```
library(knitr)
knitr::opts_chunk$set(echo =
TRUE,tidy=TRUE,message=FALSE,warning=FALSE,strip.white=TRUE,prompt=FALSE,
                      cache=TRUE, size="scriptsize",fig.width=6, fig.height=5)
```

```
#install.packages("rmarkdown") #probably already installed
#install.packages("ggplot2") #plotting with ggplot
#install.packages("ggfortify")
#install.packages("MASS")
#install.packages("dplyr")
#install.packages("magrittr")
#install.packages("tidyverse")
#install.packages("caret")
library(magrittr)
library(knitr)
library(rmarkdown)
library(ggplot2)
library(ggfortify)
library(MASS)
library(dplyr)
library(randomForest)
library(tidyverse)
library(caret)
```

\clearpage

Classification

Overall

Classification algorithms have categorical responses. In classification we build a function f(X) that takes a vector of input variables X and predicts its class membership, such that Y in C.

Possibilities of models

There are classifiers as logistic regression, Decision tree, Perceptron / Neural networks, K-nearest-neighbors, linear and quadratic logistic regression, Bayes …

Some indicators

Sensitivity and recall

The sensitivity (also named recall) is the percentage of true defaulters that are identified (True positive tests). For example, probability of predicting disease given true state is disease.

$$sensitivity = recall = \frac{TruePositiveTests}{PositivePopulation}$$

Specificity

The specificity is the percentage of non-defaulters that are correctly identified (True negative tests). 1 - specificity is the Type 1 error, it is the false positive rate. For example, probability of predicting non-disease given true state is non- disease.

$$specificity = \frac{TrueNegativeTests}{NegativePopulation}$$

Precision

The precision is the proportion of true positive tests among the positive tests.

$$precision = \frac{TruePositiveTests}{PositiveTests}$$

F-Mesure

The traditional F measure is calculated as follows:

$$F_Measure = \frac{(2 * Precision * Recall)}{(Precision + Recall)}$$

Rand index

The rand index is a mesure of similarity between two partitions from a single set.

Given two partitions π_1 and π_2 in E : \begin{itemize} \item a, the number of elements in π_1 and π_2 \item b, the number of elements in π_1 and not in π_2 \item c, the number of elements in π_2 and not in π_1 \item d, the number of elements not in both π_1 and π_2 \end{itemize}

\begin{center} \begin{tabular} { | c | c | c |} \hline & in π_2 & not in π_2 \ \hline in π_1 & a & b \ not in π_1 & c & d \ \hline \end{tabular} \end{center}

$$RI(\pi_1, \pi_2) = \frac{a+d}{a+b+c+d}$$

Accuracy of a model

How do we determine which model is best? Various statistics can be used to judge the quality of a model. \ These include Mallow's C_p, Akaike information criterion (AIC), Bayesian information criterion (BIC), and adjusted R^2.

Let's define the mean squared error or MSE.

$$MSE = \frac{1}{n} \sum_i 1_{y_i - \hat{f}(x_i)} \$\$ where : \$\$ 1_{y_i - \hat{f}(x_i)} = \begin{cases} 1 & \text{if } y_i != \hat{f}(x_i) \\ 0 & \text{otherwise} \end{cases}$$

Recall : $\$RSS = MSE * n\$$

RSS and R^2 are not suitable for selecting the best model among a collection of models with different numbers of predictors.

Mallow's Cp

TO DO

If there are d predictors :

$$C_p = \frac{RSS + 2d\hat{\sigma}^2}{n}$$

AIC : Akaike information criterion

TO DO

The AIC criterion is defined for a large class of models fit by maximum likelihood.

$$AIC = \frac{RSS + 2d\hat{\sigma}^2}{n\hat{\sigma}^2}$$

To use AIC for model selection, we simply choose the model giving small- est AIC over the set of models considered.

BIC : Bayesian information criterion

TO DO

BIC is derived from a Bayesian point of view, but ends up looking similar to Cp (and AIC) as well. For the least squares model with d predictors, the BIC is, up to irrelevant constants, given by

$$BIC = \frac{RSS + log(n)d\hat{\sigma}^2}{n}$$

Adjusted R statistic

TO DO

$$AdustedR^2 = 1 - \frac{\frac{RSS}{n-d-1}}{\frac{TSS}{n-1}}$$

Logistic Regression

How it works

In logistic regression, for covariates (X_1 , . . . , X_p), we want to estimate $p_i = P_r(Y_i = 1|X_1, \ldots, X_p)$

$$p_i = \frac{e^{\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4} + \ldots}}{1 + e^{\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4} + \ldots}}$$

To come back to linear regression we define the logistic function as follow.

$$logit(p_i) = log(\frac{p_i}{1-p_i}) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4} + \ldots$$

We can define the odds :

$$\frac{odds(Y_i = 1|X1 = x_{i1} + 1)}{odds(Y_i = 1|X1 = x_{i1})} = e^{\beta_1}$$

Which indicator to construct the model ?

We use Maximum Likehood :

$$L(\beta) = \Pi_{i=1}^{n} p_i^{y_i} * (1 - p_i)^{y_i}$$

The goal is to maximise it by adjusting β vector.

Example on the the Wimbledon tennis tournament

We use a dataset from the Wimbledon tennis tournament for Women in 2013. We will predict the result for player 1 (win=1 or loose=0) based on the number of aces won by each player and the number of unforced errors commited by both players. The data set is a subset of a data set from https://archive.ics.uci.edu/ml/datasets/Tennis+Major+Tournament+Match+Statistics.

```
id <- "1GNbIhjdhuwPOBr0Qz82JMkdjUVBuSoZd"
tennis <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download",id), header
= T)
head(tennis)

# test and train set
n = dim(tennis)[1]
n2 = n*(3/4)
set.seed(1234)
train = sample(c(1:n), replace = F)[1:n2]
```

On R

```
# reduction to two variables
tennis$ACEdiff = tennis$ACE.1 - tennis$ACE.2
tennis$UFEdiff = tennis$UFE.1 - tennis$UFE.2
head(tennis)
tennisTest = tennis[-train, ]
tennisTrain = tennis[train, ]
r.tennis2 = glm(Result ~ ACEdiff + UFEdiff, data = tennisTrain, family = "binomial")
summary(r.tennis2)
```

With the model, we can draw the slope which indicates the category of a point.

```
#We calculate the slope
glm.b = -r.tennis2$coefficients[2]/r.tennis2$coefficients[3]
glm.a = -r.tennis2$coefficients[1]/r.tennis2$coefficients[3]

ggplot() + geom_point(aes(ACEdiff, UFEdiff, color = factor(Result)), data = tennisTrain,
) + scale_color_manual(values = c("red", "green")) +
  geom_abline(slope = glm.b, intercept = glm.a) +
  theme_minimal()
```

We can write :

$$logit(p_i) = log(\frac{p_i}{1-p_i}) = 0,31318 + 0,20856 * ACEDiff - 0,08272 * UFEDiff$$

We can observe AIC = 105.1

The confusion matrix is :

```
glm.Result_probs = predict(r.tennis2, newdata = tennisTest)
glm.Result_pred = ifelse(glm.Result_probs > 0.5, 1, 0)
glm.confusion_matrix = table(glm.Result_pred, tennisTest$Result)
glm.confusion_matrix
```

The accuracy rate is $\frac{15+8}{30} = 0.7667$.

The sensitivity is the percentage of true output giving Player1-winner among the population of true Player1-winner :

```
glm.sensitivity = glm.confusion_matrix[2,2]/(glm.confusion_matrix[1,2] +
glm.confusion_matrix[2,2])
glm.sensitivity
```

The specificity is the percentage of true output giving Player2-winner (= Player1-looser) among the population of true Player2-winner:

```
glm.specificity = glm.confusion_matrix[1,1]/(glm.confusion_matrix[1,1] +
glm.confusion_matrix[2,1])
glm.specificity
```

The precision is the percentage of true output giving Player1-winner among all the outputs giving Player1-winner (even if not winner) :

```
glm.precision = glm.confusion_matrix[2,2]/(glm.confusion_matrix[2,1] +
glm.confusion_matrix[2,2])
glm.precision
```

So the F_Mesure is :

```
glm.fmesure = (2*glm.precision*glm.sensitivity)/(glm.sensitivity + glm.precision)
glm.fmesure
```

In python with scikit learn

Decision trees and Random Forest

How it works

TO DO

Which indicator to construct the model ?

Entropy

TO DO

Gine

TO DO

Example on the the Wimbledon tennis tournament

On R

```
accuracyrate <- rep(NA,40)
deg = 1:40
for (d in deg) {
  model <-  randomForest(Result ~ ACE.1 + ACE.2 + UFE.1 + UFE.2, tennisTrain,
                          mtry = 2, ntree = 500, nodesize=d,importance = TRUE)
  yRandomForest = predict(model, newdata = tennisTest)
  model.Result_probs = predict(model, newdata = tennisTest)
  model.Result_pred = ifelse(model.Result_probs > 0.5, 1, 0)
  model.confusion_matrix = table(model.Result_pred, tennisTest$Result)
  model.accuracyrate = (model.confusion_matrix[2,2] + model.confusion_matrix[1,1]) /30
  accuracyrate[d] = model.accuracyrate
}

#The model with the smallest MSE has 14 nodesizes
which.min(accuracyrate)

#The best model is
model <-  randomForest(Result ~ ACE.1 + ACE.2 + UFE.1 + UFE.2, tennisTrain,
                        mtry = 2, ntree = 500,
nodesize=which.min(accuracyrate),importance = TRUE)

predictions <- model %>% predict(tennisTest)
data.frame( MSE = mean((predictions - tennisTest$Result)^2),
            R2 = R2(predictions, tennisTest$Result),
            RMSE = RMSE(predictions, tennisTest$Result),
            MAE = MAE(predictions, tennisTest$Result))
```

The confusion matrix is :

```
model.Result_probs = predict(model, newdata = tennisTest)
model.Result_pred = ifelse(model.Result_probs > 0.5, 1, 0)
model.confusion_matrix = table(model.Result_pred, tennisTest$Result)
model.confusion_matrix
```

The accuracy rate is :

```
model.accuracyrate = (model.confusion_matrix[2,2] + model.confusion_matrix[1,1]) /30
model.accuracyrate
```

The sensitivity is the percentage of true output giving Player1-winner among the population of true Player1-winner :

```
model.sensitivity = model.confusion_matrix[2,2]/(model.confusion_matrix[1,2] +
model.confusion_matrix[2,2])
model.sensitivity
```

The specificity is the percentage of true output giving Player2-winner (= Player1-looser) among the population of true Player2-winner:

```
model.specificity = model.confusion_matrix[1,1]/(model.confusion_matrix[1,1] +
model.confusion_matrix[2,1])
model.specificity
```

The precision is the percentage of true output giving Player1-winner among all the outputs giving Player1-winner (even if not winner) :

```
model.precision = model.confusion_matrix[2,2]/(model.confusion_matrix[2,1] +
model.confusion_matrix[2,2])
model.precision
```

So the F_Mesure is :

```
model.fmesure = (2*model.precision*model.sensitivity)/(model.sensitivity +
model.precision)
model.fmesure
```

In python with scikit learn

```python
import pandas as pd
import urllib
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
```

Tennis dataset

Classification

Collecting the dataset

```
tennis_dataset_url = "https://docs.google.com/uc?id=%s&export=download"
urlRequest = urllib.request.Request(tennis_dataset_url)
datasetFile = urllib.request.urlopen(urlRequest)
```

```
tennis_dataset = pd.read_csv(datasetFile, header=0)
```

Overview of the data

```
len(tennis_dataset)
```

```
118
```

```
tennis_dataset.shape
```

```
(118, 7)
```

```
tennis_dataset.head()
```

	Player1	Player2	Result	ACE.1	UFE.1	ACE.2	UFE.2
0	M.Koehler	V.Azarenka	0	2	18	3	14
1	E.Baltacha	F.Pennetta	0	0	10	4	14
2	S-W.Hsieh	T.Maria	1	1	13	2	29
3	A.Cornet	V.King	1	4	30	0	45
4	Y.Putintseva	K.Flipkens	0	2	28	6	19

```
tennis_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 118 entries, 0 to 117
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Player1  118 non-null    object
 1   Player2  118 non-null    object
 2   Result   118 non-null    int64
 3   ACE.1    118 non-null    int64
 4   UFE.1    118 non-null    int64
 5   ACE.2    118 non-null    int64
 6   UFE.2    118 non-null    int64
dtypes: int64(5), object(2)
memory usage: 6.6+ KB
```

```
tennis_dataset.describe()
```

	Result	ACE.1	UFE.1	ACE.2	UFE.2
count	118.000000	118.000000	118.000000	118.000000	118.000000
mean	0.533898	2.974576	20.177966	3.271186	20.466102
std	0.500977	2.835857	10.248728	3.188283	11.444912
min	0.000000	0.000000	4.000000	0.000000	2.000000
25%	0.000000	1.000000	13.000000	1.000000	12.000000
50%	1.000000	2.000000	18.000000	2.000000	18.000000
75%	1.000000	4.000000	25.750000	5.000000	27.000000
max	1.000000	14.000000	54.000000	15.000000	55.000000

```
tennis_dataset["Result"].value_counts() # Check if the dataset is balanced
```

```
1    63
0    55
Name: Result, dtype: int64
```

Feature selection

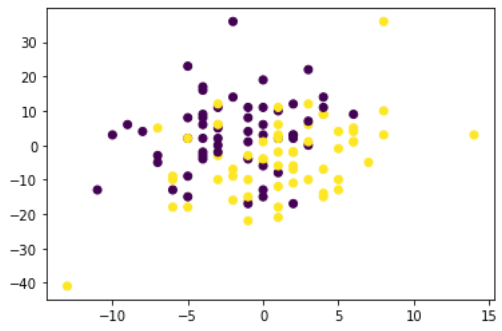
```
tennis_dataset['ACE'] = tennis_dataset['ACE.1'] - tennis_dataset["ACE.2"]
tennis_dataset['UFE'] = tennis_dataset['UFE.1'] - tennis_dataset["UFE.2"]
tennis_dataset.describe()
```

	Result	ACE.1	UFE.1	ACE.2	UFE.2	ACE	UFE
count	118.000000	118.000000	118.000000	118.000000	118.000000	118.000000	118.000000
mean	0.533898	2.974576	20.177966	3.271186	20.466102	-0.296610	-0.288136
std	0.500977	2.835857	10.248728	3.188283	11.444912	4.356564	11.410822
min	0.000000	0.000000	4.000000	0.000000	2.000000	-13.000000	-41.000000
25%	0.000000	1.000000	13.000000	1.000000	12.000000	-3.000000	-7.750000
50%	1.000000	2.000000	18.000000	2.000000	18.000000	0.000000	1.000000
75%	1.000000	4.000000	25.750000	5.000000	27.000000	2.000000	6.000000
max	1.000000	14.000000	54.000000	15.000000	55.000000	14.000000	36.000000

```
tennis_dataset = tennis_dataset.drop(columns=["ACE.1","UFE.1","ACE.2", "UFE.2"])
tennis_dataset.describe()
```

	Result	ACE	UFE
count	118.000000	118.000000	118.000000
mean	0.533898	-0.296610	-0.288136
std	0.500977	4.356564	11.410822
min	0.000000	-13.000000	-41.000000
25%	0.000000	-3.000000	-7.750000
50%	1.000000	0.000000	1.000000
75%	1.000000	2.000000	6.000000
max	1.000000	14.000000	36.000000

```
plt.scatter(tennis_dataset["ACE"], tennis_dataset["UFE"], c=tennis_dataset["Result"])
plt.show()
```



Trying Models

Logistic Regression

```
def defineTestTrainDatasetRandomly():
    msk = np.random.rand(len(tennis_dataset)) < 0.75

    train = tennis_dataset[msk]
    test = tennis_dataset[~msk]

    trainX = train[['ACE', 'UFE']]
    trainY = train[['Result']]

    testX = test[['ACE', 'UFE']]
    testY = test[['Result']]
    return (trainX, trainY, testX, testY)
```

```
meanAccuracy = 0
meanPerfModel = [0,0,0,0] # mean respectively of tn, fp, fn, tp
nbrOfIteration = 100
for i in range(0,nbrOfIteration):
    trainX, trainY, testX, testY = defineTestTrainDatasetRandomly()
    clf = LogisticRegression(C=1e5).fit(trainX, trainY.values.ravel())
    labelsPredicted = clf.predict(testX)
    meanPerfModel += confusion_matrix(labelsPredicted, testY.values.ravel()).ravel()
    meanAccuracy += clf.score(testX, testY.values.ravel())
meanAccuracy /= nbrOfIteration
meanPerfModel = [i/nbrOfIteration for i in meanPerfModel]
print(meanAccuracy)
print(meanPerfModel)
```

```
0.7431425827232663
[9.48, 3.23, 4.4, 12.63]
```

Using the Logistic Regression model in scikit we obtain an accuracy of 0.74. In average, this model has 9.48 True Positive, 3.23 False Positive, 4.4 False Negative, 12.63 True Positive.

```
sensivity = meanPerfModel[0]/(meanPerfModel[0]+meanPerfModel[2])
print("The Sensitivity is : " + str(sensivity))
specificity = meanPerfModel[3]/(meanPerfModel[3]+meanPerfModel[1])
print("The specificity is : " + str(specificity))
precision = meanPerfModel[0]/(meanPerfModel[0]+meanPerfModel[1])
print("The precision is : " + str(precision))
Fmesure = (2*precision*sensivity)/(precision+sensivity)
print("So, we can deduce that the F-mesure is : " + str(Fmesure))
```

```
The Sensitivity is : 0.6829971181556196
The specificity is : 0.7963430012610341
The precision is : 0.7458693941778127
So, we can deduce that the F-mesure is : 0.7130500188040616
```

Random forest

```
meanAccuracy = 0
meanPerfModel = [0,0,0,0] # mean respectively of tn, fp, fn, tp
nbrOfIteration = 100
for i in range(0,nbrOfIteration):
    trainX, trainY, testX, testY = defineTestTrainDatasetRandomly()
    clf = RandomForestClassifier(max_depth=6, random_state=0).fit(trainX,
trainY.values.ravel())
    labelsPredicted = clf.predict(testX)
    meanPerfModel += confusion_matrix(labelsPredicted, testY.values.ravel()).ravel()
    meanAccuracy += clf.score(testX, testY.values.ravel())
meanAccuracy /= nbrOfIteration
meanPerfModel = [i/nbrOfIteration for i in meanPerfModel]
print(meanAccuracy)
print(meanPerfModel)
```

```
0.6698461736596538
[8.79, 4.71, 5.15, 11.19]
```

```
sensivity = meanPerfModel[0]/(meanPerfModel[0]+meanPerfModel[2])
print("The Sensitivity is : " + str(sensivity))
specificity = meanPerfModel[3]/(meanPerfModel[3]+meanPerfModel[1])
print("The specificity is : " + str(specificity))
precision = meanPerfModel[0]/(meanPerfModel[0]+meanPerfModel[1])
print("The precision is : " + str(precision))
Fmesure = (2*precision*sensivity)/(precision+sensivity)
print("So, we can deduce that the F-mesure is : " + str(Fmesure))
```

```
The Sensitivity is : 0.6305595408895265
The specificity is : 0.7037735849056604
The precision is : 0.6511111111111111
So, we can deduce that the F-mesure is : 0.64067055393586
```

\clearpage

Regression

Overall

TO DO

Possibilities of models

TO DO

Accuracy of a model

MSE : Mean Squarred Error

The MSE mesures the mean accuracy of the predicted responses values for given observations. There are two MSE : the train MSE and the test MSE. \ The train MSE is use to fit a model while training. \ The test MSE is use to choose between models already trained. \

Let's define the mean squared error or MSE.

$$MSE = \frac{1}{n}\sum_i (y_i - \hat{f}(x_i))^2$$

Then the expected test MSE refers to the average test MSE that we would obtain if we repeatedly estimated f using a large number of training sets, and tested each at x_0. So that the expected test MSE is :

$$E(y_0 - \hat{f}(x_0))^2$$

$$E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + (f(x_0) - E(\hat{f}(x_0)))^2 + Var(\varepsilon)$$

$Var(\varepsilon)$ represents the irreductible error. This term can not be reduced regardless how well our statstical model fits the data.

$(f(x_0) - E(\hat{f}(x_0))^2 = [Bias(\hat{f}(x_0))]^2$ is the squared Bias and refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model. If the bias is low the model gives a prediction which is close to the true value.

$Var(\hat{f}(x_0))$ is the Variance of the prediction at $\hat{f}(x_0)$ and refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. If the variance is high, there is a large uncertainty associated with the prediction.

RMSE : Root Mean Squared Error

Root Mean Squared Error (RMSE), which measures the average prediction error made by the model in predicting the outcome for an observation. That is, the average difference between the observed known outcome values and the values predicted by the model. The lower the RMSE, the better the model.

RSS : Residual Sum of Squares

We define the residual sum of squares (RSS) as :

$$RSS = \Sigma(y_i - \hat{y}_i)^2$$

We want to minimize the RSS.

RSE : Residual Standard Error

TO DO

$$RSE = \sqrt{\frac{1}{n-2}RSS}$$

R statistic

TO DO

$$R^2 = 1 - \frac{RSS}{TSS}$$

TSS = \Sigma (y_i - \bar{y}_i)^2$$ is the total sum of squares. TSS measures the total variance in the response Y.

TSS − RSS measures the amount of variability in the response that is explained.

$R^2$ measures the proportion of variability in Y that can be explained using X.

### MAE

Mean Absolute Error (MAE), an alternative to the RMSE that is less sensitive to outliers. It corresponds to the average absolute difference between

## Simple Linear Regression

### Definition

TO DO

DEFINITION

WHICH INDICATORS CAN WE USE

Simple linear regression lives up to its name: it is a very straightforward approach for predicting a quantitative response Y on the basis of a single pr
$$Y ≈ \beta_0 + \beta_1 * X

Hospital Costs dataset

The next dataset (source F. E. Harrell, Regression Modeling Strategies) contains the total hospital costs of 9105 patients with certain diseases in American hospitals between 1989 and 1991. The different variables are :

```
id <- "1heRtzi8vBoBGMaM2-ivBQI5Ki3HgJTmO" # google file ID
hospitaldata <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download",
id), header = T)
head(hospitaldata)
```

```
# We only look at complete cases
hospitaldata <- hospitaldata[complete.cases(hospitaldata), ]
hospitaldata <- hospitaldata[hospitaldata$totcst > 0, ]
```

```
# histograms
par(mfrow = c(3, 3))
hist(hospitaldata$age, main = 'age')
hist(hospitaldata$num.co, main = 'num.co')
hist(hospitaldata$edu, main = 'edu')
hist(hospitaldata$scoma, main = 'scoma')
hist(hospitaldata$totcst, main = 'totcst')
hist(hospitaldata$meanbp, main = 'meanbp')
hist(hospitaldata$hrt, main = 'hrt')
hist(hospitaldata$resp, main = 'resp')
hist(hospitaldata$temp, main = 'temp')
hist(hospitaldata$pafi, main = 'pafi')

#transformation
par(mfrow = c(1, 2))
hist(hospitaldata$totcst, main = 'totcst')
hist(log(hospitaldata$totcst), main = 'log(totcst)')
```

```
ggplot() + geom_point(aes(age, totcst, color = as.factor(dzgroup) ), data = hospitaldata)
```

On R

We would like to build models that help us to understand which predictors are mostly driving the total cost.

Looking at the distribution of the cost we see we should apply a log transformation for a better distribution. Moreover it seems that only age and disease have an impact.

```
set.seed(12345)
train.proportion = 0.7
train.ind = sample(1:nrow(hospitaldata), train.proportion* nrow(hospitaldata))
hospitaldata.train = hospitaldata[train.ind, ]
hospitaldata.test = hospitaldata[-train.ind, ]

fit = lm(log(totcst)~ age + temp + edu + resp + num.co + as.factor(dzgroup),
         data = hospitaldata.train)
summary(fit)
```

We can that just age and dzgroup seem to have an impact on totcst.

```
fit = lm(log(totcst)~ age + as.factor(dzgroup) , data = hospitaldata.train)
summary(fit)
```

We can write :

$$log(totcost) = 8.0823597 - 0.0069950 * age + x_{ij} * \beta_j$$

where x_{ij} is 1 if patient i has disease j and β_j is the coefficient matchinf the disease in the previous tab.

We can calculate the MSE on the test set to evaluate the simple linear regression model.

```
predictions <- fit %>% predict(hospitaldata.test)
data.frame( MSE = mean((predictions - log(hospitaldata.test$totcst))^2),
            R2 = R2(predictions, log(hospitaldata.test$totcst)),
            RMSE = RMSE(predictions, log(hospitaldata.test$totcst)),
            MAE = MAE(predictions, log(hospitaldata.test$totcst)))
```

In python with scikit learn

```python
import pandas as pd
import urllib
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_validate
```

Hopital Cost dataset

Collecting the data

```
hopital_dataset_url = "https://docs.google.com/uc?id=1heRtzi8vBoBGMaM2-
ivBQI5Ki3HgJTmO&export=download"
urlRequest = urllib.request.Request(hopital_dataset_url)
datasetFile = urllib.request.urlopen(urlRequest)
hopital_dataset = pd.read_csv(datasetFile, header=0)
```

Overview of the dataset

```
len(hopital_dataset)
```

```
9105
```

```
hopital_dataset.tail()
```

	age	dzgroup	num.co	edu	income	scoma	totcst	race	meanbp	h
9100	66.07	ARF/MOSF w/Sepsis	1	8.0	NaN	0.0	34329.31	white	109.0	104.
9101	55.15	Coma	1	11.0	NaN	41.0	23558.50	white	43.0	0.0
9102	70.38	ARF/MOSF w/Sepsis	1	NaN	NaN	0.0	31409.02	white	111.0	83.0
9103	47.02	MOSF w/Malig	1	13.0	NaN	0.0	NaN	white	99.0	110.
9104	81.54	ARF/MOSF w/Sepsis	1	8.0	$11-$25k	0.0	10605.76	white	75.0	69.0

We can see that there is a lot of NaN values

```
hopital_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9105 entries, 0 to 9104
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       9105 non-null   float64
 1   dzgroup   9105 non-null   object 
 2   num.co    9105 non-null   int64  
 3   edu       7471 non-null   float64
 4   income    6123 non-null   object 
 5   scoma     9104 non-null   float64
 6   totcst    8217 non-null   float64
 7   race      9063 non-null   object 
 8   meanbp    9104 non-null   float64
 9   hrt       9104 non-null   float64
 10  resp      9104 non-null   float64
 11  temp      9104 non-null   float64
 12  pafi      6780 non-null   float64
dtypes: float64(9), int64(1), object(3)
memory usage: 924.9+ KB
```

```
hopital_dataset.describe()
```

	age	num.co	edu	scoma	totcst	meanbp	
count	9105.000000	9105.000000	7471.000000	9104.000000	8217.000000	9104.000000	9104.0
mean	62.650490	1.868644	11.747691	12.058546	30825.868110	84.546408	97.156
std	15.593675	1.344409	3.447743	24.636694	45780.821374	27.687692	31.559
min	18.040000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	52.800000	1.000000	10.000000	0.000000	5929.570000	63.000000	72.000
50%	64.860000	2.000000	12.000000	0.000000	14452.730000	77.000000	100.00
75%	74.000000	3.000000	14.000000	9.000000	36087.940000	107.000000	120.00
max	101.850000	9.000000	31.000000	100.000000	633212.000000	195.000000	300.00

```
hopital_dataset.isnull().sum()
```

```
age           0
dzgroup       0
num.co        0
edu        1634
income     2982
scoma         1
totcst      888
race         42
meanbp        1
hrt           1
resp          1
temp          1
pafi       2325
dtype: int64
```

```
hopital_dataset = hopital_dataset.dropna()
```

```
hopital_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3852 entries, 17 to 9104
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   age      3852 non-null   float64
 1   dzgroup  3852 non-null   object 
 2   num.co   3852 non-null   int64  
 3   edu      3852 non-null   float64
 4   income   3852 non-null   object 
 5   scoma    3852 non-null   float64
 6   totcst   3852 non-null   float64
 7   race     3852 non-null   object 
 8   meanbp   3852 non-null   float64
 9   hrt      3852 non-null   float64
 10  resp     3852 non-null   float64
 11  temp     3852 non-null   float64
 12  pafi     3852 non-null   float64
dtypes: float64(9), int64(1), object(3)
memory usage: 421.3+ KB
```

Remove NaN and null data

```
hopital_dataset.isnull().sum()
```

```
age        0
dzgroup    0
num.co     0
edu        0
income     0
scoma      0
totcst     0
race       0
meanbp     0
hrt        0
resp       0
temp       0
pafi       0
dtype: int64
```

```
hopital_dataset = hopital_dataset[hopital_dataset["totcst"]>0]
```

```
hopital_dataset["totcst"]
```

```
17      156674.13
22      288592.25
29      304749.25
31       28262.70
35      103090.44
          ...    
9091     13088.14
9096     12187.20
9097      4100.55
9099      1847.38
9104     10605.76
Name: totcst, Length: 3839, dtype: float64
```

```
# cost to log(cost)
hopital_dataset["totcst"] = np.log(hopital_dataset["totcst"])
# change the text labels to numbers because it's easier to process
hopital_dataset["dzgroup"] = pd.factorize(hopital_dataset["dzgroup"])[0]
```

```
hopital_dataset = hopital_dataset.drop("scoma", axis=1)
hopital_dataset = hopital_dataset.drop("race", axis=1)
hopital_dataset = hopital_dataset.drop("meanbp", axis=1)
hopital_dataset = hopital_dataset.drop("income", axis=1)
hopital_dataset = hopital_dataset.drop("hrt", axis=1)
hopital_dataset = hopital_dataset.drop("pafi", axis=1)
```

```
def defineTestTrainDatasetRandomly():
    msk = np.random.rand(len(hopital_dataset)) < 0.75

    train = hopital_dataset[msk]
    test = hopital_dataset[~msk]

    trainX = train.drop("totcst", axis=1)
    trainY = train[['totcst']]

    testX = test.drop("totcst", axis=1)
    testY = test[['totcst']]
    return (trainX, trainY, testX, testY)
```

```
trainX, trainY, testX, testY = defineTestTrainDatasetRandomly()
trainX
```

	age	dzgroup	num.co	edu	resp	temp
17	63.66	0	0	22.0	22.0	36.70
22	49.61	0	1	12.0	48.0	38.90
31	55.73	1	2	8.0	18.0	37.40
35	57.53	1	1	14.0	18.0	37.59
36	68.99	2	1	10.0	26.0	36.59
...	...	...	...	...	...	...
9079	53.32	5	2	12.0	20.0	36.40
9085	18.41	0	1	12.0	30.0	36.00
9086	70.48	2	2	12.0	32.0	36.00
9090	64.51	2	2	8.0	34.0	36.00
9104	81.54	0	1	8.0	24.0	36.20

2907 rows × 6 columns

```
trainX, trainY, testX, testY = defineTestTrainDatasetRandomly()
lm_reg = linear_model.Ridge(alpha=.5)
lm_reg.fit(trainX, trainY.values.ravel())
print(lm_reg.coef_)

lm_reg.intercept_
```

```
[-0.00862595 -0.16897914 -0.17462313  0.03123692 -0.00433157  0.12588791]
```

```
5.97605554119929
```

```
cv_results = cross_validate(lm_reg, trainX, trainY, cv=5,
                            scoring={'r2':'r2', 'MSE': 'neg_mean_squared_error',
                                     'MAE':"neg_median_absolute_error",
                                     'RMSE': "neg_root_mean_squared_error"})
```

```
cv_results
```

```
{'fit_time': array([0.00454998, 0.00381041, 0.00367999, 0.00376892, 0.00349855]),
 'score_time': array([0.00687885, 0.00751424, 0.0059278 , 0.00661039, 0.00594759]),
 'test_r2': array([0.03264289, 0.16992049, 0.21237732, 0.09746144, 0.00523087]),
 'test_MSE': array([-1.43892554, -0.93668451, -1.06340127, -1.16556294, -1.30473453]),
 'test_MAE': array([-0.85371531, -0.68837916, -0.72152959, -0.82103541, -0.87052994]),
 'test_RMSE': array([-1.19955222, -0.96782463, -1.03121349, -1.0796124 , -1.14224977])}
```

Multiple linear regression

Definition

TO DO

DEFINITION

WHICH INDICATORS ?

Hospital Costs dataset

On R

We use the same example than for simple linear regression.

```
fit_multiple = lm(log(totcst)~age*as.factor(dzgroup), data = hospitaldata.train)
summary(fit_multiple)
```

We can calculate the MSE on the test set to evaluate the multiple linear regression model.

```
predictions <- fit_multiple %>% predict(hospitaldata.test)
data.frame( MSE = mean((predictions - log(hospitaldata.test$totcst))^2),
            R2 = R2(predictions, log(hospitaldata.test$totcst)),
            RMSE = RMSE(predictions, log(hospitaldata.test$totcst)),
            MAE = MAE(predictions, log(hospitaldata.test$totcst)))
```

The MSE-test for multiple linear regression is worst than for simple linear regression.

Simple linear regression is the best model so far for this problem.

In python with scikit learn

\clearpage

Validation techniques

Sampling

This consists in dividing the dataset into a training set and a test set.

Cross validation

R2, RMSE and MAE are used to measure the regression model performance during cross-validation.

Validation set approach

TO DO

Example on R

```
# Split the data into training and test set
set.seed(123)
training.samples <- log(hospitaldata$totcst) %>% createDataPartition(p = 0.8, list =
FALSE)
hospitaldata.train2  <- hospitaldata[training.samples, ]
hospitaldata.test2 <- hospitaldata[-training.samples, ]
# Build the model
model <- lm(log(totcst) ~ age + as.factor(dzgroup), data = hospitaldata.train2)
print(model)
# Make predictions and compute the R2, RMSE and MAE
predictions <- model %>% predict(hospitaldata.test2)
data.frame( MSE = mean((predictions - log(hospitaldata.test2$totcst))^2),
            R2 = R2(predictions, log(hospitaldata.test2$totcst)),
            RMSE = RMSE(predictions, log(hospitaldata.test2$totcst)),
            MAE = MAE(predictions, log(hospitaldata.test2$totcst)))
```

Leave One out cross-validation

TO DO

This method works as follow:

Leave out one data point and build the model on the rest of the data set Test the model against the data point that is left out at step 1 and record the test error associated with the prediction Repeat the process for all data points Compute the overall prediction error by taking the average of all these test error estimates recorded at step 2.

Example on R

```
# Define training control
train.control <- trainControl(method = "LOOCV")
# Train the model
model <- train(log(totcst) ~ age + as.factor(dzgroup), data = hospitaldata, method =
"lm",
               trControl = train.control)
# Summarize the results
print(model)
```

k-Fold Cross-Validation

TO DO

We divide the set of data in k equals part and we use k-1 parts to train the model and 1 to test. We do do that k times in order to use each part as a test part.

Here are the steps :

1.Split the dataset into k equal partitions (or “folds”)

2.For each fold

One fold is used as the testing set and the union of the other folds as the training set

Calculate testing accuracy for this fold :

$$\hat{f}_i = \frac{1}{K} \sum_{j \in N_0} (y_j)$$

$$MSE = \frac{k}{n} \sum_i I(y_i \neq \hat{y}_i)$$

3.Use the average testing accuracy as the estimate of out-of-sample accuracy :

We would use the cross-validation error :

$$CV_k = \frac{1}{k} \sum_i MSE_i$$

with $I(y_i \neq \hat{y}_i) = 1$ if $y_i \neq \hat{y}_i$, 0 else. So that we calculate the average of wrong predicted values.

Example on R

```
# Define training control
train.control <- trainControl(method = "cv", number = 10)
# Train the model
model <- train(log(totcst) ~ age + as.factor(dzgroup), data = hospitaldata, method =
"lm",
               trControl = train.control)
# Summarize the results
print(model)
```

\clearpage

Comparaison between R and sckit-learn in python

On classification

Logistic Regression

TO DO : comparaison between R and python

\begin{center} \begin{tabular} { | c | c | c |} \hline & R & Scikit-learn \ \hline sensitivity & 0.6153846 & \ \hline specificity & 0.8823529 & \ \hline precision & 0.8 & \ \hline f mesure & 0.6956522 & \ \hline AIC & 105.1 & \ \hline \end{tabular} \end{center}

Decision trees

TO DO : comparaison between R and python

either knn, or decsion trees, or linear discriminant analysis or quadratic discriminant analysis

\begin{center} \begin{tabular} { | c | c | c |} \hline & R & Scikit-learn \ \hline sensitivity & 0.6923077 & \ \hline specificity & 0.5882353 & \ \hline precision & 0.5625 & \ \hline f mesure & 0.6206897 & \ \hline AIC & & \ \hline \end{tabular} \end{center}

On Regression

Simple Linear Regression

\begin{center} \begin{tabular} { | c | c | c |} \hline & R & Scikit-learn \ \hline RMSE & 0.9496248 & \ \hline Rsquared & 0.3660489 & \ \hline MAE & 0.751934 & \ \hline \end{tabular} \end{center}

Multiple Linear Regression

\begin{center} \begin{tabular} { | c | c | c |} \hline & R & Scikit-learn \ \hline RMSE & 0.9475793 & \ \hline Rsquared & 0.3687867 & \ \hline MAE & 0.7486429 & \ \hline \end{tabular} \end{center}

On Cross Validation

Validation set approach

\begin{center} \begin{tabular} { | c | c | c |} \hline & R & Scikit-learn \ \hline RMSE & 0.956954 & \ \hline Rsquared & 0.3497368 & \ \hline MAE & 0.7521536 & \ \hline \end{tabular} \end{center}

Leave One out cross-validation

\begin{center} \begin{tabular} { | c | c | c |} \hline & R & Scikit-learn \ \hline RMSE & 0.944522 & \ \hline Rsquared & 0.3656192 & \ \hline MAE & 0.7552076 & \ \hline \end{tabular} \end{center}

k-fold Cross Validation

\begin{center} \begin{tabular} { | c | c | c |} \hline & R & Scikit-learn \ \hline RMSE & 0.9442358 & \ \hline Rsquared & 0.3678628 & \ \hline MAE & 0.7551859 & \ \hline \end{tabular} \end{center}