PROGRAMA DE VERÃO DO LNCC

JORNADA DE CIÊNCIA DE DADOS

R Basics

CEFET/RJ

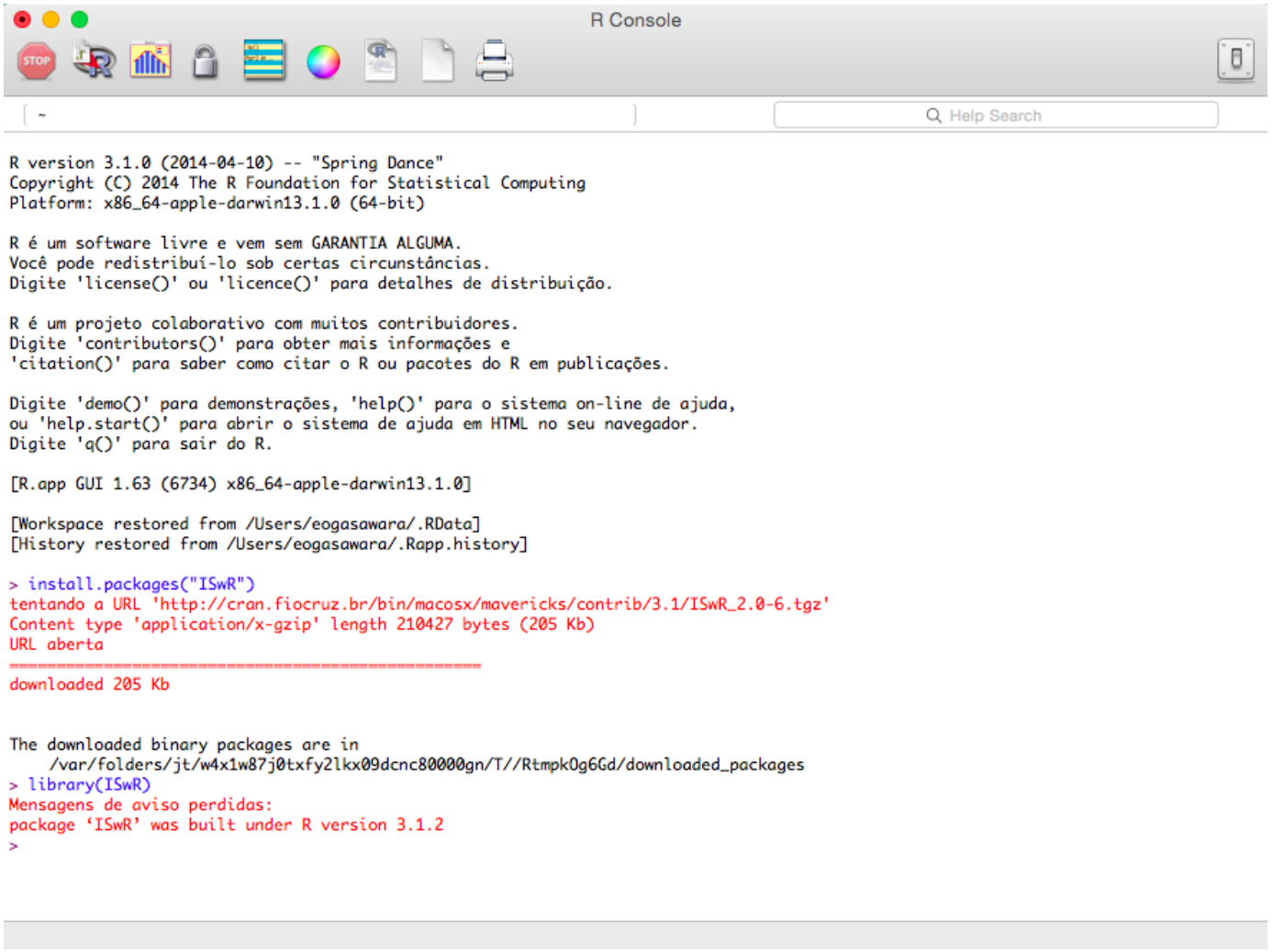
Eduardo Ogasawara

http://eic.cefet-rj.br/~eogasawara

Introduction to R

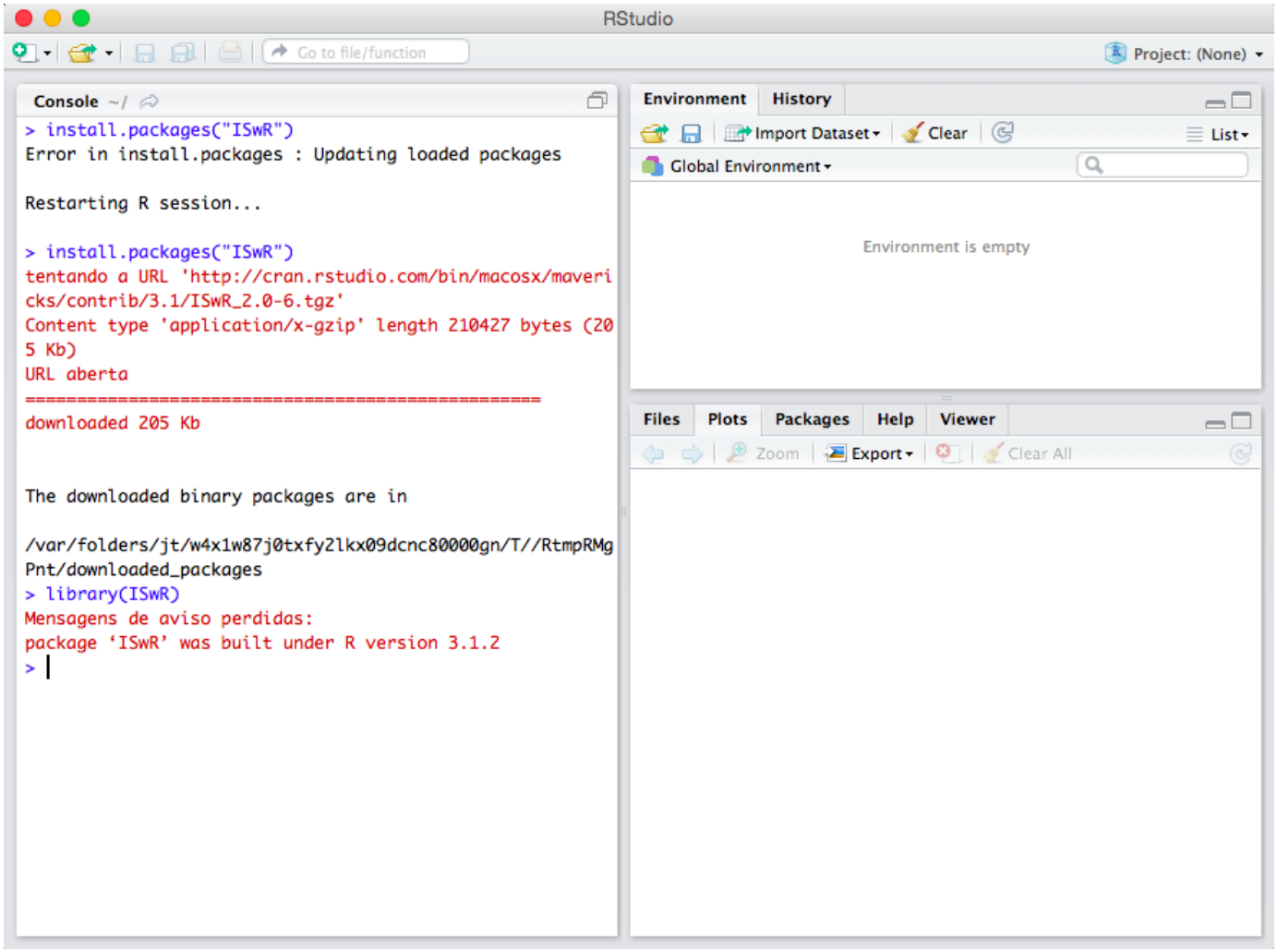
- R is a programming language and free software environment for statistical computing
  - Supported by the R Foundation for Statistical Computing
- Created by Ross Ihaka and Robert Gentleman at Auckland University, New Zealand
- R was derived by S (Bell Laboratories - AT&T)
- R is a language broadly used by statisticians, data miners, and data scientists

R Console



Available for Windows, Mac, Linux

R Studio
http://www.rstudio.com



Great advantages: IDE with data visualization, debugging

CRAN Packages

- A broad number of packages (CRAN)
  - https://cran.r-project.org
- Strong Point of R
  - More than 14000 available packages (apr/2019)
  - http://cran.r-project.org/web/packages/
- Package installation
- Package loading

```
install.packages("TSPred")
install.packages("STMotif")
```

```
package 'TSPred' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\eduar\AppData\Local\Temp\RtmpMr5h0i\downloaded_packages
package 'STMotif' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\eduar\AppData\Local\Temp\RtmpMr5h0i\downloaded_packages
```

```
require(ggplot2)
require(TSPred)
require(STMotif)
```

```
Loading required package: ggplot2
Loading required package: TSPred
Warning message:
"package 'TSPred' was built under R version 3.5.3"Loading required package: STMotif
Warning message:
"package 'STMotif' was built under R version 3.5.3"
```

Basic concepts

- Assignment
- Value display
- Logical test
- Vector definition
  - Computing BMI
- Printing values

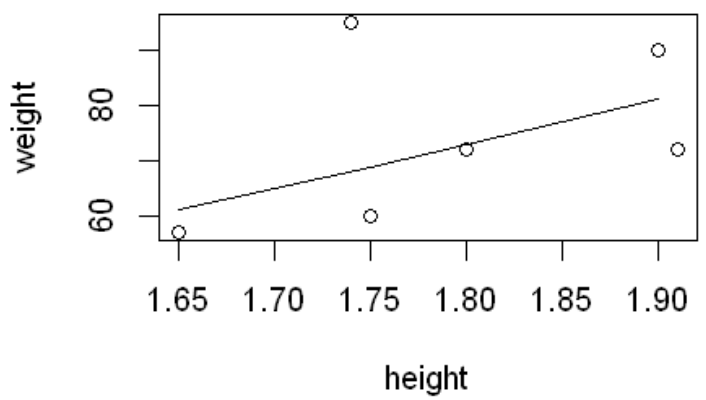
```r
x <- 2 # variable assignment

x # variable evaluation

is.numeric(x) # variable

weight = c(60, 72, 57, 90, 95, 72) # vector with six obervations

height = c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)

bmi = weight/height^2

print(bmi)

print(sprintf("%.2f +/- %.2f", mean(bmi), sd(bmi)))
```

```
2

TRUE

[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
[1] "23.13 +/- 4.49"
```

Plotting graphics & Statistical analysis

- Plotting a scatter graphics
  - Canvas is active until the next plot
- Test theoretical value of BMI equals to 22.5
  - Null hypothesis: no difference observed (p-value > 5%)
  - Alternative hypothesis: they are different

```
plot(height, weight)

hh = c(1.65, 1.70, 1.75, 1.80, 1.85, 1.90)
lines(hh, 22.5 * hh^2)
```




```
t.test(bmi, mu=22.5)
```

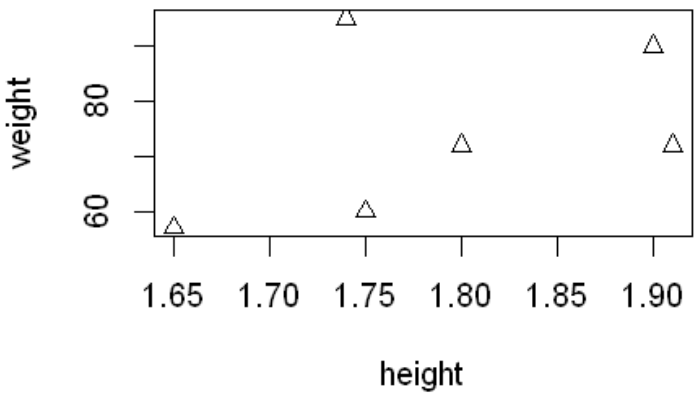
```
        One Sample t-test

data:  bmi
t = 0.34488, df = 5, p-value = 0.7442
alternative hypothesis: true mean is not equal to 22.5
95 percent confidence interval:
 18.41734 27.84791
sample estimates:
mean of x
 23.13262
```

Default arguments and help for functions

- Functions have default values
- View parameters of the function
- Use online help

```
plot(height, weight, pch=2)

args(plot.default)

?graphics::plot
```

```
function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
    log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
    ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
    panel.last = NULL, asp = NA, ...)
NULL
```



More about vectors

- Operations with NA
- Name of observations
- Scalar multiplication

```
x <- c(A=1, B=NA, C=3)

mean(x)

mean(x, na.rm=TRUE)

names(x)

x["B"] <- 2

x["B"]*x
```

```
<NA>

2

'A'  'B'  'C'
```

A	2
B	4
C	6

Matrix

- Creation
- Creation by rows
- Names for rows and columns
- Transpose
- Determinant

```
m <- 1:9
dim(m) <- c(3,3)
m

mb <- matrix(1:9, nrow=3,byrow=TRUE)
rownames(mb) = LETTERS[1:3]
mb

t(m)

m*x

det(m)
```

```
1 4 7
2 5 8
3 6 9
```

```
  1 2 3
A
B 4 5 6
C 7 8 9
```

```
1 2 3
4 5 6
7 8 9
```

```
1  4  7
4 10 16
9 18 27
```

```
0
```

Factors

- Factors are variables in R that refer to categorical data
- Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed
- Both numeric and character variables can be made into factors, but a factor's levels are always character values

```
pain = c(0,3,2,2,1)
fpain = factor(pain,levels=0:3)
levels(fpain) = c("none","mild","medium","severe")

fpain

as.numeric(fpain)

levels(fpain)
```

none severe medium medium mild

▶ **Levels**:

1 4 3 3 2

'none' 'mild' 'medium' 'severe'

Lists

- Lists are the R objects which contain elements of different types, such as numbers, strings, vectors, matrix, data frame, and another list inside it.
- A list can also contain a matrix or a function as its elements
- A list is created using the list() function

```
x = c(5260,5470,5640,6180,6390,
      6515,6805,7515,7515,8230,8770)
y = c(3910,4220,3885,5160,5645,
      4680,5265,5975,6790,6900,7335)

lst <- list(A=x, B=y)

lst

lst$A
```

```
$A
 5260  5470  5640  6180  6390  6515  6805  7515  7515  8230  8770
$B
 3910  4220  3885  5160  5645  4680  5265  5975  6790  6900  7335


 5260  5470  5640  6180  6390  6515  6805  7515  7515  8230  8770
```

Data frames

- A data frame is a table where each column corresponds to attributes, and each row corresponds to a tuple (object)

```
d <- data.frame(A=lst$A,B=lst$B)
d

df <- d[d$A > 7000 | d$A < 6000,]
df
```

A	B
5260	3910
5470	4220
5640	3885
6180	5160
6390	5645
6515	4680
6805	5265
7515	5975
7515	6790
8230	6900
8770	7335

	A	B
1	5260	3910
2	5470	4220
3	5640	3885
8	7515	5975
9	7515	6790
10	8230	6900
11	8770	7335

Implicitly Loops – sapply, lapply

- lapply, sapply executes a function for each column
  - The first character defines the return type
    - l – list, s – simple (vector or matrix)
  - The second parameter is the function to invoke
  - Following parameters are passed to the invoked function
- apply is the generic function
  - The second parameter defines if it calls the function for each row (1) or each column (2)

```
lapply(d, min, na.rm=TRUE)

sapply(d, min, na.rm=TRUE)

apply(d, 1, min)

apply(d, 2, min)
```

```
$A
5260
$B
3885
```

```
A 5260
B 3885
```

```
3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335
```

```
A 5260
B 3885
```

Sort and order

```
sort(d$B)
o <- order(d$B)
o
ds <- d[o,]
ds
```

```
3885  3910  4220  4680  5160  5265  5645  5975  6790  6900  7335
```

```
3  1  2  6  4  7  5  8  9  10  11
```

	A	B
3	5640	3885
1	5260	3910
2	5470	4220
6	6515	4680
4	6180	5160
7	6805	5265
5	6390	5645
8	7515	5975
9	7515	6790
10	8230	6900
11	8770	7335

Loading and saving files

```
wine = read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data",
    header = TRUE, sep = ",")
head(wine)
save(wine, file="wine.RData")

rm(wine)

load("wine.RData")
write.table(wine, file="wine.csv", row.names=FALSE, quote = FALSE)
```

X1	X14.23	X1.71	X2.43	X15.6	X127	X2.8	X3.06	X.28	X2.29	X5.64	X1.04	X3.92	X1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
1	14.39	1.87	2.45	14.6	96	2.50	2.52	0.30	1.98	5.25	1.02	3.58	1290

Creating functions

```
create_dataset <- function() {
  data <- read.table(text = "Year Months Flights Delays
                      2016 Jan-Mar 11 6
                      2016 Apr-Jun 12 5
                      2016 Jul-Sep 13 3
                      2016 Oct-Dec 12 5
                      2017 Jan-Mar 10 4
                      2017 Apr-Jun 9 3
                      2017 Jul-Sep 11 4
                      2017 Oct-Dec 25 15
                      2018 Jan-Mar 14 3
                      2018 Apr-Jun 12 5
                      2018 Jul-Sep 13 3
                      2018 Oct-Dec 15 4",
                      header = TRUE,sep = "")
  data$OnTime <- data$Flights - data$Delays
  data$Perc <- round(100 * data$Delays / data$Flights)
  return(data)
}

data <- create_dataset()
head(data)
```

Year	Months	Flights	Delays	OnTime	Perc
2016	Jan-Mar	11	6	5	55
2016	Apr-Jun	12	5	7	42
2016	Jul-Sep	13	3	10	23
2016	Oct-Dec	12	5	7	42
2017	Jan-Mar	10	4	6	40
2017	Apr-Jun	9	3	6	33

Pipelines

```
loadlibrary("dplyr")

data_sd <- create_dataset() %>%
  select(variable=Months, value=Delays) %>%
  group_by(variable) %>%
  summarize(sd = sd(value), value = mean(value))

data_sd$variable <- factor(data_sd$variable,
    levels = c('Jan-Mar','Apr-Jun','Jul-Sep','Oct-Dec'))

head(data_sd)
```

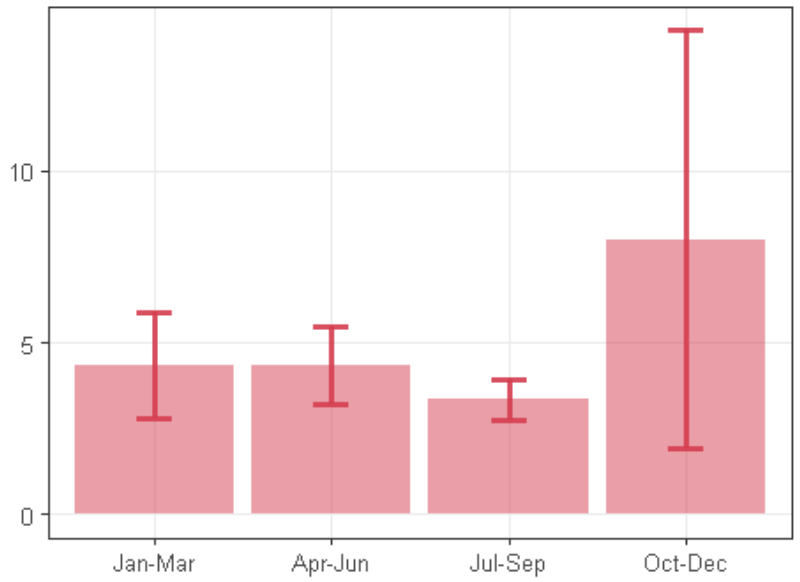
variable	sd	value
Apr-Jun	1.1547005	4.333333
Jan-Mar	1.5275252	4.333333
Jul-Sep	0.5773503	3.333333
Oct-Dec	6.0827625	8.000000

The **dplyr** is an important package to know

Pipeline dataset %>% operators %>% first parameter of functions is implicit from the pipeline

The ggplot graphics

```
loadlibrary("RColorBrewer")

col_set <- brewer.pal(11, 'Spectral')

grf <- plot.bar(data_sd, colors=col_set[2], alpha=0.5)
grf <- grf + geom_errorbar(
    aes(x=variable, ymin=value-sd, ymax=value+sd),
    width=0.2, colour=col_set[2], alpha=0.9, size=1.1)

plot(grf)
```



RColorBrewer is a nice package to setup colors
GGPlot is a nice tool to plot graphics

The melt function

```
loadlibrary("reshape")
adjust_dataset <- function(data) {
  data <- melt(data[,c('Year', 'Months', 'Flights', 'Delays', 'OnTime', 'Perc')],
               id.vars = c(1,2))
  data$x <- sprintf("%d-%s", data$Year, data$Months)
  data$x <- factor(data$x,levels = data$x[1:12])
  return(data)
}
data <- create_dataset()
head(data)
data <- adjust_dataset(data)
head(data)
```

Year	Months	Flights	Delays	OnTime	Perc
2016	Jan-Mar	11	6	5	55
2016	Apr-Jun	12	5	7	42
2016	Jul-Sep	13	3	10	23
2016	Oct-Dec	12	5	7	42
2017	Jan-Mar	10	4	6	40
2017	Apr-Jun	9	3	6	33

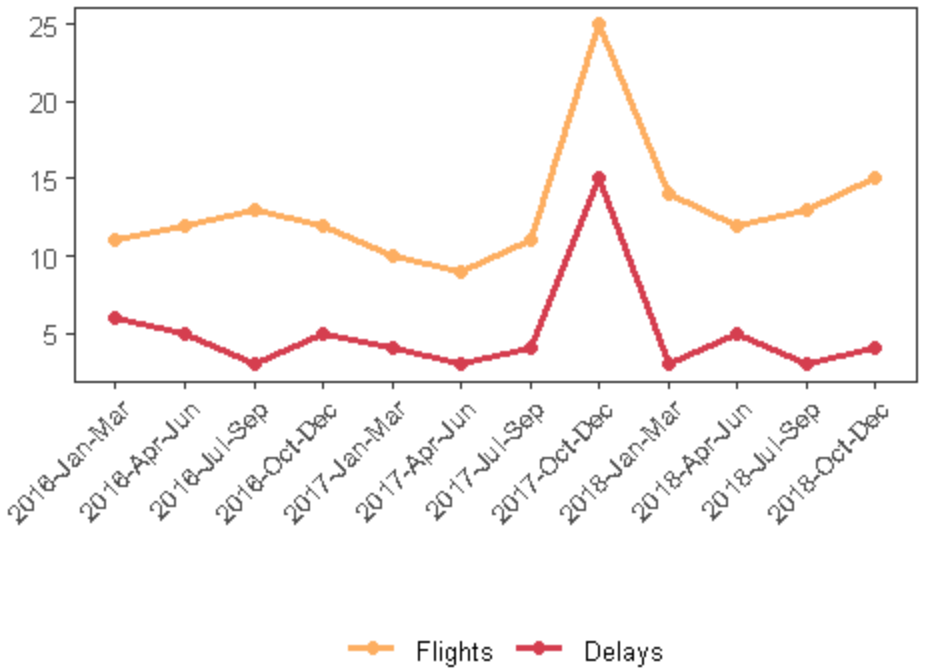
Year	Months	variable	value	x
2016	Jan-Mar	Flights	11	2016-Jan-Mar
2016	Apr-Jun	Flights	12	2016-Apr-Jun
2016	Jul-Sep	Flights	13	2016-Jul-Sep
2016	Oct-Dec	Flights	12	2016-Oct-Dec
2017	Jan-Mar	Flights	10	2017-Jan-Mar
2017	Apr-Jun	Flights	9	2017-Apr-Jun

The **melt** function transforms columns values into rows grouped by **id.vars**.

The name of columns is used to fill the **variable** attribute created during the **melt**.

Line graphics

```
grf <- plot.series(data %>% filter(variable %in% c('Flights', 'Delays')),
                   colors=col_set[c(4,2)])
grf <- grf + theme(axis.text.x = element_text(angle=45, hjust=1))

plot(grf)
```



Take some time studying myGraphics.ipynb

Joining data frames

```r
stores <- data.frame(
    city = c("Rio de Janeiro", "Sao Paulo", "Paris", "New York", "Tokyo"),
    value = c(10, 12, 20, 25, 18))
head(stores)


divisions <- data.frame(
    city = c("Rio de Janeiro", "Sao Paulo", "Paris", "New York", "Tokyo"),
    country = c("Brazil", "Brazil", "France", "US", "Japan"))
head(divisions)

data <- merge(stores, divisions, by.x="city", by.y="city")
head(data)

result <- data %>% group_by(country) %>% summarize(count = n(), amount = sum(value))
head(result)
```

city	value
Rio de Janeiro	10
Sao Paulo	12
Paris	20
New York	25
Tokyo	18

city	country
Rio de Janeiro	Brazil
Sao Paulo	Brazil
Paris	France
New York	US
Tokyo	Japan

city	value	country
New York	25	US
Paris	20	France
Rio de Janeiro	10	Brazil
Sao Paulo	12	Brazil
Tokyo	18	Japan

country	count	amount
Brazil	2	22
France	1	20
Japan	1	18
US	1	25

Loops and Conditional

- R supports loops and conditionals in a similar way as in Java
- Loops should be used when strictly needed

```
for (i in 1:nrow(result)) {
  value <- result$amount[i]
  if (result$count[i] > 1) {
      value <- 0.8*value
  }
  print(sprintf("%6s - %.1f", result$country[i], value))
}
```

```
[1] "Brazil - 17.6"
[1] "France - 20.0"
[1] " Japan - 18.0"
[1] "    US - 25.0"
```

Practicing

- Take some time to practice the examples
  - https://nbviewer.jupyter.org/github/eogasawara/mylibrary/blob/master/myIntroduction.ipynb
- Take a look at how to prepare nice graphics using ggplot2
  - https://nbviewer.jupyter.org/github/eogasawara/mylibrary/blob/master/myGraphics.ipynb

Main References