Lecture 3

Introduction to R

Assoc. Prof. Dr. Burkay Genç

2024-03-04

Packages used in these slides

```
# For creating these slides
library(knitr)

# For nicely printed data frames
library(tibble)

# For many data wrangling operations
library(dplyr)
```

Seed used in these slides

```
# For result reproducibility
set.seed(1024)
```

Data Frames

Data Frames

- Similar to matrices
  - But each column can have a different type
- Indeed, more like lists

Name	Surname	Age
Burkay	Genç	45
Hasan	Kaya	55
Ayşe	Çam	40
Fatma	Gürgen	23

Data Frames

- Each row is an **observation** (or case (or record))
- Each column is a **feature** (or an attribute (or a variable))

```
my.dataset <- data.frame(site = c('A', 'B', 'A', 'A', 'B', 'B'),
                         season = c('Winter', 'Summer', 'Summer',
                                    'Spring', 'Fall', 'Spring'),
                         pH = c(7.4, 6.3, 8.6, 7.2, 8.9, 8.4))
kable(my.dataset)
```

site	season	pH
A	Winter	7.4
B	Summer	6.3
A	Summer	8.6
A	Spring	7.2
B	Fall	8.9
B	Spring	8.4

```
my.dataset[3, 2]
```

```
## [1] "Summer"
```

Data Frames - Kable

```
kable(my.dataset)
```

site	season	pH
A	Winter	7.4
B	Summer	6.3
A	Summer	8.6
A	Spring	7.2
B	Fall	8.9
B	Spring	8.4

Note that the `kable()` function is used to prettify the table output. Normally, you **DON'T** need to use it to print tables.

```
my.dataset
```

```
##   site season  pH
## 1    A Winter 7.4
## 2    B Summer 6.3
## 3    A Summer 8.6
## 4    A Spring 7.2
## 5    B   Fall 8.9
## 6    B Spring 8.4
```

Data Frames - Indexing

```
my.dataset$site
```

```
## [1] "A" "B" "A" "A" "B" "B"
```

```
my.dataset$pH[3]
```

```
## [1] 8.6
```

```
# Rows are data frames
my.dataset[2,]
```

```
##   site season  pH
## 2    B Summer 6.3
```

```
# Columns are vectors
my.dataset[,3]
```

```
## [1] 7.4 6.3 8.6 7.2 8.9 8.4
```

Data Frames - Indexing

This returns a **vector**.

```
my.dataset[,3]
```

```
## [1] 7.4 6.3 8.6 7.2 8.9 8.4
```

This returns a **data frame**!

```
my.dataset[3]
```

```
##    pH
## 1 7.4
## 2 6.3
## 3 8.6
## 4 7.2
## 5 8.9
## 6 8.4
```

Data Frames - `head` and `tail`

You can print first or last few rows of a data frame:

```
head(my.dataset, 3)
```

```
##   site season  pH
## 1    A Winter 7.4
## 2    B Summer 6.3
## 3    A Summer 8.6
```

```
tail(my.dataset, 3)
```

```
##   site season  pH
## 4    A Spring 7.2
## 5    B   Fall 8.9
## 6    B Spring 8.4
```

Data Frames - Indexing

This returns a **vector**.

```
my.dataset$pH
```

```
## [1] 7.4 6.3 8.6 7.2 8.9 8.4
```

This returns a **data frame**!

```
my.dataset["pH"]
```

```
##    pH
## 1 7.4
## 2 6.3
## 3 8.6
## 4 7.2
## 5 8.9
## 6 8.4
```

Data Frames - Indexing

This returns a **vector**.

```
my.dataset[, "pH"]
```

```
## [1] 7.4 6.3 8.6 7.2 8.9 8.4
```

This returns a **data frame**!

```
my.dataset[, "pH", drop = F]
```

```
##    pH
## 1 7.4
## 2 6.3
## 3 8.6
## 4 7.2
## 5 8.9
## 6 8.4
```

Data Frame - Subsetting

```
kable(my.dataset[my.dataset$pH > 7, ])
```

	site	season	pH
1	A	Winter	7.4
3	A	Summer	8.6
4	A	Spring	7.2
5	B	Fall	8.9
6	B	Spring	8.4

```
kable(my.dataset[my.dataset$pH > 7 & my.dataset$site == "A", ])
```

	site	season	pH
1	A	Winter	7.4
3	A	Summer	8.6
4	A	Spring	7.2

Data Frame - Subsetting

```
my.dataset[my.dataset$pH > 7 & my.dataset$site == "A", "season"]
```

```
## [1] "Winter" "Summer" "Spring"
```

Data Frame - attach, detach

- You can not directly access data frame columns

```
my.dataset[pH > 7,]
```

```
## Error in eval(expr, envir, enclos): object 'pH' not found
```

- but you can do it, if you **attach** the data frame

```
attach(my.dataset)
my.dataset[pH > 7,]
```

```
##   site season  pH
## 1    A Winter 7.4
## 3    A Summer 8.6
## 4    A Spring 7.2
## 5    B   Fall 8.9
## 6    B Spring 8.4
```

```
detach(my.dataset)
my.dataset[pH > 7,]
```

```
## Error in eval(expr, envir, enclos): object 'pH' not found
```

Data Frame - adding columns

```
my.dataset$newColumn <- c(T, F, F, T, T, F)
kable(my.dataset)
```

site	season	pH	newColumn
A	Winter	7.4	TRUE
B	Summer	6.3	FALSE
A	Summer	8.6	FALSE
A	Spring	7.2	TRUE
B	Fall	8.9	TRUE
B	Spring	8.4	FALSE

Data Frame - deleting columns

```
my.dataset$newColumn <- NULL
kable(my.dataset)
```

site	season	pH
A	Winter	7.4
B	Summer	6.3
A	Summer	8.6
A	Spring	7.2
B	Fall	8.9
B	Spring	8.4

Data Frame - adding rows

This is a bit messy!

- define a new data frame
- add it as a new row

```
my.dataset[6, ] <- data.frame(
   site = "B",
   season = "Winter",
   pH = 6.3)
kable(my.dataset)
```

site	season	pH
A	Winter	7.4
B	Summer	6.3
A	Summer	8.6
A	Spring	7.2
B	Fall	8.9
B	Winter	6.3

You could also `rbind` the above.

- define each feature of a new row
  - one by one

```
my.dataset[7, "site"] <- "A"
my.dataset[7, "season"] <- "Fall"
my.dataset[7, "pH"] <- 6.4
kable(my.dataset)
```

site	season	pH
A	Winter	7.4
B	Summer	6.3
A	Summer	8.6
A	Spring	7.2
B	Fall	8.9
B	Winter	6.3
A	Fall	6.4

Data Frame - deleting rows

```
kable(my.dataset)
```

site	season	pH
A	Winter	7.4
B	Summer	6.3
A	Summer	8.6
A	Spring	7.2
B	Fall	8.9
B	Winter	6.3
A	Fall	6.4

```
my.dataset <- my.dataset[-(2:3),]
kable(my.dataset)
```

	site	season	pH
1	A	Winter	7.4
4	A	Spring	7.2
5	B	Fall	8.9
6	B	Winter	6.3
7	A	Fall	6.4

Data Frame - dimensions

```
kable(my.dataset)
```

site	season	pH
A	Winter	7.4
B	Summer	6.3
A	Summer	8.6
A	Spring	7.2
B	Fall	8.9
B	Winter	6.3
A	Fall	6.4

```
nrow(my.dataset)
```

```
## [1] 7
```

```
ncol(my.dataset)
```

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

```
dim(my.dataset)
```

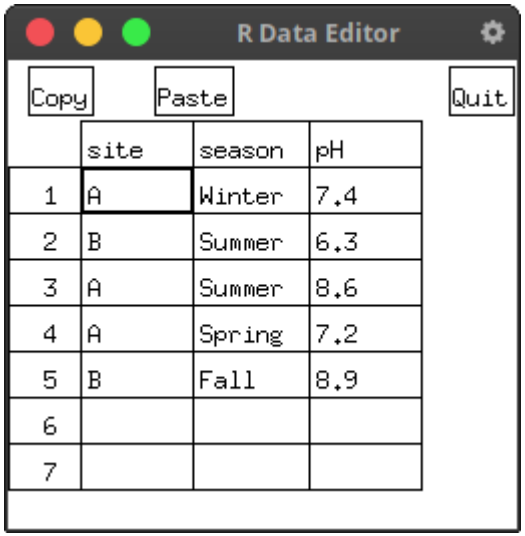
```
## [1] 7 3
```

Data Frame - editing

You can edit small data frames in R through a GUI:

```
my.dataset <- edit(my.dataset)
```

I strongly recommend you to avoid this!



Data Frame - column names

```
kable(my.dataset)
```

site	season	pH
A	Winter	7.4
B	Summer	6.3
A	Summer	8.6
A	Spring	7.2
B	Fall	8.9
B	Winter	6.3
A	Fall	6.4

```
names(my.dataset)
```

```
## [1] "site"   "season" "pH"
```

```
colnames(my.dataset)
```

```
## [1] "site"   "season" "pH"
```

```
names(my.dataset) <-
   c("Site", "Season", "pH_val")
kable(my.dataset)
```

Site	Season	pH_val
A	Winter	7.4
B	Summer	6.3
A	Summer	8.6
A	Spring	7.2
B	Fall	8.9
B	Winter	6.3
A	Fall	6.4

Data Frame - string coercion

~~Data frames automatically convert strings into factors~~:

Since 4.0, data frame do not convert strings into factors anymore:

```
df <- data.frame(name = c("burkay", "ali", "burkay", "veli"))
df$name
```

```
## [1] "burkay" "ali"    "burkay" "veli"
```

But, you can force it to do so:

```
df <- data.frame(name = c("burkay", "ali", "burkay", "veli"),
                 stringsAsFactors = T)
df$name
```

```
## [1] burkay ali    burkay veli
## Levels: ali burkay veli
```

Tibbles

Tibbles

- Provided by the `tibble` package

```
library(tibble)
```

- An advanced version of data frames
  - tibbles **never change character columns into factors** ~~as data frames do by default~~
    - this is now also the default behaviour for data frame!
  - data frame are more relaxed in terms of **naming of the columns**
  - the **printing** methods of tibbles are more convenient particularly with large datasets
    - tibbles do not try to print everything on the screen

Tibbles

```
my.dataset <- data.frame(
   site = c('A', 'B', 'A', 'A', 'B'),
   season = c('Winter', 'Summer', 'Summer', 'Spring', 'Fall'),
   pH = c(7.4, 6.3, 8.6, 7.2, 8.9))
my.dataset
```

```
##   site season  pH
## 1    A Winter 7.4
## 2    B Summer 6.3
## 3    A Summer 8.6
## 4    A Spring 7.2
## 5    B   Fall 8.9
```

```
my.tibble <- tibble(
   site = c('A', 'B', 'A', 'A', 'B'),
   season = c('Winter', 'Summer', 'Summer', 'Spring', 'Fall'),
   pH = c(7.4, 6.3, 8.6, 7.2, 8.9))
my.tibble
```

```
## # A tibble: 5 × 3
##   site  season    pH
##   <chr> <chr>  <dbl>
## 1 A     Winter   7.4
## 2 B     Summer   6.3
## 3 A     Summer   8.6
## 4 A     Spring   7.2
## 5 B     Fall     8.9
```

Tibbles

You can construct a tibble from a data frame:

```
my.dataset <- data.frame(
   site = c('A', 'B', 'A', 'A', 'B'),
   season = c('Winter', 'Summer', 'Summer', 'Spring', 'Fall'),
   pH = c(7.4, 6.3, 8.6, 7.2, 8.9))

my.tibble <- tibble(my.dataset)
my.tibble
```

```
## # A tibble: 5 × 3
##   site  season    pH
##   <chr> <chr>  <dbl>
## 1 A     Winter   7.4
## 2 B     Summer   6.3
## 3 A     Summer   8.6
## 4 A     Spring   7.2
## 5 B     Fall     8.9
```

Tibbles

You cannot use shorthand to access a column with tibbles:

```
my.tibble
```

```
## # A tibble: 5 × 3
##   site  season    pH
##   <chr> <chr>  <dbl>
## 1 A     Winter   7.4
## 2 B     Summer   6.3
## 3 A     Summer   8.6
## 4 A     Spring   7.2
## 5 B     Fall     8.9
```

```
my.tibble$sea
```

```
## Warning: Unknown or uninitialised column: `sea`.
```

```
## NULL
```

```
my.dataset$sea
```

```
## [1] "Winter" "Summer" "Summer" "Spring" "Fall"
```

Tibbles - Column generation

- Tibbles can generate new columns on the fly from other columns

```
dat <- tibble(TempCels = sample(-10:40, size = 100, replace = TRUE),
              TempFahr = TempCels * 9 / 5 + 32,           # <<<<<<<
              Location = rep(letters[1:2], each = 50))
dat
```

```
## # A tibble: 100 × 3
##    TempCels TempFahr Location
##       <int>    <dbl> <chr>
##  1       10     50   a
##  2       11     51.8 a
##  3        2     35.6 a
##  4       21     69.8 a
##  5       36     96.8 a
##  6       34     93.2 a
##  7        5     41   a
##  8       -4     24.8 a
##  9       -3     26.6 a
## 10       32     89.6 a
## # i 90 more rows
```

Tibbles - Output

Data frames try to print everything on the screen

```
data(iris)
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 24           5.1         3.3          1.7         0.5     setosa
## 25           4.8         3.4          1.9         0.2     setosa
## 26           5.0         3.0          1.6         0.2     setosa
## 27           5.0         3.4          1.6         0.4     setosa
## 28           5.2         3.5          1.5         0.2     setosa
## 29           5.2         3.4          1.4         0.2     setosa
## 30           4.7         3.2          1.6         0.2     setosa
## 31           4.8         3.1          1.6         0.2     setosa
## 32           5.4         3.4          1.5         0.4     setosa
## 33           5.2         4.1          1.5         0.1     setosa
```

Tibbles - Output

Tibbles are much smarter and only print what you can see:

```
iris_tibble <- tibble(iris)
iris_tibble
```

```
## # A tibble: 150 × 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # i 140 more rows
```

Tibbles - Output

You can force limits on the print output:

```
print(iris_tibble, n = 5, width = 50)
```

```
## # A tibble: 150 × 5
##   Sepal.Length Sepal.Width Petal.Length
##          <dbl>       <dbl>        <dbl>
## 1          5.1         3.5          1.4
## 2          4.9         3            1.4
## 3          4.7         3.2          1.3
## 4          4.6         3.1          1.5
## 5          5           3.6          1.4
## # i 145 more rows
## # i 2 more variables: Petal.Width <dbl>,
## #   Species <fct>
```

```
print(iris_tibble, n = 3, width = 30)
```

```
## # A tibble: 150 × 5
##   Sepal.Length Sepal.Width
##          <dbl>       <dbl>
## 1          5.1         3.5
## 2          4.9         3
## 3          4.7         3.2
## # i 147 more rows
## # i 3 more variables:
## #   Petal.Length <dbl>,
## #   Petal.Width <dbl>,
## #   Species <fct>
```

Tibbles - Subsetting

Subsetting works differently in data frames and tibbles:

```
iris[1:15, "Petal.Width"]
```

```
##  [1] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 0.2 0.2 0.1 0.1 0.2
```

```
iris_tibble[1:15, "Petal.Width"]
```

```
## # A tibble: 15 × 1
##    Petal.Width
##          <dbl>
##  1         0.2
##  2         0.2
##  3         0.2
##  4         0.2
##  5         0.2
##  6         0.4
##  7         0.3
##  8         0.2
##  9         0.2
## 10         0.1
## 11         0.2
## 12         0.2
## 13         0.1
## 14         0.1
## 15         0.2
```

- There is no need to `drop = F`
- You always get a new tibble after subsetting

Tibbles - Subsetting

```
iris$Species[1:10]
```

```
##  [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
iris_tibble$Species[1:10]
```

```
##  [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

Tibbles - Subsetting

```
iris[1:10,]$Species
```

```
##  [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
iris_tibble[1:10,]$Species
```

```
##  [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

Tibbles - Subsetting

```
iris[1:10, "Species"]
```

```
##  [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
iris_tibble[1:10, "Species"]
```

```
## # A tibble: 10 × 1
##    Species
##    <fct>
##  1 setosa
##  2 setosa
##  3 setosa
##  4 setosa
##  5 setosa
##  6 setosa
##  7 setosa
##  8 setosa
##  9 setosa
## 10 setosa
```

Tibbles - Subsetting

```
iris[1:10, 5]
```

```
##  [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
iris_tibble[1:10, 5]
```

```
## # A tibble: 10 × 1
##    Species
##    <fct>
##  1 setosa
##  2 setosa
##  3 setosa
##  4 setosa
##  5 setosa
##  6 setosa
##  7 setosa
##  8 setosa
##  9 setosa
## 10 setosa
```

Subsetting Hell

- This is annoying!
- Subsetting can work almost randomly in R
  - Actually it is not, it appears to be so to beginners
    - Data frames always convert to vectors
    - Unless `drop=F`
  - `$` acts like a forced conversion to vector
  - Otherwise, tibbles produce tibbles when subset
- You have to be very careful about this
- My advice:
  - Stick to one type of subsetting and always use that

Dplyr

Dplyr

- `dplyr` is a very useful package for data manipulation
- We will see it in detail later in the course
- But some examples:

```
## Petal widths and lengths of setosas
select(filter(iris_tibble,
              Species == "setosa"),
       Petal.Width, Petal.Length)
```

```
## # A tibble: 50 × 2
##    Petal.Width Petal.Length
##          <dbl>        <dbl>
##  1         0.2          1.4
##  2         0.2          1.4
##  3         0.2          1.3
##  4         0.2          1.5
##  5         0.2          1.4
##  6         0.4          1.7
##  7         0.3          1.4
##  8         0.2          1.5
##  9         0.2          1.4
## 10         0.1          1.5
## # i 40 more rows
```

Dplyr

- `select` selects the desired columns from the dataset
- `filter` selects the matching rows from the dataset
- `mutate` allows creating new columns from existing ones
- `summarise` provides summary statistics
- `group_by` allows grouping rows and obtaining summary statistics
- there are many more useful functions

Dplyr - pipe operator

- `dplyr` also declares the pipe operator `%>%`
- the pipe operator makes the left hand side the first argument of the right hand side
- `T %>% foo(a, b)` is the same as `foo(T, a, b)`

```
iris_tibble %>%
   filter(Species == "setosa") %>%
   select(Petal.Width, Petal.Length)
```

```
## # A tibble: 50 × 2
##    Petal.Width Petal.Length
##          <dbl>        <dbl>
##  1         0.2          1.4
##  2         0.2          1.4
##  3         0.2          1.3
##  4         0.2          1.5
##  5         0.2          1.4
##  6         0.4          1.7
##  7         0.3          1.4
##  8         0.2          1.5
##  9         0.2          1.4
## 10         0.1          1.5
## # i 40 more rows
```

Dplyr - pipe operator

```
foo_d(foo_c(foo_b(foo_a(x))))
```

vs.

```
x %>% foo_a() %>% foo_b() %>% foo_c() %>% foo_d()
```

vs.

```
x %>%
   foo_a() %>%
   foo_b() %>%
   foo_c() %>%
   foo_d()
```

pipe examples

```
x <- tibble(a=1:100, b=100:1, c=sample(1:100, 100))
x %>%
  filter(a > 60) %>%
  filter(b < 20)
```

```
## # A tibble: 19 × 3
##        a     b     c
##    <int> <int> <int>
##  1    82    19    59
##  2    83    18    37
##  3    84    17    67
##  4    85    16    69
##  5    86    15    71
##  6    87    14    90
##  7    88    13    39
##  8    89    12    99
##  9    90    11    10
## 10    91    10    22
## 11    92     9    34
## 12    93     8    61
## 13    94     7    14
## 14    95     6    13
## 15    96     5    33
## 16    97     4    73
## 17    98     3    27
## 18    99     2    82
## 19   100     1    60
```

pipe examples

```
x %>%
  filter(a * b > 1000) %>%
  select(c)
```

```
## # A tibble: 78 × 1
##        c
##    <int>
##  1    17
##  2    66
##  3    44
##  4    51
##  5    55
##  6    86
##  7    31
##  8    15
##  9    41
## 10    65
## # i 68 more rows
```

pipe examples

```
x %>%
  filter(a * b > 1000) %>%
  select(c) %>%
  mean()
```

```
## Warning in mean.default(.): argument is not numeric or logical: returning NA
```

```
## [1] NA
```

- `mean` requires a vector but received a tibble.
- What to do now?

pipe examples

```
x %>%
  filter(a * b > 1000) %>%
  pull(c) %>%
  mean()
```

```
## [1] 52.08974
```

- `pull` gets a vector out of a tibble

Saving and Loading

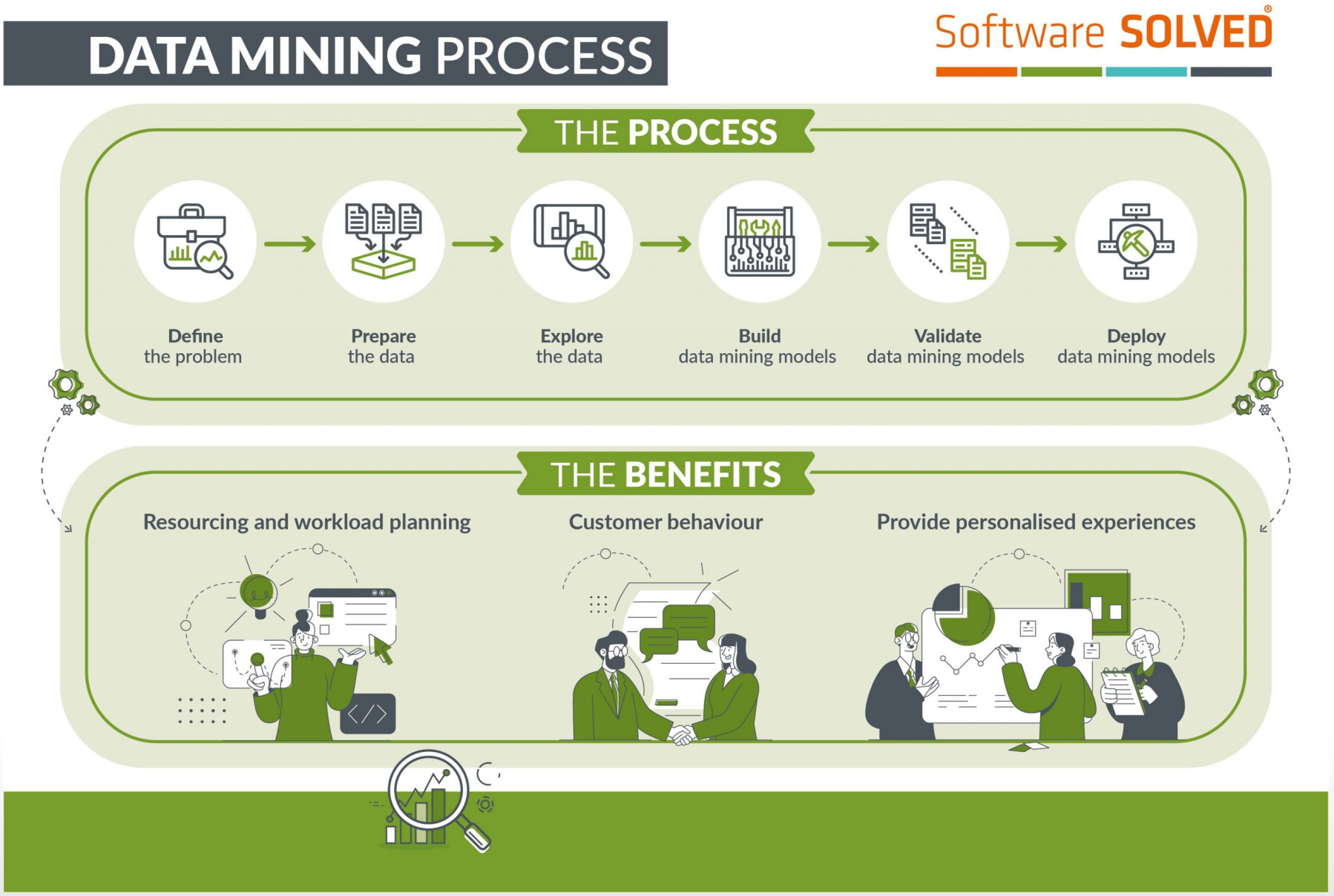
- It is useful to save and load very large R objects
- This is much faster than reading from other data types
- Save objects `f` and `my.dataset` into `mysession.RData`

```
save(f, my.dataset, file='mysession.RData')
```

- Load objects stored in `mysession.RData`

```
load('mysession.RData')
```

Workflow


DATA MINING PROCESS
Software SOLVED
THE PROCESS
Define
the problem
Prepare
the data
Explore
the data
Build
data mining models
Validate
data mining models
Deploy
data mining models
THE BENEFITS
Resourcing and workload planning
Customer behaviour
Provide personalised experiences

Data Types

- Data can come in many flavors
  - Text, numbers, dates, images, videos, audio, maps, …
- Most data mining (machine learning) algorithms can only handle tabular data
- Data taxonomy
  - Continuous (quantitative)
    - Integer
    - Numeric (real)
  - Discrete (categorical)
    - Nominal (incomparable)
    - Ordinal (comparable)
- All data can be reduced to one or more of these

Data Sources

- Text files (.csv)
- Databases (ODB, etc.)
- Spreadsheets (.xls)
- Other formats (SPSS, etc.)

Text Files

- `x.csv`

```
ID, Name, Age
23424, Ana, 45
11234, Charles, 23
77654, Susanne, 76
```

- We will use the `readr` package

```
library(readr)
dat <- read_csv("x.csv")
```

```
## Rows: 3 Columns: 3
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (1): Name
## dbl (2): ID, Age
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dat
```

```
## # A tibble: 3 × 3
##      ID Name      Age
##   <dbl> <chr>   <dbl>
## 1 23424 Ana        45
## 2 11234 Charles    23
## 3 77654 Susanne    76
```

Text Files

- `x.txt`

```
ID Name Age Phone
23424 Ana 45 ???
11234 Charles 23 3456789
77654 Susanne 76 2345678
```

- We will use `read_delim()`

```
dat <- read_delim("x.txt", delim = " ", na = "???")
```

```
## Rows: 3 Columns: 4
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: " "
## chr (1): Name
## dbl (3): ID, Age, Phone
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dat
```

```
## # A tibble: 3 × 4
##      ID Name      Age   Phone
##   <dbl> <chr>   <dbl>   <dbl>
## 1 23424 Ana        45      NA
## 2 11234 Charles    23 3456789
## 3 77654 Susanne    76 2345678
```

- You could use a vector for `na`

Database Connections

```
library(DBI)
library(RMySQL)

# The DBMS-specific code starts here
drv <- dbDriver("MySQL") # Loading the MySQL driver
con <- dbConnect(drv,dbname="transDB", # connecting to the DBMS
                 username="myuser",password="mypasswd",
                 host="localhost")
# The DBMS-specific code ends here

# getting the results of an SQL query as a data frame
data <- dbGetQuery(con,"SELECT * FROM clients")

# closing up stuff
dbDisconnect(con)
dbUnloadDriver(drv)
```

Database Connections - dplyr

```
library(RMySQL)
library(dplyr)
dbConn <- src_mysql("sonae",
                    host = "localhost",
                    user = "prodUser",
                    password = "myPassword")

sensors <- tbl(dbConn,"sensor_values")

sensors %>%
   filter(sid==274,value > 100) %>%
   select(time,value)
```

```
Source:   query [?? x 2]
Database: mysql 5.7.14 [prodUser@localhost:/sonae]

                   time   value
                  <chr>   <dbl>
1  2009-04-01 06:31:56  100.60
2  2009-04-01 06:32:04  103.11
3  2009-04-01 06:38:21  104.05
4  2009-04-01 06:38:29  103.87
5  2009-04-01 06:44:46  101.29
6  2009-04-01 06:44:54  100.16
7  2009-04-01 08:00:01  100.25
8  2009-04-01 08:55:52  101.64
9  2009-04-01 09:00:14  100.44
10 2009-04-01 09:11:50  102.33
# ... with more rows
```

Spreadsheets

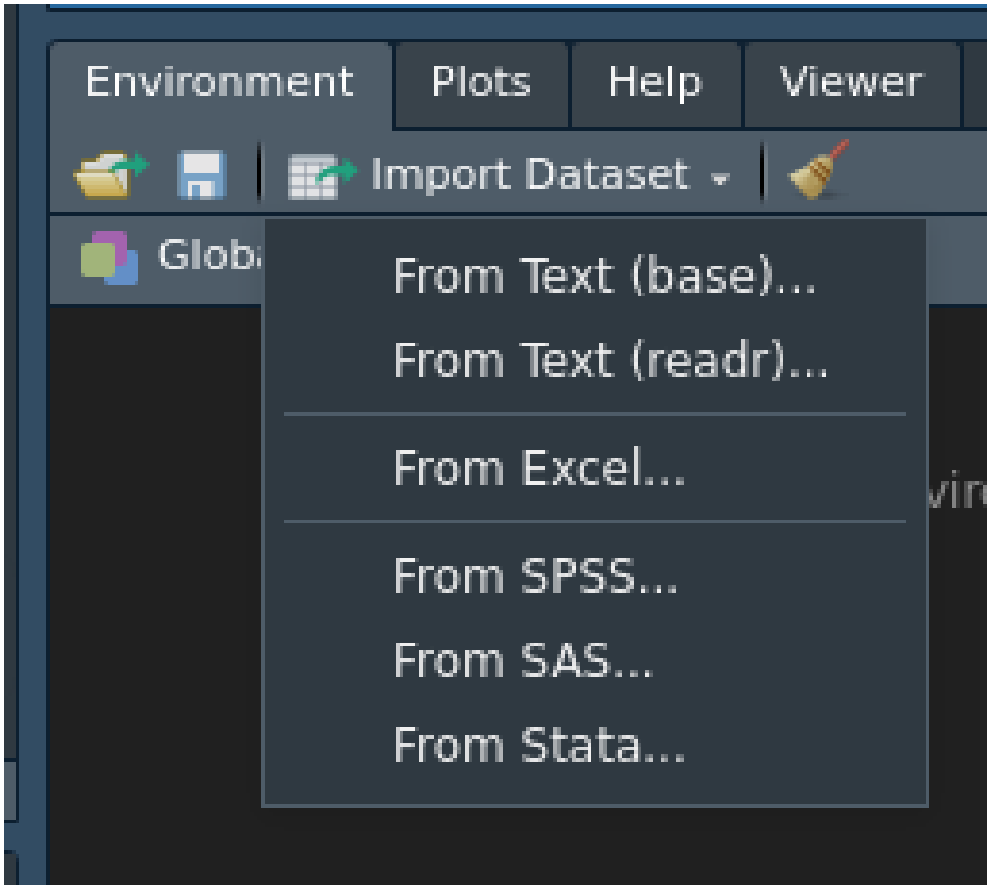
- Read from clipboard
  - Open the file in the spreadsheet editor (Excel?)
  - Copy the part that you want to import
  - Run the following command in R

```
d <- read.table("clipboard",
                header = TRUE)
```

- Read from the file

```
library(readxl)
fc <- "c:\\Documents and Settings\\xpto\\My
Documents\\calc.xls"
dat <- read_excel(fc, sheet = 1)
```

- Use RStudio



Other formats

- Use the `foreign` package

Tasks

Write a script to do the following things in a batch

1. Find a dataset in CSV format and load it into R
   - Print out the number of rows and columns
   - Convert the dataset into a tibble
   - Print the first 20 rows of the dataset
   - Save the object as `csv_object.dat`
2. Find a dataset in XLS format and load it into R
   - Print out the number of rows and columns
   - Convert the dataset into a tibble
   - Create a new tibble from the first 3 columns and 20 rows
   - Save the original and trimmed tibbles as `xls_object.dat`
3. Remove all objects from the environment
4. Load all objects back from the `.dat` files