Importing and Working with data in R

By the end of this section, you will be able to:

- Import data into R from excel, SPSS and csv files
- Identify different data structures and variable types
- Convert variables from one type to another
- Order, filter and group data
- Summarise data
- Create new variables from data

The *Tidyverse* set of packages

- A 'toolkit' of packages that are very useful for organsing and manipulating data
- We will use the *haven* package to import SPSS files
- We will use the *dplyr* to organise data
- Also includes the *ggplot2* and *tidyR* packages which we will use later

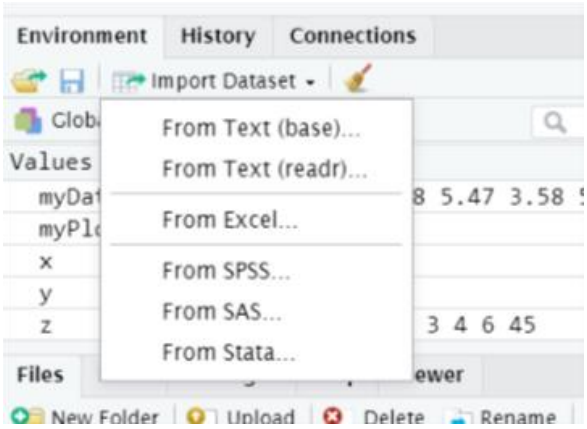
To install:

```
install.packages("tidyverse")
```

Importing .csv data into R

- Can use typed commands or the menu:

```
library(readr)
studentData <- read_csv("Datasets/studentData.csv")
```



Importing SPSS data files into R

```
library(haven)
mySPSSData <- read_sav("mySPSSFile.sav")
```

R can store many different data types

- Vectors: One-dimensional
- Data frames: Two-dimensional
- Matrices: Two-dimensional
- Arrays, Lists etc…

A data matrix:

```
> v=c(3,4)
> A*v
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    3   33   63   93  123  153  183  213  243   273
 [2,]    8   48   88  128  168  208  248  288  328   368
 [3,]    9   39   69   99  129  159  189  219  249   279
 [4,]   16   56   96  136  176  216  256  296  336   376
 [5,]   15   45   75  105  135  165  195  225  255   285
 [6,]   24   64  104  144  184  224  264  304  344   384
 [7,]   21   51   81  111  141  171  201  231  261   291
 [8,]   32   72  112  152  192  232  272  312  352   392
 [9,]   27   57   87  117  147  177  207  237  267   297
[10,]   40   80  120  160  200  240  280  320  360   400
```

The data frame

- A data frame is like a table or a two-dimensional array or matrix
- Each column contains values of one variable
- Each row contains one set of values
- Each column name must be unique

```
# view the first few rows of this dataframe
head(studentData)
```

```
## # A tibble: 6 x 6
##      X1 route    grades hoursOfStudy hasDepdendants satisfactionLevel
##   <dbl> <chr>     <dbl>        <dbl> <chr>          <chr>
## 1     1 FullTime     56            3 Yes            Very
## 2     2 FullTime     47            1 Yes            Not at all
## 3     3 FullTime     72            8 Yes            Not at all
## 4     4 FullTime     79            0 Yes            Very
## 5     5 FullTime     79            4 Yes            Somewhat
## 6     6 FullTime     80            3 Yes            Somewhat
```

Checking the structure of the data

- The **str()** command will allow us to check how our data is structured:

```
str(studentData)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 100 obs. of  6 va
riables:
##  $ X1               : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ route            : chr  "FullTime" "FullTime" "FullTime" "FullTime" ...
##  $ grades           : num  56 47 72 79 79 80 76 39 85 41 ...
##  $ hoursOfStudy     : num  3 1 8 0 4 3 1 6 2 5 ...
##  $ hasDepdendants   : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ satisfactionLevel: chr  "Very" "Not at all" "Not at all" "Very" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   X1 = col_double(),
##   ..   route = col_character(),
##   ..   grades = col_double(),
##   ..   hoursOfStudy = col_double(),
##   ..   hasDepdendants = col_character(),
##   ..   satisfactionLevel = col_character()
##   .. )
```

Notice that some of the variable types are incorrect

Changing variables from one data type to another

```
studentData$route <- as.factor(studentData$route)
studentData$hasDepdendants <- as.factor(studentData$hasDepdendants)
studentData$satisfactionLevel <- as.ordered(studentData$satisfactionLevel)
```

Changing variables from one data type to another #2

Let's check the structure again:

```
str(studentData)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 100 obs. of  6 va
riables:
##  $ X1               : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ route            : Factor w/ 2 levels "FullTime","PartTime": 1 1 1 1 1
1 1 1 1 1 ...
##  $ grades           : num  56 47 72 79 79 80 76 39 85 41 ...
##  $ hoursOfStudy     : num  3 1 8 0 4 3 1 6 2 5 ...
##  $ hasDepdendants   : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 .
..
##  $ satisfactionLevel: Ord.factor w/ 3 levels "Not at all"<"Somewhat"<..: 3
1 1 3 2 2 3 2 1 2 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   X1 = col_double(),
##   ..   route = col_character(),
##   ..   grades = col_double(),
##   ..   hoursOfStudy = col_double(),
##   ..   hasDepdendants = col_character(),
```

```
##   ..   satisfactionLevel = col_character()
##   .. )

str(studentData$satisfactionLevel)

##  Ord.factor w/ 3 levels "Not at all"<"Somewhat"<..: 3 1 1 3 2 2 3 2 1 2 ..
.
```

Changing variables from one data type to another #3

Let's give a proper name to the row ID:

```
library(dplyr) # Loading the dplyr library
#Overwriting studentData with a new version, where the ID column has been cor
rectly named
studentData <- studentData %>% rename(ID = X1)
head(studentData) # viewingthe first 5 rows of the data

## # A tibble: 6 x 6
##      ID route    grades hoursOfStudy hasDepdendants satisfactionLevel
##   <dbl> <fct>     <dbl>        <dbl> <fct>          <ord>
## 1     1 FullTime     56            3 Yes            Very
## 2     2 FullTime     47            1 Yes            Not at all
## 3     3 FullTime     72            8 Yes            Not at all
## 4     4 FullTime     79            0 Yes            Very
## 5     5 FullTime     79            4 Yes            Somewhat
## 6     6 FullTime     80            3 Yes            Somewhat
```

Sorting data

- Using the *dplyr* package, we can arrange our data according to student grade:

```
library(dplyr)
arrange(studentData, grades)

## # A tibble: 100 x 6
##       ID route    grades hoursOfStudy hasDepdendants satisfactionLevel
##    <dbl> <fct>     <dbl>        <dbl> <fct>          <ord>
##  1    64 PartTime     35            7 No             Somewhat
##  2    93 PartTime     36            8 No             Somewhat
##  3    25 FullTime     37            7 Yes            Not at all
##  4    39 FullTime     37            2 Yes            Somewhat
##  5    46 FullTime     37            6 Yes            Very
##  6    61 PartTime     38            2 No             Somewhat
##  7     8 FullTime     39            6 Yes            Somewhat
##  8    31 FullTime     39            4 Yes            Somewhat
##  9    58 PartTime     39            7 No             Very
## 10    67 PartTime     40            6 No             Not at all
## # ... with 90 more rows

arrange(studentData, desc(grades)) # Arrange in descending order
```

```
## # A tibble: 100 x 6
##       ID route    grades hoursOfStudy hasDepdendants satisfactionLevel
##    <dbl> <fct>     <dbl>        <dbl> <fct>           <ord>
##  1     9 FullTime     85            2 Yes             Not at all
##  2    43 FullTime     85            0 Yes             Somewhat
##  3    59 PartTime     85            4 No              Very
##  4    76 PartTime     83            6 No              Not at all
##  5    48 FullTime     82            6 Yes             Not at all
##  6    68 PartTime     82            6 No              Very
##  7    71 PartTime     82            4 No              Somewhat
##  8    12 FullTime     81            8 Yes             Not at all
##  9    74 PartTime     81            5 No              Very
## 10     6 FullTime     80            3 Yes             Somewhat
## # ... with 90 more rows
```

Filtering data

- Show students who acheived a grade of less than 40%

```
library(dplyr)
filter(studentData, grades < 40)
```

```
## # A tibble: 9 x 6
##       ID route    grades hoursOfStudy hasDepdendants satisfactionLevel
##    <dbl> <fct>     <dbl>        <dbl> <fct>           <ord>
## 1     8 FullTime     39            6 Yes             Somewhat
## 2    25 FullTime     37            7 Yes             Not at all
## 3    31 FullTime     39            4 Yes             Somewhat
## 4    39 FullTime     37            2 Yes             Somewhat
## 5    46 FullTime     37            6 Yes             Very
## 6    58 PartTime     39            7 No              Very
## 7    61 PartTime     38            2 No              Somewhat
## 8    64 PartTime     35            7 No              Somewhat
## 9    93 PartTime     36            8 No              Somewhat
```

Filtering data #2

- Show part-time students who scored above 70%

```
library(dplyr)
filter(studentData, grades > 70 & route == "PartTime")
```

```
## # A tibble: 17 x 6
##       ID route    grades hoursOfStudy hasDepdendants satisfactionLevel
##    <dbl> <fct>     <dbl>        <dbl> <fct>           <ord>
##  1    52 PartTime     72            4 No              Very
##  2    59 PartTime     85            4 No              Very
##  3    60 PartTime     73            2 No              Very
##  4    62 PartTime     80            0 No              Somewhat
##  5    68 PartTime     82            6 No              Very
##  6    69 PartTime     74            7 No              Very
##  7    71 PartTime     82            4 No              Somewhat
##  8    73 PartTime     75            8 No              Very
```

```
##  9    74 PartTime     81            5 No              Very
## 10    75 PartTime     75            1 No              Very
## 11    76 PartTime     83            6 No              Not at all
## 12    78 PartTime     77            0 No              Very
## 13    81 PartTime     77            7 No              Somewhat
## 14    87 PartTime     75            1 No              Somewhat
## 15    88 PartTime     71            1 No              Not at all
## 16    97 PartTime     80            5 No              Somewhat
## 17    98 PartTime     76            5 No              Somewhat
```

Using the "pipe" *%>%* with dplyr

- The pipe %>% allows us to:
  - write R commands in a way that is easier to read
  - Chain multiple commands together
- For example: filteredData <- studentData %>% filter(grades > 70 & route == "PartTime")

Grouping data

- It is possible to organise the data into groups and performe analysis on each group:

```
library(dplyr)

 studentData %>% group_by(hasDepdendants) %>%
   summarise(mean = mean(grades), sd = sd(grades))
```

```
## # A tibble: 2 x 3
##   hasDepdendants  mean    sd
##   <fct>          <dbl> <dbl>
## 1 No              62.4  14.3
## 2 Yes             62.3  14.8
```

Remember: we can store that summary data as an object and call it later:

```
library(dplyr)

summaryTable <- studentData %>% group_by(hasDepdendants) %>%
  summarise(mean = mean(grades), sd = sd(grades))

summaryTable
```

```
## # A tibble: 2 x 3
##   hasDepdendants  mean    sd
##   <fct>          <dbl> <dbl>
## 1 No              62.4  14.3
## 2 Yes             62.3  14.8
```

Create new variables from data

We can create new variables from existing data using **mutuate**

```
library(dplyr)

studentData %>% mutate(passFail = ifelse(grades > 40, "Pass", "Fail"))
```

```
## # A tibble: 100 x 7
##       ID route grades hoursOfStudy hasDepdendants satisfactionLev~ passFai
l
##    <dbl> <fct>  <dbl>        <dbl> <fct>          <ord>            <chr>
##  1     1 Full~     56            3 Yes            Very             Pass
##  2     2 Full~     47            1 Yes            Not at all       Pass
##  3     3 Full~     72            8 Yes            Not at all       Pass
##  4     4 Full~     79            0 Yes            Very             Pass
##  5     5 Full~     79            4 Yes            Somewhat         Pass
##  6     6 Full~     80            3 Yes            Somewhat         Pass
##  7     7 Full~     76            1 Yes            Very             Pass
##  8     8 Full~     39            6 Yes            Somewhat         Fail
##  9     9 Full~     85            2 Yes            Not at all       Pass
## 10    10 Full~     41            5 Yes            Somewhat         Pass
## # ... with 90 more rows
```

Importing and Working with data in R – Tasks:

0. Install the **tidyverse** package. When this has been done, enter the command **library("tidyverse")**
1. Use the **read.sav()** command to import the **"Datasets/salesData.sav"** file to a new object called **salesData**
2. Use the **str()** command to check the structure of **salesData** . There should be 5 variables: salary, married, numberOfVisits, age & valueOfSales
3. Change the variable **married** to a factor
4. Arrange the data from lowest to highest **sales** . What was the lowest **sales** value?
5. What is the mean **salary** ?
6. Filter the data to only display customers who are **married** (1 = married, 2 = not married)
7. Create a summary of the data to compare the mean and standard deviation of **sales** for married and non-married customers (1 = married, 2 = not married)
8. Create a new variable called **VIP** and label customers who spent over £500 as "VIP" and other customers as "Non-VIP"