# R Data Structures and Import Export

## Table of Contents

# 1 R data structures

## 1.1 Vectors

- Vector is a sequence of some sort of values (for example numeric, character, logical, and many more) of specified length.
- An atomic vector may contains only one type of data.
- To create a vector with 10 random deviates from a standard normal distribution, and store its elements in an object called vector1, write the following R code:

```
vector1 <- rnorm(10)
vector1

##  [1]  1.2240818  0.3598138  0.4007715  0.1106827 -0.5558411  1.7869131
0.4978505
##  [8] -1.9666172  0.7013559 -0.4727914
```

- Let's now see the contents of our newly created vector1:

```
vector1
```

```
##  [1]  1.2240818  0.3598138  0.4007715  0.1106827 -0.5558411  1.7869131
0.4978505
##  [8] -1.9666172  0.7013559 -0.4727914
```

- As we drew random values, next time if we run the same command, vector1 most likely will contain different elements from the ones shown in the preceding example.

```
vector1 <- rnorm(10)
vector1

##  [1] -1.0678237 -0.2179749 -1.0260044 -0.7288912 -0.6250393 -1.6866933
0.8377870
##  [8]  0.1533731 -1.1381369  1.2538149
```

Let’s make sure that my new vector (vector2) contains same value, we need to set a seed from which we will be drawing the values:

```
set.seed(123)
vector2 <- rnorm(10, mean=3, sd=2)
vector2

##  [1] 1.8790487 2.5396450 6.1174166 3.1410168 3.2585755 6.4301300 3.9218324
##  [8] 0.4698775 1.6262943 2.1086761
```

- In the preceding code we’ve set the seed to an arbitrary number (123) in order to allow you to replicate the values of elements stored in vector2
- we’ve also used some optional parameters of the rnorm() function, which enabled us to specify two characteristics of our data, that is the arithmetic mean (set to 3) and standard deviation (set to 2).

```
set.seed(123)
vector2 <- rnorm(10, mean=3, sd=2)
vector2

##  [1] 1.8790487 2.5396450 6.1174166 3.1410168 3.2585755 6.4301300 3.9218324
##  [8] 0.4698775 1.6262943 2.1086761
```

- Note that vector2 contains the same elements as before.

- The most common way of creating a vector of data is by using the c() function (c stands for concatenate) and then explicitly passing the values of each element of the vector:

```
vector3 <- c(6, 8, 7, 1, 2, 3, 9, 6, 7, 6)
vector3

##  [1] 6 8 7 1 2 3 9 6 7 6
```

- In the preceding example we’ve created vector3 with 10 numeric elements.
- You can use the length() function of any data structure to inspect the number of elements:

```
length(vector3)
```

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

- The class() function allows you to determine how to handle the elements of a vector.

```
class(vector3)

## [1] "numeric"
```

- The mode() function allows you to determine how the data are stored in a vector.

```
mode(vector3)

## [1] "numeric"
```

- The difference between both functions becomes clearer if we create a vector that holds levels of categorical variables (known as a factor in R) with character values:

```
vector4 <- c("poor", "good", "good", "average", "average", "good", "poor",
"good", "average", "good")
vector4

##  [1] "poor"    "good"    "good"    "average" "average" "good"    "poor"   
##  [8] "good"    "average" "good"   

class(vector4)

## [1] "character"

mode(vector4)

## [1] "character"

levels(vector4)

## NULL
```

- In the preceding example, both the class() and mode() outputs of our character vector are the same, because
- we still haven't set it to be treated as a categorical variable, and
- we haven't defined its levels (the content of the levels() function is empty-NULL).
- Following code will explicitly set the vector to be recognized as categorical with three levels:

```
vector4 <- factor(vector4, levels = c("poor", "average", "good"))
vector4

##  [1] poor    good    good    average average good    poor    good   
average
## [10] good   
## Levels: poor average good
```

- The sequence of levels doesn't imply that our vector is ordered.
- We can order the levels of factors in R using the ordered() command.
- For example, To arrange the levels of vector4 in reverse order, starting from "good":

```
vector4.ord <- ordered(vector4, levels = c("good", "average", "poor"))
vector4.ord

##  [1] poor    good    good    average average good    poor    good
average
## [10] good
## Levels: good < average < poor
```

- Output shows that R has now properly recognized the order of our levels, which we had defined.
- We can now apply class() and mode() functions on the vector4.ord object:

```
class(vector4.ord)

## [1] "ordered" "factor"

mode(vector4.ord)

## [1] "numeric"
```

- You may be wondering why the mode() function returned the "numeric" type instead of "character"?
- The answer is simple. By setting the levels of our factor, R has assigned values 1, 2, and 3 to "good", "average", and "poor" respectively, exactly in the same order as we had defined them in the ordered() function.
- You can check this using the levels() and str() functions:

```
levels(vector4.ord)

## [1] "good"    "average" "poor"

str(vector4.ord)

##  Ord.factor w/ 3 levels "good"<"average"<..: 3 1 1 2 2 1 3 1 2 1
```

- Let's create a logical vector, that contains only TRUE and FALSE values:

```
vector5 <- c(TRUE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE,
FALSE)
vector5

##  [1]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

## 1.2 Vector equivalent examples in Python

To create a vector with 10 random deviates from a standard normal distribution, and store its elements in an object called vector1, write the following Python code:

```
import numpy as np
# mean is 0 and std is 1 (Default in R)
vector1 = np.random.normal(0,1,10)
vector1
```

```
## array([-0.56641304, -0.75694758,  1.14628986,  2.70452094, -0.76687951,
##         0.86275206,  1.06666086,  1.04790025, -0.58103393, -1.36004765])
```

- Next time if we run the same code most likely we will be getting different result

```
import numpy as np
# mean is 0 and std is 1 (Default in R)
vector1 = np.random.normal(0,1,10)
vector1

## array([ 0.69309676,  1.42565041,  0.8917772 ,  0.39562409,  1.06123575,
##        -0.16679835, -0.42311024,  0.12609508,  0.01264005,  2.54992217])
```

- To set a seed value to 123 in Python

```
import random
random.seed(123)
# Here mean is 3 and std deviation is 2
vector2 = np.random.normal(3,2,10)
vector2

## array([-0.25762253,  2.06189874,  1.49288671,  8.82397155,  6.08786403,
##         3.640455  ,  1.93049096,  2.11581087, -0.61542344,  4.29731867])
```

- If we execute the code again:

```
import random
# Here mean is 3 and std deviation is 2
vector2 = np.random.normal(3,2,10)
vector2

## array([ 3.58746589, -1.24624096,  3.13744468,  4.24690144,  2.93363097,
##         0.79080031,  4.12245904,  5.29256084,  5.49705847,  3.71986537])
```

- Note that vector2 contains the same elements as before.
- A vector in a simple term can be considered as a single-dimensional array.
- With respect to Python, a vector is a one-dimensional array of lists.
- It occupies the elements in a similar manner as that of a Python list.
- Let us now understand the Creation of a vector in Python.
- Python NumPy module is used to create a vector.
- We use numpy.array() method to create a one-dimensional array i.e. a vector.

```
list = [6, 8, 7, 1, 2, 3, 9, 6, 7, 6]
len(list)

## 10

vector3 = np.array(list)
len(vector3)
```

```
## 10

vector3

## array([6, 8, 7, 1, 2, 3, 9, 6, 7, 6])
```

- In the preceding example we've created vector3 with 10 numeric elements.
- You can get the length of vector3 using NumPy.size:

```
vector3.size

## 10
```

- Using Python's type() function we can determine the type of elements in vector3 (R uses class() function)

```
type(vector3)

## <class 'numpy.ndarray'>
```

## 1.3 Scalars

- Think of scalars as one-element vectors that are traditionally used to hold some constant values, for example:

```
a1 <- 5
a1

## [1] 5
```

- You may use scalars in computations and also assign any one-element outputs of mathematical or statistical operations to another, arbitrary named scalar for example:

```
a2 <- 4
a3 <- a1 + a2
a3

## [1] 9
```

## 1.4 Python equivalent examples for Scalars

- Scalars are traditionally used to hold some constant values, for example:

```
a1 = 5
a1

## 5
```

- You may use scalars in computations and also assign any one-element outputs of mathematical or statistical operations to another, arbitrary named scalar for example:

```
a2 = 4
a3 = a1 + a2
a3

## 9
```

## 1.5 Matrices

- A matrix is a two-dimensional R data structure in which each of its elements must be of the same type; that is numeric, character, or logical.
- As matrices consist of rows and columns, their shape resembles tables.
- When creating a matrix, you can specify how you want to distribute values across its rows and columns, for example:

```
y <- matrix(1:20, nrow=5, ncol=4)
y

##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

- In the preceding example we have allocated a sequence of 20 values (from 1 to 20) into five rows and four columns, and by default they have been distributed by column.

- Let us create another matrix in which we will distribute the values by rows and give names to rows and columns using the dimnames argument (dimnames stands for names of dimensions) in the matrix() function:

```
rows <- c("R1", "R2", "R3", "R4", "R5")
columns <- c("C1", "C2", "C3", "C4")
z <- matrix(1:20, nrow=5, ncol=4, byrow=TRUE, dimnames=list(rows, columns))
z

##    C1 C2 C3 C4
## R1  1  2  3  4
## R2  5  6  7  8
## R3  9 10 11 12
## R4 13 14 15 16
## R5 17 18 19 20
```

- How to extract specific elements stored in a matrix?
- Looking at the matrix y, for which we didn't define any names for its rows and columns, notice how R denotes them. The row numbers come in the format

  $$r,$$

  , where r is a consecutive number of a row, whereas columns are identified by

  $$,c$$

  , where c is a consecutive number of a column.
- To extract a value stored in the fourth row of the second column of our matrix y, type:

```
y[4,2]

## [1] 9
```

- To extract the whole column number three from our matrix y, type the following:

```
y[,3]

## [1] 11 12 13 14 15
```

- To extract three values stored in the second, third, and fifth rows of the first column in our vector z with named rows and columns.
- Notice that for several values to extract we have to specify their row locations as a vector; hence we will put their row coordinates inside the c() function that we had previously used to create vectors:

```
z[c(2, 3, 5), 1]

## R2 R3 R5
##  5  9 17
```

- Similar rules for extracting data will apply to other data structures in R such as arrays, lists, and data frames.

- To extract the whole column number three from our matrix y, type the following:

```
y[,3]

## [1] 11 12 13 14 15
```

## 1.6 Python Matrices

- When creating a matrix, you can specify how you want to distribute values across its rows and columns, for example:

```
# R   y <- matrix(1:20, nrow=5, ncol=4)
y = np.arange(1,21).reshape(5, 4)
y

## array([[ 1,  2,  3,  4],
##        [ 5,  6,  7,  8],
##        [ 9, 10, 11, 12],
##        [13, 14, 15, 16],
##        [17, 18, 19, 20]])
```

- How to extract specific elements stored in a matrix?
- To extract a value stored in the fourth row of the second column of our matrix y, type:

```
# In R z[4,2]
y[4-1,2-1]

## 14
```

- To extract three values stored in the second, third, and fifth rows of the first column in our vector z with named rows and columns.

- Notice that for several values to extract we have to specify their row locations as a vector; hence we will put their row coordinates inside the c() function that we had previously used to create vectors:

```
# In R z[c(2, 3, 5), 1]
y[[2-1, 3-1, 5-1], 1-1]

## array([ 5,  9, 17])
```

- To extract the whole column number three from our matrix y, type the following:

```
y[:,3-1]

## array([ 3,  7, 11, 15, 19])
```

## 1.7 Arrays

- Arrays are very similar to matrices with only one exception: they contain more dimensions. However, just like matrices or vectors, they may only hold one type of data.
- Arrays are created using the array() function:

```
array1 <- array(1:20, dim=c(2,2,5))
array1

## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
##
## , , 3
##
##      [,1] [,2]
## [1,]    9   11
## [2,]   10   12
##
## , , 4
##
##      [,1] [,2]
## [1,]   13   15
## [2,]   14   16
##
## , , 5
##
##      [,1] [,2]
## [1,]   17   19
## [2,]   18   20
```

- The dim argument, which was used within the array() function, specifies how many dimensions you want to distribute your data across.
- As we had 20 values (from 1 to 20) we had to make sure that our array can hold all 20 elements; therefore we will assign them into two rows, two columns, and five dimensions (2 x 2 x 5 = 20).
- You can check the dimensionality of your multi-dimensional R objects with the dim() command:

```
dim(array1)

## [1] 2 2 5
```

- If you want to extract a specific value located in the second row of the first column in the fourth dimension of our array1:

```
array1[2, 1, 4]

## [1] 14
```

- If you need to find a location of a specific value, for example 11, within the array, you can simply type the following line:

```
which(array1==11, arr.ind=TRUE)

##      dim1 dim2 dim3
## [1,]    1    2    3
```

- The which() function returns indices of the array (arr.ind=TRUE), where the sought value equals 11.
- As we had only one instance of value 11 in our array, there is only one row specifying its location in the output.
- If we had more instances of 11, additional rows would be returned indicating indices for each element equal to 11.

## 1.8 Python Arrays

- Arrays are very similar to matrices with only one exception: they contain more dimensions. However, just like matrices or vectors, they may only hold one type of data.
- Arrays are created using the array() function:

```
# In R array1 <- array(1:20, dim=c(2,2,5))
array1 = np.arange(1,21).reshape(2,2,5)
array1

## array([[[ 1,  2,  3,  4,  5],
##          [ 6,  7,  8,  9, 10]],
##
##         [[11, 12, 13, 14, 15],
##          [16, 17, 18, 19, 20]]])
```

```
array1 = np.arange(1,21).reshape(5,2,2)
array1

## array([[[ 1,  2],
##          [ 3,  4]],
## 
##         [[ 5,  6],
##          [ 7,  8]],
## 
##         [[ 9, 10],
##          [11, 12]],
## 
##         [[13, 14],
##          [15, 16]],
## 
##         [[17, 18],
##          [19, 20]]])
```

- You can check the dimensionality of your multi-dimensional R objects with the dim() command:

```
# In R dim(array1)
array1.shape

## (5, 2, 2)
```

- If you want to extract a specific value located in the second row of the first column in the fourth dimension of our array1:

```
# In R array1[2, 1, 4]
array1[4-1, 2-1, 1-1]

## 15
```

- If you need to find a location of a specific value, for example 11, within the array, you can simply type the following line:

```
np.where(array1==11)

## (array([2], dtype=int64), array([1], dtype=int64), array([0],
dtype=int64))
```

## 1.9 Data Frames

- Data frames are one of the most widely used R data structures.
- Data frames are very similar to matrices, but they may contain different types of data.
- Its a typical rectangular data set with rows and columns or observations and variables.
- Most data sets are imported into R as data frames.
- You can also create a simple data frame manually with the data.frame() function, but as each column in the data frame may be of a different type, we must first create vectors that will hold data for specific columns:

```
subjectID <- c(1:10)
age <- c(37,23,42,25,22,25,48,19,22,38)
gender <- c("male", "male", "male", "male", "male", "female",
"female","female", "female", "female")
lifesat <- c(9,7,8,10,4,10,8,7,8,9)
health <- c("good", "average", "average", "good", "poor", "average", "good",
"poor", "average", "good")
paid <- c(T, F, F, T, T, T, F, F, F, T)
dataset <- data.frame(subjectID, age, gender, lifesat, health, paid)
dataset

##    subjectID age gender lifesat  health  paid
## 1          1  37   male       9    good  TRUE
## 2          2  23   male       7 average FALSE
## 3          3  42   male       8 average FALSE
## 4          4  25   male      10    good  TRUE
## 5          5  22   male       4    poor  TRUE
## 6          6  25 female      10 average  TRUE
## 7          7  48 female       8    good FALSE
## 8          8  19 female       7    poor FALSE
## 9          9  22 female       8 average FALSE
## 10        10  38 female       9    good  TRUE
```

- The preceding example presents a simple data frame which contains a sample from a basic psychological experiment, which measured the subjects' life satisfaction (lifesat) and their health status (health) and also collected other sociodemographic information such as age and gender, and whether the participant was a paid subject or a volunteer.
- As we deal with various types of data, the elements for each column had to be amalgamated into a single structure of a data frame using the data.frame() command and specifying the names of objects (vectors) in which we stored all values.
- You can inspect the structure of this data frame with the str() function:

```
str(dataset)

## 'data.frame':    10 obs. of  6 variables:
##  $ subjectID: int  1 2 3 4 5 6 7 8 9 10
##  $ age      : num  37 23 42 25 22 25 48 19 22 38
##  $ gender   : chr  "male" "male" "male" "male" ...
##  $ lifesat  : num  9 7 8 10 4 10 8 7 8 9
##  $ health   : chr  "good" "average" "average" "good" ...
##  $ paid     : logi  TRUE FALSE FALSE TRUE TRUE TRUE ...
```

- The output of str() gives you some basic insights into the shape and format of your data in the dataset object, for example, the number of observations and variables, the names of variables, types of data they hold, and examples of values for each variable.

- Suppose you want to print only those columns which contain age, gender, and life satisfaction information from our dataset data frame. You may use the following two alternatives:

```
dataset[,2:4]

##    age gender lifesat
## 1   37   male       9
## 2   23   male       7
## 3   42   male       8
## 4   25   male      10
## 5   22   male       4
## 6   25 female      10
## 7   48 female       8
## 8   19 female       7
## 9   22 female       8
## 10  38 female       9

# or
dataset[, c("age", "gender", "lifesat")]

##    age gender lifesat
## 1   37   male       9
## 2   23   male       7
## 3   42   male       8
## 4   25   male      10
## 5   22   male       4
## 6   25 female      10
## 7   48 female       8
## 8   19 female       7
## 9   22 female       8
## 10  38 female       9

dataset[, c("gender", "lifesat", "age")]

##    gender lifesat age
## 1    male       9  37
## 2    male       7  23
## 3    male       8  42
## 4    male      10  25
## 5    male       4  22
## 6  female      10  25
## 7  female       8  48
## 8  female       7  19
## 9  female       8  22
## 10 female       9  38
```

- Both lines of code will produce exactly the same results.
- The subset() function however gives you the additional capabilities of defining conditional statements that will filter the data, based on the output of logical operators.

- You can replicate the preceding output using subset() in the following way:

```
subset(dataset[c("age", "gender", "lifesat")])

##    age gender lifesat
## 1   37   male       9
## 2   23   male       7
## 3   42   male       8
## 4   25   male      10
## 5   22   male       4
## 6   25 female      10
## 7   48 female       8
## 8   19 female       7
## 9   22 female       8
## 10  38 female       9
```

- Assume now that you want to create a subset with all subjects who are over 30 years old, and with a score of greater than or equal to eight on the life satisfaction scale (lifesat).
- The subset() function comes in very handy:

```
subset(dataset, age > 30 & lifesat >= 8)

##    subjectID age gender lifesat  health  paid
## 1          1  37   male       9    good  TRUE
## 3          3  42   male       8 average FALSE
## 7          7  48 female       8    good FALSE
## 10        10  38 female       9    good  TRUE
```

- To produce an output with two socio-demographic variables of age and gender, relating only to those subjects who were paid to participate in this experiment:

```
subset(dataset, paid==TRUE, select=c("age", "gender"))

##    age gender
## 1   37   male
## 4   25   male
## 5   22   male
## 6   25 female
## 10  38 female
```

## 1.10 Python Data Frames

```
# In R
# subjectID = c(1:10)
# age <- c(37,23,42,25,22,25,48,19,22,38)
# gender <- c("male", "male", "male", "male", "male", "female",
"female","female", "female", "female")
# lifesat <- c(9,7,8,10,4,10,8,7,8,9)
# health <- c("good", "average", "average", "good", "poor", "average",
"good", "poor", "average", "good")
# paid <- c(T, F, F, T, T, T, F, F, F, T)
# dataset <- data.frame(subjectID, age, gender, lifesat, health, paid)
# dataset
# library(reticulate)
# py_install("pandas")
import pandas as pd
subjectID = [*range(1, 11)]
age = [37,23,42,25,22,25,48,19,22,38]
gender = ["male", "male", "male", "male", "male", "female",
"female","female", "female", "female"]
lifesat = [9,7,8,10,4,10,8,7,8,9]
health = ["good", "average", "average", "good", "poor", "average", "good",
"poor", "average", "good"]
paid = [True, False, False, True, True, True, False, False, False, True]
# zip the above lists
# data = [ subjectID, age, gender, lifesat, health, paid ]
data = zip(subjectID, age, gender, lifesat, health, paid)
# dataset = pd.DataFrame(data )
dataset = pd.DataFrame(data, columns=['subjectID', 'age', 'gender',
'lifesat', 'health','paid'] )
dataset
```

```
##    subjectID  age  gender  lifesat   health   paid
## 0          1   37    male        9     good   True
## 1          2   23    male        7  average  False
## 2          3   42    male        8  average  False
## 3          4   25    male       10     good   True
## 4          5   22    male        4     poor   True
## 5          6   25  female       10  average   True
## 6          7   48  female        8     good  False
## 7          8   19  female        7     poor  False
## 8          9   22  female        8  average  False
## 9         10   38  female        9     good   True
```

```
# In R
# subjectID = c(1:10)
# age <- c(37,23,42,25,22,25,48,19,22,38)
# gender <- c("male", "male", "male", "male", "male", "female",
"female","female", "female", "female")
# lifesat <- c(9,7,8,10,4,10,8,7,8,9)
# health <- c("good", "average", "average", "good", "poor", "average",
"good", "poor", "average", "good")
# paid <- c(T, F, F, T, T, T, F, F, F, T)
# dataset <- data.frame(subjectID, age, gender, lifesat, health, paid)
# dataset
# library(reticulate)
# py_install("pandas")
import pandas as pd
# initialize data of lists.
data = {
  'subjectID' : [*range(1, 11)],
  'age' : [37,23,42,25,22,25,48,19,22,38],
  'gender' : ["male", "male", "male", "male", "male", "female",
"female","female", "female", "female"],
  'lifesat' : [9,7,8,10,4,10,8,7,8,9],
  'health' : ["good", "average", "average", "good", "poor", "average",
"good", "poor", "average", "good"],
  'paid' : [True, False, False, True, True, True, False, False, False, True]
}
dataset = pd.DataFrame(data, columns=['subjectID', 'age', 'gender',
'lifesat', 'health','paid'] )
dataset = pd.DataFrame(data )
dataset

##    subjectID  age  gender  lifesat   health   paid
## 0          1   37    male        9     good   True
## 1          2   23    male        7  average  False
## 2          3   42    male        8  average  False
## 3          4   25    male       10     good   True
## 4          5   22    male        4     poor   True
## 5          6   25  female       10  average   True
## 6          7   48  female        8     good  False
## 7          8   19  female        7     poor  False
## 8          9   22  female        8  average  False
## 9         10   38  female        9     good   True
```

### 1.11 Lists

- A list in R is a data structure, which is a collection of other objects.
- In the list you can store vectors, scalars, matrices, arrays, data function) that will include a variety of other data structures:

```
simple.vector1 <- c(1, 29, 21, 3, 4, 55)
simple.matrix <- matrix(1:24, nrow=4, ncol=6, byrow=TRUE)
simple.scalar1 <- 5
simple.scalar2 <- "The List"
simple.vector2 <- c("easy", "moderate", "difficult")
simple.list <- list(name=simple.scalar2, matrix=simple.matrix,
vector=simple.vector1, scalar=simple.scalar1, difficulty=simple.vector2)
simple.list

## $name
## [1] "The List"
##
## $matrix
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    3    4    5    6
## [2,]    7    8    9   10   11   12
## [3,]   13   14   15   16   17   18
## [4,]   19   20   21   22   23   24
##
## $vector
## [1]  1 29 21  3  4 55
##
```

```
## $scalar
## [1] 5
##
## $difficulty
## [1] "easy"      "moderate"  "difficult"
```

```
str(simple.list)
```

```
## List of 5
##  $ name      : chr "The List"
##  $ matrix    : int [1:4, 1:6] 1 7 13 19 2 8 14 20 3 9 ...
##  $ vector    : num [1:6] 1 29 21 3 4 55
##  $ scalar    : num 5
##  $ difficulty: chr [1:3] "easy" "moderate" "difficult"
```

- In the preceding output, we have assigned names to each component in our list
- The str() function prints them as if they were variables of a standard rectangular data set.
- In order to extract specific elements from a list, you first need to use a double square bracket notation

$$\[x$$

] to identify a component x within the list.

- For example, to print an element stored in its first row and the third column of the second component, you may use the following line in R:

```
simple.list[[2]][1,3]
```

```
## [1] 3
```

## 2 Exporting R Data Objects

- Sometimes, it may happen that you need to exit RStudio or shut your PC down.
- If you do not save your created objects, you will lose all of them, the moment you close RStudio.
- This might turn out to be quite costly, especially if you had not saved your original R script.
- In order to prevent the objects from being deleted, you can save all or selected ones as .RData.
- In the first case, you may use the save.image() function, which saves your whole current workspace with all objects to your current working directory:

```
save.image(file = "workspace.RData")
```

- You can reduce the size of the saved objects using one of the compression methods available.
- For example, the above workspace.RData file was 3,751 bytes in size without compression, but when xz compression was applied the size of the resulting file decreased to 3,568 bytes.

```
save.image(file = "workspace2.RData", compress = "xz")
```

- It gets much more significant for bigger data structures.
- The trade-off with applying one of the compression methods is the time it takes for R to save and load .RData files.
- If you prefer to save only chosen objects (for example the dataset data frame and simple.list list) you can achieve this with the save() function:

```
save(dataset, simple.list, file = "two_objects.RData")
```

- You may now test whether the above solutions worked by cleaning your global environment of all objects, and then loading one of the created files, for example:

```
list=ls()
list

##  [1] "a1"             "a2"             "a3"             "age"           
##  [5] "array1"         "columns"        "dataset"        "gender"        
##  [9] "grel"           "grel.data"      "health"         "l"             
## [13] "lifesat"        "paid"           "rows"           "simple.list"   
## [17] "simple.matrix"  "simple.scalar1" "simple.scalar2" "simple.vector1"
## [21] "simple.vector2" "subjectID"      "vector1"        "vector2"       
## [25] "vector3"        "vector4"        "vector4.ord"    "vector5"       
## [29] "x"              "y"              "z"

rm(list=ls())

load("workspace2.RData")
#rm(db)
```

- Other functions that allow you to write text representations of R objects, for example dump() or dput().
- Run the following commands and compare the returned outputs:

```
#dump(ls(), file = "dump.R", append = FALSE)
dump(ls(dataset), file = "dump1.R", append = FALSE)

dput(dataset, file = "dput.txt")
```

- The save.image() and save() functions only create images of your workspace or selected objects on the hard drive.
- The easiest way to export R objects to generic file formats like CSV, TXT, or TAB is through the cat() function, but it only works on atomic vectors:

```
cat(age, file="age.txt", sep=",", fill=TRUE, labels=NULL, append=TRUE)

cat(age, file="age.csv", sep=",", fill=TRUE, labels=NULL, append=TRUE)
```

- The preceding code creates two files, one as a text file and another one as a comma separated format, both of which contain values from the age vector that we had previously created.

- The sep argument is a character vector of strings to append after each element.
- The fill option is a logical argument that controls whether the output is automatically broken into lines (if set to TRUE),
- The labels parameter allows you to add a character vector of labels for each printed line of data in the file, and
- The append logical argument enables you to append the output of the call to the already existing file with the same name.
- In order to export vectors and matrices to TXT, CSV, or TAB formats, you can use the write() function, which writes out a matrix or a vector in a specified number of columns for example:

```
write(age, file="agedata.csv", ncolumns=2, append=TRUE, sep=",")

write(y, file="matrix_y.tab", ncolumns=2, append=FALSE, sep="\t")
```

- Another method of exporting matrices provides the MASS package (make sure you install it with the install.packages("MASS") function) through the write.matrix() command:

```
library(MASS)
write.matrix(y, file="ymatrix.txt", sep=",")
```

- For large matrices, the write.matrix() function allows users to specify the size of blocks in which the data are written through the blocksize argument.
- One of the most common R data structure that you are going to export to different file formats will be a data frame.
- The generic write.table() function gives you an option to save your processed data frame objects to standard data formats, for example TAB, TXT, or CSV:

```
write.table(dataset, file="dataset1.txt", append=F, sep=",", na="NA",
col.names=T, row.names=F, dec=".")
```

- In the na option you may specify an arbitrary string to use for missing values in the data.
- The logical parameter col.names allows users to append the names of columns to the output file, and the dec parameter sets the string used for decimal points and must be a single character.
- In the example, we used row.names set to FALSE, as the names of the rows in the data are the same as the values of the subjectID column.
- However, it is very likely that in other data sets the ID variable may differ from the names (or numbers) of rows, so you may want to control it depending on the characteristics of your data.

- Two similar functions write.csv() and write.csv2() are just convenience wrappers for saving CSV files, and they only differ from the generic write.table() function by default settings of some of their parameters, for example sep and dec.

- Another very useful way of writing R objects to the XLSX format is provided by the openxlsx package through the write.xlsx() function, which, apart from data frames, also allows lists to be easily written to Excel spreadsheets.

- Windows users may need to install the Rtools package in order to use openxlsx functionalities.

- The write.xlsx() function gives you a large choice of possible options to set, including a custom style to apply to column names (through the headerStyle argument), the color of cell borders (borderColour), or even line style (borderStyle).

- The following example utilizes only the most common and minimal arguments required to write a list to the XLSX file:

```
#install.packages("xlsx")
library("xlsx")
write.xlsx(dataset, file = "dataset1.xlsx",sheetName = NULL, append = F)
write.xlsx(dataset, file = "dataset1.xlsx",sheetName = "First", append = F)
write.xlsx(dataset, file = "dataset1.xlsx",sheetName = "Second", append = T)
```

- A third-party package called foreign makes it possible to write data frames to other formats used by well-known statistical tools such as SPSS, Stata, or SAS.
- When creating files, the write.foreign() function requires users to specify the names of both the data and code files.
- Data files hold raw data, whereas
- code files contain scripts with the data structure and metadata (value and variable labels, variable formats, and so on) written in the proprietary syntax.
- In the following example, the code writes the dataset data frame to the SPSS format:

```
library(foreign)
write.foreign(dataset, "datafile.txt", "codefile.txt", package="SPSS")
```

- Finally, another package called rio contains only three functions, allowing users to quickly import(), export(), and convert() data between a large array of file formats, (for example TSV, CSV, RDS, RData, JSON, DTA, SAV, and many more).
- The rio package does not introduce any new functionalities apart from the default arguments characteristic for underlying export functions, so you still need to be familiar with the original functions and their parameters if you require more advanced exporting capabilities. But, if you are only looking for a no-fuss general export function, the rio package is definitely a good shortcut to take:

```
library(rio)
export(dataset, format = "stata")
export(dataset, "dataset1.csv", col.names = TRUE, na = "NA")
```

## 3 Importing data from different formats

- In the same way you exported R objects to external files on your hard drive, you can just as easily import data into R from almost all available file formats.

- R also connects very well with traditional SQL and NoSQL databases, other statistical tools such as SPSS, SAS, Minitab, Stata, Systat, and many others, online data repositories, and distributed file systems (such as HDFS);

- Besides, it allows you to do web scraping , or import and filter textual data.

- All in all, R is a great tool for serious data mining

- The first set of data that we will be exploring comes from Queen's University in (Northern Ireland) and is based on the Northern Ireland Life and Times Survey (NILT) 2012.

- The NILT 2012 Teaching Dataset, which, apart from the standard socio-demographic NILT variables, contains several additional variables concerning the Good relations topic, which refers to social attitudes of people living in Northern Ireland towards ethnic minorities, migrants, and cultural diversity.

- Upon downloading the data to the current working directory, we are now ready to import it to the R session.

- As the dataset comes in the SPSS format, we will use the foreign package to upload it to the workspace:

```
library(foreign)
grel <- read.spss("NILT2012GR.sav", to.data.frame=TRUE)

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE):
NILT2012GR.sav:
## Unrecognized record type 7, subtype 10 encountered in system file

## re-encoding from CP1252

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
36, 37, 38,
## 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
57, 58, 59,
## 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80,
## 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 97 added in
variable: rage

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 18, 19, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39, 40,
## 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58,
59, 60, 61,
## 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
80, 81, 82,
## 83, 84, 87, 88 added in variable: spage

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 1, 2, 3, 4, 5, 6 added in variable: nadult

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 0, 1, 2, 3, 4, 5 added in variable: nkids

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 0, 1, 2 added in variable: nelderly

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 1, 2, 3, 4, 5, 6, 7, 8, 10 added in variable: nfamily

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23,
## 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44,
## 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65,
## 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 80, 81, 82, 83, 84,
85, 86, 87,
## 88, 89, 92, 94, 97 added in variable: livearea

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 4, 6, 9, 10, 12, 13, 15, 16, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30,
31, 32,
## 33, 34, 35, 36, 37, 38, 39, 40, 43, 45, 48, 49, 50, 53, 55, 56, 60, 65,
70, 80, 84,
## 96, 100 added in variable: rhourswk

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22,
## 23, 24, 25, 30, 32, 34, 35, 37, 40, 46, 48, 50, 60, 70, 82, 85, 88, 100,
112, 120,
## 130, 150, 200, 250, 300, 344, 800 added in variable: rmany
## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 1, 4, 6, 7, 8, 10, 16, 18, 20, 22, 24, 25, 26, 27, 28, 30, 31, 32, 35, 36,
37, 38,
## 39, 40, 42, 43, 45, 47, 48, 50, 55, 60, 64, 70, 72, 85 added in variable:
shourswk

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 18, 20, 24, 25, 31, 40, 50,
70, 80,
## 100, 200, 300, 1000, 1090 added in variable: smany

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 260, 780, 1300, 1820, 2340, 2860, 3380, 3900, 4420, 4940, 5720, 6760,
7800, 8840,
## 9880, 10920, 11960, 13000, 14040, 15080, 16120, 17160, 18200, 19240,
20280, 22100,
## 24700, 27300, 29900, 32500, 35100, 37700, 40300, 42900, 45500, 48100,
50700, 75000
## added in variable: persinc2

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 1300, 1820, 2340, 2860, 3380, 3900, 4420, 4940, 5720, 6760, 7800, 8840,
9880,
## 10920, 11960, 13000, 14040, 15080, 16120, 17160, 18200, 19240, 20280,
22100, 24700,
## 27300, 29900, 32500, 35100, 37700, 40300, 42900, 45500, 48100, 50700,
75000 added
## in variable: hhldinc2

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 910, 1300, 1430, 1560, 1708.57142857143, 1820, 1950, 1976, 2184, 2210,
2340,
## 2451.42857142857, 2470, 2513.33333333333, 2600, 2730, 2748.57142857143,
2808,
## 2860, 2990, 3016, 3250, 3293.33333333333, 3380, 3432, 3510, 3640,
3683.33333333333,
## 3770, 3848, 3900, 3986.66666666667, 4056, 4116.66666666667, 4290,
4333.33333333333,
## 4420, 4550, 4680, 4940, 5014.28571428571, 5026.66666666667, 5070, 5362.5,
## 5373.33333333333, 5416.66666666667, 5460, 5525, 5720, 5980,
6066.66666666667,
## 6175, 6283.33333333333, 6413.33333333333, 6500, 6716.66666666667, 6760,
6825, 7020,
## 7366.66666666667, 7475, 7540, 7800, 8060, 8125, 8233.33333333333, 8580,
8775, 8840,
## 9100, 9425, 9620, 9880, 9966.66666666667, 10075, 10140, 10725,
10833.3333333333,
## 10920, 11050, 11375, 11700, 11960, 12025, 12350, 12500, 12566.6666666667,
## 12675, 13000, 13433.3333333333, 13650, 14040, 14300, 14950, 15000, 15080,
## 15166.6666666667, 16033.3333333333, 16120, 16250, 16900, 17160, 17550,
18200,
## 18750, 18850, 19240, 20150, 20280, 21450, 22100, 22750, 24050, 24700,
25000, 25350,
## 27300, 29900, 32500, 35100, 37500, 37700, 40300, 42900, 48100, 75000 added
in
## variable: percapti1

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 3293.33333333333, 4420, 5460, 5720, 6760, 7540, 7800, 8580, 8840, 9100,
9880,
## 10140, 10920, 11050, 11960, 12350, 13000, 13650, 14040, 14950, 15080,
16120, 16250,
## 17160, 17550, 18200, 18850, 19240, 20280, 21450, 22100, 24050, 24700,
25000, 27300,
## 29900, 32500, 35100, 37500, 37700, 40300, 42900, 45500, 48100, 50700,
75000 added
## in variable: percapti2

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 0, 1, 2, 3, 4, 5, 6 added in variable: eesocdist

## Warning in read.spss("NILT2012GR.sav", to.data.frame = TRUE): Undeclared
level(s)
## 0, 1, 2, 3, 4, 5, 6 added in variable: mussocdist
```

- Check the structure of the newly created grel data frame by listing the top lines of the dataset:

```
head(grel, n=5)

##   serial househld rage spage   rsex nadult nkids nelderly nfamily
## 1      1        4   44    52   Male      3     1        0       4
## 2      2        2   86    81   Male      2     0        2       2
## 3      3        3   26    26 Female      2     1        0       3
## 4      4        1   72  <NA> Female      1     0        1       1
## 5      5        2   40    38   Male      2     0        0       2
##                                            rmarstat livearea
## 1 Married/civil partnership and living with partner       44
## 2 Married/civil partnership and living with partner       86
## 3                                  Living as married        4
## 4                                            Widowed       47
## 5 Married/civil partnership and living with partner       10
##                                    placeliv                hincpast intwww
umineth
## 1 the suburbs or outskirts of a big city ...fallen behind prices    Yes
No
```

```
## 2          a farm or home in the country ...fallen behind prices     No
No
## 3                    a small city or town ...fallen behind prices    Yes
No
## 4 the suburbs or outskirts of a big city     kept up with prices    Yes
No
## 5 the suburbs or outskirts of a big city ...fallen behind prices    Yes
No
##   eqnow1 eqnow2 eqnow3 eqnow4 eqnow5 eqnow6 eqnow7 eqnow8 eqnow9 eqnow10
eqnow11
## 1     No     No     No     No     No     No     No     No     No      No
No
## 2     No     No    Yes     No     No     No     No     No     No      No
Yes
## 3     No     No    Yes     No     No    Yes     No     No     No      No
Yes
## 4   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>    <NA>
<NA>
## 5     No     No    Yes     No     No    Yes     No     No     No      No
Yes
##                             eqpast1                            eqpast2
## 1 Treated the same as 5 years ago Treated the same as 5 years ago
## 2 Treated the same as 5 years ago Treated the same as 5 years ago
## 3 Treated the same as 5 years ago  Treated worse than 5 years ago
## 4                            <NA>                            <NA>
## 5 Treated the same as 5 years ago Treated the same as 5 years ago
##                             eqpast3                            eqpast4
## 1 Treated better than 5 years ago Treated better than 5 years ago
## 2 Treated the same as 5 years ago Treated the same as 5 years ago
## 3  Treated worse than 5 years ago Treated better than 5 years ago
## 4                            <NA>                            <NA>
## 5 Treated the same as 5 years ago Treated better than 5 years ago
##                             eqpast5                            eqpast6
## 1 Treated the same as 5 years ago Treated the same as 5 years ago
## 2 Treated the same as 5 years ago Treated the same as 5 years ago
## 3 Treated better than 5 years ago Treated the same as 5 years ago
## 4                            <NA>                            <NA>
## 5 Treated the same as 5 years ago Treated the same as 5 years ago
##                             eqpast7                            eqpast8
## 1 Treated better than 5 years ago Treated the same as 5 years ago
## 2 Treated the same as 5 years ago Treated the same as 5 years ago
## 3 Treated the same as 5 years ago Treated the same as 5 years ago
## 4                            <NA>                            <NA>
## 5  Treated worse than 5 years ago Treated the same as 5 years ago
##                             eqpast9                           eqpast10
## 1 Treated the same as 5 years ago Treated the same as 5 years ago
## 2 Treated the same as 5 years ago Treated the same as 5 years ago
## 3 Treated the same as 5 years ago Treated the same as 5 years ago
## 4                            <NA>                            <NA>
## 5 Treated the same as 5 years ago Treated the same as 5 years ago
##                           eqpast11                               tenshort
## 1 Treated the same as 5 years ago Buying with help of a mortgage or loan
## 2 Treated the same as 5 years ago                        Own it outright
## 3 Treated better than 5 years ago                Rent – private landlord
## 4                            <NA>                        Own it outright
## 5  Treated worse than 5 years ago Buying with help of a mortgage or loan
##                  highqual         tea                      work rhourswk
rsuper
## 1   GCE A level or equiv          16                  Employee       40
No
## 2      No qualifications 15 or under             Self-employed     <NA>
<NA>
## 3      GCSE A-C or equiv          16 Foreperson or supervisor     <NA>
Yes
## 4 Degree level or higher  19 or over             Self-employed     <NA>
<NA>
## 5      GCSE A-C or equiv          16 Foreperson or supervisor       50
Yes
##   rmany         rsect tunionsa
ansseca
## 1  <NA> Public sector      Yes                          7 Routine
occupations
## 2  <NA>          <NA>       No       4 Small employers and own account
workers
## 3     2 Public sector       No 2 Lower managerial and professional
occupations
## 4  <NA>          <NA>       No       4 Small employers and own account
workers
## 5    46 Public sector      Yes                     3 Intermediate
occupations
##   shourswk ssuper smany          ssect   religcat   famrelig   prtnrrlg
## 1       20     No  <NA> Private sector   Catholic   Catholic   Catholic
## 2     <NA>   <NA>  <NA>           <NA> Protestant Protestant Protestant
## 3       50   <NA>  <NA>           <NA> Protestant       <NA>   Catholic
## 4     <NA>   <NA>  <NA>           <NA>       <NA>       <NA>       <NA>
## 5       37    Yes     5  Public sector   Catholic   Catholic   Catholic
##      chattnd2 carehome caresep anyhcond   hcondact persinc2 hhldinc2
percapti1
## 1 Once a week       No     Yes       No       <NA>    22100    29900
7475
## 2 Once a week       No      No      Yes Yes, a lot     6760    10920
5460
## 3 Once a week      Yes      No       No       <NA>     9880    10920
3640
## 4        <NA>       No     Yes       No       <NA>     <NA>     <NA>
<NA>
## 5       Never       No      No       No       <NA>    27300    48100
24050
##   percapti2                        orient      polpart2      ruhappy
healthyr
```

```
## 1     14950 I am heterosexual or straight     Sinn Fein Fairly happy
Good
## 2     10920 I am heterosexual or straight          <NA> Fairly happy
Poor
## 3     10920 I am heterosexual or straight None of these   Very happy
Good
## 4      <NA>                          <NA>          <NA> Can't choose Can't
choose
## 5     48100 I am heterosexual or straight     Sinn Fein   Very happy
Excellent
##    racprej       racprejm             racprejf                   racownkd
## 1 A little       More now More in 5 years time Neither agree nor disagree
## 2 A little About the same       About the same                      Agree
## 3    A lot About the same       About the same                   Disagree
## 4    A lot About the same       About the same                   Disagree
## 5 A little       More now More in 5 years time          Strongly disagree
##   travsocdist eesocdist megsocdist mussocdist contegrp
uprejmeg
## 1           1         3          1          1        3   A little
prejudiced
## 2           4         2          2          6        0 Not prejudiced at
all
## 3           0         5          5          5        3   A little
prejudiced
## 4           0         0          0          0        2 Not prejudiced at
all
## 5           0         0          0          0        2 Not prejudiced at
all
##                        livwkeu1                      livwkeu2 umworker
mil10yrs
## 1 Neither welcome nor unwelcome Neither welcome nor unwelcome       No
3
## 2 Neither welcome nor unwelcome Neither welcome nor unwelcome       No
3
## 3                  Very welcome                Fairly welcome       No 5
Neither
## 4                  Very welcome                  Very welcome       No
7
## 5                  Very welcome                  Very welcome       No
7
##     miecono  micultur       rlrelago       rlrelfut
mxrlgngh
## 1         4 5 Neither About the same         Better Mixed religion
neighbourhood
## 2         2 5 Neither         Better About the same Mixed religion
neighbourhood
## 3 5 Neither         7 About the same          Worse Mixed religion
neighbourhood
## 4         6         8           <NA>           <NA>
<NA>
```

```
## 5         7         8         Better         Better Mixed religion
neighbourhood
##                   mxrlgwrk          ownmxsch        ninatid
omarrrlg
## 1 Mixed religion workplace Own religion only Northern Irish Would mind a
little
## 2 Mixed religion workplace Own religion only          Irish Would mind a
little
## 3 Mixed religion workplace Own religion only        British Would mind a
little
## 4                     <NA>              <NA>        British
<NA>
## 5 Mixed religion workplace Own religion only          Irish Would mind a
little
##         smarrrlg repmural repmur2              morrflag loymural loymur2
## 1 Would not mind       No      No                  Less       No     Yes
## 2 Would not mind       No      No                  Less       No      No
## 3 Would not mind       No      No                  More       No      No
## 4           <NA>     <NA>    <NA>                  <NA>     <NA>    <NA>
## 5 Would not mind       No      No About the same number      Yes     Yes
##                 morlflag flaglamp     comdiv commopen        avoidpwk
## 1 About the same number       No      A lot        4   Yes, probably
## 2                   Less       No Not at all        4  Definitely not
## 3                   More       No   A little        4 Yes, definitely
## 4                   <NA>     <NA>       <NA>        0            <NA>
## 5                   More       No Not at all        4    Probably not
##          avoidcwk       srelfrnd        srelngh
target1a
## 1  Definitely not           Half           Half
5
## 2  Definitely not           Half           Most
3
## 3 Yes, definitely Less than half Less than half 1 - definitely not
achieved
## 4            <NA>           <NA>           <NA>
5
## 5  Definitely not           Half           <NA>
3
##                      target2a                    target3a
## 1                           2                           7
## 2                           3                           6
## 3 1 - definitely not achieved 1 - definitely not achieved
## 4                           5                           5
## 5 1 - definitely not achieved                           6
##                      target4a                    target5a
## 1                           7                           7
## 2                           5                           3
## 3 1 - definitely not achieved 1 - definitely not achieved
## 4                           5                           5
## 5                           6                           5
##                      target6a target7a target8a                 irbrit
## 1                           7        5        5 More Irish than British
## 2                           2        4        3 More Irish than British
## 3 1 - definitely not achieved        4        4       British not Irish
## 4                           5        5        5                    <NA>
## 5                           5        5        8       Irish not British
##      uninatid      uninatst
## 1 Nationalist Fairly strong
## 2     Neither          <NA>
## 3     Neither          <NA>
## 4        <NA>          <NA>
## 5 Nationalist Fairly strong
##                                                         nirelnd2
## 1 To remain part of the United Kingdom, with devolved government
## 2                          To reunify with the rest of Ireland?
## 3 To remain part of the United Kingdom, with devolved government
## 4                                                           <NA>
## 5                          To reunify with the rest of Ireland?
##                               satmlas                   pubvoice
aachieve
## 1                 Very dissatisfied It is making no difference       A
little
## 2                 Very dissatisfied It is making no difference       A
little
## 3 Neither satisfied nor dissatisfied It is making no difference Nothing at
all
## 4                                <NA>                       <NA>
<NA>
## 5                 Very dissatisfied It is making no difference       A
little
##                  mixdprim                mixdgram                 mixdliv
## 1 Keep things as they are Keep things as they are Keep things as they are
## 2        Much more mixing        Much more mixing        Much more mixing
## 3         Bit more mixing         Bit more mixing Keep things as they are
## 4 Keep things as they are Keep things as they are Keep things as they are
## 5         Bit more mixing         Bit more mixing         Bit more mixing
##                  mixdwork                mixdleis                mixdmarr
suppint
## 1         Bit more mixing Keep things as they are Keep things as they are
1
## 2        Much more mixing        Much more mixing        Much more mixing
6
## 3         Bit more mixing Keep things as they are         Bit more mixing
4
## 4 Keep things as they are Keep things as they are Keep things as they are
0
## 5         Bit more mixing         Bit more mixing         Bit more mixing
6
##                                     whnuflg2
## 1                                      Never
## 2 Only for a few weeks around special events
## 3                                       Never
## 4                              Can't choose
## 5                                       Never
##                                     whentri2                 shopuflg
## 1                                       Never            Less willing
## 2 Only for a few weeks around special events Would make no difference
## 3                                       Never Would make no difference
## 4                              Can't choose Would make no difference
## 5                                       Never            Less willing
##                    shoptri riotngh                              whyriot
wtfactor
## 1             Less willing      No A specific incident usually sparks it
1.640327
## 2 Would make no difference      No                          Can't choose
1.093551
## 3 Would make no difference      No A specific incident usually sparks it
1.093551
## 4 Would make no difference    <NA>                          Can't choose
0.546776
## 5             Less willing      No            Paramilitaries organise it
1.093551
```

```
str(grel)
```

```
## 'data.frame':    1204 obs. of  133 variables:
##  $ serial     : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ househld   : num  4 2 3 1 2 2 1 1 2 1 ...
##  $ rage       : Factor w/ 78 levels "18","19","20",..: 27 69 9 55 23 27 59
6 72 39 ...
##  $ spage      : Factor w/ 67 levels "18","19","22",..: 33 62 7 NA 19 NA NA
NA NA NA ...
##  $ rsex       : Factor w/ 2 levels "Male","Female": 1 1 2 2 1 2 2 1 2 1
...
##  $ nadult     : Factor w/ 6 levels "1","2","3","4",..: 3 2 2 1 2 1 1 1 2 1
...
##  $ nkids      : Factor w/ 6 levels "0","1","2","3",..: 2 1 2 1 1 2 1 1 1 1
...
##  $ nelderly   : Factor w/ 3 levels "0","1","2": 1 3 1 2 1 1 2 1 3 2 ...
##  $ nfamily    : Factor w/ 9 levels "1","2","3","4",..: 4 2 3 1 2 2 1 1 2 1
...
##  $ rmarstat   : Factor w/ 6 levels "Single (never married)",..: 2 2 3 6 2
5 6 1 6 6 ...
##  $ livearea   : Factor w/ 92 levels "Less than 1 year",..: 40 85 35 43 3 7
21 35 58 19 ...
##  $ placeliv   : Factor w/ 5 levels "...a big city",..: 2 5 3 2 2 2 2 2 3 3
...
##  $ hincpast   : Factor w/ 3 levels "...fallen behind prices",..: 1 1 1 2 1
1 1 1 1 1 ...
##  $ intwww     : Factor w/ 2 levels "Yes","No": 1 2 1 1 1 1 2 1 2 1 ...
##  $ umineth    : Factor w/ 2 levels "Yes","No": 2 2 2 2 2 2 2 2 2 2 ...
##  $ eqnow1     : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 2 2 1 2 2 ...
##  $ eqnow2     : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 2 2 1 2 2 ...
##  $ eqnow3     : Factor w/ 2 levels "Yes","No": 2 1 1 NA 1 1 2 1 2 1 ...
##  $ eqnow4     : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 1 2 1 2 2 ...
##  $ eqnow5     : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 1 2 1 2 2 ...
##  $ eqnow6     : Factor w/ 2 levels "Yes","No": 2 2 1 NA 1 2 2 1 2 2 ...
##  $ eqnow7     : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 2 2 1 2 2 ...
##  $ eqnow8     : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 1 2 1 2 2 ...
##  $ eqnow9     : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 2 2 1 2 2 ...
##  $ eqnow10    : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 1 2 1 2 2 ...
##  $ eqnow11    : Factor w/ 2 levels "Yes","No": 2 1 1 NA 1 1 2 1 2 1 ...
##  $ eqpast1    : Factor w/ 3 levels "Treated better than 5 years ago",..: 2
2 2 NA 2 1 1 1 2 2 ...
##  $ eqpast2    : Factor w/ 3 levels "Treated better than 5 years ago",..: 2
2 3 NA 2 3 1 2 2 2 ...
##  $ eqpast3    : Factor w/ 3 levels "Treated better than 5 years ago",..: 1
2 3 NA 2 1 2 1 3 1 ...
##  $ eqpast4    : Factor w/ 3 levels "Treated better than 5 years ago",..: 1
2 1 NA 1 3 1 1 1 2 ...
##  $ eqpast5    : Factor w/ 3 levels "Treated better than 5 years ago",..: 2
2 1 NA 2 3 1 1 1 2 ...
##  $ eqpast6    : Factor w/ 3 levels "Treated better than 5 years ago",..: 2
2 2 NA 2 3 2 2 3 3 ...
##  $ eqpast7    : Factor w/ 3 levels "Treated better than 5 years ago",..: 1
2 2 NA 3 1 1 1 3 1 ...
##  $ eqpast8    : Factor w/ 3 levels "Treated better than 5 years ago",..: 2
2 2 NA 2 3 1 1 2 2 ...
##  $ eqpast9    : Factor w/ 3 levels "Treated better than 5 years ago",..: 2
2 2 NA 2 1 1 1 2 2 ...
##  $ eqpast10   : Factor w/ 3 levels "Treated better than 5 years ago",..: 2
2 2 NA 2 3 1 1 2 2 ...
##  $ eqpast11   : Factor w/ 3 levels "Treated better than 5 years ago",..: 2
2 1 NA 3 3 1 NA 2 1 ...
##  $ tenshort   : Factor w/ 7 levels "Own it outright",..: 2 1 6 1 2 6 1 2 1
4 ...
##  $ highqual   : Factor w/ 8 levels "Degree level or higher",..: 3 6 4 1 4
1 5 1 5 4 ...
##  $ tea        : Factor w/ 8 levels "15 or under",..: 2 1 2 5 2 5 3 5 1 2
...
##  $ work       : Factor w/ 4 levels "Employee","Foreperson or
supervisor",..: 1 3 2 3 2 1 1 1 2 1 ...
##  $ rhourswk   : Factor w/ 44 levels "4","6","9","10",..: 29 NA NA NA 34 19
NA NA NA 29 ...
##  $ rsuper     : Factor w/ 2 levels "Yes","No": 2 NA 1 NA 1 2 2 2 1 2 ...
##  $ rmany      : Factor w/ 50 levels "0","1","2","3",..: NA NA 3 NA 33 NA
NA NA 6 NA ...
##  $ rsect      : Factor w/ 4 levels "Public sector",..: 1 NA 1 NA 1 1 2 1 2
1 ...
##  $ tunionsa   : Factor w/ 2 levels "Yes","No": 1 2 2 2 1 2 1 2 1 1 ...
##  $ ansseca    : Factor w/ 10 levels "1.1 Large employers and higher
managerial occupations",..: 8 5 3 5 4 3 4 4 7 4 ...
##  $ shourswk   : Factor w/ 36 levels "1","4","6","7",..: 9 NA 30 NA 21 NA
NA NA NA NA ...
##  $ ssuper     : Factor w/ 2 levels "Yes","No": 2 NA NA NA 1 NA NA NA NA NA
...
##  $ smany      : Factor w/ 28 levels "0","1","2","3",..: NA NA NA NA 6 NA
NA NA NA NA ...
##  $ ssect      : Factor w/ 4 levels "Public sector",..: 2 NA NA NA 1 NA NA
NA NA NA ...
##  $ religcat   : Factor w/ 3 levels "Catholic","Protestant",..: 1 2 2 NA 1
2 2 2 2 3 ...
##  $ famrelig   : Factor w/ 3 levels "Catholic","Protestant",..: 1 2 NA NA 1
2 2 2 2 3 ...
##  $ prtnrrlg   : Factor w/ 3 levels "Catholic","Protestant",..: 1 2 1 NA 1
NA NA NA NA NA ...
##  $ chattnd2   : Factor w/ 8 levels "Several times a week",..: 2 2 2 NA 8 2
8 8 2 NA ...
##  $ carehome   : Factor w/ 2 levels "Yes","No": 2 2 1 2 2 2 2 2 2 2 ...
##  $ caresep    : Factor w/ 2 levels "Yes","No": 1 2 2 1 2 1 2 2 2 2 ...
##  $ anyhcond   : Factor w/ 2 levels "Yes","No": 2 1 2 2 2 2 1 2 1 2 ...
##  $ hcondact   : Factor w/ 3 levels "Yes, a lot","Yes, a little",..: NA 1
NA NA NA NA 1 NA 1 NA ...
##  $ persinc2   : Factor w/ 38 levels "260","780","1300",..: 26 12 15 NA 28
21 NA NA NA 22 ...
##  $ hhldinc2   : Factor w/ 36 levels "1300","1820",..: 27 14 14 NA 34 19 NA
NA NA 20 ...
##  $ percapti1  : Factor w/ 126 levels "910","1300","1430",..: 61 47 27 NA
113 64 NA NA NA 102 ...
##  $ percapti2  : Factor w/ 46 levels "3293.33333333333",..: 20 13 13 NA 44
NA NA NA NA NA ...
##  $ orient     : Factor w/ 4 levels "I am heterosexual or straight",..: 1 1
1 NA 1 1 1 1 1 1 ...
##  $ polpart2   : Factor w/ 7 levels "Democratic Unionist Party (DUP)",..: 2
NA 7 NA 2 7 3 7 1 7 ...
##  $ ruhappy    : Factor w/ 5 levels "Very happy","Fairly happy",..: 2 2 1 5
1 2 3 3 1 3 ...
##  $ healthyr   : Factor w/ 6 levels "Excellent","Good",..: 2 4 2 6 1 4 4 2
3 2 ...
##  $ racprej    : Factor w/ 3 levels "A lot","A little",..: 2 2 1 1 2 1 3 1
2 2 ...
##  $ racprejm   : Factor w/ 4 levels "More now","Less now",..: 1 3 3 3 1 1 3
3 3 3 ...
##  $ racprejf   : Factor w/ 4 levels "More in 5 years time",..: 1 3 3 3 1 1
3 3 2 2 ...
##  $ racownkd   : Factor w/ 5 levels "Strongly agree",..: 3 2 4 4 5 3 2 4 4
3 ...
##  $ travsocdist: num  1 4 0 0 0 2 4 0 0 4 ...
##  $ eesocdist  : Factor w/ 7 levels "0","1","2","3",..: 4 3 6 1 1 6 7 1 2 7
...
```

```
##  $ megsocdist : num  1 2 5 0 0 0 3 0 0 6 ...
##  $ mussocdist : Factor w/ 7 levels "0","1","2","3",..: 2 7 6 1 1 1 7 1 2 7
...
##  $ contegrp   : num  3 0 3 2 2 2 0 2 0 3 ...
##  $ uprejmeg   : Factor w/ 4 levels "Very prejudiced",..: 2 3 2 3 3 2 3 3 3
2 ...
##  $ livwkeu1   : Factor w/ 5 levels "Very welcome",..: 3 3 1 1 1 1 2 1 2 1
...
##  $ livwkeu2   : Factor w/ 5 levels "Very welcome",..: 3 3 2 1 1 3 2 1 2 1
...
##  $ umworker   : Factor w/ 2 levels "Yes","No": 2 2 2 2 2 2 2 2 2 2 ...
##  $ mil10yrs   : Factor w/ 11 levels "0 Extremely bad",..: 4 4 6 8 8 2 3 4
8 9 ...
##  $ miecono    : Factor w/ 11 levels "0 Extremely bad for economy",..: 5 3
6 7 8 1 2 1 9 8 ...
##  $ micultur   : Factor w/ 11 levels "0 Cultural life undermined",..: 6 6 8
9 9 3 4 11 9 8 ...
##  $ rlrelago   : Factor w/ 4 levels "Better","About the same",..: 2 1 2 NA
1 1 1 3 1 2 ...
##  $ rlrelfut   : Factor w/ 4 levels "Better","About the same",..: 1 2 3 NA
1 2 1 3 1 1 ...
##  $ mxrlgngh   : Factor w/ 3 levels "Own religion only",..: 2 2 2 NA 2 2 1
2 2 2 ...
##  $ mxrlgwrk   : Factor w/ 3 levels "Own religion only",..: 2 2 2 NA 2 2 1
2 2 2 ...
##  $ ownmxsch   : Factor w/ 3 levels "Own religion only",..: 1 1 1 NA 1 2 2
2 2 2 ...
##  $ ninatid    : Factor w/ 7 levels "British","Irish",..: 4 2 1 1 2 5 1 4 1
1 ...
##  $ omarrrlg   : Factor w/ 3 levels "Would mind a lot",..: 2 2 2 NA 2 3 3 3
2 2 ...
##  $ smarrrlg   : Factor w/ 3 levels "Would mind a lot",..: 3 3 3 NA 3 3 3 3
3 3 ...
##  $ repmural   : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 2 2 2 2 2 ...
##  $ repmur2    : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 2 2 1 2 1 ...
##  $ morrflag   : Factor w/ 3 levels "More","Less",..: 2 2 1 NA 3 3 3 2 3 3
...
##  $ loymural   : Factor w/ 2 levels "Yes","No": 2 2 2 NA 1 2 NA 2 2 2 ...
##  $ loymur2    : Factor w/ 2 levels "Yes","No": 1 2 2 NA 1 2 2 1 2 1 ...
##  $ morlflag   : Factor w/ 3 levels "More","Less",..: 3 2 1 NA 1 2 3 2 3 3
...
##  $ flaglamp   : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 2 2 1 2 2 ...
##  $ comdiv     : Factor w/ 3 levels "A lot","A little",..: 1 3 2 NA 3 3 2 3
2 2 ...
##   [list output truncated]
##  - attr(*, "variable.labels")= Named chr [1:133] "Serial number of
respondent" "s1Q1 How many people are there in your household?" "s1Q1 Age of
respondent" "s1Q1 Age of spouse/partner" ...
##   ..- attr(*, "names")= chr [1:133] "serial" "househld" "rage" "spage" ...
##  - attr(*, "codepage")= int 1252
```

- From the output, you can see that we are dealing with 1,204 observations over 133 variables.
- However not all variables will be useful during the analysis, so for our convenience we can simply create a smaller subset and store it in a separate object:

```
grel.data <- subset(grel[, c(1:3, 5, 7:8, 10, 12, 16:19, 38:39, 41, 47,52,
55, 60, 64, 66, 76:77, 80:83, 105:112)])

str(grel.data)

## 'data.frame':    1204 obs. of  35 variables:
##  $ serial  : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ househld: num  4 2 3 1 2 2 1 1 2 1 ...
##  $ rage    : Factor w/ 78 levels "18","19","20",..: 27 69 9 55 23 27 59 6
72 39 ...
##  $ rsex    : Factor w/ 2 levels "Male","Female": 1 1 2 2 1 2 2 1 2 1 ...
##  $ nkids   : Factor w/ 6 levels "0","1","2","3",..: 2 1 2 1 1 2 1 1 1 1
...
##  $ nelderly: Factor w/ 3 levels "0","1","2": 1 3 1 2 1 1 2 1 3 2 ...
##  $ rmarstat: Factor w/ 6 levels "Single (never married)",..: 2 2 3 6 2 5 6
1 6 6 ...
##  $ placeliv: Factor w/ 5 levels "...a big city",..: 2 5 3 2 2 2 2 2 3 3
...
##  $ eqnow1  : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 2 2 1 2 2 ...
##  $ eqnow2  : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 2 2 1 2 2 ...
##  $ eqnow3  : Factor w/ 2 levels "Yes","No": 2 1 1 NA 1 1 2 1 2 1 ...
##  $ eqnow4  : Factor w/ 2 levels "Yes","No": 2 2 2 NA 2 1 2 1 2 2 ...
##  $ tenshort: Factor w/ 7 levels "Own it outright",..: 2 1 6 1 2 6 1 2 1 4
...
##  $ highqual: Factor w/ 8 levels "Degree level or higher",..: 3 6 4 1 4 1 5
1 5 4 ...
##  $ work    : Factor w/ 4 levels "Employee","Foreperson or supervisor",..:
1 3 2 3 2 1 1 1 2 1 ...
##  $ ansseca : Factor w/ 10 levels "1.1 Large employers and higher
managerial occupations",..: 8 5 3 5 4 3 4 4 7 4 ...
##  $ religcat: Factor w/ 3 levels "Catholic","Protestant",..: 1 2 2 NA 1 2 2
2 2 3 ...
##  $ chattnd2: Factor w/ 8 levels "Several times a week",..: 2 2 2 NA 8 2 8
8 2 NA ...
##  $ persinc2: Factor w/ 38 levels "260","780","1300",..: 26 12 15 NA 28 21
NA NA NA 22 ...
##  $ orient  : Factor w/ 4 levels "I am heterosexual or straight",..: 1 1 1
NA 1 1 1 1 1 1 ...
##  $ ruhappy : Factor w/ 5 levels "Very happy","Fairly happy",..: 2 2 1 5 1
2 3 3 1 3 ...
##  $ contegrp: num  3 0 3 2 2 2 0 2 0 3 ...
##  $ uprejmeg: Factor w/ 4 levels "Very prejudiced",..: 2 3 2 3 3 2 3 3 3 2
...
##  $ umworker: Factor w/ 2 levels "Yes","No": 2 2 2 2 2 2 2 2 2 2 ...
##  $ mil10yrs: Factor w/ 11 levels "0 Extremely bad",..: 4 4 6 8 8 2 3 4 8 9
...
```

```
##  $ miecono : Factor w/ 11 levels "0 Extremely bad for economy",..: 5 3 6 7
8 1 2 1 9 8 ...
##  $ micultur: Factor w/ 11 levels "0 Cultural life undermined",..: 6 6 8 9
9 3 4 11 9 8 ...
##  $ target1a: Factor w/ 10 levels "1 - definitely not achieved",..: 5 3 1 5
3 3 3 1 4 3 ...
##  $ target2a: Factor w/ 10 levels "1 - definitely not achieved",..: 2 3 1 5
1 2 3 2 4 2 ...
##  $ target3a: Factor w/ 10 levels "1 - definitely not achieved",..: 7 6 1 5
6 10 4 1 4 3 ...
##  $ target4a: Factor w/ 10 levels "1 - definitely not achieved",..: 7 5 1 5
6 9 7 3 5 5 ...
##  $ target5a: Factor w/ 10 levels "1 - definitely not achieved",..: 7 3 1 5
5 5 5 3 5 5 ...
##  $ target6a: Factor w/ 10 levels "1 - definitely not achieved",..: 7 2 1 5
5 6 5 5 6 6 ...
##  $ target7a: Factor w/ 10 levels "1 - definitely not achieved",..: 5 4 4 5
5 5 5 5 6 6 ...
##  $ target8a: Factor w/ 10 levels "1 - definitely not achieved",..: 5 3 4 5
8 5 5 6 5 7 ...
```

```
require(rmarkdown)
render("R_DS.qmd", pdf_document(toc=T, number_sections = T))
```