384	0.616
	Not Educated	0.436	0.564
	Student	0.282	0.718

There are a number of functions specifically designed to create cross-tables that are somewhat easier to use, but can be somewhat less flexible. Generally, they are most useful for one independent variable and one dependent variable. I tend to use the `CrossTable()` function from the `gmodels` package frequently.

```
# Load gmodels
library(gmodels)

# Generate cross tab of Sex and Dep.Var in which
# the row proportions are displayed, but table
# proportions, column proportions, and
# contribution to chi-square are suppressed, with
# 0 decimal values displayed, and missing
# combinations included.
CrossTable(td$Sex, td$Dep.Var, prop.r = TRUE, prop.c = FALSE,
  prop.t = FALSE, prop.chisq = FALSE, format = "SPSS",
  digits = 0, missing.include = TRUE)
```

Cell Contents

Count
Row Percent

Total Observations in Table: 1189

	td\$Sex	td\$Dep.Var	Deletion	Realized	Row Total
	F		188	471	659
			29%	71%	55%
	M		198	332	530
			37%	63%	45%
Column Total			386	803	1189

For the `CrossTable()` function you can set the denominator to row total with the option `prop.r=TRUE`. If instead you wanted to the proportion by column, you set `prop.c = TRUE`, and if you want the proportion across the entire table you can set `prop.t = TRUE`. You can actually set all of these to `TRUE` to get all three. There are other values that can be generated, including values for calculating chi-square (see the

`CrossTable()` documentation here⁵). The above code includes the minimal number of options needed to generate the type of cross-table we generally want.

To produce proportions using the `tidy` method, we combine the `group_by()` and `summarize()` functions with the `mutate()` discussed in an earlier section⁶.

```
# Generate tibble of combination of Sex and
# Dep.Var with token counts and proportion of
# each level of Dep.Var by Sex
td %>%
  group_by(Sex, Dep.Var) %>%
  summarize(Count = n()) %>%
  mutate(Prop = Count/sum(Count))

# A tibble: 4 x 4
# Groups:   Sex [2]
  Sex  Dep.Var  Count  Prop
<fct> <fct>    <int> <dbl>
1 F      Deletion    188 0.285
2 F      Realized    471 0.715
3 M      Deletion    198 0.374
4 M      Realized    332 0.626
```

After grouping the data by `Sex` and `Dep.Var`, we create a new column `Count` with values equal to the number of tokens for the particular combination, then we create a new column using `mutate()` and a math equation to generate proportions. It is important here that your dependent variable `Dep.Var` is the last grouping variable. If we change the order, instead of generating the proportion of `Realized` and `Deletion` tokens, it will instead return the percentage of `Realized` tokens that are `M` and the percentage that are `F`, which is the incorrect denominator for our purposes.

```
# Generate tibble of combination of Dep.Var and
# Sex with token counts and proportion of each
# level of Sex by Dep.Var
td %>%
  group_by(Dep.Var, Sex) %>%
  summarize(Count = n()) %>%
  mutate(Prop = Count/sum(Count))

# A tibble: 4 x 4
# Groups:   Dep.Var [2]
  Dep.Var Sex  Count  Prop
<fct>    <fct> <int> <dbl>
1 Deletion F      188 0.487
2 Deletion M      198 0.513
3 Realized F      471 0.587
4 Realized M      332 0.413
```

Unlike the `CrossTable()` function, we can include multiple independent variables. To include every combination (including those for which there are no tokens), we can add `.drop = FALSE` to the `group_by()` function.

⁵<https://www.rdocumentation.org/packages/gmodels/versions/2.18.1.1/topics/CrossTable>

⁶https://lingmethodshub.github.io/content/R/lvc_r/040_lvc.html

```
# Generate tibble of combination of Sex,
# Education, Age.Group, and Dep.Var with all
# combinations included, with token counts and
# proportion of each level of Dep.Var by each
# combination of other variables
td %>%
  group_by(Sex, Education, Age.Group, Dep.Var, .drop = FALSE) %>%
  summarize(Count = n()) %>%
  mutate(Prop = Count/sum(Count)) %>%
  print(n = Inf)
```

```
# A tibble: 36 x 6
```

```
# Groups:   Sex, Education, Age.Group [18]
```

	Sex	Education	Age.Group	Dep.Var	Count	Prop
	<fct>	<fct>	<fct>	<fct>	<int>	<dbl>
1	F	Educated	Old	Deletion	2	0.0625
2	F	Educated	Old	Realized	30	0.938
3	F	Educated	Middle	Deletion	68	0.308
4	F	Educated	Middle	Realized	153	0.692
5	F	Educated	Young	Deletion	20	0.278
6	F	Educated	Young	Realized	52	0.722
7	F	Not Educated	Old	Deletion	41	0.347
8	F	Not Educated	Old	Realized	77	0.653
9	F	Not Educated	Middle	Deletion	5	0.294
10	F	Not Educated	Middle	Realized	12	0.706
11	F	Not Educated	Young	Deletion	0	NaN
12	F	Not Educated	Young	Realized	0	NaN
13	F	Student	Old	Deletion	0	NaN
14	F	Student	Old	Realized	0	NaN
15	F	Student	Middle	Deletion	0	NaN
16	F	Student	Middle	Realized	0	NaN
17	F	Student	Young	Deletion	52	0.261
18	F	Student	Young	Realized	147	0.739
19	M	Educated	Old	Deletion	0	NaN
20	M	Educated	Old	Realized	0	NaN
21	M	Educated	Middle	Deletion	16	0.348
22	M	Educated	Middle	Realized	30	0.652
23	M	Educated	Young	Deletion	48	0.384
24	M	Educated	Young	Realized	77	0.616
25	M	Not Educated	Old	Deletion	24	0.471
26	M	Not Educated	Old	Realized	27	0.529
27	M	Not Educated	Middle	Deletion	36	0.474
28	M	Not Educated	Middle	Realized	40	0.526
29	M	Not Educated	Young	Deletion	24	0.436
30	M	Not Educated	Young	Realized	31	0.564
31	M	Student	Old	Deletion	0	NaN
32	M	Student	Old	Realized	0	NaN
33	M	Student	Middle	Deletion	0	NaN
34	M	Student	Middle	Realized	0	NaN
35	M	Student	Young	Deletion	50	0.282
36	M	Student	Young	Realized	127	0.718

Notice that for the missing combinations the `count()` is 0, and the percentage is `NaN`, which stands for “not a number”, the result of trying to divide 0 by something. `NaN` is similar to `NA`, but `NA` stands for “no data”,

and is used for empty cells.

```
# Assign the tibble generated in the previous
# code to an object called results
results <- td %>%
  group_by(Sex, Education, Age.Group, Dep.Var, .drop = FALSE) %>%
  summarize(Count = n()) %>%
  mutate(Prop = Count/sum(Count))

# Recode all NaN in results to 0
results$Prop[is.nan(results$Prop)] <- 0
# Print results
print(results, n = Inf)
```

A tibble: 36 x 6

Groups: Sex, Education, Age.Group [18]

	Sex	Education	Age.Group	Dep.Var	Count	Prop
	<fct>	<fct>	<fct>	<fct>	<int>	<dbl>
1	F	Educated	Old	Deletion	2	0.0625
2	F	Educated	Old	Realized	30	0.938
3	F	Educated	Middle	Deletion	68	0.308
4	F	Educated	Middle	Realized	153	0.692
5	F	Educated	Young	Deletion	20	0.278
6	F	Educated	Young	Realized	52	0.722
7	F	Not Educated	Old	Deletion	41	0.347
8	F	Not Educated	Old	Realized	77	0.653
9	F	Not Educated	Middle	Deletion	5	0.294
10	F	Not Educated	Middle	Realized	12	0.706
11	F	Not Educated	Young	Deletion	0	0
12	F	Not Educated	Young	Realized	0	0
13	F	Student	Old	Deletion	0	0
14	F	Student	Old	Realized	0	0
15	F	Student	Middle	Deletion	0	0
16	F	Student	Middle	Realized	0	0
17	F	Student	Young	Deletion	52	0.261
18	F	Student	Young	Realized	147	0.739
19	M	Educated	Old	Deletion	0	0
20	M	Educated	Old	Realized	0	0
21	M	Educated	Middle	Deletion	16	0.348
22	M	Educated	Middle	Realized	30	0.652
23	M	Educated	Young	Deletion	48	0.384
24	M	Educated	Young	Realized	77	0.616
25	M	Not Educated	Old	Deletion	24	0.471
26	M	Not Educated	Old	Realized	27	0.529
27	M	Not Educated	Middle	Deletion	36	0.474
28	M	Not Educated	Middle	Realized	40	0.526
29	M	Not Educated	Young	Deletion	24	0.436
30	M	Not Educated	Young	Realized	31	0.564
31	M	Student	Old	Deletion	0	0
32	M	Student	Old	Realized	0	0
33	M	Student	Middle	Deletion	0	0
34	M	Student	Middle	Realized	0	0
35	M	Student	Young	Deletion	50	0.282
36	M	Student	Young	Realized	127	0.718

The easiest way to convert **NaN** (or **Na**) to 0 is to assign the above to a variable, then replace **NaN** with 0 using the function `is.nan()`. If there were **NA** values, you can do the same thing as above, but replace `is.nan()` with `is.na()`

When we report proportions in sociolinguistics manuscripts, we often only report the proportion of one level of the dependent variable (called the application value). To only display one of the two levels of **Dep.Var** — for instance, if we just want to show the rates of **Deletion**, which we might decide is our application value — we can use the `subset()` function.

```
# Create the results object, but subsetted to
# include only Deletion tokens
results <- td %>%
  group_by(Sex, Education, Age.Group, Dep.Var, .drop = FALSE) %>%
  summarize(Count = n()) %>%
  mutate(Prop = Count/sum(Count)) %>%
  subset(Dep.Var == "Deletion")

# Recode NaN to 0
results$Prop[is.nan(results$Prop)] <- 0
# Print results
print(results, n = Inf)
```

A tibble: 18 x 6

Groups: Sex, Education, Age.Group [18]

	Sex	Education	Age.Group	Dep.Var	Count	Prop
	<fct>	<fct>	<fct>	<fct>	<int>	<dbl>
1	F	Educated	Old	Deletion	2	0.0625
2	F	Educated	Middle	Deletion	68	0.308
3	F	Educated	Young	Deletion	20	0.278
4	F	Not Educated	Old	Deletion	41	0.347
5	F	Not Educated	Middle	Deletion	5	0.294
6	F	Not Educated	Young	Deletion	0	0
7	F	Student	Old	Deletion	0	0
8	F	Student	Middle	Deletion	0	0
9	F	Student	Young	Deletion	52	0.261
10	M	Educated	Old	Deletion	0	0
11	M	Educated	Middle	Deletion	16	0.348
12	M	Educated	Young	Deletion	48	0.384
13	M	Not Educated	Old	Deletion	24	0.471
14	M	Not Educated	Middle	Deletion	36	0.474
15	M	Not Educated	Young	Deletion	24	0.436
16	M	Student	Old	Deletion	0	0
17	M	Student	Middle	Deletion	0	0
18	M	Student	Young	Deletion	50	0.282

Finally, if we also want to add the total number of tokens per category (something we usually report alongside the application value) we can add another column using `mutate()`. Also, if we want the percentage instead of proportion, we can add `100 *` to the proportion equation (as percentage is proportion \times 100)

```
# Generate results object with percentage instead
# of proportion and a column with total tokens
# per combination.
results <- td %>%
  group_by(Sex, Education, Age.Group, Dep.Var, .drop = FALSE) %>%
```

```

summarize(Count = n()) %>%
mutate(Percentage = 100 * Count/sum(Count), Total.N = sum(Count)) %>%
subset(Dep.Var == "Deletion")

# Recode NaN to 0
results$Percentage[is.nan(results$Percentage)] <- 0
# Print results
print(results, n = Inf)

```

A tibble: 18 x 7

Groups: Sex, Education, Age.Group [18]

	Sex	Education	Age.Group	Dep.Var	Count	Percentage	Total.N
	<fct>	<fct>	<fct>	<fct>	<int>	<dbl>	<int>
1	F	Educated	Old	Deletion	2	6.25	32
2	F	Educated	Middle	Deletion	68	30.8	221
3	F	Educated	Young	Deletion	20	27.8	72
4	F	Not Educated	Old	Deletion	41	34.7	118
5	F	Not Educated	Middle	Deletion	5	29.4	17
6	F	Not Educated	Young	Deletion	0	0	0
7	F	Student	Old	Deletion	0	0	0
8	F	Student	Middle	Deletion	0	0	0
9	F	Student	Young	Deletion	52	26.1	199
10	M	Educated	Old	Deletion	0	0	0
11	M	Educated	Middle	Deletion	16	34.8	46
12	M	Educated	Young	Deletion	48	38.4	125
13	M	Not Educated	Old	Deletion	24	47.1	51
14	M	Not Educated	Middle	Deletion	36	47.4	76
15	M	Not Educated	Young	Deletion	24	43.6	55
16	M	Student	Old	Deletion	0	0	0
17	M	Student	Middle	Deletion	0	0	0
18	M	Student	Young	Deletion	50	28.2	177

The above results show that there are 32 tokens from old, educated females, 2 of which (or 6.25%) are **Deletion**.