Business Intelligence

G. Mudare

Special data types

Special data types

BELGIUM CAMPUS iTversity
It's the way we're wired

- A dataframe can contain six types of data. These are summarized in the table below:

Data type	Description	Example
numeric	Any number	c(1, 12.3491, 10/2, 10*6)
character	Character strings	c("E. saligna", "HFE", "a b c")
factor	Categorical variable	factor(c("Control","Fertilized","Irrigated"))
logical	Either TRUE or FALSE	10 == 100/10
Date	Special Date class	as.Date("2010-6-21")
POSIXct	Special Date-time class	Sys.time()

- R has a very useful built-in data type to represent missing values. This is represented by NA (Not Available)

Working with factors

- The *factor* data type is used to represent qualitative, categorical data.
- When reading data from file, for example with read.csv, **R** will automatically convert any variable to a factor if it is unable to convert it to a numeric variable.
- You can use as.factor to convert it to a factor if its already numeric

```
# Read pupae data
pupae <- read.csv("pupae.csv")

# This dataset contains a temperature (T_treatment) and CO2 treatment (CO2_treatment).
# Both should logically be factors, however, CO2_treatment is read as numeric:
str(pupae)

## 'data.frame': 84 obs. of  5 variables:
##  $ T_treatment  : Factor w/ 2 levels "ambient","elevated": 1 1 1 1 1 1 1 1 1 1 ...
##  $ CO2_treatment: int  280 280 280 280 280 280 280 280 280 280 ...
##  $ Gender       : int  0 1 0 0 0 1 0 1 0 1 ...
```

Working with factors


```
##  $ PupalWeight  : num  0.244 0.319 0.221 0.28 0.257 0.333 0.275 0.312 0.254 0.356 ...
##  $ Frass        : num  1.9 2.77 NA 2 1.07 ...

# To convert it to a factor, we use:
pupae$CO2_treatment <- as.factor(pupae$CO2_treatment)


# Compare with the above,
str(pupae)

## 'data.frame': 84 obs. of  5 variables:
##  $ T_treatment  : Factor w/ 2 levels "ambient","elevated": 1 1 1 1 1 1 1 1 1 1 ...
##  $ CO2_treatment: Factor w/ 2 levels "280","400": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Gender       : int  0 1 0 0 0 1 0 1 0 1 ...
##  $ PupalWeight  : num  0.244 0.319 0.221 0.28 0.257 0.333 0.275 0.312 0.254 0.356 ...
##  $ Frass        : num  1.9 2.77 NA 2 1.07 ...
```

Working with factors

- A factor variable has a number of 'levels', which are the text values that the variable has in the dataset.

```
levels(allom$species)

## [1] "PIMO" "PIPO" "PSME"
```

This Shows the three unique species in this dataset

We can count the number of rows in the dataframe for each species

```
table(allom$species)

##
##  PIMO  PIPO  PSME
##    19    22    22
```

Working with factors

- when the dataframe is read, the levels are assigned based on alphabetical order(Often, not very logical)

```
allom$species <- factor(allom$species, levels=c("PSME","PIMO","PIPO"))
```

We can generate new factors, and add them to the dataframe

```
# Add a new variable to allom:  'small' when diameter is less than 10, 'large' otherwise.
allom$treeSizeClass <- factor(ifelse(allom$diameter < 10, "small", "large"))

# Now, look how many trees fall in each class.
# Note that somewhat confusingly, 'large' is printed before 'small'.
# Once again, this is because the order of the factor levels is alphabetical by default.
table(allom$treeSizeClass)
```

```
##
## large small
##    56     7
```

Working with factors

BELGIUM CAMPUS iTversity
It's the way we're wired

- to add a new factor based on a numeric variable with more than two levels:

```
# The cut function takes a numeric vectors and cuts it into a categorical variable.
# Continuing the example above, let's make 'small','medium' and 'large' tree size classes:
allom$treeSizeClass <- cut(allom$diameter, breaks=c(0,25,50,75),
                           labels=c("small","medium","large"))

# And the results,
table(allom$treeSizeClass)

##
##  small medium  large
##     22     24     17
```

Working with factors

- To add a new factor based on a numeric variable with more than two levels:

```
# The cut function takes a numeric vectors and cuts it into a categorical variable.
# Continuing the example above, let's make 'small','medium' and 'large' tree size classes:
allom$treeSizeClass <- cut(allom$diameter, breaks=c(0,25,50,75),
                           labels=c("small","medium","large"))

# And the results,
table(allom$treeSizeClass)

##
##  small medium  large
##     22     24     17
```

Empty factor levels

- Each unique value of a factor variable is assigned a level, which is used every time you summarize your data by the factor variable.
- Even when you delete data, the original factor *level* is still present
- Sometimes it is more convenient to drop empty factor levels with the drop levels function.

Empty factor levels

BELGIUM CAMPUS iTversity
It's the way we're wired

```
# Read the Pupae data:
pupae <- read.csv("pupae.csv")

# Note that 'T_treatment' (temperature treatment) is a factor with two levels,
# with 37 and 47 observations in total:
table(pupae$T_treatment)

##
##  ambient elevated
##       37       47

# Suppose we decide to keep only the ambient treatment:
pupae_amb <- subset(pupae, T_treatment == "ambient")

# Now, the level is still present, although empty:
table(pupae_amb$T_treatment)

##
##  ambient elevated
##       37        0

# In this case, we don't want to keep the empty factor level.
# Use droplevels to get rid of any empty levels:
pupae_amb2 <- droplevels(pupae_amb)
```

Changing the levels of a factor

BELGIUM CAMPUS iTversity
It's the way we're wired

- If you want to change the levels of a factor, to replace abbreviations with more readable labels.
- To do this, you can assign new values with the levels function,

```
# Change the levels of T_treatment by assigning a character vector to the levels.
levels(pupae$T_treatment) <- c("Ambient","Ambient + 3C")

# Or only change the first level, using subscripting.
levels(pupae$T_treatment)[1] <- "Control"
```

Working with logical data

- Some data can only take two values: true, or false.
- **R** has the *logical* data type.
- Logical data are coded by integer numbers (0 = FALSE, 1= TRUE]

Working with logical data

BELGIUM CAMPUS iTversity
It's the way we're wired

```
# Answers to (in)equalities are always logical:
10 > 5
```

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

```
101 == 100 + 1
```

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

```
# ... or use objects for comparison:
apple <- 2
pear <- 3
apple == pear
```

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

```
# NOT equal to.
apple != pear
```

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

```
# Logical comparisons like these also work for vectors, for example:
nums <- c(10,21,5,6,0,1,12)
nums > 5
```

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

```
# Find which of the numbers are larger than 5:
which(nums > 5)
```

```
## [1] 1 2 4 7
```

```
# Other useful functions are 'any' and 'all':
# Are any numbers larger than 25?
```

Working with logical data

BELGIUM CAMPUS iTversity
It's the way we're wired

```
any(nums > 25)
```

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

```
# Are all numbers less than or equal to 10?
all(nums <= 10)
```

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

```
# Use & for AND, for example to take subsets where two conditions are met:
subset(pupae, PupalWeight > 0.4 & Frass > 3)
```

```
##     T_treatment CO2_treatment Gender PupalWeight Frass
## 25      ambient           400      1        0.405 3.117
```

```
# Use | for OR
nums[nums < 2 | nums > 20]
```

```
## [1] 21  0  1
```

```
# How many numbers are larger than 5?
#- Short solution
sum(nums > 5)
```

```
## [1] 4
```

```
#- Long solution
length(nums[nums > 5])
```

```
## [1] 4
```

Business Intelligence

G. Mudare

Special data types

Working with missing values

- In **R**, missing values are represented with NA, a special data type that indicates the data is simply *Not Available*.
- Never use 'NA' as an abbreviation for anything (like North America).
- Many functions can handle missing data, usually in different ways

```
myvec1 <- c(11,13,5,6,NA,9)
```

- In order to calculate the mean, we might want to either exclude the missing value or we might want mean(myvec1) to fail (produce an error).

Working with missing values

```
# Calculate mean: this fails if there are missing values
mean(myvec1)
```

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

```
# Calculate mean after removing the missing values
mean(myvec1, na.rm=TRUE)
```

```
## [1] 8.8
```

The function is.na returns TRUE when a value is missing, which can be useful to see which values are missing, or how many,

Making missing values

- In many cases it is useful to change some bad data values to NA by using indexes

```
# Some vector that contains bad values coded as -9999
datavec <- c(2,-9999,100,3,-9999,5)

# Assign NA to the values that were -9999
datavec[datavec == -9999] <- NA
```

Making missing values

- Missing values may arise when certain operations did not produce the desired result.

```
# A character vector, some of these look like numbers:
myvec <- c("101","289","12.3","abc","99")

# Convert the vector to numeric:
as.numeric(myvec)

## Warning:  NAs introduced by coercion

## [1] 101.0 289.0  12.3    NA  99.0
```

The warning message NAs introduced by coercion means that missing values were produced by when we tried to turn one data type (character) to another (numeric).

Not A Number

- Another type of missing value is the result of calculations that went wrong:

```
# Attempt to take the logarithm of a negative number:
log(-1)

## Warning in log(-1):  NaNs produced

## [1] NaN
```

- The result is NaN, short for *Not A Number*
- Dividing by zero is not usually meaningful, but **R** does not produce a missing value:

```
1000/0

## [1] Inf
```

Missing values in dataframes

- When working with dataframes, you often want to remove missing values for a particular analysis:

```
# Read the data
pupae <- read.csv("pupae.csv")

# Look at a summary to see if there are missing values:
summary(pupae)
```

```
##     T_treatment CO2_treatment        Gender          PupalWeight
##   ambient :37   Min.    :280.0   Min.    :0.0000   Min.    :0.1720
##   elevated:47   1st Qu.:280.0   1st Qu.:0.0000   1st Qu.:0.2562
##                 Median :400.0   Median :0.0000   Median :0.2975
##                 Mean   :344.3   Mean   :0.4487   Mean   :0.3110
##                 3rd Qu.:400.0   3rd Qu.:1.0000   3rd Qu.:0.3560
##                 Max.   :400.0   Max.   :1.0000   Max.   :0.4730
##                                 NA's   :6
##       Frass
##   Min.   :0.986
##   1st Qu.:1.515
##   Median :1.818
##   Mean   :1.846
##   3rd Qu.:2.095
##   Max.   :3.117
##   NA's   :1
```

Missing values in dataframes

BELGIUM CAMPUS iTversity
It's the way we're wired

```
# Notice there are 6 NA's (missing values) for Gender, and 1 for Frass.

# Option 1: take subset of data where Gender is not missing:
pupae_subs1 <- subset(pupae, !is.na(Gender))

# Option 2: take subset of data where Frass AND Gender are not missing
pupae_subs2 <- subset(pupae, !is.na(Frass) & !is.na(Gender))

# A more rigorous subset: remove all rows from a dataset where ANY variable
# has a missing value:
pupae_nona <- pupae[complete.cases(pupae),]
```

Subsetting when there are missing values

BELGIUM CAMPUS iTversity
It's the way we're wired

- use which to drop missing values when subsetting

```
# A small dataframe
dfr <- data.frame(a=1:4, b=c(4,NA,6,NA))

# subset drops all missing values
subset(dfr, b > 4, select=b)

##   b
## 3 6

# square bracket notation keeps them
dfr[dfr$b > 4,"b"]

## [1] NA  6 NA

# ... but drops them when we use 'which'
dfr[which(dfr$b > 4),"b"]

## [1] 6
```

Working with text

- Learn how to modify, extract, and analyse text-based ('character') variables.

```
# Count number of characters in a bit of text:
sentence <- "Not a very long sentence."
nchar(sentence)

## [1] 25

# Extract the first 3 characters:
substr(sentence, 1, 3)

## [1] "Not"
```

Working with text

- When we have character vectors:

```
# Substring all elements of a vector
substr(c("good","good riddance","good on ya"),1,4)

## [1] "good" "good" "good"

# Number of characters of all elements of a vector
nchar(c("hey","hi","how","ya","doin"))
```

```
## [1] 3 2 3 2 4
```

Working with text

- To glue bits of text together, use the paste function, like so:

```
# Add a suffix to each text element of a vector:
txt <- c("apple","pear","banana")
paste(txt, "-fruit")

## [1] "apple -fruit"  "pear -fruit"   "banana -fruit"

# Glue them all together into a single string using the collapse argument
paste(txt, collapse="-")

## [1] "apple-pear-banana"

# Combine numbers and text:
paste("Question", 1:3)

## [1] "Question 1" "Question 2" "Question 3"

# This can be of use to make new variables in a dataframe,
# as in this example where we combine two factors to create a new one:
pupae$T_CO2 <- with(pupae, paste(T_treatment, CO2_treatment, sep="-"))
head(pupae$T_CO2)

## [1] "ambient-280" "ambient-280" "ambient-280" "ambient-280" "ambient-280"
## [6] "ambient-280"
```

Column names

BELGIUM CAMPUS iTversity
It's the way we're wired

```
# Change the names of a dataframe:
hydro <- read.csv("hydro.csv")
names(hydro)  # first print the old names

## [1] "Date"    "storage"

names(hydro) <- c("Date","Dam_Storage") # then change the names


# Change only the first name (you can index names() just like you can a vector!)
names(hydro)[1] <- "Datum"
```

Column names

- Sometimes it is useful to find out which columns have particular names→use the match function

```
match(c("diameter","leafarea"), names(allom))

## [1] 2 4
```

Text in dataframes and grep

BELGIUM CAMPUS iTversity
It's the way we're wired

- When you read in a dataset (with read.csv, read.table or similar), any variable that R cannot convert to numeric is automatically converted to a factor. sometimes we want a variable to be treated like text

```
# Read data, tell R to treat the first variable ('Cereal.name') as character, not factor
cereal <- read.csv("cereals.csv", stringsAsFactors=FALSE)

# Make sure that the Cereal name is really a character vector:
is.character(cereal$Cereal.name)

## [1] TRUE

# The above example avoids converting any variable to a factor,
# what if we want to just convert one variable to character?
cereal <- read.csv("cereals.csv")
```