Working With Data

Programming in R for Data Science

Anders Stockmarr, Kasper Kristensen, Anders Nielsen

Data frames; adding and removing columns

- Look at the data frame dat:

```
> dat <- data.frame(x=LETTERS[1:3], y=1:3)
> dat
  x y
1 A 1
2 B 2
3 C 3
> dat[,1]
[1] A B C
Levels: A B C
> dat$x
[1] A B C
Levels: A B C
```

- It is simple to add or remove a column:

```
> dat$z <- dat$y^2
> dat$name <- c("Cat", "Vic", "Osc")
> dat$y<-NULL
> dat
  x z name
1 A 1  Cat
2 B 4  Vic
3 C 9  Osc
```

Merging data frames

- If we have two data sets:

```
> dat1
  name age
1  Cat   9
2  Vic   7
3  Osc   4
```

```
> dat2
  names gender
1   Vic   Male
2   Cat Female
3   Osc   Male
```

- Then we can merge that information into one data set by:

```
> dat <- merge(dat1, dat2, by.x="name", by.y="names")
> dat
  name age gender
1  Cat   9 Female
2  Osc   4   Male
3  Vic   7   Male
```

Getting dimension and column info

Given the dataset:

```
> df
  name age gender
1  Cat   9 Female
2  Osc   4   Male
3  Vic   7   Male
```

- names

```
> names(df)
[1] "name"   "age"    "gender"
```

- class

```
> class(df$name)
[1] "factor"
> class(df$age)
[1] "numeric"
```

- dim

```
> dim(df)
[1] 3 3
> nrow(df)
[1] 3
> ncol(df)
[1] 3
```

Object structure

- Get overview of the object structure:

  > *str(df)*

```
'data.frame':        3 obs. of  3 variables:
 $ name  : Factor w/ 3 levels "Cat","Osc","Vic": 1 2 3
 $ age   : num  9 4 7
 $ gender: Factor w/ 2 levels "Female","Male": 1 2 2
```

First and last rows

- First rows of a data frame:

```
> head(airquality, 3)
```

```
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
```

- Last rows of a data frame:

```
> tail(airquality, 3)
```

```
    Ozone Solar.R Wind Temp Month Day
151    14     191 14.3   75     9  28
152    18     131  8.0   76     9  29
153    20     223 11.5   68     9  30
```

The subset() function

Let's look at the `airquality` data again:

```
> head(airquality, 3)
```

```
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
```

- Logical indexing applies to data frames:

  ```
  > datA <- airquality[airquality$Temp>80,c("Ozone","Temp")]
  ```

- But a neat function is built in for making subsets of data

  ```
  > datA <- subset(airquality, Temp > 80, select = c(Ozone, Temp))
  > datB <- subset(airquality, Day == 1, select = -Temp)
  > datC <- subset(airquality, select = Ozone:Wind)
  ```

The summary() function

- The **summary()** function gives you a range of statistics

  > *summary(airquality$Wind)*

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.700   7.400   9.700   9.958  11.500  20.700
```

  that you could alternatively obtain using the **R** functions **min()**, **max()**, **mean()**, **median()**, **quantile()**.

- The summary of a data frame gives the summary of each column:

  > *summary(airquality)*

```
     Ozone           Solar.R           Wind             Temp           Month            Day
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00   Min.   :5.000   Min.   : 1.0
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00   1st Qu.:6.000   1st Qu.: 8.0
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00   Median :7.000   Median :16.0
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88   Mean   :6.993   Mean   :15.8
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00   3rd Qu.:8.000   3rd Qu.:23.0
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00   Max.   :9.000   Max.   :31.0
 NA's   :37       NA's   :7
```

Missing values

- **R** uses the special value **NA** to code missing values.
- The result of arithmetic involving NAs becomes NA as well:

  ```
  > colMeans(airquality)
      Ozone   Solar.R      Wind      Temp     Month       Day
         NA        NA  9.957516 77.882353  6.993464 15.803922
  ```

- This also holds for the comparison operator:

  ```
  > NA == NA
  [1] NA
  ```

Missing values

- We need a special function **is.na** to filter out NAs:

```
> is.na(NA)
[1] TRUE
```

- To get rid of NAs in a column we can use

```
> s <- subset(airquality, !is.na(Ozone) )
> colMeans(s)
    Ozone   Solar.R      Wind      Temp     Month       Day
42.129310        NA  9.862069 77.870690  7.198276 15.534483
```

- Note that the argument **na.rm=TRUE** can be passed to most summary functions e.g. **sum()**, **mean()**, **sd()**:

```
> mean(airquality$Ozone, na.rm=TRUE)
[1] 42.12931
```

Text manipulation

Text mining/text analytics is a relatively new and evolving science; online text mining has evolved in this century.

- Here are some examples:
  - Tweets.
  - Google Flu Trends (now terminated).
  - Questionnaires with freeform text.
  - Extraction of quantitative information from messy text sources.
- Text mining can be done in R.
- **R** has some simple tools that can be readily used for text searches.

Text search and replace

- Overview:

Function	Description
grep() grepl()	Text pattern search
sub()	Text pattern search and replace first occurrence
gsub()	Text pattern search and replace all occurrences

- Note common arguments:
  - By default pattern is a *regular expression*: **fixed = FALSE**.
  - By default matching is *case sensitive*: **ignore.case = FALSE**.
- Lets try some examples...

Text search example

- Some example text

```
> txt <- c("Hello, my",
+          "name is",
+          "anders."
+          )
```

- Search for string 'name' returning indices:

```
> grep("name", txt)
```

```
[1] 2
```

or logical indices

```
> grepl("name", txt)
```

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

- Replace 'anders' with 'Anders':

```
> sub("anders", "Anders", txt)
```

```
[1] "Hello, my" "name is"   "Anders."
```

Text manipulation - Regular expressions

- Messy dataset: Number of fruits eaten by person.

  > *df*

```
  person.ID                                      fruit
1         1     apple: 3  Orange : 9 banana:2
2         2     Orange:1 Apple: 3  banana: 10
3         3 banana: 3 Apple: 3  Orange : 04
```

- We want the number of oranges eaten for each person. How to get this information automatically?
- Answer: Construct a regular expression to extract it...

Regular expressions

- Regular expression to find the orange count:

.*	A sequence of arbitrary characters (possibly empty) followed by
orange	followed by
[:]*	a sequence of whitespace OR colon followed by
([0-9]*)	a sequence of digits followed by
.*	a sequence of arbitrary characters (possibly empty).

  - Parenthesis around the digit pattern stores this match. The match is accessed through a so-called *back reference* "\\1".

- Now try it:

```
> pattern <- ".*orange[ :]*([0-9]*).*"
> sub(pattern, "\\1", df$fruit, ignore.case=TRUE)
```

```
[1] "9"  "1"  "04"
```

- Use **as.numeric()** to convert result to a number.

Date and time objects

- **R** has classes to handle dates **?as.Date** and dates with time information **?as.POSIXct**.
- You can construct these manually from characters:

```
> as.Date("2016-03-10")
[1] "2016-03-10"
> as.POSIXct("2016-03-10 13:53:38 CET")
[1] "2016-03-10 13:53:38 CET"
```

- But if you are lucky your SQL server already uses a suitable datetime format that will be properly imported by **R** :

```
> conn <- odbcDriverConnect(connStr)
> df <- sqlQuery(conn, "SELECT TOP 2000 * FROM bi.sentiment")
> class(df$Date)
[1] "POSIXct" "POSIXt"
```

Working with dates

- Many useful methods exist for dates (see **methods(class="Date")**):
- You can do math with dates / datetimes.

```
> mean(df$Date)
[1] "2014-01-13 16:40:04 CET"
> mean(df$Date+10)
[1] "2014-01-13 16:40:14 CET"
```

- However, note the unit difference:

```
> mean(as.Date(df$Date))
[1] "2014-01-12"
> mean(as.Date(df$Date)+10)
[1] "2014-01-22"
```

- Some useful conversion methods:

```
> table(weekdays(df$Date))
lørdag onsdag
   819   1181
> table(months(df$Date))
februar  januar
    819    1181
```