# STAT 210
# Applied Statistics and Data Analysis
# Tables in R

Joaquin Ortega

Fall 2020

# Reading Files

# Reading Files

We start by looking at ways of retrieving data stored in a .txt, .csv, or .xlsx file.

For a tutorial on this, go to:
https://www.datacamp.com/community/tutorials/

and search for **Reading and Importing Excel Files into R**.

We are going to work with the file `Human_data` which has been saved from the original `.xlsx` version into `.txt` and `.csv` formats to be used as an example.

# Reading .txt files

Use the function read.table

```r
humans1 <- read.table('data/Human_data.txt',
                      header = TRUE)
```

```r
dim(humans1)
```

```
## [1] 500  10
```

# Reading .txt files

The `str` command in R shows the structure of the file:

```r
str(humans1, vec.len = 2)
```

```
## 'data.frame':    500 obs. of  10 variables:
##  $ Index         : int  1 2 3 4 5 ...
##  $ Gender        : Factor w/ 2 levels "F","M": 2 1 2 1 1 ...
##  $ age           : int  22 33 46 24 37 ...
##  $ Ocupation     : Factor w/ 3 levels "Nothing","student",..: 1 1 3
##  $ Head_size     : num  34.4 28 27 24.8 30.1 ...
##  $ Height_cm     : num  206 163 ...
##  $ Weight_kg     : num  105.3 71.3 ...
##  $ Salary        : int  0 0 19268 2034 14829 ...
##  $ blood_type    : int  4 4 4 3 2 ...
##  $ Sugar_in_blood: num  95.2 83.5 ...
```

# Reading .txt files

`head` shows the first few (`n=6` by default) values in the file:

```r
head(humans1, n=4)
```

```
##   Index Gender age Ocupation Head_size Height_cm Weight_kg
## 1     1      M  22   Nothing      34.4     205.5     105.3
## 2     2      F  33   Nothing      28.0     162.8      71.3
## 3     3      M  46      Work      27.0     162.4      94.7
## 4     4      F  24   student      24.8     156.0      56.0
##   Salary blood_type Sugar_in_blood
## 1      0          4           95.2
## 2      0          4           83.5
## 3  19268          4           92.7
## 4   2034          3           95.8
```

# Reading .csv files

Use the function `read.csv` if the values are separated by commas (,) or `read.csv2` if they are separated by semicolons (;)

```r
humans2 <- read.csv('data/Human_data.csv')
str(humans2, vec.len = 2)
```

```
## 'data.frame':    500 obs. of  10 variables:
##  $ Index         : int  1 2 3 4 5 ...
##  $ Gender        : Factor w/ 2 levels "F","M": 2 1 2 1 1 ...
##  $ age           : int  22 33 46 24 37 ...
##  $ Ocupation     : Factor w/ 3 levels "Nothing","student",..: 1 1 3
##  $ Head_size     : num  34.4 28 27 24.8 30.1 ...
##  $ Height_cm     : num  206 163 ...
##  $ Weight_kg     : num  105.3 71.3 ...
##  $ Salary        : int  0 0 19268 2034 14829 ...
##  $ blood_type    : int  4 4 4 3 2 ...
##  $ Sugar_in_blood: num  95.2 83.5 ...
```

# Reading .csv files

```r
head(humans2)
```

```
##   Index Gender age Ocupation Head_size Height_cm Weight_kg
## 1     1      M  22   Nothing      34.4     205.5     105.3
## 2     2      F  33   Nothing      28.0     162.8      71.3
## 3     3      M  46      Work      27.0     162.4      94.7
## 4     4      F  24   student      24.8     156.0      56.0
## 5     5      F  37      Work      30.1     172.7     103.3
## 6     6      F  31      Work      26.6     157.7      47.0
##   Salary blood_type Sugar_in_blood
## 1      0          4           95.2
## 2      0          4           83.5
## 3  19268          4           92.7
## 4   2034          3           95.8
## 5  14829          2          114.1
## 6  10586          3           95.1
```

# Reading `.xlsx` files

Reading `.xlsx` is more complicated and requires loading a package. There are several packages for doing this.

I was not able to make the packages `xlxs` and `XLConnect` work on my computer. There are problems with the Java installation that I was not able to solve.

I could load `gdata`, and it worked, but beware, you may need to know where your PERL executable is. In my case (fortunately!) it worked without requiring this path.

```
library(gdata)
humans <- read.xls('data/Human_data.xlsx')
```

# Reading .xlsx files

```r
str(humans, vec.len = 2)
```

```
## 'data.frame':    500 obs. of  10 variables:
##  $ Index          : int  1 2 3 4 5 ...
##  $ Gender         : Factor w/ 2 levels "F","M": 2 1 2 1 1 ...
##  $ age            : int  22 33 46 24 37 ...
##  $ Ocupation      : Factor w/ 3 levels "Nothing","student",..: 1 1 3
##  $ Head_size      : num  34.4 28 27 24.8 30.1 ...
##  $ Height_cm      : num  206 163 ...
##  $ Weight_kg      : num  105.3 71.3 ...
##  $ Salary         : int  0 0 19268 2034 14829 ...
##  $ blood_type     : int  4 4 4 3 2 ...
##  $ Sugar_in_blood: num  95.2 83.5 ...
```

# Reading .xlsx files

```r
head(humans)
```

```
##   Index Gender age Ocupation Head_size Height_cm Weight_kg
## 1     1      M  22   Nothing      34.4     205.5     105.3
## 2     2      F  33   Nothing      28.0     162.8      71.3
## 3     3      M  46      Work      27.0     162.4      94.7
## 4     4      F  24   student      24.8     156.0      56.0
## 5     5      F  37      Work      30.1     172.7     103.3
## 6     6      F  31      Work      26.6     157.7      47.0
##   Salary blood_type Sugar_in_blood
## 1      0          4           95.2
## 2      0          4           83.5
## 3  19268          4           92.7
## 4   2034          3           95.8
## 5  14829          2          114.1
## 6  10586          3           95.1
```

# Selecting data

# Selecting data

For listing all the names of the columns of the table use `names()`

```r
names(humans)
```

```
##  [1] "Index"            "Gender"           "age"
##  [4] "Ocupation"        "Head_size"        "Height_cm"
##  [7] "Weight_kg"        "Salary"           "blood_type"
## [10] "Sugar_in_blood"
```

We can select a variable (column) of the table using the string `humans$name` where `name` stands for the name of the variable:

```r
humans$Gender[1:5]
```

```
## [1] M F M F F
## Levels: F M
```

# Selecting data

Select only female humans.

Note the use of the “,” to indicate select all the columns:

```
fem_h<-humans[humans$Gender=='F', ]
str(fem_h, vec.len = 2)
```

```
## 'data.frame':    272 obs. of  10 variables:
##  $ Index          : int  2 4 5 6 7 ...
##  $ Gender         : Factor w/ 2 levels "F","M": 1 1 1 1 1 ...
##  $ age            : int  33 24 37 31 38 ...
##  $ Ocupation      : Factor w/ 3 levels "Nothing","student",..: 1 2 3
##  $ Head_size      : num  28 24.8 30.1 26.6 25.6 ...
##  $ Height_cm      : num  163 156 ...
##  $ Weight_kg      : num  71.3 56 ...
##  $ Salary         : int  0 2034 14829 10586 11272 ...
##  $ blood_type     : int  4 3 2 3 4 ...
##  $ Sugar_in_blood: num  83.5 95.8 ...
```

# Selecting data

Another method

```
fem_h<-humans[humans[,2]=="F" , ]
```

# Selecting data

A third option is to use the `subset()` function which is very useful for extracting data that satisfies several conditions:

```
fem_h <- subset(humans,Gender=='F')
str(fem_h, vec.len = 2)
```

```
## 'data.frame':    272 obs. of  10 variables:
##  $ Index         : int  2 4 5 6 7 ...
##  $ Gender        : Factor w/ 2 levels "F","M": 1 1 1 1 1 ...
##  $ age           : int  33 24 37 31 38 ...
##  $ Ocupation     : Factor w/ 3 levels "Nothing","student",..: 1 2 3
##  $ Head_size     : num  28 24.8 30.1 26.6 25.6 ...
##  $ Height_cm     : num  163 156 ...
##  $ Weight_kg     : num  71.3 56 ...
##  $ Salary        : int  0 2034 14829 10586 11272 ...
##  $ blood_type    : int  4 3 2 3 4 ...
##  $ Sugar_in_blood: num  83.5 95.8 ...
```

# Selecting data

We now select only females over 35 years and columns 3 to 7:

```
fem_h_AGE<-humans[humans$Gender=='F' & humans$age > 35,3:7]
str(fem_h_AGE, vec.len = 2)
```

```
## 'data.frame':    162 obs. of  5 variables:
##  $ age      : int  37 38 38 49 49 ...
##  $ Ocupation: Factor w/ 3 levels "Nothing","student",..: 3 3 3 1 3 .
##  $ Head_size: num  30.1 25.6 25.6 30.1 26.5 ...
##  $ Height_cm: num  173 152 ...
##  $ Weight_kg: num  103.3 46.5 ...
```

Using subset this can be done as follows

```
fem_h_AGE<- subset(humans, Gender=='F' & age > 35, select = 3:7)
```

# Factors

Observe that the blood type is coded according to the following correspondence

| Code | Blood type |
|---|---|
| 1 | O |
| 2 | A |
| 3 | B |
| 4 | AB |

We want to write the blood type according to the code in the file.

# Factors

The function `factor()` takes a numeric variable and converts it into into categorical data:

```
blood_factor<-factor(humans$blood_type,
                     labels=c("O", "A", "B", "AB"))
head(blood_factor)
```

```
## [1] AB AB AB B  A  B
## Levels: O A B AB
```

# Tables

# Contingency Tables

Tabular summaries of data are a frequent starting point for statistical analysis.

A **contingency table** in Statistics is a table that displays the multivariate frequency distribution of two or more variables.

The entries in the cells of a two-way table are the frequency counts or relative frequencies, and the table is usually presented as a matrix.

Karl Pearson first used the name in 1904.

Before looking at the usual statistical techniques for analysis of contingency tables, let us review some of the available tools for producing them.

# Contingency Tables

## Classifying observations according to a numeric variable

Let's classify people according to their salary:

- from 0 until 5000 is a low salary
- from 5001 until 12000 is medium and
- greater than 12000 is a high salary.

The function `cut` performs this work.

- ‘breaks‘ defines the break points of each level or class
- ‘labels‘ specifies the value to use when an observation falls in one class
- ‘right‘ specifies the type of interval: ‘right=F‘ is for $[a, b)$ and ‘right=T‘ is for $(a, b]$.

# Tables

```
Sal_class <-cut(humans$Salary,
      breaks=c(min(humans$Salary),5000,
               12000, (max(humans$Salary)+1)),
              labels=c("low","med","high"), right=F )
str(Sal_class)
```

```
##  Factor w/ 3 levels "low","med","high": 1 1 3 1 3 2 2 3 1 3 ...
```

```
head(Sal_class)
```

```
## [1] low  low  high low  high med
## Levels: low med high
```

The cut function produces a factor with three levels.

# Tables

We can now use the `table()` function to compute frequencies:

```
(freq_gender_sal<- table(Sal_class,humans$Gender))
```

```
##
## Sal_class   F   M
##       low  107  77
##       med   69  58
##       high  96  93
```

The table has the frequency count for each of the six possible categories, which are a combination of two levels of `Gender` and three levels of income.

# Tables

## Table of age and ocupation

```r
(f_ocup_age<-table(humans$Ocupation,humans$age))
```

```
##
##            18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
##   Nothing   3  2  3  2  3  3  2  7  5  0  2  2  1  2  1  3
##   student  10 13 12  8 14  6 12  0  0  0  0  0  0  0  0  0
##   Work      0  0  0  0  0  0  0  0  9  7  8 10  7 12  4  4
##
##            34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
##   Nothing   2  4  6  4  5  5  2  3  0  0  1  4  2  1  1  4
##   student   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##   Work     12 12 10 10 12  9 10  8 14 11 12 11 10  6  7 13
##
##            50 51 52 53 54 55 56 57 58 59 60
##   Nothing   1  5  5  3  6  0  0  2  0  1  2
##   student   0  0  0  0  0  0  0  0  0  0  0
##   Work     14  6 15  8  5 10  3  8  5  6  7
```

# Tables

Can reduce the size ‘cutting’ age into decades:

```
age_class <- cut(humans$age,
                 breaks = c(10,20,30,40,50,60,70),
                 right = FALSE)
(f_ocup_age2<-table(age_class,humans$Ocupation))
```

```
##
## age_class Nothing student Work
##   [10,20)       5      23    0
##   [20,30)      29      52   34
##   [30,40)      33       0   92
##   [40,50)      18       0  102
##   [50,60)      23       0   80
##   [60,70)       2       0    7
```

# Tables

We want to store the Sal_class vector we obtained as a new variable (column) in the data frame humans, and we will name this new variables Salary_class. One way to do this is to use the command

```
humans$Salary_class <- Sal_class
```

A second way is to use the function within to create the variable using the cut function and store it in the data frame. We do this next as an example of the use of the within function.

```
humans <- within(humans, Salary_class <-cut(humans$Salary,
              breaks=c(min(humans$Salary),5000,12000,
              (max(humans$Salary)+1)),
              labels=c("low","med","high"), right=F ))
```

# Tables

```r
str(humans, vec.len = 2)
```

```
## 'data.frame':	500 obs. of  11 variables:
##  $ Index          : int  1 2 3 4 5 ...
##  $ Gender         : Factor w/ 2 levels "F","M": 2 1 2 1 1 ...
##  $ age            : int  22 33 46 24 37 ...
##  $ Ocupation      : Factor w/ 3 levels "Nothing","student",..: 1 1 3
##  $ Head_size      : num  34.4 28 27 24.8 30.1 ...
##  $ Height_cm      : num  206 163 ...
##  $ Weight_kg      : num  105.3 71.3 ...
##  $ Salary         : int  0 0 19268 2034 14829 ...
##  $ blood_type     : int  4 4 4 3 2 ...
##  $ Sugar_in_blood: num  95.2 83.5 ...
##  $ Salary_class   : Factor w/ 3 levels "low","med","high": 1 1 3 1 3
```

Observe that a new factor variable has been created in the `humans` data frame with levels `low`, `med`, `high`.

# Tables

We can store the resulting table in a file that will go to the working directory unless we specify a different path in the command.

In this case, we will use the ‘csv’ format with the function `write.csv`, but there are other alternatives such as `write.table`.

The first argument of the function is the table we want to download, and the second is the name, including the path if we do not want to store it in the working directory.

```
write.csv(freq_gender_sal,file="results.csv")
```

# Summaries of tables

# Summaries of tables

Proportion with respect of the total sample

```
(freq_gender_sal_prop <- freq_gender_sal/sum(freq_gender_sal))
```

```
##
## Sal_class     F     M
##      low  0.214 0.154
##      med  0.138 0.116
##      high 0.192 0.186
```

# Summaries of tables

Another way of doing this is using the function `prop.table`:

```
prop.table(freq_gender_sal)
```

```
##
## Sal_class     F     M
##      low  0.214 0.154
##      med  0.138 0.116
##      high 0.192 0.186
```

# Summaries of tables

## Proportion of females and males

```
(prop_fem<-sum(freq_gender_sal[,1])/sum(freq_gender_sal))
```

```
## [1] 0.544
```

```
(prop_male<-1-prop_fem)
```

```
## [1] 0.456
```

# Summaries of tables

### Proportion of females and males

We can also use the combination of `table` and `prop.table` to obtain this

```
prop.table(table(humans$Gender))
```

```
##
##     F     M
## 0.544 0.456
```

# Summaries of tables

## Proportion of salary classes

```r
sum(freq_gender_sal[1,])/sum(freq_gender_sal)
sum(freq_gender_sal[2,])/sum(freq_gender_sal)
sum(freq_gender_sal[3,])/sum(freq_gender_sal)
```

```
## [1] 0.368
## [1] 0.254
## [1] 0.378
```

```r
prop.table(table(Sal_class))
```

```
## Sal_class
##   low   med  high
## 0.368 0.254 0.378
```

# Summaries of tables

**Proportions for females and males**

Suppose we want now to look at the proportion of each of the three income classes among the female population.

To obtain this, we need to divide the elements of the first column of

```
##
## Sal_class   F   M
##       low 107  77
##       med  69  58
##      high  96  93
```

by their sum.

Similarly, for the male population, the proportion requires dividing by the sum of the second column.

# Summaries of tables

We can obtain these sums with the function `apply`, which applies a given function to the rows or columns of an array, according to the value of the `MARGIN` option.

The syntax of this function is

```
apply(X, MARGIN, FUN, ...)
```

where `X` is an array, `MARGIN` is a number that indicates that the operation should be performed row-wise (1) or column-wise (2), and `FUN` is the function to be applied.

```
(sum.col <- apply(freq_gender_sal,2,sum))
```

```
##   F   M
## 272 228
```

# Summaries of tables

Another way to do this is to use the function `colSums`

```
colSums(freq_gender_sal)
```

```
##   F   M
## 272 228
```

There is also a function `rowSums` that computes row totals.

# Summaries of tables

We now need to divide the elements of each column by the corresponding sums to obtain the proportions for each of the two populations.

A naive (and wrong!) way to attempt to do this would be to write `freq_gender_sal/sum.com`.

The problem with this approach is that `R` stores matrices column-wise, and therefore the division is performed in this sense, recycling the vector `col.sum` as required.

# Summaries of tables

Let's see an example

```r
(mat1 <- matrix(c(1,2,3,4,5,6), ncol=2))
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```r
mat1/c(1,2)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    1    5
## [3,]    3    3
```

To obtain what we want, we need the function `sweep`.

# Summaries of tables

The syntax for this function is

```
sweep(x, MARGIN, STATS, FUN = '-')
```

`x` and `MARGIN` have the same meaning as before. `STATS` is the summary statistic that is to be swept out (the sum in our case), and `FUN` is the function to be used to carry out the sweeping, which by default is the difference.

```
sweep(mat1,2,c(1,2),'/')
```

```
##      [,1] [,2]
## [1,]    1  2.0
## [2,]    2  2.5
## [3,]    3  3.0
```

# Summaries of tables

Use this for our problem

```r
(freq_gender_sal1 <-sweep(freq_gender_sal,2,sum.col,'/'))
```

```
##
## Sal_class         F         M
##      low  0.3933824 0.3377193
##      med  0.2536765 0.2543860
##      high 0.3529412 0.4078947
```

Observe that the columns add up to 1.

# Summaries of tables

We see that around 35.3% of the female population has large incomes while the proportion for the male population is about 40.8%.

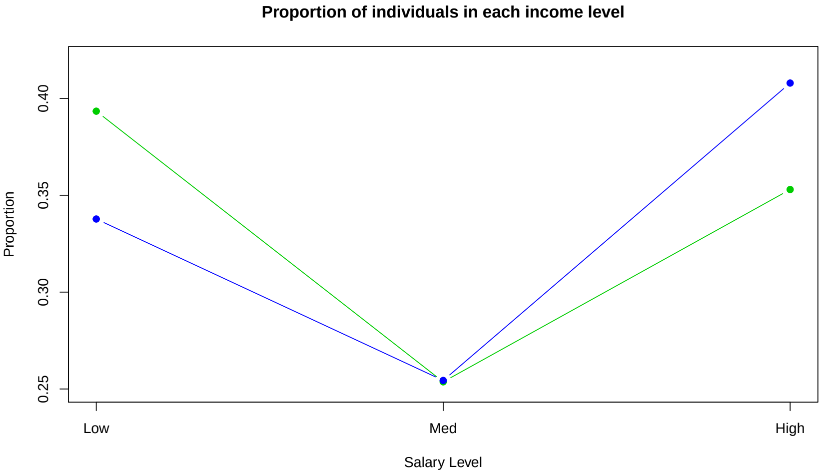
A simpler way to do this for tables is to use the function `prop.tables` again with the argument `margin` that determines which of the variables used to build the table should be used to calculate the proportions. In our case, `gender` is in the columns, so we set `margin` equal to 2:

```
prop.table(freq_gender_sal, margin = 2)
```

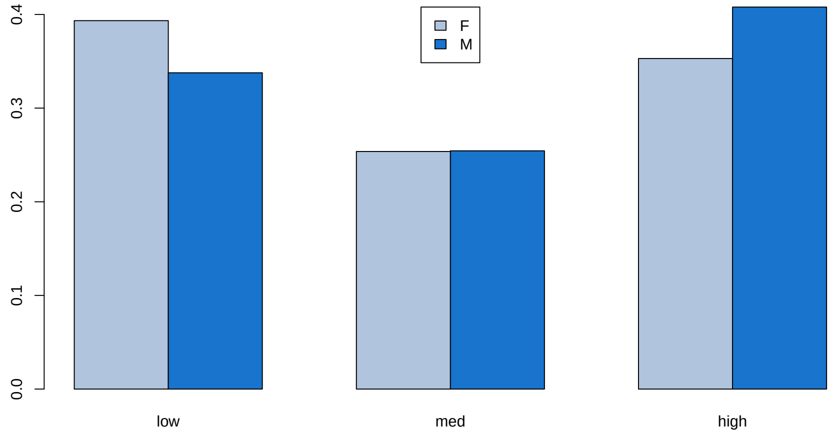
```
##
## Sal_class         F         M
##      low  0.3933824 0.3377193
##      med  0.2536765 0.2543860
##      high 0.3529412 0.4078947
```

# Graphical representations

# Graphical representations



# Graphical representations



# Mosaic plots

A more elegant way of graphing the contents of a contingency table is the mosaic plot, which gives an overview of the data and helps display possible relationships among variables.

These plots were initially proposed by Hartigan and Kleiner in 1981 and later expanded by M. Friendly (1994).

A mosaic plot represents the data in a contingency table using rectangles.

In the case of a $2 \times 2$ table, each variable is assigned to an axis.

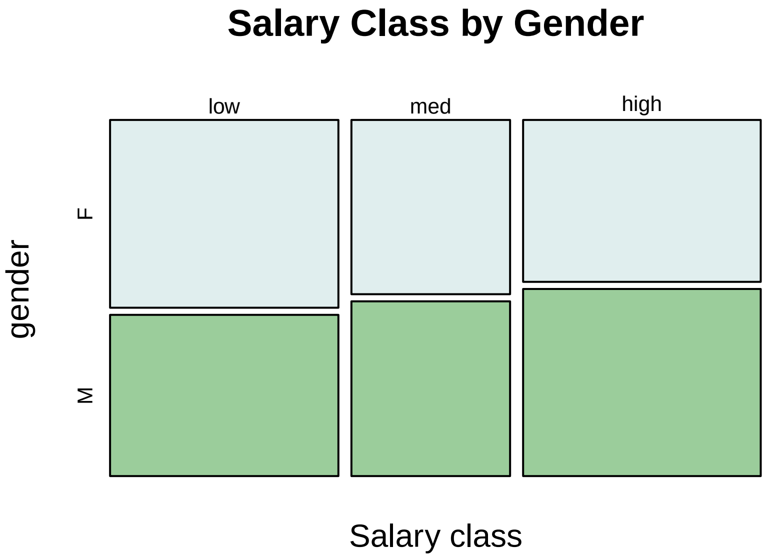
# Mosaic plots

The different categories are represented so that the length corresponding to each one is proportionate to their relative frequency in the table. This gives rectangles with sides proportional to the relative frequencies of each category and area proportional to the relative frequency of the table cell.

One way of graphing a mosaic plot is by using the function `mosaicplot()`. The main input to this function is the contingency table we want to plot.

# Mosaic plots

```
mosaicplot(freq_gender_sal1, xlab='Salary class', ylab='gender',
    main = 'Salary Class by Gender', col = c('azure2','darkseagreen3'))
```



# Mosaic plots

We can see in the graph how the proportion of males increses as we move from low to high income. The function can also be used with a formula instead of a contingency table. The command

```
mosaicplot(~ Sal_class + humans$Gender,
           xlab='Salary class', ylab='gender',
           main = 'Salary Class by Gender',
           col = c('azure2','darkseagreen3'))
```

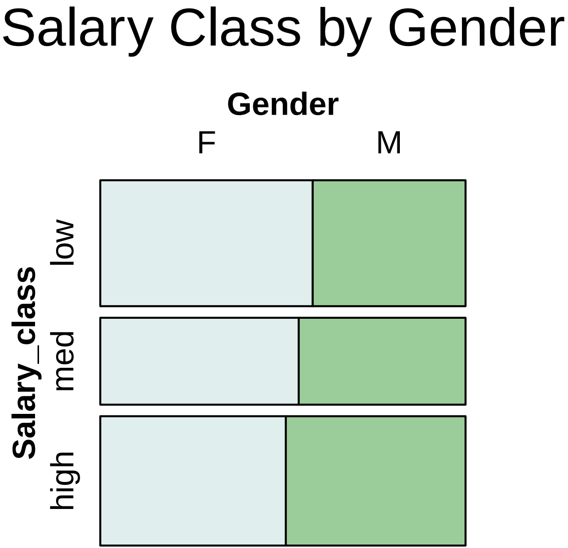
produces the same mosaic plot.

# Mosaic plots

Another way of graphing mosaic plots is by using the `vcd` package, where `vcd` stands for visualizing categorical data. The function `mosaic` in this package also produces this type of graph.

```
library(vcd)
mosaic(~ Salary_class + Gender, data = humans,
       main = 'Salary Class by Gender',
       highlighting = 'Gender',
       highlighting_fill = c('azure2','darkseagreen3'))
```

# Mosaic plots



# Mosaic plots

Observe that the order of the variables in the equation determines which axis the variable will occupy:

```
mosaic(~ Gender +  Salary_class, data = humans,
       main = 'Salary Class by Gender',
       highlighting = 'Gender',
       highlighting_fill = c('azure2','darkseagreen3'))
```

# Mosaic plots