Data Science avec R

Fousseynou Bah

2019-02-12

Contents

1 Introduction **5**
1.1 Un autre livre sur la *data science*! Vraiment? . . . 5
1.2 La *data science* . . . 5
1.3 Le *data scientist* . . . 6
1.4 R . . . 7
1.5 RStudio . . . 11

2 Objets dans R **13**
2.1 Objectifs et outils . . . 13
2.2 La notion d'objet dans R . . . 13
2.3 Vecteurs . . . 17
2.4 Matrices . . . 24
2.5 *Data frames* . . . 27
2.6 Listes . . . 30
2.7 Conclusion . . . 31

Chapter 1

Introduction

1.1 Un autre livre sur la *data science*! Vraiment?

En décidant d'écrire un livre sur la *data science*, j'ai longuement débattu dans ma propre tête, je me suis posé plusieurs questions dont une qui revenait constamment: "a-t-on vraiment besoin d'un autre livre sur la *data science*?" "N'en-t-on pas assez?" Avec le succès dont jouit la discipline, ce n'est certainement pas les ressources qui manquent, aussi bien en ligne que dans les librairies. Et surtout, je me demandais bien "qu'avais-je à dire qui n'avait pas été dit"? Et pourtant, quelques raisons m'ont poussé à reconsidérer ma position.

La première est assez égoïte. On n'apprend jamais aussi bien qu'en enseignant. Pour m'assurer que j'avais bien assimilé les connaissances que j'avais acquises dans ce domaine, il n'y avait rien de mieux que de me livrer à un exercice de pédagogue. Expliquer à d'autres ce que j'avais appris. N'est-ce pas là que réside l'ultime test pour un apprenant! C'est partant de cette idée que je me suis mis à faire des diapositives dans le cadre des mes enseignements. Très tôt, j'ai réalisé que les diapositives ne sauraient jouer leur rôle, qui est d'offrir un aperçu synthétique d'une idée développée par un narrateur, et satifaire l'apprenant qui souhaiterai obtenir des explications détaillées. Ce travail revient au narrateur, à défaut de qui l'on se tourne vers un manuel. Donc, il me fallait bien accompagner les diapositives d'un support plus détaillé pour mieux outiller mes étudiants.

La seconde raison est le contexte. Malgré l'abondance et la qualité des ressources disponibles sur la *data science* et malgré l'accès de plus en plus facile - coût faible et gratuité pour beaucoup -, il demeure que l'étudiant africain peut souvent se sentir éloigné du contexte à travers lequel la *data science* est présentée. Or, cell-ci est avant tout une discipline de contexte. Bien que mélangeant informatique, mathématiques, statistiques... et bien d'autres expertises, elle est avant tout un outil, mobilisée pour répondre à des questions. Et ces questions sont très contextuelles. Il ne fait aucun doute que le disponibilité et l'accéssibilité des données sur le monde industrialisé rend leur utilisation commode pour introduire la *data science* à un jeune africain est très commode. Mais la distance entre le contexte présenté et celui qui est vécu par le bénéficiaire pose un problème. Elle empêche l'appropriation de la discipline. De ce fait, je me suis trouvé dans ce constat une raison de m'engager dans ce projet et surtout de me forcer à utiliser des données sur le contexte local. Après tout, l'être humain n'est-il pas plus enclin à vous prêter attention quand vous lui parlez de lui-même?

1.2 La *data science*

Comme toute discipline qui connaît une expansion rapide, il est difficile de définir la *data science*. Elle est vaste et riche, tant de par les disciplines dont elle emprunte des morceaux pour se contituer en entité que de par les branches qu'elle pousse avec sa propre croissance.

Commençons par quelques exemples

Fait de la *data science*:

- l'économiste qui examine le niveau du PIB sur 30 ans et cherche à dégager des scénarii pour des futures évolutions;
- le sociologue qui s'appuie le taux de natalité et le taux de participation des femmes au marché du travail pour comprendre l'évolution de la place de la femme dans la société;
- le météorologue qui cherche à prédire la pluviométrie de la semaine à venir en modélisant les données historiques;
- l'épidémiologue qui cartographie le taux de prévalence du paludisme pour appuyer un programme stratégique;
- etc.

Le caractère transversale de la *data science* apparait ici quand on sait que ces individus sont de disciplines différentes et poursuivent des questions tout aussi distantes les unes des autres. Et pourtant, les données les réunissent tous. Ils ont chacun besoin de trouver dans celles-ci un appui pour améliorer leur propre compréhension du phénomène étudié, tester leurs hypothèses, fonder leurs recommandations ou même... reconforter leurs propres idées ou mieux s'armer pour rejetter celles de leurs adversaires (les données ne sont aussi neutres que celui qui les manipule!)

Selon Wikipédia, la *data science* est un champ interdisciplinaire qui utilise les méthode, processus, algorithmes et systèmes scientifiques pour extraire des données - tant structurées que non structurées - des informations utiles à la compréhension et à la prise de décision. De ce fait, elle s'appuie sur diverses méthodes (mathématiques, statistiques, informatiques, etc.) pour tirer des données une compréhension meilleure de phénomènes d'intérêt.

1.3 Le *data scientist*

Et le *data scientist* dans tout ça? Il est apparait désormais comme la perle rare. Un individu capable de parler aux hommes, aux machines et aux données. Aux:

- hommes, il pose les questions auxquelles il a la charge d'offrir des réponses.
- machines, il parle à travers des langages spécifiques (R, Python, Julia,...), des langages qui ressemblent à bien d'égards à ceux avec lesquels il s'entretient avec les humains car ils sont basés sur des règles précises et sont vivants et évolutifs;
- données, il applique des méthodes d'investigation où l'expérience, l'intuition, le sens artistique interviennent tout autant que la connaissance du domaine d'intervention. Dans les données disponibles, il cherche à séparer les bonnes des mauvaises, les utiles des nuisibles. A celles qu'il sélectionne, il cherche le bon format, la bonne structure. Sur celles qu'il retient, il teste des modèles, sans oublier la place importante de la visualisation à tous les niveaux. Bref, un vrai détective!

Face à la génération massive des données, le besoin de *data scientist* se fait pressant partout. De ce fait l'engouement ne manque pas pour les jeunes désireux de se lancer. Mais le portrait de super-homme généralement fait du *data scientist* (ne cherchez pas plus loin que les lignes d'en dessus!), l'on peut croire qu'il faut être spécial pour embrasser la profession. Du tout! Celà dit, certaines compétences sont utiles.

Alors, qu'est-ce qu'il faut pour être *data scientist*?

- pas nécéssairement un diplôme avancé en mathématiques ou en statistiques... quoiqu'il est utile de maîtriser des concepts de bases (les concepts algébriques comme le vecteur, la matrice,... et les notions statistiques comme la moyenne, l'écart-type, etc.);
- pas forcément un diplôme en informatique ou en programmation... quoiqu'il est utile de connaître les notions de bases (qu'est-ce qu'un objet, un environnement? quels types d'objets peut-on manipuler dans un environnement donnée...?);

- une connaissance avérée dans un domaine spécifique dans lequel l'on peut soulever des questions, mobiliser des outils théoriques auxquels on confronte les résultats de l'analyse conduite sur les données;

- un esprit curieux, quelle que soit l'avenue que l'on emprunte.

Vous pourrez avoir une meilleure idée en surfant sur le net (Google est votre ami!)

1.4 R

1.4.1 Qu'est-ce que c'est que R?

Voici basiquement ce que Wikipédia dit. R est un langage de programmation et un logiciel gratuit et libre. Il est surtout utilisé pour le développement de programmes statistiques et des analyses de données. Il gagne en popularité depuis quelques années avec l'émergence de la *data science* et du fait qu'il est gratuit et ouvert (*open-source*). R est née d'un projet de recherche mené par deux chercheurs, Ross Ihaka et Robert Gentleman à l'université d'Auckland (Nouvelle-Zélande) en 1993. En 1997 est mis en place le *Comprehension R Archive Network (CRAN)* qui centralise les contributions au projet

Depuis le projet connaît une croissance soutenue, grâce à des contributions de la part de milliers de personnes à travers le monde.

1.4.2 Pourquoi R?

Pour un apprenti *data scientist*, le choix du langage et/ou du programme est une décision critique. Considérant le temps qu'il investira en apprentissage et le retour qu'il espéra à travers l'utilisation de ses nouvelles connaissances dans sa profession, il est utile de considerer divers critères dont:

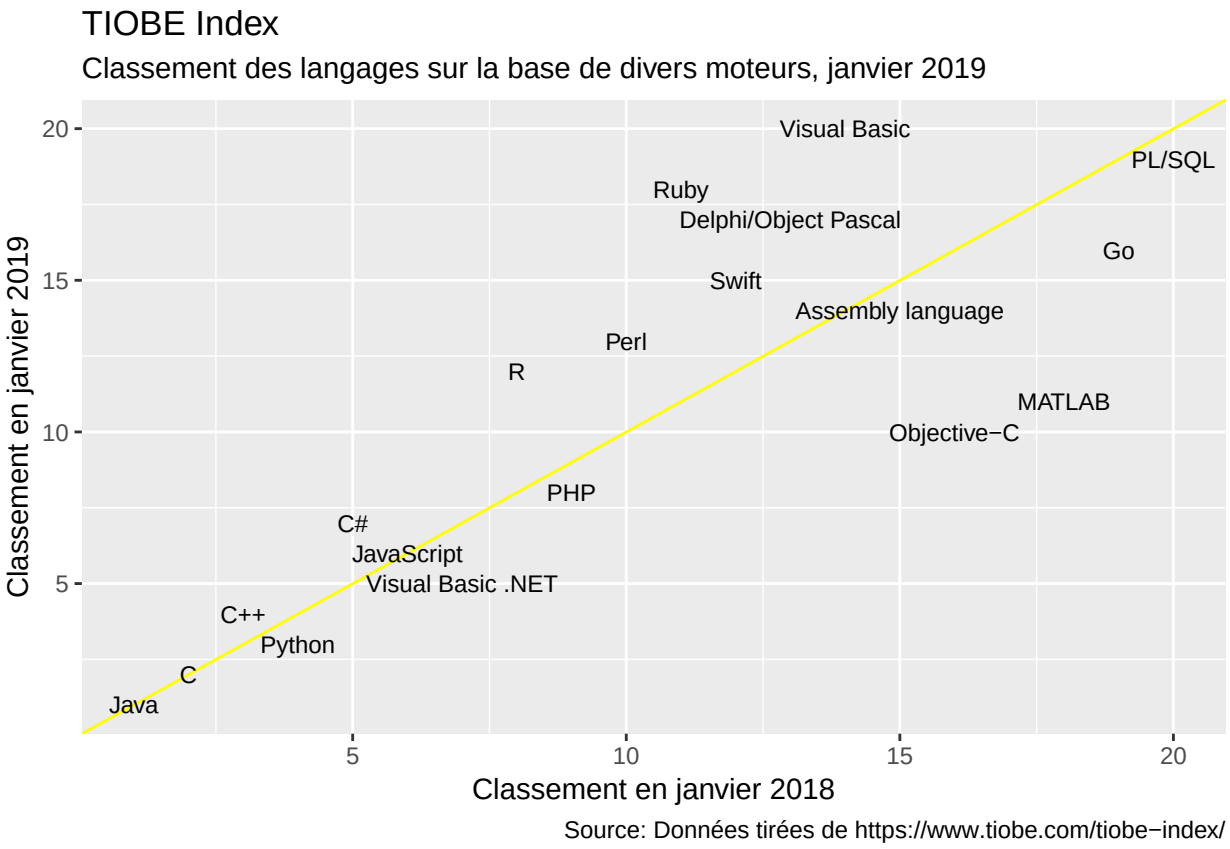
- l'accessibilité de l'outil en termes de coûts: tous les langages de programmation ne sont pas gratuits comme R! Certains coûtent... chers mêmes ;

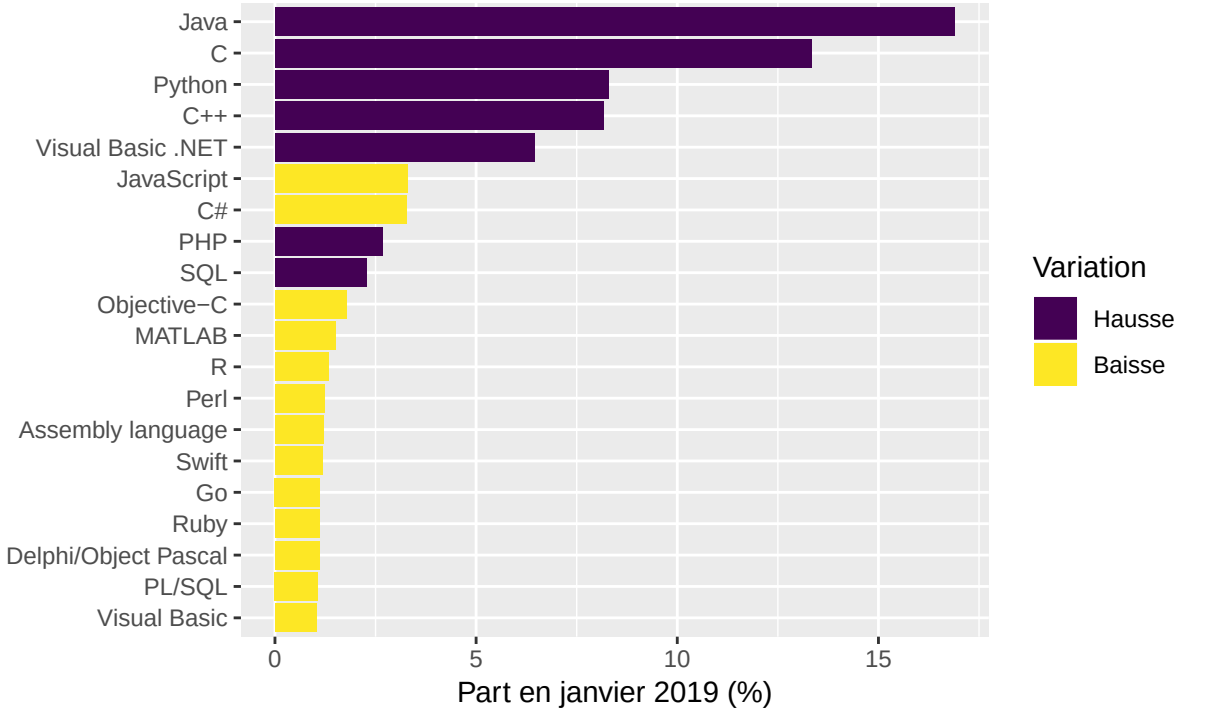
- l'accessibilité du langage en termes de syntaxe: R est très compréhensible (surtout pour quelqu'un qui se retrouve un peu avec la langue anglaise);

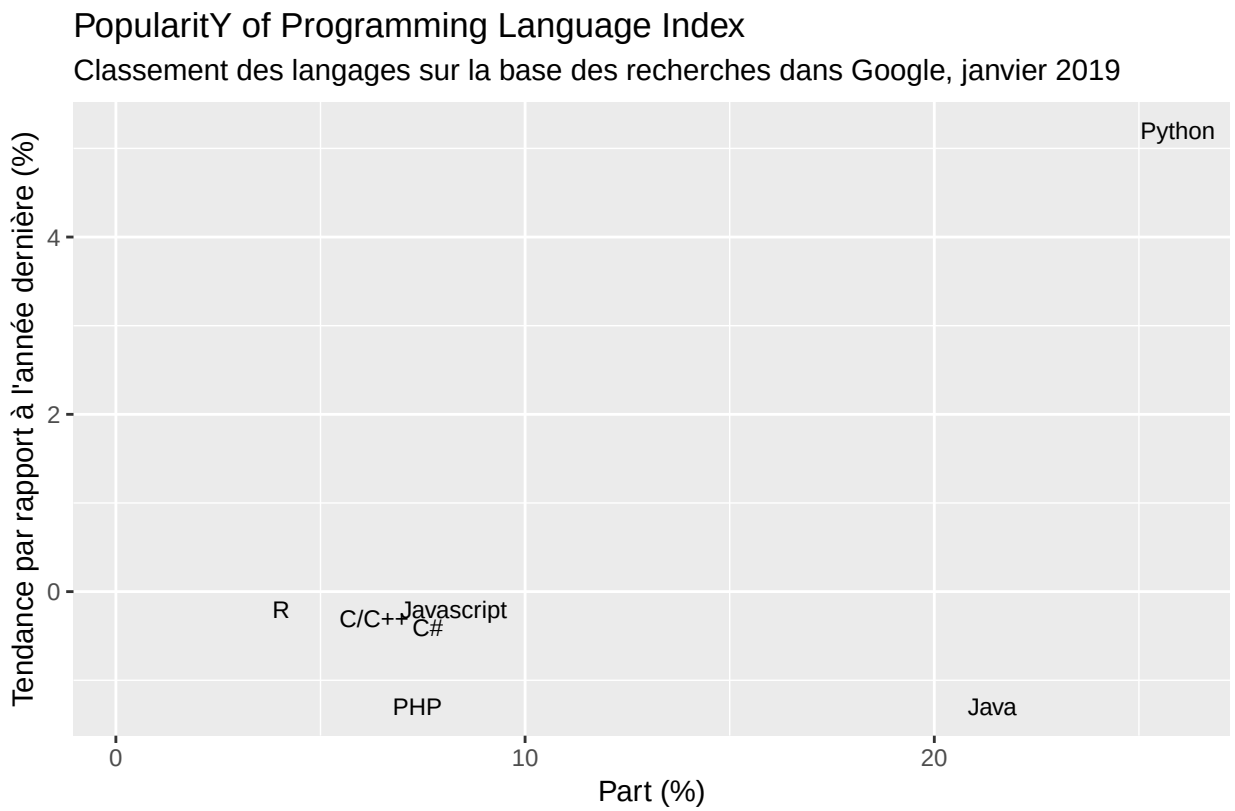
- la popularité du langage parmi les paires: tout le monde s'est mis à l'anglais, même dans les pays où ce n'est pas la langue dominante. N'est-ce pas? De la même façon, il est important pour le *data scientist* d'embrasser un langage qui est aussi utilisé par ceux avec lesquels il sera amené à collaborer. A ce niveau, R est très populaire.

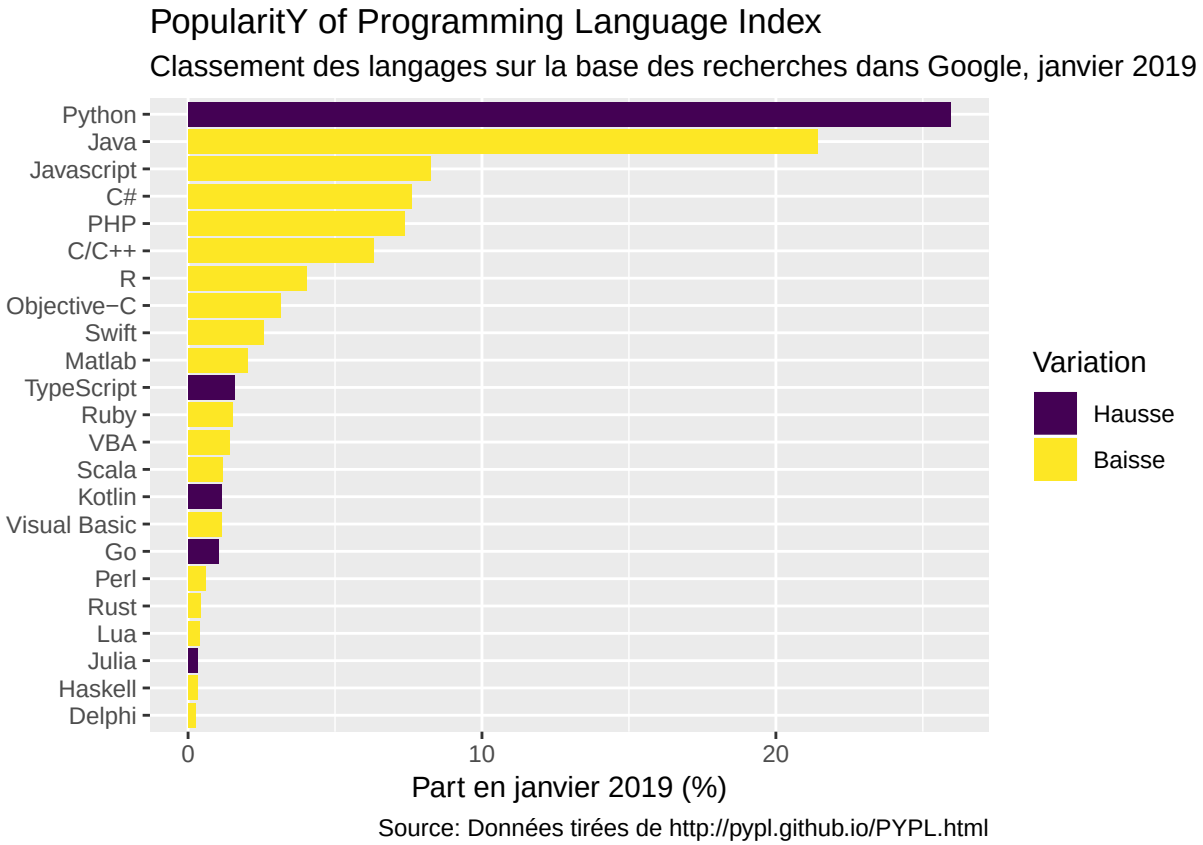
- la dynamique de développement du langage: le langage étant un investissement en soit, il est important de miser sur ceux qui présentent un avenir. Et ceux-ci sont ceux qui mutent avec la technologie et les besoins des utilisateurs. A ce niveau encore, R présente des arguments. Il dispose du réseau *CRAN* alimenté par des milliers de contributeurs, divers aussi bien de par leur position dans le monde que de par leur discipline.

1.4.3 R dans l'écosystème des langages




TIOBE Index
Classement des langages sur la base de divers moteurs de recherche, jan
Java
C
Python
C++
Visual Basic .NET
JavaScript
C#
PHP
SQL
Objective−C
MATLAB
R
Perl
Assembly language
Swift
Go
Ruby
Delphi/Object Pascal
PL/SQL
Visual Basic
0
5
10
15
Part en janvier 2019 (%)
Variation
Hausse
Baisse
Source: Données tirées de https://www.tiobe.com/tiobe−index/


PopularitY of Programming Language Index
Classement des langages sur la base des recherches dans Google, janvier 2019
Tendance par rapport à l'année dernière (%)
4
2
0
Python
R
C/C++
Javascript
C#
PHP
Java
0
10
20
Part (%)
Source: Données tirées de http://pypl.github.io/PYPL.html



Ce qui apparait des différentes figures, c'est que R parvient à se tailler une place parmi les langages les plus populaires au monde. Et celà, malgré le fait que c'est une langage spécialisée. Si sur les dix dernières années, le langage s'est enrichi avec la diversification de ses contributeurs, il reste à la base un langage élaboré par des statisticiens pour des statisticiens. De ce fait, il est excéllent pour l'analyse de données, mais fort peu utile pour certaines tâches... comme le développement d'un site web.

1.5 RStudio

1.5.1 Qu'est-ce que c'est que RStudio

- C'est une IDE (*Integrated Development Environment*) ou Environnement Intégré de Développement
- Il sert d'interface entre R et l'utilisateur, offre à celui diverses commodités d'utilisation

Maintenant, vous avez les outils nécéssaires pour commencer la formidable aventuRe!

Chapter 2

Objets dans R

2.1 Objectifs et outils

Dans ce chapitre, nous allons :

- introduire la notion d'objet dans R;
- présenter un certain nombre d'entre eux;
- et illustrer avec quelques exemples.

Que nous faudra-t-il?

- R (évidemment)
- RStudio (de préférence)

2.2 La notion d'objet dans R

2.2.1 Qu'est-ce qu'un objet?

Dans R, un objet représente un concept, une idée. Il se matérialise par une entité qui possède sa propre identité. Dans celle-ci, l'on compte deux aspects majeurs:

- la structure interne;
- le comportement.

Illustrons pour comprendre. Commençons par créer des objets.

Imaginez que vous voulez créer et conserver des bouts d'information dans R sur les présidents qui se sont succédés à la tête de la République du Mali. Commençons par le premier président, Mobido Keïta. Créeons des objects relatifs à son nom et son prénom.

```
nom <- "Keïta"
prenom <- "Mobido"
```

L'acte d'assignation d'une valeur à un objet se fait par le signe `<-` qui est équivalent à `=`. Chez beaucoup d'utilisateurs, la préférence est donnée à la première. Ceci peut se comprendre par le fait qu'avec `<-`, l'acte d'assignation se différencie plus facilement d'autres utilisations du signe `=` (dont notamment à l'intérieur de fonctions). Désormais, ces informations sont stockées dans notre environnement. Pour vérifier appellons-les! Ceci revient à les saisir dans notre console et à taper "Entrée"!

```
nom
```

```
## [1] "Keïta"
```

```
prenom
```

```
## [1] "Mobido"
```

2.2.2 Oranges et bananes

Enrichissons notre environnement des objets additionnels. Ajoutons l'année d'accession au pouvoir. Appelons cet objet `annee_arrivee_pouvoir`.

```
annee_arrivee_pouvoir <- 1960
```

Comme pour les objets précédent, celui-ci aussi peut être invoqué:

```
annee_arrivee_pouvoir
```

```
## [1] 1960
```

A l'instar de l'orange et de la banane, fort différentes bien que toutes les deux des fruits, ici aussi nos objets diffèrent. Peut-on les additionner?

```
nom + annee_arrivee_pouvoir
```

```
## Error in nom + annee_arrivee_pouvoir: non-numeric argument to binary operator
```

Non, en l'occurence! On a un message d'erreur. R, c'est comme la vraie vie! Les oranges et les bananes ne se mélangent.

2.2.3 Ce qui se ressemblent s'assemblent

Les choses qui diffèrent ne s'assemblent pas Illustration d'une propriété des objets: le comportement. Regardons les choses qui marchent.

```
## [1] 2
```

Maintenant stockons ce résultat dans un objet.

```
objet1 <- 1 + 1
```

Créons-en un autre.

```
objet2 <- 2 + 2
```

Amusons à faire diverses opérations avec ces deux objets

```
objet1 + objet2
```

```
## [1] 6
```

```
objet1 - objet2
```

```
## [1] -2
```

```
objet1 * objet2
```

```
## [1] 8
```

```
objet1 / objet2
```

```
## [1] 0.5
```

Bref, vous voyez l'idée! Les propriétés des objets déterminent les intéractions auxquelles elles se prêtent. Et ce sont justement ces intéractions qui constituent le coeur de l'analyse de données. D'où l'importance de la notion d'objet.

2.2.4 Quelques objets dans R

Dans R, l'on distingue plusieurs types d'objets. Nous en retiendrons ici 5, qui nous serons utiles tout le long de l'ouvrage. Il s'agit des:

- caractères (*strings* en anglais);
- nombres (entiers ou réels);
- dates;
- valeurs logiques qui ne prennent que deux valeurs: `TRUE` (vrai) ou `FALSE` (faux);
- facteurs qui sont un format spécial dans R prévu pour les variables catégorielles.

Revenons à notre exemple *présidentiel*! Nous avons déjà le nom et le prénom...

```
# Caractères
nom <- "Keïta"
prenom <- "Mobido"
```

...ainsi que l'année d'arrivée au pouvoir.

```
# Nombre
annee_arrivee_pouvoir <- 1960
```

Ajoutons la date de naissance,

```
# Date
date_naissance <- as.Date("1915-06-04")
```

une valeur logique indiquant s'il a eu un parcours militaire ou pas,

```
# Valeur logique
parcours_militaire <- FALSE
```

et enfin la région de naissance.

```
# Facteur
region_naissance <- as.factor("Bamako")
```

2.2.5 La notion de classe et de type

Quand on a faire à des objets dont on ignore l'identité, l'on peut s'appuyer la fonction `class`. Celle-ci permeet de connaître la classe de l'objet. La classe est un attribut qui contribue à la formation de l'idée d'un objet. "Avec quoi se mélange-t-il?" "A quelles règles de transformation se soumet-il?" Basiquement, la classe dicte les principes régissant la manipulation de cet objet. Testons la fonction sur les objets que nous venons de créer pour bien confirmer les identités qu'on leur a attribuées.

```
class(nom)
```

```
## [1] "character"
```

```
class(prenom)
```

```
## [1] "character"
```

```
class(annee_arrivee_pouvoir)
```

```
## [1] "numeric"
```

```
class(date_naissance)
```

```
## [1] "Date"
```

```
class(parcours_militaire)
```

```
## [1] "logical"
```

```
class(region_naissance)
```

```
## [1] "factor"
```

Nous voyons que les résultats sont bien conformes aux dénominations que nous leur avons données plus haut.

Dans R, il y a aussi la fonction `typeof` (ou `mode`, mais nous resterons avec la première) qui permet de connaître le mode de stockage d'un objet. Testons!

```
typeof(nom)
```

```
## [1] "character"
```

```
typeof(prenom)
```

```
## [1] "character"
```

```
typeof(annee_arrivee_pouvoir)
```

```
## [1] "double"
```

```
typeof(date_naissance)
```

```
## [1] "double"
```

```
typeof(parcours_militaire)
```

```
## [1] "logical"
```

```
typeof(region_naissance)
```

```
## [1] "integer"
```

Si pour les objets `nom` et `prenom` qui sont des lettres, la classe et le type se confondent, la question est tout autre pour d'autres objets. Regardons `region_naissance`, par exemple. En termes de classe, c'est un facteur. Par contre, en terme de type, R l'a coercé en entier (*integer*).

Les types sont assez génériques car présentant pratiquement les mêmes nomenclatures d'un langage à un autre. Dans R, nous allons plus nous intéressér aux types suivants:

- logique (*logical*);
- entier (*integer*);
- réel (*double*);
- caractère (*character*);
- liste (*list*);
- valeur nulle (*NULL*).

Les objets que nous avons vus là peuvent être pensés comme des briques. Ils entrent à leur tour dans la formation d'autres objets qui varient les uns des autres. Tout comme les constructions peuvent différer entre elles.

2.2.6 Vers d'autres types d'objets

La deuxième catégorie d'objets que nous allons voir ici peuvent être pensés comme des objets composites. Nous en verrons quatre types:

- le vecteur;
- la matrice;
- le *data frame* (cadre de données ou données rectangulaires);
- la liste.

2.3 Vecteurs

2.3.1 Qu'est-ce qu'un vecteur?

De façon très simple, un vecteur est un ensemble d'éléments de même nature. Revenons à notre exemple pour mieux comprendre. Nous avons défini l'objet `nom`, n'est-ce pas? Est-ce un vecteur? A quoi peut-on voir si c'est un vecteur ou pas? La réponse:

```
is.vector(nom)
```

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

Donc nous avons créé des vecteurs depuis longtemps et on voit qu'un objet d'un seul élément peut être un vecteur. Maintenant, compte le nombre d'éléments que compte ce vecteur.

```
length(nom)
```

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

C'est vraiment un singleton qu'on a là... pour le moment!

2.3.2 Créons-en, des vecteurs!

Décidons d'étendre nos observations à tous les présidents de la République du Mali. En voici de quoi nous faire revisiter nos livres d'histoire... ou juste visiter Wikipedia!

```
# Omettons les périodes de transition (la valeur pédagogique est ce qui est recherché ici!)
nom <- c("Keïta", "Traoré", "Konaré", "Touré", "Keïta")
prenom <- c("Modibo", "Moussa", "Alpha Oumar", "Amadou Toumani", "Ibrahim Boubacar")
date_naissance <- as.Date(c("1915-06-04", "1936-09-25", "1946-02-02", "1948-11-04", "1945-01-29"))
region_naissance <- as.factor(c("Bamako", "Kayes", "Kayes", "Mopti", "Koutiala"))
annee_arrivee_pouvoir <- c(1960, 1968, 1992, 2002, 2013)
parcours_militaire <- c(FALSE, TRUE, FALSE, TRUE, FALSE)
```

Maintenant, expérimentons! Commençons avec `nom` que nous avons écrasé avec de nouvelles valeurs.

```
is.vector(nom)
```

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

```
length(nom)
```

```
## [1] 5
```

```
class(nom)
```

```
## [1] "character"
```

```
typeof(nom)
```

```
## [1] "character"
```

"nom" est un **vecteur**, un ensemble de **5** éléments en **charactères**. Amusez-vous à expérimenter avec les autres vecteurs.

2.3.3 Vrai pour un, vrai pour plusieurs

Vous vous rappelez que plus haut, nous voyions que les opérations n'étaient pas possible entre de différentes natures. Et bien, cette règle, valable à l'échelle des objets élémentaires, l'est aussi aux échelles supérieures.

Prenons nos données et cherchons à déterminer l'âge des présidents à leur arrivée au pouvoir. On a les éléments nécéssaires pour ce faire, la date de naissance et l'année d'arrivée au pouvoir. Toutefois, ces deux vecteurs ne sont pas de même nature.

```
age_arrivee_pouvoir <- annee_arrivee_pouvoir - date_naissance
```

```
## Error in `-.Date`(annee_arrivee_pouvoir, date_naissance): can only subtract from "Date" objects
```

On a un message d'erreur. Apparemment l'opération n'est pas possible. Il faudrait procéder à une transformation: déduire de la date de naissance l'année pour conduire l'opération avec celle-ci.

```
annee_naissance <- as.numeric(format(date_naissance,'%Y'))
```

Testons si le nouveau vecteur est de même nature de celui de `annee_arrivee_pouvoir`.

```
class(annee_naissance)
```

```
## [1] "numeric"
```

Maintenant, nous pouvons procéder à l'opération

```
age_arrivee_pouvoir <- annee_arrivee_pouvoir - annee_naissance
age_arrivee_pouvoir
```

```
## [1] 45 32 46 54 68
```

On le confirme: les oranges et les bananes ne se mélangent pas. Toutefois, R nous fait souvent des cocktails de fruits en coerçant certains éléments. Imaginons que l'on veuille rassembler le prénom et le nom dans un seul vecteur. Collons avec une fonction de base dans R, `paste`, (ne vous en faites pas, vous ferez progressivement connaissance avec les fonctions!)

```
prenom_nom <- paste(prenom, nom)
prenom_nom
```

```
## [1] "Modibo Keïta"           "Moussa Traoré"
## [3] "Alpha Oumar Konaré"     "Amadou Toumani Touré"
## [5] "Ibrahim Boubacar Keïta"
```

On peut être enclin à dire que ceci est passé sans souci parce que `nom` et `prenom` sont tous les deux des vecteurs en caractères. Maintenant, et si l'on ajoutait l'année d'arrivée au pouvoir?

```
prenom_nom_age <- paste(prenom, nom, ",", age_arrivee_pouvoir)
prenom_nom_age
```

```
## [1] "Modibo Keïta , 45"           "Moussa Traoré , 32"
## [3] "Alpha Oumar Konaré , 46"     "Amadou Toumani Touré , 54"
## [5] "Ibrahim Boubacar Keïta , 68"
```

C'est passé comme une lettre à la poste (pour la génération email, voici ce qu'est la poste). Car R a une hiérarchie entre les objets. Avant de déclarer forfait avec un message d'erreur, il tente tant bien que mal

d'exécuter l'opération. Sur la base de cette hiérarchie, il coerce certains éléments à se conformer à d'autres, partant du plus flexible au moins flexible: `logique` < `entier` < `réel` < `caractère`. Pour comprendre ça, créons un vecteur de valeurs logiques.

```
vecteur_logique <- c(TRUE, FALSE)
```

Confirmons sa classe.

```
class(vecteur_logique)
```

```
## [1] "logical"
```

Ajoutons un troisième élément qui sera un entier. Disons 1.

```
vecteur_entier <- c(vecteur_logique, 1)
```

Qu'obtenons-nous?

```
vecteur_entier
```

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

Des entiers! R a coercé `TRUE` en 1 et `FALSE` en 0.

```
class(vecteur_entier)
```

```
## [1] "numeric"
```

Ajoutons un quatrième élément, cette fois-ci une réel: 2.5 (dans R, comme en anglais, les décimales viennent après un `.`, pas une `,`, qui sert plutôt de séparateur de milliers).

```
vecteur_reel <- c(vecteur_entier, 2.5)
vecteur_reel
```

```
## [1] 1.0 0.0 1.0 2.5
```

```
class(vecteur_reel)
```

```
## [1] "numeric"
```

La mutation se voit au fait que R a affecté aux trois premiers éléments des décimales, bien qu'initialement c'étaient des entiers. Maintenant, ajoutons un cinquième élément: un prénom.

```
vecteur_caractere <- c(vecteur_reel, "Mariam")
vecteur_caractere
```

```
## [1] "1"      "0"      "1"      "2.5"    "Mariam"
```

```
class(vecteur_caractere)
```

```
## [1] "character"
```

Là aussi, la coercion se voit.

2.3.4 Nommer les éléments d'un vecteur

Jusque là, ce sont des objets à part intégrale que nous avons nommés. On les a assignés des noms pour les garder dans notre environnement de travail. Maintenant, nous allons donner un nom aux éléments de vecteur. Dressons l'analogie suivante. Notre environnement dans R est comme une rue. Dans celle-ci, nous avons des concessions dont les portes sont toutes numérotées: ce sont les noms des objets. A l'intérieur des concessions, nous avons des individus: ce sont les éléments à l'intérieur de nos objets. Tout comme ces individus portent des prénoms, nous pouvons donner des appélations aux éléments contenus dans nos objets.

Considerons que nous voulons associer à chaque date de naissance le nom du président en question.

```
names(date_naissance) <- prenom_nom
```

Voyons ce que ça donne

```
date_naissance
```

```
##           Modibo Keïta          Moussa Traoré     Alpha Oumar Konaré
##           "1915-06-04"           "1936-09-25"           "1946-02-02"
##   Amadou Toumani Touré Ibrahim Boubacar Keïta
##           "1948-11-04"           "1945-01-29"
```

C'est beau non! Il est intéressant de noter que quand on conduit des opérations sur des vecteurs aux éléments nommés, le résultat peut hériter de ces propriétés. Reprenons l'opération de déduction de l'âge à l'arrivée au pouvoir. Rappelons les deux vecteurs.

```
annee_naissance
```

```
## [1] 1915 1936 1946 1948 1945
```

```
annee_arrivee_pouvoir
```

```
## [1] 1960 1968 1992 2002 2013
```

Nommons juste un des deux vecteurs.

```
names(annee_naissance) <- prenom_nom
annee_naissance
```

```
##           Modibo Keïta          Moussa Traoré     Alpha Oumar Konaré
##                   1915                   1936                   1946
##   Amadou Toumani Touré Ibrahim Boubacar Keïta
##                   1948                   1945
```

Procédons à l'opération.

```
age_arrivee_pouvoir <- annee_arrivee_pouvoir - annee_naissance
age_arrivee_pouvoir
```

```
##           Modibo Keïta          Moussa Traoré     Alpha Oumar Konaré
##                     45                     32                     46
##   Amadou Toumani Touré Ibrahim Boubacar Keïta
##                     54                     68
```

Le vecteur `age_arrivee_pouvoir` a hérité des noms d'éléments.

Cette règle n'est pas toutefois immuable. Quand les éléments sont coercés à prendre une autre classe que leur classe de départ, ils peuvent perdre leur nom, qui n'est qu'un de leurs attributs (qui sont subordonnés à leur classe). Reprenons la déduction de l'année de naissance à partir de la date de naissance.

```
annee_naissance <- format(date_naissance,'%Y')
annee_naissance
```

```
##           Modibo Keïta          Moussa Traoré     Alpha Oumar Konaré
##                 "1915"                 "1936"                 "1946"
##   Amadou Toumani Touré Ibrahim Boubacar Keïta
##                 "1948"                 "1945"
```

```
class(annee_naissance)
```

```
## [1] "character"
```

Ici, l'année n'a pas été coercé. Elle a été extraite par la fonction sous format de caractères. En voulant conformer le vecteur à la classe de nombre (on descend dans la hiérarchie), on coerce les éléments.

```
annee_naissance <- as.numeric(format(date_naissance,'%Y'))
annee_naissance
```

```
## [1] 1915 1936 1946 1948 1945
```

```
class(annee_naissance)
```

```
## [1] "numeric"
```

Avec la coercion, les noms se perdent. Il est donc utile de se rappeler que les noms d'éléments ne sont pas immunes à la coercion. Toutefois, quand les opérations se passent entre des éléments de même nature, les noms sont bien saufs!

2.3.5 Opérations sur vecteurs

2.3.5.1 Sélection explicite

Il arrive souvent qu'on ne soit intéréssée que par un élément précis d'un vecteur. Peut-être l'on souhaite connaître seulement l'âge du premier président lors de son accès au pouvoir. C'est le premier élément du vecteur `age_arrivee_pouvoir`.

```
age_arrivee_pouvoir[1]
```

```
## Modibo Keïta
##           45
```

Peut-être nous voulons l'information pour le 1er et le 3ème présidents. Ce sont les 1er et 3ème éléments du vecteur.

```
age_arrivee_pouvoir[c(1, 3)]
```

```
##       Modibo Keïta Alpha Oumar Konaré
##                 45                 46
```

Peut-être que nous voulons l'information du 1er au 3ème président.

```
age_arrivee_pouvoir[c(1:3)]
```

```
##       Modibo Keïta      Moussa Traoré Alpha Oumar Konaré
##                 45                 32                 46
```

On peut aussi souhaiter exclure certains éléments. Imaginons que l'on veuille seulement regarder les informations sans les deux derniers éléments du vecteur.

```
age_arrivee_pouvoir[-c(4, 5)]
```

```
##       Modibo Keïta      Moussa Traoré Alpha Oumar Konaré
##                 45                 32                 46
```

Le signe `[]` agit comme une porte d'entrée à l'intérieur du vecteur tandis que les chiffres indiqués sont des index qui indiquent les éléments intérêt. L'opération peut consister en une sélection ou une exclusion selon que l'opérateur `c` est précédé du signe `-` (exclusion) ou pas (sélection).

2.3.5.2 Sélection à partir de logiques

La sélection à l'intérieur d'un vecteur peut aussi se faire à partir de valeurs logiques. L'on peut poser des critères auxquels certains éléments répondraient. Et sur la base de leur confirmité au(x) critère(s) posé(s), l'on pourra effectuer la sélection (ou l'exclusion). Cette fonctionnalité est très utile le data scientist d'utiliser les questions qu'il se pose pour avoir un aperçu des données qui sont à sa disposition.

Explorons la question suivante: quels sont les présidents arrivés au pouvoir avant l'âge de 50 ans?

```
president_avant_50ans <- age_arrivee_pouvoir < 50
president_avant_50ans
```

```
##           Modibo Keïta          Moussa Traoré     Alpha Oumar Konaré
##                   TRUE                   TRUE                   TRUE
##   Amadou Toumani Touré Ibrahim Boubacar Keïta
##                  FALSE                  FALSE
```

On transforme maintenant ce vecteur de valeurs logiques en outil de sélection. On peut soir regarder le nom de ses présidents:

```
prenom_nom[president_avant_50ans]
```

```
## [1] "Modibo Keïta"       "Moussa Traoré"      "Alpha Oumar Konaré"
```

Le résultat nous donne le nom des présidents pour lesquels le vecteur de valeurs logiques affiche `TRUE`. On peut utiliser le même critère sur d'autres vecteurs. Voyons le vecteur d'âge d'arrivée au pouvoir: quel âge avec les présidents qui sont arrivés au pouvoir avant l'âge de 50 ans?

```
age_arrivee_pouvoir[age_arrivee_pouvoir < 50]
```

```
##       Modibo Keïta      Moussa Traoré Alpha Oumar Konaré
##                 45                 32                 46
```

Pendant qu'on y est, dans quelle région sont-ils nés?

```
names(region_naissance) <- prenom_nom # nommons d'abord les éléments
region_naissance[age_arrivee_pouvoir < 50]
```

```
##       Modibo Keïta      Moussa Traoré Alpha Oumar Konaré
##             Bamako              Kayes              Kayes
## Levels: Bamako Kayes Koutiala Mopti
```

Vous comprenez la logique.

2.3.5.3 Statistiques sommaires

Une fois le vecteur constitué, il peut en lui-même faire l'objet d'opérations diverses. Posons diverses questions avec le vecteur `age_arrivee_pouvoir`. Quelle est la moyenne d'âge d'arrivée au pouvoir sur la base des éléments disponibles?

```
mean(age_arrivee_pouvoir)
```

```
## [1] 49
```

```
# une alternative donnat le même résultat.
sum(age_arrivee_pouvoir)/length(age_arrivee_pouvoir)
```

```
## [1] 49
```

Quel est l'âge d'arrivée au pouvoir le plus bas ?

```
min(age_arrivee_pouvoir)
```

```
## [1] 32
```

Quel est l'âge d'arrivée au pouvoir le plus élevé ?

```
## [1] 68
```

2.3.5.4 Ajustement et recyclage

Maintenant, revenons-en un peu aux opérations entre deux vecteurs. Imaginez maintenant, que l'on veuille connaître l'âge auquel les présidents ont quitté le pouvoir. Rappellons d'abord le vecteur `age_arrivee_pouvoir` que nous avions déjà généré.

```
age_arrivee_pouvoir
```

```
##           Modibo Keïta          Moussa Traoré     Alpha Oumar Konaré
##                     45                     32                     46
##   Amadou Toumani Touré Ibrahim Boubacar Keïta
##                     54                     68
```

Construisons ensuite un vecteur avec le nombre d'années passées au pouvoir.

```
annee_en_pouvoir <- c(8, 23, 10, 10)
```

Maintenant calculons l'année de départ du pouvoir en ajoutant à l'âge d'arrivée au pouvoir le nombre d'années qui y ont été passé.

```
age_depart_pouvoir <- age_arrivee_pouvoir + annee_en_pouvoir
```

```
## Warning in age_arrivee_pouvoir + annee_en_pouvoir: longer object length is
## not a multiple of shorter object length
```

```
age_depart_pouvoir
```

```
##           Modibo Keïta          Moussa Traoré     Alpha Oumar Konaré
##                     53                     55                     56
##   Amadou Toumani Touré Ibrahim Boubacar Keïta
##                     64                     76
```

Parvenez-vous à décéler l'erreur? Nous avons additionné une vecteur de 5 éléments, `age_arrivee_pouvoir`, avec un vecteur de 4 éléments `annee_en_pouvoir`. R a recyclé le premier élément du vecteur court (4) pour poursuivre l'opération d'addition entre les deux vecteur et l'a ajouté au 5ème élément du vecteur long. D'où la valeur de `76`.

```
68 + 8
```

```
## [1] 76
```

R avertit, mais conduit l'opération. De ce fait, même si les opérations entre les vecteurs de même nature s'exécute sans problème majeur, il reste utile de vérifier leur longueur. Pour éviter le recyclage, il faudrait ne pas laisser de videe dans le vecteur court, s'assurer que les vecteurs impliqués dans l'opératio sont de la même taille. Sur nos 5 présidents, nous n'avons pas ajouté le nombre d'années passées au pouvoir (car le mandat est encore en cours pendant la rédaction du présent document). Une solution serait de remplir la position dans le vecteur avec la valeur `NA`, indiquant un valeur manquante. Ajoutons-le à è "

```
annee_en_pouvoir <- c(annee_en_pouvoir, NA)
annee_en_pouvoir
```

```
## [1]  8 23 10 10 NA
```

Reprenons l'opération.

```
age_depart_pouvoir <- age_arrivee_pouvoir + annee_en_pouvoir
age_depart_pouvoir
```

```
##           Modibo Keïta          Moussa Traoré     Alpha Oumar Konaré
##                     53                     55                     56
##   Amadou Toumani Touré Ibrahim Boubacar Keïta
##                     64                     NA
```

Ne sachant pas comment faire l'opération pour la dernière entrée du vecteur car l'un des composante est `NA`, R reconduit cette valeur. Ainsi le recyclage est évité.

2.4 Matrices

2.4.1 La matrice, un ensemble de vecteurs

De façon basique, une matrice n'est autre qu'une collection de vecteurs. De ce fait, elle hérite d'une propriété fondamentale du vecteur: ne peuvent former une matrice que des éléments de même nature.

Retournons à notre exemple. Associons les noms et prénoms en une matrice car tous deux sont en charactères.

Solution 1: coller horizontalement les deux vecteurs

```
prenom_nom_hmatrix <- rbind(prenom, nom)
prenom_nom_hmatrix
```

```
##        [,1]     [,2]     [,3]           [,4]              [,5]
## prenom "Modibo" "Moussa" "Alpha Oumar" "Amadou Toumani" "Ibrahim Boubacar"
## nom    "Keïta"  "Traoré" "Konaré"      "Touré"           "Keïta"
```

Solution 2: coller verticalement les deux vecteurs

```
prenom_nom_vmatrix <- cbind(prenom, nom)
prenom_nom_vmatrix
```

```
##      prenom              nom
## [1,] "Modibo"            "Keïta"
## [2,] "Moussa"            "Traoré"
## [3,] "Alpha Oumar"       "Konaré"
## [4,] "Amadou Toumani"    "Touré"
## [5,] "Ibrahim Boubacar" "Keïta"
```

On voit que la matrice hérite des noms donnés aux différents vecteurs.

Bien que l'on puisse créer une matrice en combinant différents vecteurs, horizontalement avec `rbind` ou verticalement avec `cbind`, il existe aussi une fonction qui permet de créer directement une matrice: `matrix`. Il est toutefois utile de connaitre l'ordre de positionnement des éléments. Reprenons la création avec `matrix`, horizontalement...

```
prenom_nom_hmatrix <- matrix(c("Modibo", "Moussa", "Alpha Oumar", "Amadou Toumani", "Ibrahim Boubacar",
                               "Keïta",  "Traoré", "Konaré", "Touré", "Keïta"),
                             byrow = TRUE,
                             nrow = 2,
                             dimnames = list(c("prenom", "nom"), NULL)
                             )
prenom_nom_hmatrix
```

```
##        [,1]     [,2]     [,3]          [,4]              [,5]
## prenom "Modibo" "Moussa" "Alpha Oumar" "Amadou Toumani" "Ibrahim Boubacar"
## nom    "Keïta"  "Traoré" "Konaré"      "Touré"           "Keïta"
```

...et verticalement

```
prenom_nom_vmatrix <- matrix(c("Modibo", "Keïta",
                               "Moussa", "Traoré",
                               "Alpha Oumar", "Konaré",
                               "Amadou Toumani", "Touré",
                               "Ibrahim Boubacar", "Keïta"),
                             byrow = TRUE,
                             ncol = 2,
                             dimnames = list(NULL, c("prenom", "nom"))
                             )
prenom_nom_vmatrix
```

```
##      prenom              nom
## [1,] "Modibo"            "Keïta"
## [2,] "Moussa"            "Traoré"
## [3,] "Alpha Oumar"       "Konaré"
## [4,] "Amadou Toumani"    "Touré"
## [5,] "Ibrahim Boubacar" "Keïta"
```

Dans la fonction `matrix`, les arguments `nrow`, `ncol`, `byrow` et `bycol` servent à celà. Fonction, arguments... ne vous en faites pas! On y viendra.

2.4.2 La matrice, un objet bidimensionnel

La matrice n'est pas seulement un ensemble de vecteurs. Elle se distingue aussi de par sa bidimensionnalité. Pendant que le vecteur est soit une ligne de plusieurs éléments (1 x n) soit une colonne de plusieurs éléments éléments (n x 1), la matrice, elle, est faite de plusieurs lignes (n rows) et de plusieurs colonnes (n columns). Ici n étant bien sûr supérieur à 1. Nous avions noté que pour connaître le nombre d'éléments dans un vecteur on utilisait la fonction `length`.

```
length(prenom_nom)
```

```
## [1] 5
```

La même chose marche-t-elle pour le vecteur?

```
length(prenom_nom_vmatrix)
```

```
## [1] 10
```

En l'occurence, non! `length` ne rend pas compte de la bidimensionnalité. Il y a une autre fonction pour ça: `dim`.

```
dim(prenom_nom_vmatrix)
```

```
## [1] 5 2
```

La bidimensionnalité se lit aussi dans le nom des rangées.

```
dimnames(prenom_nom_vmatrix)
```

```
## [[1]]
## NULL
##
## [[2]]
## [1] "prenom" "nom"
```

Comme nous avons vu plus haut, l'on peut nommer les rangées depuis la création de la matrice. Reprenons la création de `prenom_nom_vmatrix` en nommant toutes les rangées, aussi bien horizontales que verticales.

```
prenom_nom_vmatrix <- matrix(c("Modibo", "Keïta",
                               "Moussa", "Traoré",
                               "Alpha Oumar", "Konaré",
                               "Amadou Toumani", "Touré",
                               "Ibrahim Boubacar", "Keïta"),
                             byrow = TRUE,
                             ncol = 2,
                             dimnames = list(c("1er", "2ème", "3ème", "4ème", "5ème"),
                                             c("prenom", "nom"))
                             )
```

Imprimons la matrice.

```
print(prenom_nom_vmatrix)
```

```
##      prenom             nom
## 1er  "Modibo"           "Keïta"
## 2ème "Moussa"           "Traoré"
## 3ème "Alpha Oumar"      "Konaré"
## 4ème "Amadou Toumani"   "Touré"
## 5ème "Ibrahim Boubacar" "Keïta"
```

Examinons la matrice à travers différentes fonctions que nous avons vues en haut:

- la dimension, c'est-à-dire le nombre de lignes et le nombre de colonnes;

```
dim(prenom_nom_vmatrix)
```

```
## [1] 5 2
```

- les noms des lignes et des colonnes;

```
dimnames(prenom_nom_vmatrix)
```

```
## [[1]]
## [1] "1er"  "2ème" "3ème" "4ème" "5ème"
##
## [[2]]
## [1] "prenom" "nom"
```

- le nombre de lignes uniquement;

```
nrow(prenom_nom_vmatrix)
```

```
## [1] 5
```

- le nom des lignes uniquement;

```
rownames(prenom_nom_vmatrix)
```

```
## [1] "1er"  "2ème" "3ème" "4ème" "5ème"
```

- le nombre de colonnes uniquement;

```
ncol(prenom_nom_vmatrix)
```

```
## [1] 2
```

- le nom des colonnes uniquements.

```
colnames(prenom_nom_vmatrix)
```

```
## [1] "prenom" "nom"
```

2.4.3 Opérations sur matrices: une autre matrice

A l'instar des vecteurs, les matrices aussi sep

```r
# Reprenons les années et les âges pour former une nouvelle matrice
# Considérons les années d'évènements majeurs: naissance, arrivée au pouvoir et départ du pouvoir
annee_evenement_matrix <- matrix(c(1915, 1936, 1946, 1948, 1945,
                                   1960, 1968, 1992, 2002, 2013,
                                   1968, 1991, 2002, 2012, NA),
                                 byrow = TRUE,
                                 ncol = 5,
                                 dimnames = list(c("Naissance", "Arrivée", "Départ"),
                                                 c("M. Keïta", "M. Traoré", "A.O. Konaré", "A.T. Touré", "I.B. Keïta")
                                                 )
                                 )
annee_evenement_matrix
```

```
##           M. Keïta M. Traoré A.O. Konaré A.T. Touré I.B. Keïta
## Naissance     1915      1936        1946       1948       1945
## Arrivée       1960      1968        1992       2002       2013
## Départ        1968      1991        2002       2012         NA
```

```r
# Nous venons d'introduire une nouvelle notion: "NA"
# NA: 'Non Available' en anglais
# Cette valeur rempli la cellule dont les valeurs nous sont inconnues.
```

2.4.4 Opérations sur matrices: questions logiques

```r
# Maintenant que nous avons notre matrice, amusons-nous avec.
# Prenons un grand-père née vers 1949 (oui, il fait partie des "né vers").
# Marié à l'âge de 22 ans, père 1 an plus tard, grand-père 18 ans plus tard et décédé à l'âge de 61 ans.
# Quels sont les évènements qui se sont passés de son vivant?
ce_que_grandpa_a_vu <- annee_evenement_matrix > 1949 & annee_evenement_matrix < (1949 + 61)
ce_que_grandpa_a_vu
```

```
##           M. Keïta M. Traoré A.O. Konaré A.T. Touré I.B. Keïta
## Naissance    FALSE     FALSE       FALSE      FALSE      FALSE
## Arrivée       TRUE      TRUE        TRUE       TRUE      FALSE
## Départ        TRUE      TRUE        TRUE      FALSE         NA
```

```r
# Apparemment, il en a vu beaucoup, mais tous les présidents le dépassent en âge
# On vient d'introduier ici la notion d'addition dans les critères.
```

2.4.5 Opérations sur matrices: extraction par position

```r
# Comme pour les vecteurs, des éléments peuvent être explicitement sélectionnés à l'intérieur des matrices.
annee_evenement_matrix
```

```
##           M. Keïta M. Traoré A.O. Konaré A.T. Touré I.B. Keïta
## Naissance     1915      1936        1946       1948       1945
## Arrivée       1960      1968        1992       2002       2013
## Départ        1968      1991        2002       2012         NA
```

```r
# Supposons que l'on veuille connaître l'élément qui est dans la cellule de la 3ème ligne et le 2ème colonne.
annee_evenement_matrix[3, 2]
```

```
## [1] 1991
```

```r
# Et la 3ème ligne toute entière.
annee_evenement_matrix[3, ]
```

```
##    M. Keïta   M. Traoré A.O. Konaré  A.T. Touré  I.B. Keïta
##        1968        1991        2002        2012          NA
```

```r
# Et la 2ème colonne toute entière.
annee_evenement_matrix[ , 2]
```

```
## Naissance   Arrivée    Départ
##      1936      1968      1991
```

2.4.6 Opérations sur matrices: extraction par nom

```r
# Rappel
annee_evenement_matrix
```

```
##           M. Keïta M. Traoré A.O. Konaré A.T. Touré I.B. Keïta
## Naissance     1915      1936        1946       1948       1945
## Arrivée       1960      1968        1992       2002       2013
## Départ        1968      1991        2002       2012         NA
```

```r
# Si les rangées sont nommés, alors il est aussi possible de passer par ces noms pour sélectionner
# Vous vous rappelez "rownames()" ou "colnames()"
# Si la réponse est non, je saurai que vous n'avez pas tout suivi!
# Utilisons "rownames()" pour voir la ligne sur les années de naissance.
annee_evenement_matrix[rownames(annee_evenement_matrix) == "Naissance", ]
```

```
##    M. Keïta   M. Traoré A.O. Konaré  A.T. Touré  I.B. Keïta
##        1915        1936        1946        1948        1945
```

```r
# Et "colnames()" pour voir la colonne du premier président
annee_evenement_matrix[, colnames(annee_evenement_matrix) == "M. Keïta"]
```

```
## Naissance   Arrivée    Départ
##      1915      1960      1968
```

2.4.7 Opérations sur matrices: consolidation

```
# Il arrive qu'on souhaite consolider une matrice, y ajouter de nouvelles informations.
# Considérons ici les âges à l'arrivée au et au départ du pouvoir.
# Recalculons les à partir des matrices.
age_arrivee_pouvoir <- annee_evenement_matrix[rownames(annee_evenement_matrix) == "Arrivée", ] -
  annee_evenement_matrix[rownames(annee_evenement_matrix) == "Naissance", ]
age_depart_pouvoir <- annee_evenement_matrix[rownames(annee_evenement_matrix) == "Départ", ] -
  annee_evenement_matrix[rownames(annee_evenement_matrix) == "Naissance", ]
# Ajoutons maintenant c'est deux nouveaux vecteurs à notre matrice
annee_evenement_matrix_cons <- rbind(annee_evenement_matrix,
                                     "Âge d'arrivée au pouvoir" =  age_arrivee_pouvoir,
                                     "Âge de départ du pouvoir" = age_depart_pouvoir)
# Voyons la matrice
annee_evenement_matrix_cons
```

```
##                         M. Keïta M. Traoré A.O. Konaré A.T. Touré
## Naissance                   1915      1936        1946       1948
## Arrivée                     1960      1968        1992       2002
## Départ                      1968      1991        2002       2012
## Âge d'arrivée au pouvoir      45        32          46         54
## Âge de départ du pouvoir      53        55          56         64
##                         I.B. Keïta
## Naissance                     1945
## Arrivée                       2013
## Départ                          NA
## Âge d'arrivée au pouvoir        68
## Âge de départ du pouvoir        NA
```

```
# Nous sommes passés par la fonction "rbind()". Sachez qu'il y a plusieurs solutions!
# Remarquez-vous "NA" dans une nouvelle cellule? Pouvez-vous expliquer pourquoi?
```

2.4.8 Opérations sur matrices: calculs

```
# Comme pour les vecteurs, des calculs sont possibles sur les matrices.
# Pour ce faire, limitons-nous à deux informations de la matrice: les âges.
# Ajoutons aussi la durée de la période passée au pouvoir.
age_pouvoir_matrix <- rbind(annee_evenement_matrix_cons[c(4, 5),],
                            "Durée au pouvoir" = annee_evenement_matrix_cons[5, ] -
                              annee_evenement_matrix_cons[4, ])
age_pouvoir_matrix[, -c(1)] # -c(1): juste pour une commodité d'impression.
```

```
##                         M. Traoré A.O. Konaré A.T. Touré I.B. Keïta
## Âge d'arrivée au pouvoir        32          46         54         68
## Âge de départ du pouvoir        55          56         64         NA
## Durée au pouvoir                23          10         10         NA
```

```
# Calculons la moyenne pour chacune des lignes (âges moyens, durées moyennes)
rowMeans(age_pouvoir_matrix)
```

```
## Âge d'arrivée au pouvoir Âge de départ du pouvoir         Durée au pouvoir
##                       49                       NA                       NA
```

```
# On voit des "NA". R ne traite pas les valeurs inconnues de lui-même. On lui instruit de les ignorer.
rowMeans(age_pouvoir_matrix, na.rm = TRUE)
```

```
## Âge d'arrivée au pouvoir Âge de départ du pouvoir         Durée au pouvoir
##                    49.00                    57.00                    12.75
```

```
# Comme pour beaucoup de fonctions dans R, tout ce qu'il y avec "row" existe avec "col"
# Testez les fonctions suivantes: colSums(), rowSums(), colMeans() et rowMeans().
```

2.5 *Data frames*

2.5.1 Le *data frame*, au-délà de la matrice (1)

```
# Jusque là, nous avons travaillé avec des éléments de même nature.
# Et pourtant le data scientist ne peut pleinement mener ses
# ses investigations avec une telle contrainte.
# Il a besoin d'explorer en même temps des informations de diverses natures.
# D'où le data frame. Qu'est-ce que c'est au juste?
# Un format d'organisation de données en forme rectangulaire.
# Toutefois, contrairement à la matrice, elle respecte la nature des données qu'elle contient.
# Explorons l'idée. Rassemblons verticalement les différents vecteurs que nous avons créées
presidents_df <- cbind(nom,
                       prenom,
                       date_naissance,
                       region_naissance,
                       parcours_militaire,
                       annee_arrivee_pouvoir,
                       annee_en_pouvoir)
# Qu'est-ce que ça donne?
presidents_df
```

```
##      nom      prenom             date_naissance region_naissance
## [1,] "Keïta"  "Modibo"           "-19335"       "1"
## [2,] "Traoré" "Moussa"           "-12151"       "2"
## [3,] "Konaré" "Alpha Oumar"      "-8734"        "2"
## [4,] "Touré"  "Amadou Toumani"   "-7728"        "4"
## [5,] "Keïta"  "Ibrahim Boubacar" "-9103"        "3"
##      parcours_militaire annee_arrivee_pouvoir annee_en_pouvoir
## [1,] "FALSE"            "1960"                "8"
## [2,] "TRUE"             "1968"                "23"
## [3,] "FALSE"            "1992"                "10"
## [4,] "TRUE"             "2002"                "10"
## [5,] "FALSE"            "2013"                NA
```

2.5.2 Le *data frame*, au-délà de la matrice (2)

```
# Nous avons vu que certaines informations ont été dénaturées.
# Certaines données ont été coercées à se transformer en autre chose
# Regardons la classe de l'objet "presidents_df"
class(presidents_df)
```

```
## [1] "matrix"
```

```
typeof(presidents_df)
```

```
## [1] "character"
```

```
# Les vecteurs ont été rassemblés en matrice (class)
# Les éléments ont toutefois été coercés en charactères (typeof)
# C'est en celà que le data frame révèle sa place.
```

2.5.3 Le *data frame*, au-délà de la matrice (3)

```
# Reprenons l'opération
presidents_df <- data.frame(nom,
                            prenom,
                            date_naissance,
                            region_naissance,
                            parcours_militaire,
                            annee_arrivee_pouvoir,
                            annee_en_pouvoir,
                            stringsAsFactors = FALSE)
# Regardons à nouveau
presidents_df
```

```
##      nom           prenom date_naissance region_naissance
## 1  Keïta           Modibo     1915-06-04           Bamako
## 2 Traoré           Moussa     1936-09-25            Kayes
## 3 Konaré      Alpha Oumar     1946-02-02            Kayes
## 4  Touré   Amadou Toumani     1948-11-04            Mopti
## 5  Keïta Ibrahim Boubacar     1945-01-29         Koutiala
##   parcours_militaire annee_arrivee_pouvoir annee_en_pouvoir
## 1              FALSE                  1960               8
## 2               TRUE                  1968              23
## 3              FALSE                  1992              10
## 4               TRUE                  2002              10
## 5              FALSE                  2013              NA
```

```
# Qu'en est-il de la classe et du type
class(presidents_df)
```

```
## [1] "data.frame"
```

2.5.4 Le *data frame*, au-délà de la matrice (4)

```
# Maintenant que nous savons à quoi ressemble un data frame, essayons de le définir.
# Un data frame est une forme d'organisation de données en format rectangulaire où
# les lignes sont des observations et les colonnes des attributs de ceux-ci.
# Ici par exemple, nous organisons diverses informations sur les individus qui
# ont assumé le poste de Président de la République du Mali.
# Chaque ligne sera dédiée à un président et rassemblera tous les informations sur lui (attributs).
# Chaque colonne sera dédiée à un seul attribut et couvrira tous les présidents (observations).
# Introduisons ici la fonction "str" qui permet de visualiser la structure d'un objet dans R
# Elle permettra de rendre compte de toute la richesse du concept de data frame.
str(presidents_df)
```

```
## 'data.frame':    5 obs. of  7 variables:
##  $ nom                : chr  "Keïta" "Traoré" "Konaré" "Touré" ...
##  $ prenom             : chr  "Modibo" "Moussa" "Alpha Oumar" "Amadou Toumani" ...
##  $ date_naissance     : Date, format: "1915-06-04" "1936-09-25" ...
##  $ region_naissance   : Factor w/ 4 levels "Bamako","Kayes",..: 1 2 2 4 3
##  $ parcours_militaire : logi  FALSE TRUE FALSE TRUE FALSE
##  $ annee_arrivee_pouvoir: num  1960 1968 1992 2002 2013
##  $ annee_en_pouvoir   : num  8 23 10 10 NA
```

```
# On a une synthèse: nombre d'observations et nombre de variables - comme avec la fonction dim().
# On voit aussi que pour chaque variable, on a :
# - le nom
# - la classe
# - quelques observations
```

2.5.5 Opérations sur *data frame*: sélection de cellules

```
# En matière de sélection, la data frame hérite beaucoup de la matrice.
# Les principes demeurent
# Si l'on veut la ligne 2 de la colonne 4, on fait:
presidents_df[2, 4]
```

```
## [1] Kayes
## Levels: Bamako Kayes Koutiala Mopti
```

```
# Si l'on veut la ligne 5 (un président, une observation)
presidents_df[2, ]
```

```
##      nom prenom date_naissance region_naissance parcours_militaire
## 2 Traoré Moussa     1936-09-25            Kayes               TRUE
##   annee_arrivee_pouvoir annee_en_pouvoir
## 2                  1968              23
```

```
# Ou encore la colonne 4 (une variable, un attribut)
presidents_df[, 4]
```

```
## [1] Bamako   Kayes    Kayes    Mopti    Koutiala
## Levels: Bamako Kayes Koutiala Mopti
```

2.5.6 Opérations sur *data frame*: sélection de variables

```
# Comme pour la matrice, avec le data frame, on peut utiliser le nom des colonnes (variables)
# pour accéder aux éléments. Regardons juste la variable "date_naissance".
presidents_df[, "date_naissance"]
```

```
## [1] "1915-06-04" "1936-09-25" "1946-02-02" "1948-11-04" "1945-01-29"
```

```
# Le data frame offre en plus une alternative: les variables y sont accessibles avec le signe "$".
presidents_df$date_naissance
```

```
## [1] "1915-06-04" "1936-09-25" "1946-02-02" "1948-11-04" "1945-01-29"
```

```
# Cependant, la première solution se prête à la sélection de plusieurs variables.
presidents_df[, c("date_naissance", "region_naissance")]
```

```
##   date_naissance region_naissance
## 1     1915-06-04           Bamako
## 2     1936-09-25            Kayes
## 3     1946-02-02            Kayes
## 4     1948-11-04            Mopti
## 5     1945-01-29         Koutiala
```

2.5.7 Opérations sur *data frame*: création de variables

```
# Comme avec les matrices, souvent, l'analyse peut souhaiter ajouter une nouvelle variable à son data frame.
# Procédons comme avec les matrices à la génération de deux nouvelles variables:
# l'âge d'arrivée au pouvoir et l'âge de départ du pouvoir.
# Pour commencer, générons l'année de naissance
presidents_df$annee_naissance <- as.numeric(format(presidents_df$date_naissance,'%Y'))
# Ensuite on génère l'âge d'arrivée au pouvoir
presidents_df$age_arrivee_pouvoir <- presidents_df$annee_arrivee_pouvoir - presidents_df$annee_naissance
# Ensuite l'âge de départ du pouvoir
presidents_df$age_depart_pouvoir <- presidents_df$age_arrivee_pouvoir + presidents_df$annee_en_pouvoir
# Regardons notre nouveau data frame
str(presidents_df)
```

```
## 'data.frame':    5 obs. of  10 variables:
##  $ nom                  : chr  "Keïta" "Traoré" "Konaré" "Touré" ...
##  $ prenom               : chr  "Modibo" "Moussa" "Alpha Oumar" "Amadou Toumani" ...
##  $ date_naissance       : Date, format: "1915-06-04" "1936-09-25" ...
##  $ region_naissance     : Factor w/ 4 levels "Bamako","Kayes",..: 1 2 2 4 3
##  $ parcours_militaire   : logi  FALSE TRUE FALSE TRUE FALSE
##  $ annee_arrivee_pouvoir: num  1960 1968 1992 2002 2013
##  $ annee_en_pouvoir     : num  8 23 10 10 NA
##  $ annee_naissance      : num  1915 1936 1946 1948 1945
##  $ age_arrivee_pouvoir  : num  45 32 46 54 68
##  $ age_depart_pouvoir   : num  53 55 56 64 NA
```

```
# A travers cette création, on voit comment on peut mener des opérations entre des colonnes d'un data frame.
```

2.5.8 Opérations sur *data frame*: suppression de variables

```
# Dans notre exemple, nous avons créé l'année de naissance comme étape transitoire vers une
# autre variable. Sachant que nous avons la même information dans la date de naissance,
# l'on peut éviter la redondance, donc la supprimer.
# Comment s'y prend-on dans R?
presidents_df$annee_naissance <- NULL
# Vérifions si cette colonne est partie.
str(presidents_df)
```

```
## 'data.frame':    5 obs. of  9 variables:
##  $ nom                  : chr  "Keïta" "Traoré" "Konaré" "Touré" ...
##  $ prenom               : chr  "Modibo" "Moussa" "Alpha Oumar" "Amadou Toumani" ...
##  $ date_naissance       : Date, format: "1915-06-04" "1936-09-25" ...
##  $ region_naissance     : Factor w/ 4 levels "Bamako","Kayes",..: 1 2 2 4 3
##  $ parcours_militaire   : logi  FALSE TRUE FALSE TRUE FALSE
##  $ annee_arrivee_pouvoir: num  1960 1968 1992 2002 2013
##  $ annee_en_pouvoir     : num  8 23 10 10 NA
##  $ age_arrivee_pouvoir  : num  45 32 46 54 68
##  $ age_depart_pouvoir   : num  53 55 56 64 NA
```

```
# Mission accomplie!
```

2.5.9 Opérations sur *data frame*: sélection d'observations

```
# Nous avons vu que comme la matrice, les éléments du data frame sont accessibles grâce aux numéros de lignes.
# Ici, nous allons voir qu'il est aussi possible de passer par des critères spécifiques aux variables
# pour sélectionner des observations.
# Cherchons seulement la date de naissance des présidents nés dans la région de "Kayes".
presidents_df[presidents_df$region_naissance == "Kayes", c("nom", "prenom")]
```

```
##      nom      prenom
## 2 Traoré      Moussa
## 3 Konaré Alpha Oumar
```

```
# Il est possible d'aboutir au même résultat avec une fonction intégrée à R: "subset"
# Expérimentons
subset(x = presidents_df, subset = region_naissance == "Kayes", select = c(nom, prenom))
```

```
##      nom      prenom
## 2 Traoré      Moussa
## 3 Konaré Alpha Oumar
```

```
# Vous voyez?
# Avec R, tous les chemins mènent...à Roundé.
# (Rome est trop loin pour moi! Même s'il comment par R).
```

2.5.10 Opérations sur *data frame*: ordonner les observations

```
# On peut souvent souhaiter  ordonner son data frame selon une variable donnée.
# Rearrangeons nos données selon l'année de naissance des présidents
ordre_age <- order(presidents_df$date_naissance)
ordre_age
```

```
## [1] 1 2 5 3 4
```

```
# A l'aide de ce classement, regardons les nom, prénom et date de naissance
presidents_df[ordre_age, c("nom", "prenom", "date_naissance")]
```

```
##      nom           prenom date_naissance
## 1  Keita           Modibo     1915-06-04
## 2 Traoré           Moussa     1936-09-25
## 5  Keïta Ibrahim Boubacar     1945-01-29
## 3 Konaré      Alpha Oumar     1946-02-02
## 4  Touré   Amadou Toumani     1948-11-04
```

2.5.11 *Data frame*: le meilleur reste à venir

Le *data frame* est la pièce maîtresse de l'analyse dans R, comme dans beaucoup d'autres langages. D'ailleurs, d'autres langages ont développé des concepts similaires. En prenant Python par exemple, on trouve la notion de *DataFrame*, une adaption du concept de *data frame* tel que défini dans R. Pour dire combien l'idée englobée dans le *data frame* est puissante. D'où son rôle capital dans le reste de ce cours.

C'est avec le *data frame* que nous:

- procéderons à des manipulations de données: du nettoyage à la transformation ;
- explorerons des données par la visualisation ;
- introduirons l'application de modèles à des données.

2.6 Listes

2.6.1 Les listes, que faire de l'ordre et de la structure?

```
# La liste (list en anglais et dans R) apporte elle aussi sa particularité.
# Elle permet de créer un espace pour les données non structurées dans R
# Créons de nouveaux éléments
# Commençons par les pays voisins du Mali: un vecteur en caractères
voisins_vec_char <- c( "Algérie", "Burkina-Faso", "Côte d'Ivoire", "Guinée", "Mauritanie", "Niger", "Sénégal")
# Ajoutons des données sur le population (à partir de 1976, 1987, 1998 et 2009)
# ça nous fait une matrice en nombres (entiers).
population_matrix_int <- matrix(data = c(3123733, 3269185, 6392918,
                                         3760711, 3935638, 7696349,
                                         4856023, 4954889, 9810912,
                                         7204990, 7323672, 14528662),
                                byrow = TRUE,
                                nrow = 4,
                                dimnames = list(c(1976, 1987, 1998, 2009),
                                                c("Hommes", "Femmes", "Total")))
# Ajoutons un dernier élément: lesquels de nos présidents sont encore vivants?
# Mettons ça sous forme boléén.
presidents_en_vie_vec_logi <- c(FALSE, TRUE, TRUE, TRUE, TRUE)

# Nous avons là un beau monde. Rassemblons tout ça dans une liste!
mali_list <- list(presidents = presidents_df,
                  voisins = voisins_vec_char,
                  population = population_matrix_int,
                  presidents_en_vie = presidents_en_vie_vec_logi)
# De par leurs différences en nature, forme et taille, rien ne prédispose ses objets à être contenus
# dans le même objet! Et pourtant ça tient dans notre liste.
# Explorons-là!
```

2.6.2 Listes, un contenant de contenants (1)

```
# Commençons par la structure de la liste. Que voit-on?
str(mali_list)
```

```
## List of 4
##  $ presidents       :'data.frame':   5 obs. of  9 variables:
##   ..$ nom                 : chr [1:5] "Keïta" "Traoré" "Konaré" "Touré" ...
##   ..$ prenom              : chr [1:5] "Modibo" "Moussa" "Alpha Oumar" "Amadou Toumani" ...
##   ..$ date_naissance      : Date[1:5], format: "1915-06-04" ...
##   ..$ region_naissance    : Factor w/ 4 levels "Bamako","Kayes",..: 1 2 2 4 3
##   ..$ parcours_militaire  : logi [1:5] FALSE TRUE FALSE TRUE FALSE
##   ..$ annee_arrivee_pouvoir: num [1:5] 1960 1968 1992 2002 2013
##   ..$ annee_en_pouvoir    : num [1:5] 8 23 10 10 NA
##   ..$ age_arrivee_pouvoir : num [1:5] 45 32 46 54 68
##   ..$ age_depart_pouvoir  : num [1:5] 53 55 56 64 NA
##  $ voisins          : chr [1:7] "Algérie" "Burkina-Faso" "Côte d'Ivoire" "Guinée" ...
##  $ population       : num [1:4, 1:3] 3123733 3760711 4856023 7204990 3269185 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:4] "1976" "1987" "1998" "2009"
##   .. ..$ : chr [1:3] "Hommes" "Femmes" "Total"
##  $ presidents_en_vie: logi [1:5] FALSE TRUE TRUE TRUE TRUE
```

```
# Qu'en est-il des noms
names(mali_list)
```

```
## [1] "presidents"        "voisins"           "population"
## [4] "presidents_en_vie"
```

```
# Les noms assignés aux objets sont bien reconduits
# Voyons voir si à l'instar des matrices et des data frames, ces noms peuvent être utilisés
# pour accéder aux éléments qui y sont stockés.
knitr::asis_output("\\normalsize")
```

2.6.3 Listes, un contenant de contenants (2)

```
# Prenons le vecteur sur les pays voisins.
mali_list[["voisins"]]
```

```
## [1] "Algérie"       "Burkina-Faso"  "Côte d'Ivoire" "Guinée"
## [5] "Mauritanie"    "Niger"         "Sénégal"
```

```
# Le même résultat doit être possible par l'ordre de l'objet dans la liste, le 2ème.
mali_list[[2]]
```

```
## [1] "Algérie"       "Burkina-Faso"  "Côte d'Ivoire" "Guinée"
## [5] "Mauritanie"    "Niger"         "Sénégal"
```

```
# Peut-on utiliser le signe "$" comme avec les data frame
mali_list$voisins
```

```
## [1] "Algérie"       "Burkina-Faso"  "Côte d'Ivoire" "Guinée"
## [5] "Mauritanie"    "Niger"         "Sénégal"
```

```
# Donc, on a l'embarra du choix.
```

2.6.4 Listes, un contenant de contenants (3)

```
# Maitenant qu'on peut accéder aux objets à l'intérieur d'un liste,
# qu'en est-il des éléments stockés à l'intérieur de cet objet lui-même.
# Cherchons le 2ème élément du vecteur des pays voisins
mali_list[["voisins"]][2]
```

```
## [1] "Burkina-Faso"
```

```
# Qu'en est-il de "mali_list[[2]]" et de "mali_list$voisins[2]"?
# Testez avec eux pour voir.

# Un autre exemple: la 3ème colonne de la matrice sur la population
mali_list[["population"]][, 3]
```

```
##     1976     1987     1998     2009
##  6392918  7696349  9810912 14528662
```

```
# La ligne suivante aussi marche
mali_list[["population"]][, "Total"]
```

```
##     1976     1987     1998     2009
##  6392918  7696349  9810912 14528662
```

2.7 Conclusion

2.7.1 Et ce n'est que le début

Avec une introduction à ces objets, on pose les bases de l'analyse de données dans R. Bien que, pour des raisons pédagogiques, chaque objet ait été présenté par rapport aux limites du précédent, ils demeurent tous utiles, chacun avec ses avantages (compétitifs). Il revient au data scientist de connaître quand, où et comment faire intervenir un au lieu des autres. Contribuer à vous outiller pour faire ces choix - parmi tant d'autres - est l'un des objectifs de ce cours.