Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Exploration and Analysis of Software Engineering Data with R

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it

The 13th International Conference on Software Engineering Advances
ICSEA 2018
October 14, 2018 to October 18, 2018 - Nice, France

Motivations

- Technicians need to know if a given technique or method or tool actually works well, i.e., whether it actually provides the expected benefits.
- Managers need the knowledge need to make decisions, to estimate costs, to plan activities, etc.
- This knowledge is *quantitative*.
- To obtain this knowledge we need
  - Measures
  - Model that synthetize the measures into ready-to-use quantitative models. E.g.,
    - a model of defect proneness based on measures of internal software attributes
    - A model of the cost of development based on the measures that characterize the software application to be developed and the process used to develop it.

Contents

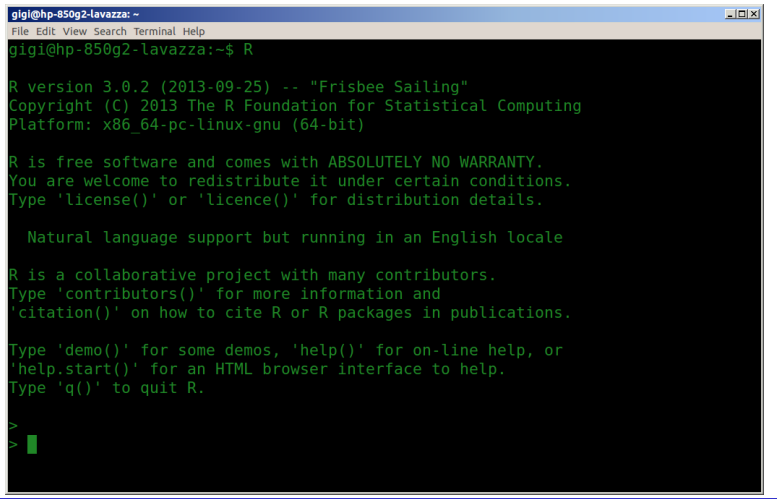
- R basics
- Data exploration
  - Visualization
  - Descriptive statistics
- Correlation
- Models
  - OLS linear regression
  - OLS log-log regression
  - Accuracy evaluation
  - Dealing with outliers
    - Identifying outliers
    - LMS regression
- Estimating probabilities
  - Binary Logistic Regression
- Classification
  - From probability estimation to classification
    - thresholds
  - Evaluating the accuracy of classifications
  - Machine Learning
    - decision trees

What is R

- *R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS*
  - http://www.r-project.org
- Open source, downloadable from CRAN (Comprehensive R Archive Network)
  - https://cran.r-project.org/mirrors.html
- It comes with a set of "basic" packages, which satisfy the most common needs of users.
- Hundreds of specialized packages are available.
  - For free
  - New packages are continuously released
- It has state-of-the-art graphics capabilities.
  - But you have to program to get a nice graphic output.

Getting started with R

- To launch R, open a shell in Unix/Linux or a command line interface window in Windows. At the prompt, type R<return>

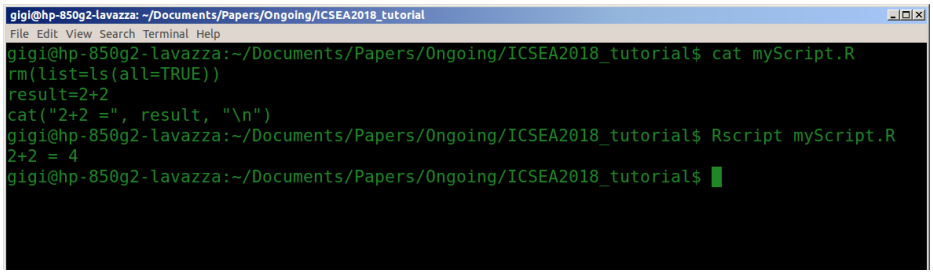


Getting started with R

- You can enter commands at the command prompt
  - help()
  - quit()
- Alternatively, you can run a set of commands (instructions) from a source file.
  - In this way, you can run a real program, as complex as you like.

R usage

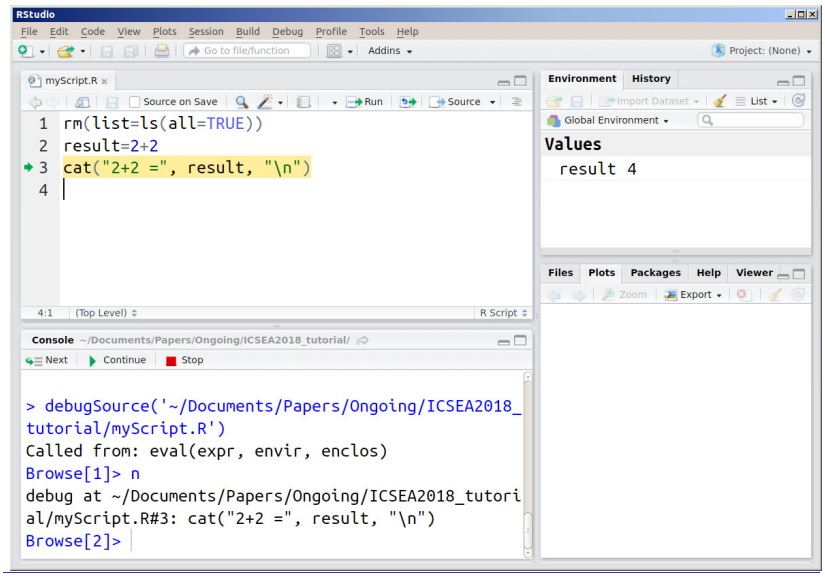
- In general, you end up performing sequences of operations, or real programs.
- Therefore, you need to save the sequences of instructions (or programs) that you want to be able to reuse.
- The most simple (though not very efficient) way of doing this is:
  - You write a R script using your preferred editor and save it in file **`myScript.R`**.
  - You launch the execution via **`Rscript myScript.R`**



RStudio

- As soon as the script becomes complex, you have to modify it and debug it, etc., this approach is no longer viable.
- RStudio is much preferable.
- Rstudio is an Integrated Development Environment for R. It supports
  - Editing
  - Executing
  - Debugging
  - Graphics display and storage
  - Etc.
- An open version is available
  - https://www.rstudio.com/

RStudio



Documentation

- Start from the CRAN site
  - https://cran.r-project.org/manuals.html
- At the CRAN site you can find the official documentation, including
  - An Introduction to R
  - R Data Import/Export
  - R Installation and Administration
  - Writing R Extensions
  - The R language definition (draft)
  - R Internals
  - The R Reference Index

Getting help

- The help function
  - help("name")
- The web is an extremely effective source of help for R users.
  - R is now one of the 10 most popular languages worldwide.
  - I almost always find an answer to my questions just by Googling them.
- Most useful sites:
  - https://www.r-bloggers.com/
  - https://stackoverflow.com/
  - …

R Help

- R provides an effective help system.
  - help.start() # general help
  - help(foo) # help about function foo
  - apropos("foo") # list all function containing string foo
  - example(foo) # show an example of function foo
  - vignette() # show available vignettes of installed packages
  - vignette("foo") # show specific vignette

R Workspace

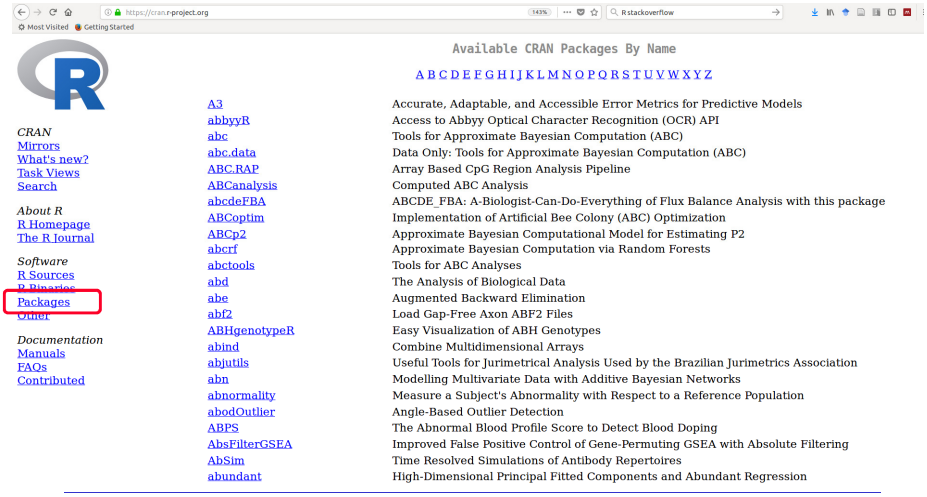
- Objects that you create during an R session are hold in memory
- The collection of objects that you currently have is called the workspace.
- This workspace is not saved on disk unless you tell R to do so.
- When you close the RGui or the R console window, the system will ask if you want to save the workspace image.
  - If you select to save the workspace image then all the objects in your current R session are saved in a file .RData.
  - .RData is a binary file located in the working directory of R, which is by default the installation directory of R.
- Your objects are lost when you close R and **not** save the objects, or when R or your system crashes.

Workspace: is it useful?

- Not necessarily
- I work on R programs and
  - Continuously save them
  - Write results to files
- So, I hardly ever need to save the workspace

R Packages

- R can easily be extended via packages.
- Many R packages have been written and made available on CRAN for



R Packages

- When you download R, several packages are also downloaded.
- When you start R not all of the downloaded packages are loaded.
- To load another package to the system you can use the **`library`** function.
  - **`library(MASS)`**

R Packages

- To show the installed packages
  - `installed.packages()`
- To install a new package
  - `install.packages()`

Data Types

- R has a wide variety of data types including scalars, vectors (numerical, character, logical), matrices, dataframes, and lists.

Vectors

```
a <- c(1,2,5.3,6,-2,4)            # numeric vector
first10<-c(1:10)
b <- c("one", "two", "three")     # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
                                  # logical vector
```

- Vector elements are referred to using subscripts.

```
a[3]      # 3rd element of vector a
a[c(2,4)] # 2nd and 4th elements of vector a
a[c(1:3)] # first 3 elements of vector a
```


Matrices

- All columns in a matrix must have the same mode(numeric, character, etc.) and the same length.
- The general format is

```
mymatrix <- matrix(vector, nrow=r, ncol=c,
    byrow=FALSE, dimnames=list(char_vector_rownames,
                               char_vector_colnames))
```

  - **byrow=TRUE** indicates that the matrix should be filled by rows. **byrow=FALSE** indicates that the matrix should be filled by columns (the default).
  - **dimnames** provides optional labels for the columns and rows.

Matrices

```
y<-matrix(1:20, nrow=5, ncol=4)
                # generates 5 x 4 numeric matrix
cells <- c(1,26,24,68)
rnames <- c("R1", "R2")
cnames <- c("C1", "C2")
mymatrix <- matrix(cells, nrow=2, ncol=2,
          byrow=TRUE, dimnames=list(rnames, cnames))
```

- Identify rows, columns or elements using subscripts.

```
x[,4]           # 4th column of matrix
x[3,]           # 3rd row of matrix
x[-3,]          # all rows of matrix except 3rd row
x[2:4,1:3]      # rows 2,3,4 of columns 1,2,3
```


Arrays

- Arrays are similar to matrices but can have more than two dimensions. See help(array) for details.

Data frames

- A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).

```
d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
mydata <- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed")
```


Identifying the elements of a dataframe

```
mydata[3:5]         # columns 3,4,5 of dataframe
mydata[c("ID", "Passed")] # columns ID and Passed
mydata$Passed       # column Passed: a vector of logical
```

Lists

- An ordered collection of objects (components).
- A list allows you to gather a variety of (possibly unrelated) objects under one name.
- Example of a list with 4 components: a string, a numeric vector, a matrix, and a number

```
w <- list(name="Fred", mynumbers=a, mymatrix=y,
          age=5.3)
```

- Identify elements of a list:

```
mylist[[2]] # 2nd component of the list
```


Factors

- Tell R that a variable is nominal by making it a factor.
- The factor stores the nominal values as a vector of integers in the range [1... k] (where k is the number of unique values in the nominal variable), and an internal vector of character strings (the original values) mapped to these integers.

```
# variable gender with 20 "male" entries and
# 30 "female" entries
gender <- c(rep("male",20), rep("female", 30))
gender <- factor(gender)
# stores gender as 20 1s and 30 2s and associates
# 1=female, 2=male internally (alphabetically)
# R now treats gender as a nominal variable
summary(gender)
```

Useful Functions

Name	effect
`length(object)`	number of elements or components
`str(object)`	structure of an object
`class(object)`	class or type of an object
`names(object)`	names
`c(object, object,...)`	combine objects into a vector
`cbind(object, object, ...)`	combine objects as columns
`rbind(object, object, ...)`	combine objects as rows
`ls()`	list current objects
`rm(object)`	delete an object
`newobject <- edit(object)`	edit copy and save a new object
`fix(object)`	edit in place

Importing Data

- You can import data into R from different sources.
- We consider here only one type of source: comma separated values (csv) files.

From a Comma Delimited Text File

```
mydata <- read.table("mydata.csv", header=TRUE, sep=",")
```

- You can specify the complete path of the file, otherwise the file is looked for in the working directory
- header=TRUE uses the contents of the first row as column names
- The expected separator is ","
- **`mydata`** is a dataframe

Exporting Data

- How to export a data frame to a .csv file:

```
write.csv(mydata, file="mydata.csv", sep="\t")
```

Missing Data

- NA = not available
- NaN = not a number
- Testing for Missing Values

```
is.na(x)  # returns TRUE of x is NA
y <- c(1,2,3,NA)
is.na(y) # returns vector (F F F T)
mean(y)  # causes an error
mean(y, na.rm=TRUE) # returns 2
```


Dataset import example

- Let us import the data from file cocomo.csv

cocomo81.csv - LibreOffice Calc

File Edit View Insert Format Tools Data Window Help

Liberation Sans 10

A1 rely

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	rely	data	cplx	time	stor	virt	turn	acap	aexp	pcap	vexp	lexp	modp	tool	sced	loc	actual
2	0.88	1.16	0.7	1	1.06	1.15	1.07	1.19	1.13	1.17	1.1	1	1.24	1.1	1.04	113	2040
3	0.88	1.16	0.85	1	1.06	1	1.07	1	0.91	1	0.9	0.95	1.1	1	1	293	1600
4	1	1.16	0.85	1	1	0.87	0.94	0.86	0.82	0.86	0.9	0.95	0.91	0.91	1	132	243
5	0.75	1.16	0.7	1	1	0.87	1	1.19	0.91	1.42	1	0.95	1.24	1	1.04	60	240
6	0.88	0.94	1	1	1	0.87	1	1	1	0.86	0.9	0.95	1.24	1	1	16	33
7	0.75	1	0.85	1	1.21	1	1	1.46	1	1.42	0.9	0.95	1.24	1.1	1	4	43
8	0.75	1	1	1	1	0.87	0.87	1	1	1	0.9	0.95	0.91	0.91	1	6.9	8
9	1.15	0.94	1.3	1.66	1.56	1.3	1	0.71	0.91	1	1.21	1.14	1.1	1.1	1.08	22	1075
10	1.15	0.94	1.3	1.3	1.21	1.15	1	0.86	1	0.86	1.1	1.07	0.91	1	1	30	423
11	1.4	0.94	1.3	1.11	1.56	1	1.07	0.86	0.82	0.86	0.9	1	1	1	1	29	321
12	1.4	0.94	1.3	1.11	1.56	1	1.07	0.86	0.82	0.86	0.9	1	1	1	1	32	218
13	1.15	0.94	1.3	1.11	1.06	1	1	0.86	0.82	0.86	1	0.95	0.91	1	1.08	37	201
14	1.15	0.94	1.3	1.11	1.06	1.15	1	0.71	1	0.7	1.1	1	0.82	1	1	25	79
15	1.15	0.94	1.65	1.3	1.56	1.15	1	0.86	1	0.7	1.1	1.07	1.1	1.24	1.23	3	60
16	1.4	0.94	1.3	1.3	1.06	1.15	0.87	0.86	1.13	0.86	1.21	1.14	0.91	1	1.23	3.9	61
17	1.4	1	1.3	1.3	1.56	1	0.87	0.86	1	0.86	1	1	1	1	1	6.1	40
18	1.4	1	1.3	1.3	1.56	1	0.87	0.86	0.82	0.86	1	1	1	1	1	3.6	9

The code

```
rm(list=ls(all=TRUE))
isBatch = FALSE
if(isBatch) {
  options(echo  =  FALSE)
} else {
  options(echo  =  TRUE)
}

modelsetName="cocomo81"
fname=paste("./Data/", modelsetName, ".csv", sep="")
dfDataset=read.csv(fname)
```


Showing data

- Display the data frame in a table

```
View(dfDataset)
```

- List the fileds

```
Browse[2]> names(dfDataset)
 [1] "rely"   "data"   "cplx"   "time"   "stor"   "virt"   "turn"   "acap"
 [9] "aexp"   "pcap"   "vexp"   "lexp"   "modp"   "tool"   "sced"   "loc"
[17] "actual"
```

We are interested in exploring the relationship between size (loc) and development effort (actual)

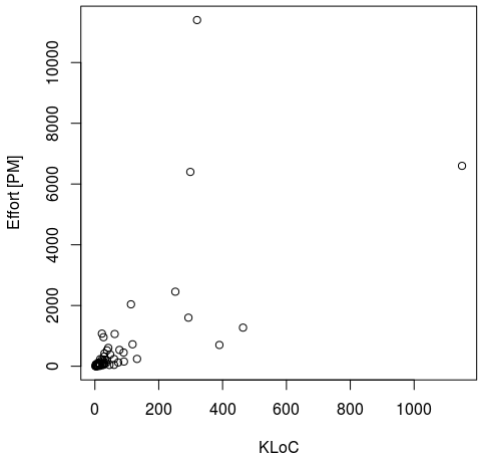
Data plotting

```
par(mar=c(4.1, 4.1, 1.1, 2.1))
```

set margins (bottom, left, top, and right)
default is 5.1 4.1 4.1 2.1

```
plot(dfDataset$loc,
     dfDataset$actual,
     xlab="KLoC",
     ylab="Effort [PM]")
```

Plots points with coordinates dfDataset$loc and dfDataset$actual.
x and y labels are also set.

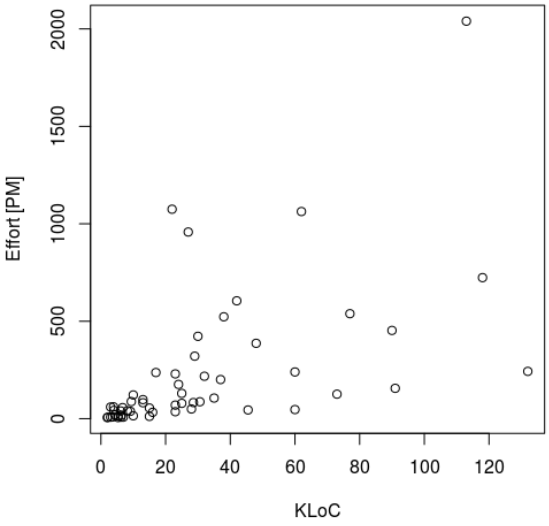


Subsetting data

- It is clear that we have too few projects with size over 200 KLoC.
- Hence we cannot build models or derive any significant conclusion for projects having size > 200 KLoC.
- Let us drop those data.
- Code:

```
dfDataset=subset(dfDataset, dfDataset$loc<=200)
plot(dfDataset$loc, dfDataset$actual,
     xlab="KLoC", ylab="Effort [PM]")
```

New plot



Descriptive statistics

- We count the data points.
- For each dimension we compute:
  - Mean
  - Standard deviation
  - Median
  - Minimum
  - Maximum

Descriptive statistics code

```
n=length(dfDataset[,1])
cat("\nThe dataset contains", n, "datapoints\n")
locMean=mean(dfDataset$loc)
locMedian=median(dfDataset$loc)
locSd=sd(dfDataset$loc)
locMin=min(dfDataset$loc)
locMax=max(dfDataset$loc)
cat("Mean KLoC =", locMean, "stdev =", locSd, "\n")
cat("Median KLoC =", locMedian, "\n")
cat("KLoC range = [", locMin, ",", locMax, "]\n")
```


Descriptive statistics: output

The dataset contains 56 datapoints
Mean KLoC = 30.28964 stdev = 31.23318
Median KLoC = 23
KLoC range = [1.98 , 132]

Functions

- Instead of repeating the code for getting descriptive statistics of effort, let us define a function, that we shall apply to both size and effort.

```
# function statDescr
# in: vData, a vector of numeric data; nData, the name of the data
# out: none
# effect: descriptive statistics are printed in output
statDescr<-function(vData, nData){
  cat("Mean", nData, " =", mean(vData), "stdev =", sd(vData), "\n")
  cat("Median", nData, " =", median(vData), "\n")
  cat(nData, "range = [", min(vData), ",", max(vData), "]\n")
}

statDescr(dfDataset$loc, "KLoC")
statDescr(dfDataset$actual, "Effort")
```

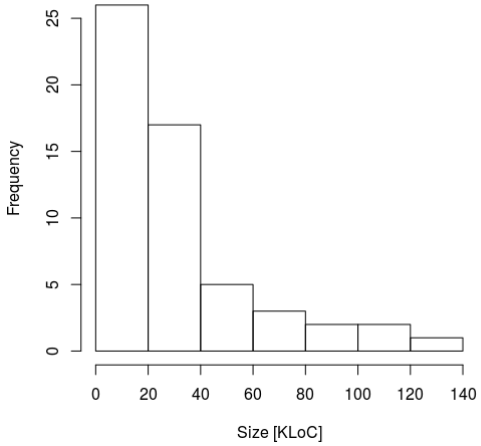

Descriptive statistics: output

```
The dataset contains 56 datapoints
Mean KLoC  = 30.28964 stdev = 31.23318
Median KLoC  = 23
KLoC range = [ 1.98 , 132 ]
Mean Effort  = 225.3607 stdev = 359.812
Median Effort  = 82.5
Effort range = [ 5.9 , 2040 ]
```

What is the frequency of project size?

- We can get very quickly an idea of the data frequency via the function that shows histograms:

```
hist(dfDataset$loc, xlab="Size [KLoC]", main="")
```



Studying correlations

- Are Size and Effort correlated?
- Several tests are available:
  - Pearson
  - Spearman
  - Kendall
- All implanted via a single R function:

```
cor.test(x, y,
    alternative = c("two.sided", "less", "greater"),
    method = c("pearson", "kendall", "spearman"),
    exact = NULL, conf.level = 0.95, continuity = FALSE,
    ...)
```

Computing Kendall's tau

```
corrTestKendall=cor.test(dfDataset$loc, dfDataset$actual,
                         method = c("kendall"))
cat("Kendall test for size and effort: tau =",
    corrTestKendall$estimate,
    " p-value =", corrTestKendall$p.value, "\n")
```

- Result:

```
Kendall test for size and effort: tau = 0.5679832 p-value =
7.365786e-10
```

- Hence, there is evidence of a statistically significant correlation

Linear regression

- Let us study the relationship between size (KLoC) and Effort (PM)
- Via Ordinary Least Square (OLS) regression.
- R provides the lm function:

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE,
   qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset,
   ...)
```

Linear regression

```
myModel = lm(actual ~ loc, data=dfDataset)
print(summary(myModel))
```

- Result:

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   29.909     56.243   0.532    0.597
loc            6.453      1.299   4.969 7.16e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 300.8 on 54 degrees of freedom
Multiple R-squared:  0.3137,    Adjusted R-squared:  0.301
F-statistic: 24.69 on 1 and 54 DF,  p-value: 7.161e-06
```


Testing the model

- Are residuals normally distributed?

- Residuals are readily available in `myModel$residuals`
- To test normality, we can use the Shapiro-Wilks test:

```
shapiro.test(myModel$residuals)
```

```
ttt=shapiro.test(myModel$residuals)
cat("normally distributed residuals: ")
if(ttt$p.value>0.05){
  cat("OK\n")
} else{
  cat("KO\n")
}
```

- Result:

```
normally distributed residuals: KO
Error in eval(expr, envir, enclos) : the End
> st
       Shapiro-Wilk normality test
data:  myModel$residuals
W = 0.7359, p-value = 1.064e-08
```


Let's try log-log transformation

- We build the model of log(y) vs. log(x).
- if log(y)=a log(x) + b, then $y = e^b x^a$

```
logX=log(dfDataset$loc)
logY=log(dfDataset$actual)
myModel = lm(logY ~ logX)
print(summary(myModel))
st=shapiro.test(myModel$residuals)
cat("normally distributed residuals: ",
    ifelse(st$p.value>0.05, "OK", "KO"), "\n")
```

The log-log model

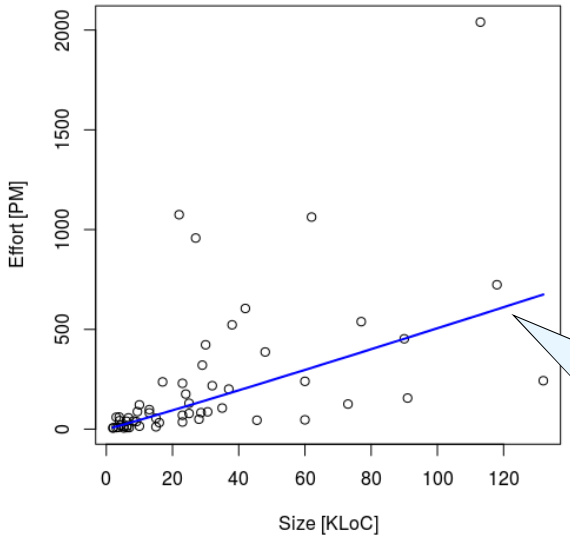
- We get a better model, and a valid one:

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.4317     0.3652   3.920 0.000251 ***
logX          1.0410     0.1185   8.785 5.44e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9673 on 54 degrees of freedom
Multiple R-squared:  0.5884,     Adjusted R-squared:  0.5807
F-statistic: 77.18 on 1 and 54 DF,  p-value: 5.443e-12

normally distributed residuals:  OK
```


Let's visualize the model

```
modelIntercept=myModel$coefficients[1]
modelCoeff=myModel$coefficients[2]
sortedLoc=sort(dfDataset$loc)
estimatedEffort=exp(modelIntercept)*sortedLoc^modelCoeff
plot(dfDataset$loc, dfDataset$actual,
     xlab="Size [KLoC]", ylab="Effort [PM]")
points(sortedLoc, estimatedEffort, type="l", lwd=2,
       col="blue")
```

Model visualization



Since the coefficient is close to one, the curve is close to a straight line.

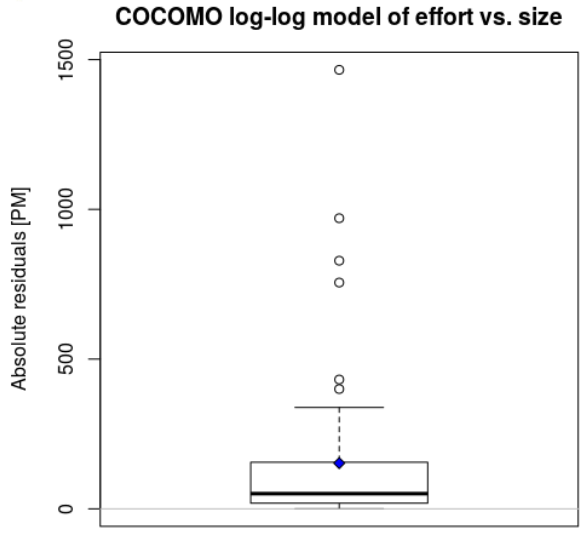
How accurate is the model?

- We compute the absolute residuals
- That is, the absolute values of estimation errors (actual-estimated)
- We are interested in the descriptive statistics of absolute residuals, and on their distribution.

Computing and showing Absolute Residuals

```
estimationErrors=dfDataset$actual-
                exp(modelIntercept)*dfDataset$loc^modelCoeff
ARs=abs(estimationErrors)
par(mar=c(1.1, 4.1, 3.1, 2.1))
boxplot(ARs, ylab="Absolute residuals [PM]",
        main="COCOMO log-log model of effort vs. size")
abline(h=0, col="gray")
points(mean(ARs), pch=23, bg="blue")
cat("Median absolute residual =", median(ARs), "PM \n")
```


Boxplot of Absolute residuals




```
Median absolute residual = 50.60129 PM
```

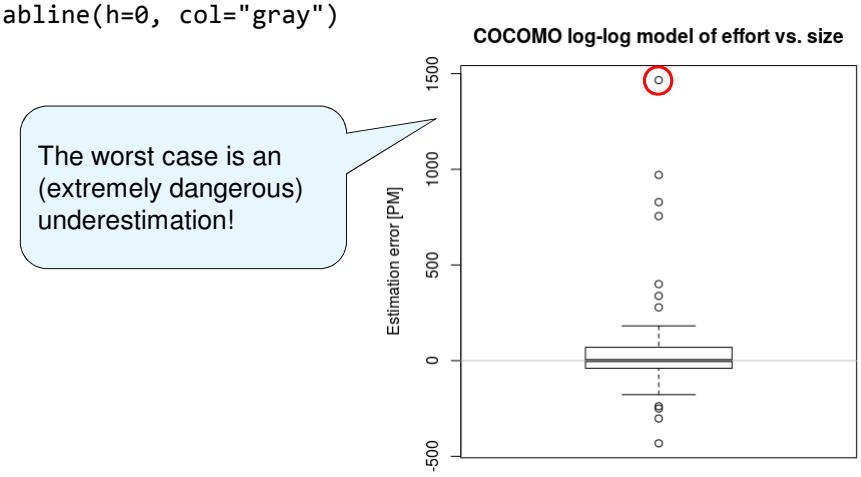
Quantiles

```
> quantile(ARs)
          0%          25%          50%          75%         100%
4.261235e-02 1.941239e+01 5.060129e+01 1.547250e+02 1.465961e+03
```

- The model is reasonably accurate in half of the cases, but is definitely very inaccurate in the worst 25% of cases (AR > 154).
- Also the maximum error is enormous (circa 1465)!

Overestimation or underestimation?

```
boxplot(estimationErrors, ylab="Estimation error [PM]",
main="COCOMO log-log model of effort vs. size")
abline(h=0, col="gray")
```



The worst case is an (extremely dangerous) underestimation!

Models with multiple independent variables

- It is generally believed that development effort depends on the size of the software to be developed as well as on its complexity.
- Let us build a model of effort based on two variables:
  - Size [KLoC]
  - Required reliability (COCOMO cost driver "rely")

Models with multiple independent variables

```
myData=data.frame(dfDataset$loc, dfDataset$rely,
dfDataset$actual)
colnames(myData)<-c("Size", "rely", "Effort")
myData=log(myData)
myModel = lm(Effort ~ Size + rely, data=myData)
print(summary(myModel))
st=shapiro.test(myModel$residuals)
cat("normally distributed residuals: ",
ifelse(st$p.value>0.05,"OK","KO"), "\n")
```

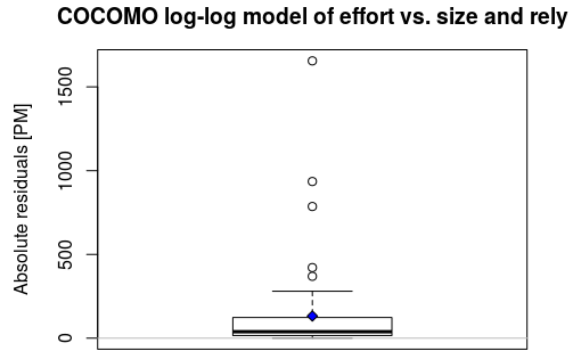
Models with multiple independent variables

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.4688     0.3271   4.490 3.88e-05 ***
Size          1.0135     0.1063   9.531 4.39e-13 ***
rely          2.4138     0.6369   3.790 0.000387 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.866 on 53 degrees of freedom
Multiple R-squared:  0.6761,     Adjusted R-squared:  0.6639
F-statistic: 55.32 on 2 and 53 DF,  p-value: 1.059e-13

normally distributed residuals:  OK
```

R^2 improved!

Accuracy

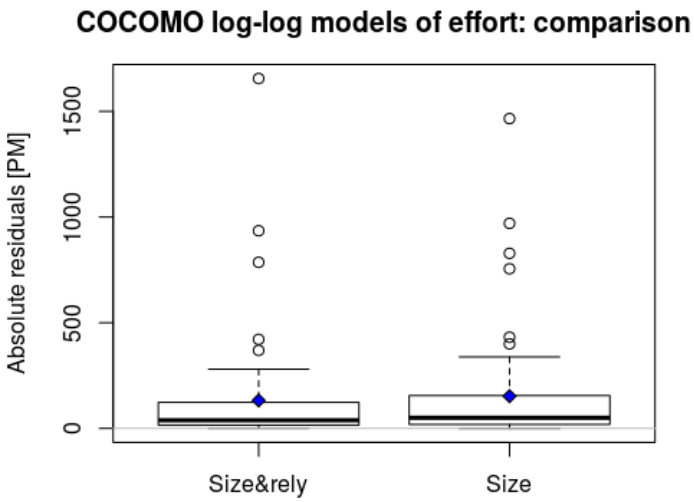
```
locOnlyARs=ARs
modelIntercept=myModel$coefficients[1]
modelCoeff1=myModel$coefficients[2]
modelCoeff2=myModel$coefficients[3]
estimatedEffort=exp(modelIntercept) *
                exp(myData$Size)^modelCoeff1 *
                exp(myData$rely)^modelCoeff2
ARs=abs(exp(myData$Effort)-estimatedEffort)
boxplot(ARs, ylab="Absolute residuals [PM]", main="COCOMO
log-log model of effort vs. size and rely")
abline(h=0, col="gray")
points(mean(ARs), pch=23, bg="blue")
cat("Median absolute residual =", median(ARs), "PM \n")
```




```
Median absolute residual = 38.46888 PM
```


Accuracy comparison

```
par(mar=c(3.1, 4.1, 3.1, 2.1))
boxplot(ARs, locOnlyARs, ylab="Absolute residuals [PM]",
        main="COCOMO log-log models of effort: comparison",
        names=c("Size&rely", "Size"))
abline(h=0, col="gray")
points(c(mean(ARs), mean(locOnlyARs)), pch=23, bg="blue")
```



What estimates for accuracy evaluation?

- In the previous slides we computed the errors of estimates concerning every project X when the estimate is computed via a model obtained from a dataset including X itself.
- This is not actually correct.
- In general, you have:
  - historical data that provide the *training set*, i.e., the dataset from which you *derive models*.
  - New applications, that provide the *test set*, i.e., the set of data to which you *apply models*.
- Two solutions (among many proposals) to simulate the situation:
  - Leave-one-out
  - N times N fold

N times N-fold cross validation

- Repeat N times the following:
  - Randomly divide the dataset into N subsets
  - For each subset S
    - Build the model using the other N-1 subsets (i.e., the dataset-S)
    - Use the obtained model to estimate the projects in subset S
    - Compute ARs and add them to the set of ARs
- The process is repeated multiple times to average out the effects of random choices.
- If the dataset is large enough, N is usually 10.
- If the dataset is relatively small, the leave-one-out method may be preferable.

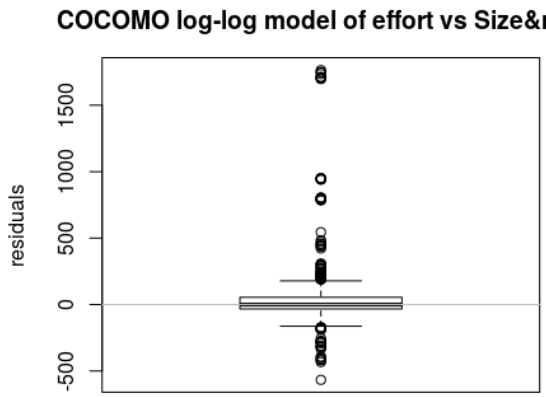
N times N-fold cross validation

```
residuals=c()
idxes=c(1:numPoints)
set.seed(12345)
for(it in 1:10){
  rndIdx=sample(idxes)  # randomized indexes
  numPointsPerteration=floor(numPoints/10)
  for(j in 1:10){
    ... # see next slide
  }
}
boxplot(residuals, main="COCOMO log-log model of effort vs Size&rely",
        ylab="residuals")
abline(h=0, col="gray")
```

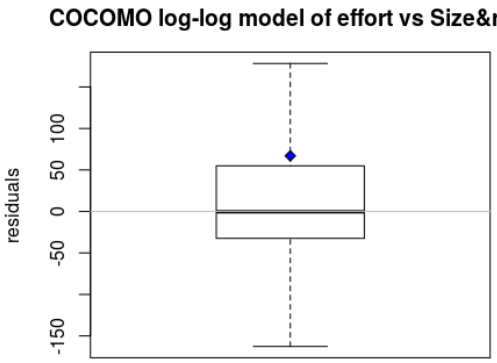
N times N-fold cross validation

```
for(j in 1:10){
  if(j==10){
    testIdxes=c((1+numPointsPerteration*(j-1)):numPoints)
  } else {
    testIdxes=c((1+numPointsPerteration*(j-1)):
                (numPointsPerteration*j))
  }
  selIdxes=rndIdx[testIdxes]
  testSet=myData[selIdxes,]
  trainingSet=myData[-selIdxes,]
  myModel = lm(Effort ~ Size + rely, data=trainingSet)
  sumModel=summary(myModel)
  vPr=sumModel$coefficients[,4]
  vPr=vPr[-1]
  st=shapiro.test(myModel$residuals)
  if(length(which(vPr>0.05))==0 && st$p.value>0.05){
    estimate=exp(predict(myModel, testSet))
    residuals=append(residuals, exp(testSet$Effort)-estimate)
  } else {} # model KO: nothing to do
```


N times N-fold cross validation



N times N-fold cross validation



- The same boxplot as in the previous slide, but without outliers.
  - Useful to understand better what happens in the central part of the distribution, without compression
- Note: the position of the mean (the blue diamond) tells that there are outliers, and that they are mostly positive!

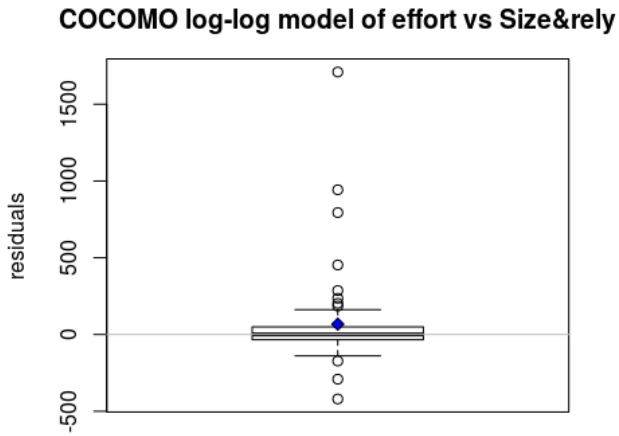
Leave-one-out evaluation

- For each data point X in the dataset:
  - Compute the model based on the dataset from which X has been removed.
  - Use the obtained model to estimate X
  - Compute the AR and add it to the set of ARs

Leave-one-out evaluation

```
numPoints=length(myData[,1])
residuals=c()
for(i in 1:numPoints){
  testSet=myData[i,]
  trainingSet=myData[-i,]
  myModel = lm(Effort ~ Size + rely, data=trainingSet)
  sumModel=summary(myModel)
  vPr=sumModel$coefficients[,4]
  vPr=vPr[-1]
  st=shapiro.test(myModel$residuals)
  if(length(which(vPr>0.05))==0 && st$p.value>0.05){
    estimate=exp(predict(myModel, testSet))
    residuals=append(residuals, exp(testSet$Effort)-estimate)
  }
}
boxplot(residuals, main="COCOMO log-log model of effort vs Size&rely",
        ylab="residuals")
points(mean(residuals), pch=23, bg="blue")
abline(h=0, col="gray")
```


Leave-one-out evaluation



Accuracy evaluation

- Quite often, accuracy of estimates is evaluated via MMRE (Mean Magnitude of Relative Error), defined as follows:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_j - \hat{y}_j|}{y_i}$$

- Problem: MMRE is a biased indicator, because it is an asymmetric measure. Example (from Shepperd, M., & MacDonell, S.G. "Evaluating prediction systems in software project estimation", Information and Software Technology 54(8), 2012):

Projects	y_i	$\hat{y}_i$	$y_i - \hat{y}_i$	$\lvert y_i - \hat{y}_i \rvert$	MMRE (%)
A	10	100	-90	90	900
B	100	10	90	90	90

- Both estimates have identical absolute residuals yet the MMRE values differ by an order of magnitude. One consequence is MMRE will be biased towards prediction systems that under-estimate.

Mean of absolute residuals

- The Mean of absolute residuals (i.e., the mean of absolute errors) is not biased.

☞ We use MAR.

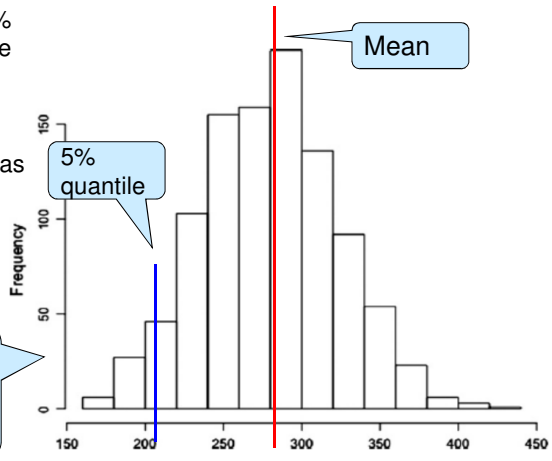
- Problem: a given value of MAR –say, 23.7– is good or bad?
- We have to be able to compare ARs.
  - All models must be better than a "baseline" model
  - If you already have a model M, a new model M' should be more accurate than M.

Baselines: the random model.

- The estimate is picked at random from the historical dataset.
- Example: in the past the effort spent for developing projects was {23, 32, 29, 15, 41, 33}.
- To estimates in a leave-one-out fashion are obtained as follows:
  - The estimate of the first project is obtained picking at random a value in {32, 29, 15, 41, 33}.
  - The estimate of the second project is obtained picking at random a value in {23, 29, 15, 41, 33}.
  - Etc.
- Of course, every time you apply this method you get a different result. To compute the MAR we need to apply the method many times (e.g., 1000 times) and take the mean of the means.

Baselines: the random model.

- The mean of the means does not provide a very strict assessment.
- It is suggested that you take the 5% quantile from random MARs.
  - The interpretation of the 5% quantile is similar to the use of α for conventional statistical inference, that is any accuracy value that is better than this threshold has a less than one in twenty chance of being a random occurrence.



The cumulative distribution of MAR values from 1000 runs

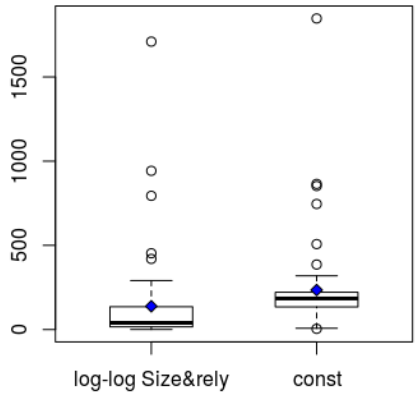
Baselines: the constant model.

- The estimate is the mean of the other values from the historical dataset.
- Example: in the past the effort spent for developing projects was {23, 32, 29, 15, 41, 33}.
- To estimates in a leave-one-out fashion are obtained as follows:
  - The estimate of the first project is the mean of {32, 29, 15, 41, 33}.
  - The estimate of the second project is the mean of {23, 29, 15, 41, 33}.
  - Etc.
- The constant model MAR is the mean of the estimates.
- Easier to compute than the random MAR.
- Provides very similar indications.

Comparing ARs

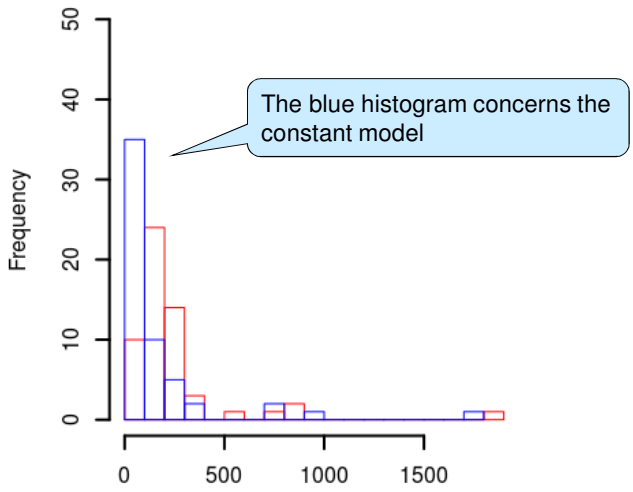
- We have two models: M1 and M2 (one can be a baseline model).
- Model M1 yields a set of residuals AR1
- Model M2 yields a set of residuals AR2
- How do we compare AR1 with AR2?
  - Via boxplots. Sometimes the visualization via boxplots is sufficient to draw conclusions.
  - Using Wilcoxon sign rank test
  - Evaluating the effect size

Boxplot comparison



- The model compared to the constant model is more accurate:
  - MAR is smaller
    - 138.4 vs. 234.7
  - The median AR is smaller
  - There are less cases with large AR

Histograms may also help



Wilcoxon sign rank test

- The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used to *compare two related samples*, matched samples, or repeated measurements on a single sample *to assess whether their population mean ranks differ*
  - it is a paired difference test.
- It can be used when the population cannot be assumed to be normally distributed.

Wilcoxon sign rank test

```
wilcox_test <- function(ARmx, ARmy, mx_name, my_name, printout){
  trWe=wilcox.test(ARmx, ARmy, alternative = "two.sided", mu = 0, paired = TRUE);
  trWg=wilcox.test(ARmy, ARmx, alternative = "greater", mu = 0, paired = TRUE);
  trWl=wilcox.test(ARmy, ARmx, alternative = "less", mu = 0, paired = TRUE);
  pmax=max(trWe$p.value, trWg$p.value, trWl$p.value)
  if(pmax==trWe$p.value && pmax>0.05){
    if(printout) {
      cat("Wilcoxon sign rank test:", my_name,
          "'s absolute residuals are equal to ", mx_name,
          "'s (p-value=", trWe$p.value, ")\n")
    }
    return("=")
  }
  if(pmax==trWg$p.value && pmax>0.05){
    if(printout) {
      cat("Wilcoxon sign rank test:", my_name,
          "'s absolute residuals are smaller than ", mx_name,
          "(p-value=", trWg$p.value, ")\n")
    }
    return(">")
  }
```

Wilcoxon sign rank test

```
 if(pmax==trWl$p.value && pmax>0.05){
    if(printout) {
      cat("Wilcoxon sign rank test:", my_name,
          "'s absolute residuals are greater than ", mx_name,
          "'s (p-value=", trWl$p.value, ")\n")
    }
    return("<")
  }
  if(printout) {
    cat("Wilcoxon sign rank test accnot draw any conclusion concerning the
absolute residuals of ",
        mx_name, " and ", my_name,"\n")
  }
  return("-")
}
```


Wilcoxon sign rank test

- Example

```
wilcox_test(abs(residuals_m1), abs(residuals_const),
"log-log model", "const model", TRUE)
```

- Result:

```
Wilcoxon sign rank test: const model's absolute
residuals are greater than log-log model's (p-value=
0.9999962)
```

The effect size

- We may found that M2 is more accurate than M1.
- Suppose you are currently using M1.
- Is switching to M2 worthwhile?
- How much is M2 more accurate than M1?

- Some indexes measure the effect size.
- We use Hedges's *g*

Effect size: code

```
require(effsize)
effectSize<- function(ARmx, ARmy, mx_name, my_name) {
  gHedges=cohen.d(ARmx, ARmy, hedges.correction=TRUE)
  es_levels=c("negligible", "small", "medium", "large")
  cat("Effect size for ", mx_name, " vs. ", my_name, " is ",
      es_levels[gHedges$magnitude], "\n")
  cat("Hedge's g =", gHedges$estimate, "\n")
  return(gHedges$estimate)
}
effectSize(abs(residuals_loo), abs(residuals_const),
           "log-log model", "const model")
```

- Result:

```
Effect size for log-log model vs. const model is small
Hedge's g = -0.3430518
```

Comparison example: conclusions

- In the example we had:
  - Visually, boxplots show that the log-log model is somewhat better than the constant model
  - Wilcoxon signed rank test confirms.
  - Hedges's g indicates that the improvement exists, but is moderate.

Dealing with outliers

- We want to check if there is any data point that affects excessively the model.
- This can be done in several ways, including by computing Cook's distance.
- Code

```
cd <- cooks.distance(lm.r);
cat("maximum Cook distance =", max(cd), "\n")
```

- Result

```
maximum Cook distance = 0.1670832
```

This is a small enough value.
If not, we could remove the responsible data point from the dataset and derive a new model.

Estimating probabilities

- Sometimes, the dependent variable is a probability (which has a value in the [0,1] range).
- In these cases, OLS and LMS regression models are not suitable
- Because they do not constrain the dependent variable in the required range.

Example

- As a running example, we consider the case when you have an object-oriented program consisting on several classes.
- For every class you know:
  - A set of OO measures
  - Whether the class is faulty (i.e., it contains one or more faults) or not.
- Objective: we want a model that –given a set of measures concerning a class– yields the probability that the class is faulty, i.e., its ***fault-proneness***.

Example

	wmc	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	loc	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	faulty
1	7	0	2	14	3	1	1	7	0.58333333	89	1.00000000	0	0.7272727	0.5714286	1	1	11.428571	2	1.2857	0
2	8	0	2	17	2	1	1	8	0.57142857	162	1.00000000	0	0.6956522	0.5833333	1	1	19.000000	8	2.3750	0
3	38	0	14	38	703	8	9	38	2.00000000	38	0.00000000	0	0.0000000	0.2421053	0	0	0.000000	1	1.0000	0
4	10	0	9	57	0	4	5	9	0.11111111	510	1.00000000	0	0.0000000	0.4250000	0	0	49.900000	3	1.1000	1
5	28	0	19	83	172	12	9	24	0.78240741	800	1.00000000	0	0.0000000	0.1785714	1	1	27.285714	23	2.5357	1
6	36	0	11	45	214	11	0	34	0.74725275	774	0.92307692	0	0.0000000	0.3314286	1	1	20.138889	52	2.9444	1
7	7	0	5	13	21	5	0	7	2.00000000	88	0.00000000	0	0.0000000	0.4285714	0	0	11.571429	4	2.0000	0
8	2	0	4	8	1	2	3	1	1.50000000	37	1.00000000	0	1.0000000	1.0000000	0	0	16.500000	0	0.0000	0
9	11	0	9	80	41	5	8	1	0.81304348	829	1.00000000	1	0.9790356	0.3818182	0	0	72.272727	7	2.0000	1
10	19	0	13	89	113	9	11	1	0.74444444	839	0.60000000	4	0.9664430	0.1754386	0	0	42.105263	7	1.4211	1
11	6	0	10	34	11	6	9	2	0.98888889	261	0.11111111	1	0.9908257	0.5000000	0	0	39.500000	1	0.6667	1
12	10	0	6	49	0	1	5	9	0.11111111	473	1.00000000	0	0.0000000	0.4250000	0	0	46.200000	3	1.1000	1
13	8	0	3	18	2	2	1	8	0.66666667	246	0.66666667	0	0.6956522	0.5833333	1	1	29.375000	7	2.0000	0
14	84	0	44	313	3022	34	44	5	0.88012048	5924	0.95000000	5	0.8490909	0.2010582	0	0	68.571429	49	2.5119	1
15	8	0	3	17	2	2	1	8	0.57142857	205	1.00000000	0	0.6956522	0.5833333	1	1	24.375000	7	2.0000	0
16	24	0	12	51	130	9	4	23	0.77173913	584	0.87500000	0	0.0000000	0.2422360	1	1	23.000000	28	2.4167	1
17	6	0	9	22	11	1	8	5	0.80000000	90	1.00000000	0	0.0000000	0.6666667	0	0	13.666667	1	0.6667	0
18	4	0	2	8	6	2	0	4	1.33333333	21	1.00000000	0	1.0000000	0.6666667	0	0	4.000000	0	0.0000	0
19	3	0	6	20	3	1	5	2	1.40000000	100	1.00000000	0	0.6666667	0.6666667	0	0	30.666667	1	0.3333	0
20	4	0	4	4	6	2	2	4	2.00000000	4	0.00000000	0	0.0000000	1.0000000	0	0	0.000000	1	1.0000	0
21	3	0	2	6	3	0	2	1	1.50000000	17	1.00000000	0	0.0000000	0.6666667	0	0	4.000000	1	0.6667	0

- This is a fragment concerning the berek project, from the PROMISEDATA dataset.

Terminology

- Positive = faulty
- Negative = not faulty

Preliminary consideration

- Given a set of classes whose actual faultiness is known, but nothing else is known, what is the probability that a given class is faulty?
- It is $\frac{AP}{n}$, where
  - n is the number of classes in the dataset
  - AP is number of Actually Positive classes in the dataset
  - In our example, n = 43, AP=16, thus AP/n = 0.372

The Binary Logistic Regression

- The Binary Logistic Regression (BLR) provides a possible solution to the problem.
- The BLR formula is:

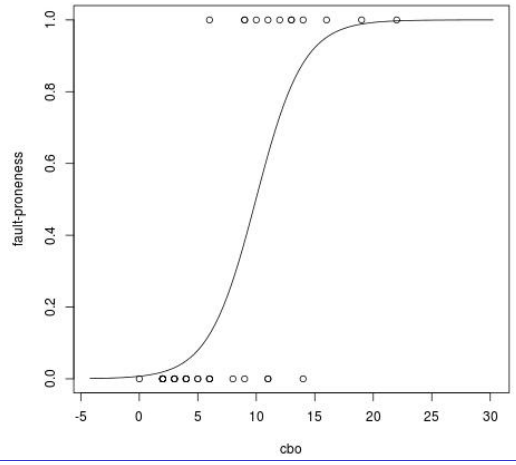
$$fp(\underline{x}) = \frac{e^{logit(\underline{x})}}{1 + e^{logit(\underline{x})}}$$

where $\underline{x}$ is a vector of variables, logit is a function of $\underline{x}$, and fp($\underline{x}$) is the fault-proneness associated to $\underline{x}$.

- Here we consider only logits of type c_0+c_1x, i.e., the logit is linear and contains a single variable.

The model

- A typical BLR model
  - The model is either monotonically increasing (as in figure below) or monotonically increasing.



Constant logit

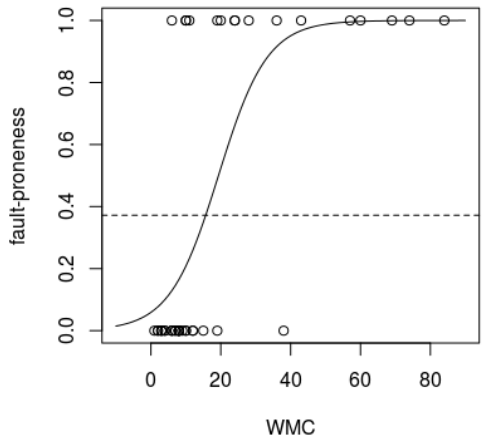
- Let us build the BLR model with no variables, that is, with a constant model
  - `lm0=glm(ErrPron ~ 1, family=binomial(link="logit"))`

No independent variables

- We get the model
  - `fp()=AP/n=0.372` !

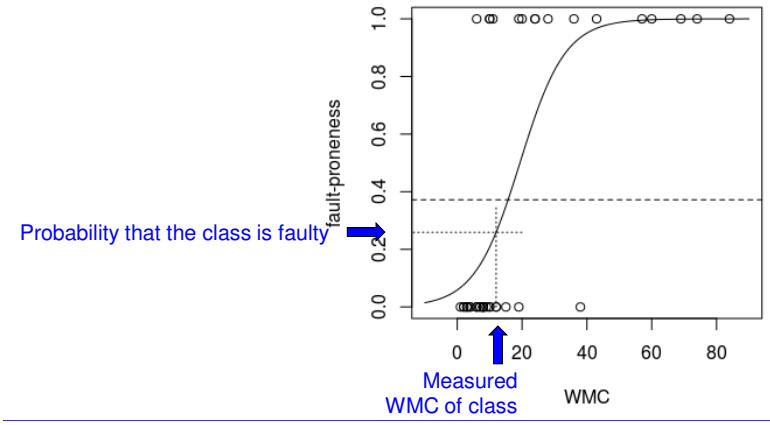
BLR computing and drawing

```
ttt=glm(faulty ~ wmc, data=S, family=binomial(link="logit"))
logitConst=ttt$coefficients[1]
logitCoeff=ttt$coefficients[2]
curve(1/(1+exp(-(logitConst+logitCoeff*x))),
      xlim=c(-10, 90), ylim=c(0,1),
      xlab="WMC", ylab="fault-proneness")
points(S$wmc, S$faulty)
abline(h=sum(S$faulty)/length(S$faulty), lty=2)
```

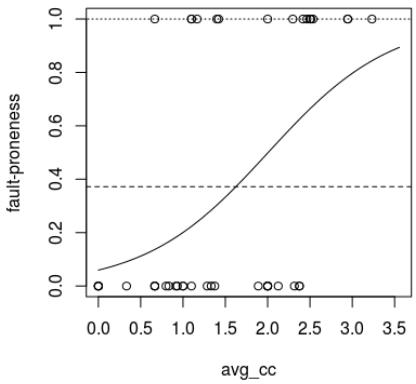
Now we can compute fault-proneness

- Now I know the probability that a class having WMC=12 is faulty.
- The probability is 0.259



Should we trust the model?

- Is the model found "good enough" to be usable?
- A few statistical tests have been proposed to test the validity of BLR models.
- However, I should always look at the resulting model, before resorting to statistical tests. For instance ….



Clearly not a good model:
- Relatively "flat"
- For a large range of x in the middle part there are approximately as many positive and negative classes.

Should we trust the model?

- A few tests have been proposed to test the validity of BLR models:
  - Sign test and Wilcoxon Signed Rank Test, to compare the Absolute Residuals of the BLR model with one independent variable with the Absolute Residuals of the BLR model with no independent variable: if the former are smaller, than it is worth building a BLR model with one independent variable
    - In R terms, you compare `abs(y-lm.g$fitted)` with `abs(y-AP/n)`
    - By the way, the same procedure is applied to check if model that uses two variables is better than a model that uses just one variable.
  - Hosmer-Lemeshow Goodness of Fit (GOF) Test
  - Wald Test for Model Coefficients
  - …

All these tests are supported by some R package.
We do not go into details.

The likely question from the practitioner

- Should I consider the class faulty or not?
- To answer, we have to move from a model of fault-proneness to a model of faultiness
  - That is, a model that provides only two possible answers:
    - Estimated positive (i.e., faulty)
    - Estimated negative (i.e., not faulty)

From fault-proneness to faultiness

- A simple solution consists in the following algorithm:

```
if (fp(x) < t)
    then faultiness=0
    else faultiness=1
```

- The difficult part is finding a proper value of t.

A "default" threshold

- Considering that AP/n is the average probability that a class is faulty, a quite "natural" solution is the following:

```
if (fp(x) < AP/n)
    then faultiness=0
    else faultiness=1
```

- Other thresholds can be set, using different criteria:
  - Risk-averse thresholds
    - lower than AP/n
  - Thresholds based on business considerations
    - E.g., given the required reliability of the product, its cost and the intended usage, one may decide that fault-proneness=0.4 is sufficient.

Accuracy evaluation

- Suppose we found a model that looks good and passes the tests.
- How can we evaluate its accuracy?
- To answer this question, we start from a basic consideration: there are four possible outcomes of estimations:
  - A positive class is estimated positive (true positive)
  - A positive class is estimated negative (false negative)
  - A negative class is estimated positive (false positive)
  - A negative class is estimated negative (true negative)
- There are several indicators that combine the count of true positive, false positives, etc.

The contingency table

- Aka confusion table or confusion matrix

EFTN \ AFTN	Negative	Positive	
Negative	TN (True Negatives)	FN (False Negatives)	EN = TN + FN (Estimated Negatives)
Positive	FP (False Positives)	TP (True Positives)	EP = FP + TP (Estimated Positives)
	AN = TN + FP (Actual Negatives)	AP = FN + TP (Actual Positives)	n = AN + AP = EN + EP

Accuracy indicators

Family	Indicator	Definition	Purpose
Positive-focused	Precision	$\frac{TP}{EP}$	extent to which we can believe that an estimated positive is actually positive
	Recall	$\frac{TP}{AP}$	extent to which actual positives are correctly estimated positive
	F-measure	$\frac{2}{\frac{1}{Precision}+\frac{1}{Recall}}$	overall estimation evaluation for positives
Negative-focused	NPV	$\frac{TN}{EN}$	extent to which we can believe that an estimated negative is actually negative
	Specificity	$\frac{TN}{AN}$	extent to which actual negatives are correctly estimated negative
	NM	$\frac{2}{\frac{1}{NPV}+\frac{1}{Specificity}}$	overall evaluation of estimation for negatives
Overall	J	$\frac{TP}{AP}-\frac{FP}{AN}$	overall evaluation of estimation for estimated positives
	Markedness	$\frac{TP}{EP}-\frac{FN}{EN}$	overall evaluation of estimation for actual positives
	ϕ	$\frac{TP\cdot TN-FP\cdot FN}{\sqrt{EN\cdot EP\cdot AN\cdot AP}}$	overall evaluation of estimation for positives and negatives

Chi-square test

- It can be shown that φ (aka Matthews correlation coefficient) is related to chi-square χ^2 , since $\varphi = \sqrt{\frac{\chi^2}{n}}$
- This is good news, since we can apply the chi square test to the contingency table and get a p-value, which tells if the result is statistically significant.

Building the contingency table

```
# ---- computes the contingency table
# in: est=estimated, yy=actuals
# out:       [,1] [,2]
#      [1,]   TP   FP
#      [2,]   FN   TN
contingency_tab <- function(est, yy){
  true_positives=yy&est; TP=sum(round(true_positives));
  true_negatives=(!yy)&(!est); TN=sum(round(true_negatives));
  false_positives=est&(!yy); FP=sum(round(false_positives));
  false_negatives=(!est)&(yy); FN=sum(round(false_negatives));
  rrr=matrix(c(TP, FN, FP, TN), nrow=2, ncol=2)
  return(rrr)
}
```


Computing and testing estimates

```
ttt=glm(faulty ~ wmc, data=S, family=binomial(link="logit"))
logitConst=ttt$coefficients[1]
logitCoeff=ttt$coefficients[2]
est=1/(1+exp(-(logitConst+logitCoeff*S$wmc)))
APonN=sum(S$faulty)/length(S$faulty)
ep=round(est>APonN)
ctab=contingency_tab(ep, S$faulty)
ctest=chisq.test(ctab)
phi=sqrt(ctest$statistic/length(S$faulty))
cat("phi =", phi, "p-value =", ctest$p.value, "\n")
```

Test set = training set

- Result

```
      [,1] [,2]
[1,]   12    2
[2,]    4   25
phi = 0.6458957 p-value = 2.281237e-05
```

Fairly good result
But 4 false negatives could be very expensive.

Advanced: ML techniques

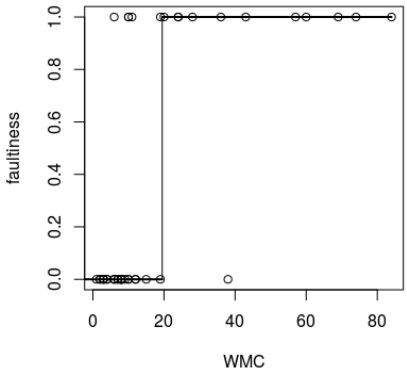
- Faultiness estimation is a (binary) classification problem.
- ML techniques can be applied to classification problems.
- There are many ML techniques.
- Here we see only

Building a J48 model

```
library(RWeka)
ttt=ifelse(S$faulty==1,"positive","negative")
S$faulty=as.factor(ttt)
modJ48 <- J48(faulty ~ wmc, data = S)
summary(modJ48) # calls evaluate_Weka_classifier()
jtab=table(S$faulty, predict(modJ48))
print(modJ48)
print(jtab)
```

Resulting J48 model

```
J48 pruned tree
------------------
wmc <= 19: negative (31.0/5.0)
wmc > 19: positive (12.0/1.0)
Number of Leaves  :   2
Size of the tree :    3

            negative positive
  negative        26        1
  positive         5       11
```



Conclusions

- Of course, there is a lot yet to say about data analysis in general and data analysis for software engineering in particular.
- Some books will help getting a more complete picture and deeper understanding.
  - Of general purpose data analysis, if not of Software Engineering data analysis.

Books

Garrett Grolemund

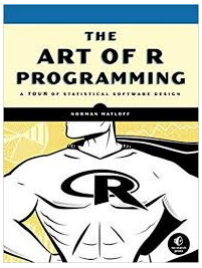
Hands-On Programming with R
Write Your Own Functions and Simulations

O'Reilly Media, 2014

Norman Matloff

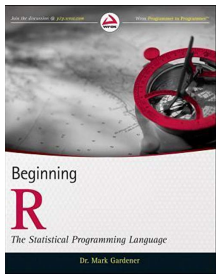
The Art of R Programming
A Tour of Statistical Software Design

2011

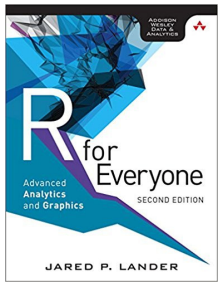


Books

- Mark Gardner
- Beginning R: The Statistical Programming
- Wiley, 2012

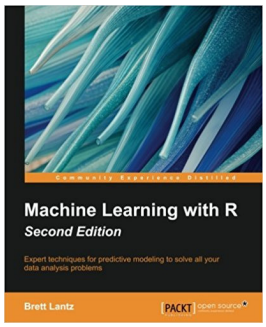


- Jared P. Lander
- R for everyone – Advanced Analytics and Graphics, 2nd ed.
- Addison-Wesley Data & Analytics Series, 2017



Books

- Brett Lantz
- Machine Learning With R, 2nd ed.



and thousands of others...