12 TRAVAUX D'ASTERIX_Marion DANYACH

Contents

Pour ces 12 travaux j'ai choisi d'apprécier le travail de mes camarades selon plusieurs critères.
-Lisibilité
-Caractère didactique.
-Sujet facilement compréhensible pour un novice en R
-Originalité du sujet.
-Illustrations graphiques.

J'ai conservé les mêmes critères tout au long de l'évaluation, ce qui n'a pas été chose facile.
En effet les travaux et les thèmes abordés sont tous tellement
différents, que j'ai été tentée de changer à plusieurs reprises.

J'ai pour ce travail mis des liens utiles en fin de rubrique, plutôt qu'une bibliographie à la fin du document.

NETWORK 3D.

Auteurs: Claire MAZZUCCATO, Adrien JUPITER.
lien Github

J'ai choisi ce package car j'ai commencé à travailler sur GEPHI dans le cadre d'un POC et je suis curieuse de voir ce que R propose.
Ce package semble être finalement la version 3D du package network.
Installons le package avec CRANmirror, car il ne se charge pas sur ma version de R studio.
Une installation via Cran Mirror ne semble pas marcher non plus.

```
#chooseCRANmirror()
```

il faut ensuite taper dans la console le numero qui correspond a la localisation la plus proche : pour moi 30.

```
#install.packages(c("network3D1"))
#getOption("repos")
#av <- available.packages(filters=list())
#av[av[, "Package"] == network3D, ]
```

Je n'ai pu installer le network 3D a cause de ma version de R.

Network 3D utilise en effet un algorithme force de spatialisation similaire à celui propose par Gephi dans les paramètres de spatialisation (Force Atlas).

Network 3D propose une jolie complémentarité avec 3 des packages suivants : - igraph pour la préparation des données et le "plotting".
- ggraph for les représentations visuelles
- networkD3 pour l'interactivité

A noter : Network 3D permet de visualiser des widgets interactifs (voir code ci-dessous bien que n'ayant pas pu installer le package). Le widget se trouve dans le lien fourni plus bas dans la rubrique liens utiles.

```
#install.packages("igraph")
#install.packages("dyplr")
#library(dplyr)
#library(igraph)
# create a dataset:
#data <- data.frame
#(from=c("A", "A", "B", "D", "C", "D", "E", "B", #"C", "D", "K", "A", "M"),
#to=c("B", "E", "F", "A", "C", "A", "B", #"Z", "A", "C", "A", "B", "K")
#)

# Plot
#p <- simpleNetwork(data, height="100px", width="100px",
#        Source = 1,                 # column number of source
#        Target = 2,                 # column number of target
#        linkDistance = 10,          # distance between node.
#Increase this value to have more space between nodes
# charge = -900,
# numeric value indicating
#either the strength of the node repulsion.
#(negative value) or #attraction (positive value)

#       fontSize = 14,               # size of the node names
```

```
#        fontFamily = "serif",       # font og node names
#linkColour = "#666",
# colour of edges, MUST be a common #colour for the whole graph
#        nodeColour = "#69b3a2",     # colour of nodes(common)
#       opacity = 0.9,              # opacity of nodes. #0=transparent. 1=no transparency
#      zoom = T                    # Can you zoom on the figure?
#       )

#p

# save the widget
# library(htmlwidgets)
# saveWidget(p, file=paste0( getwd(), #"/HtmlWidget/networkInteractive2.html"))
```

Le package offre plusieurs options de customisation (couleurs etc),
que l'on peut trouver avec la commande ci-dessous.
help(simpleNetwork).

On pourra également choisir de modifier les polices, ce qui est généralement un peu plus compliqué avec R en installant le package extrafont.

```
#install.packages('extrafont')
#library('extrafont')

#Import des fontes du système - cela peut être long
#font_import()
#fonts() # Liste des polices disponibles
#loadfonts(device = "pdf") # Indiquer device = "pdf"
```

A mon sens il s'agit d'un bon travail, très didactique.
Il se démarque également par son originalité.
Il dispose d'une bonne introduction sur les réseaux et la théorie des graphes.
Les étapes sont bien détaillées.
J'ai aimé le fait qu'ils aient construit eux même un réseau imaginaire sans repartir de jeux de données.
On regrettera uniquement l'absence de représentation graphique de réseaux même 2D.

Evaluation: 15/20

liens intéressants:
lien utile1.
lien utile3.

GGPLOT2

Auteurs: Maxime et Siva Chane
lien Github.
Il est intéressant de noter que ggplot2 est en fait une extension du package Tidyverse (package vu avec lubridate un peu plus bas.)

Avant de se lancer, on pourra lire si on le souhaite ce lien intéressant sur la **Data visualisation** i.e. l'art de représenter graphiquement ses données.

A la manière des autres packages, je le charge avec Library puis je choisis un jeu de données déjà inclus dans R ici mtcars.

```
library(ggplot2)
data(mtcars)

mtcars$cyl <- as.factor(mtcars$cyl)
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Important à noter : il vous faut convertir votre jeu de données en facteurs dans de nombreux cas avant de pouvoir l'exploiter.

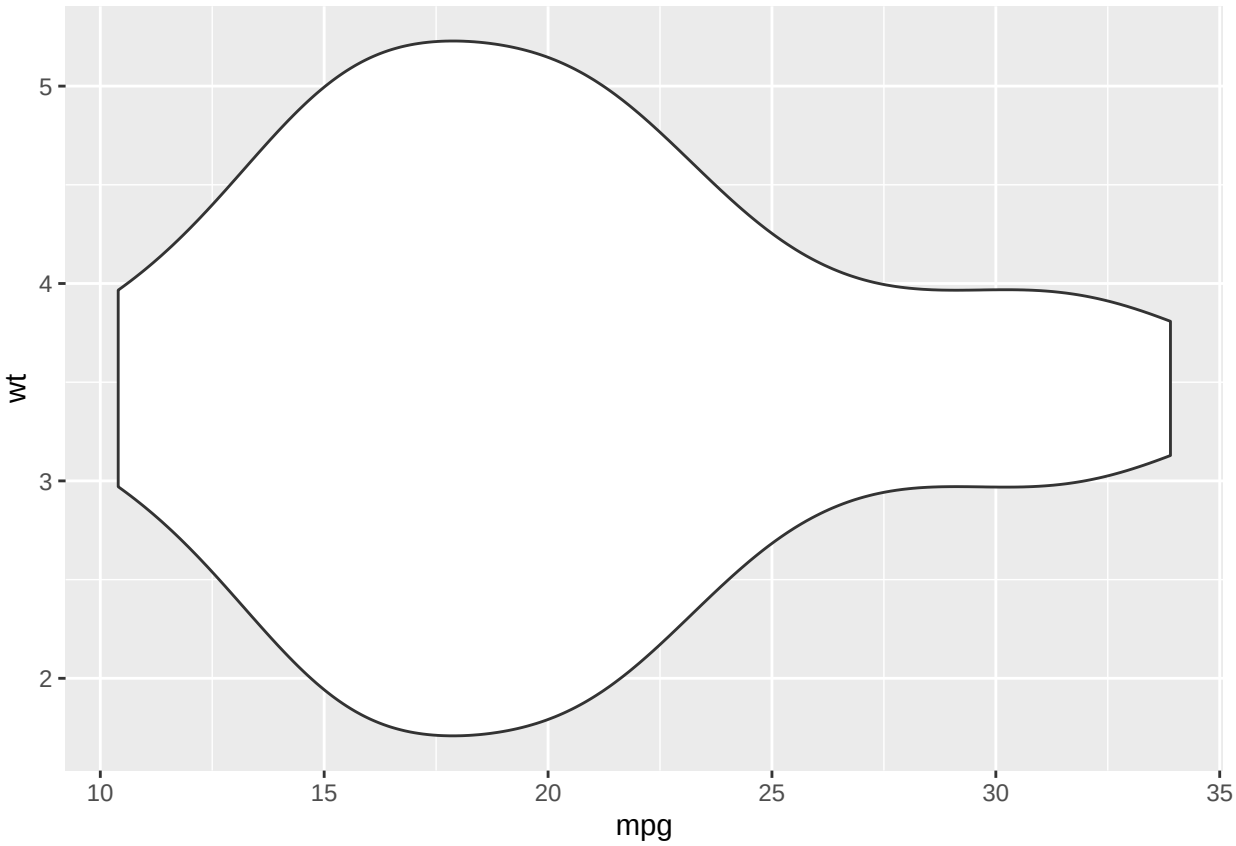
Ci-dessous un lien sur un travail sur les **facteurs**

On notera que ggplot2 fonctionne de manière très simple avec
-en premier le jeu de données que l'on souhaite représenter r - l'indication du data test à représenter
- aes qui conditionne l'esthétique du graphe (quelles valeurs pour x, y, mais aussi la couleur : on peut par exemple indiquer qu'on souhaite que les points soient colorés par variable ex par pays)
- puis des ajouts qui viendront déterminer le style de votre graphique tel que geom_point() ou geom_plot.

J'ai bien aimé découvrir l'ajout de la fonctionnalité smooth(),
en plus de ce que nous avons pu voir dans le travail. Smooth est utilisée pour ajouter des droites de régression à un nuage de points.
Il en existe d'autres ayant un usage similaire, que nous ne détaillerons pas ici. geom_smooth(), stat_smooth(),geom_abline()

J'ai également pu découvrir violin, similaire d'une certaine façon à ce qu'on peut obtenir avec les boites à moustache. Ils permettent de montrer la courbe de densité de probabilité des différentes valeurs.
J'ai fait ici uniquement un test trivial pour voir à quoi cela ressemblait. Je me suis également amusée à le tourner grâce à coord_flip().

```
p <- ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_violin()
p + coord_flip()
```



J'ai trouvé ici plusieurs extensions de ggplot plutôt intéressantes dont GGally qui a l'avantage de combiner représentation graphique de chaque variable ici wt et une rapide analyse exploratoire via le coefficient de corrélation.

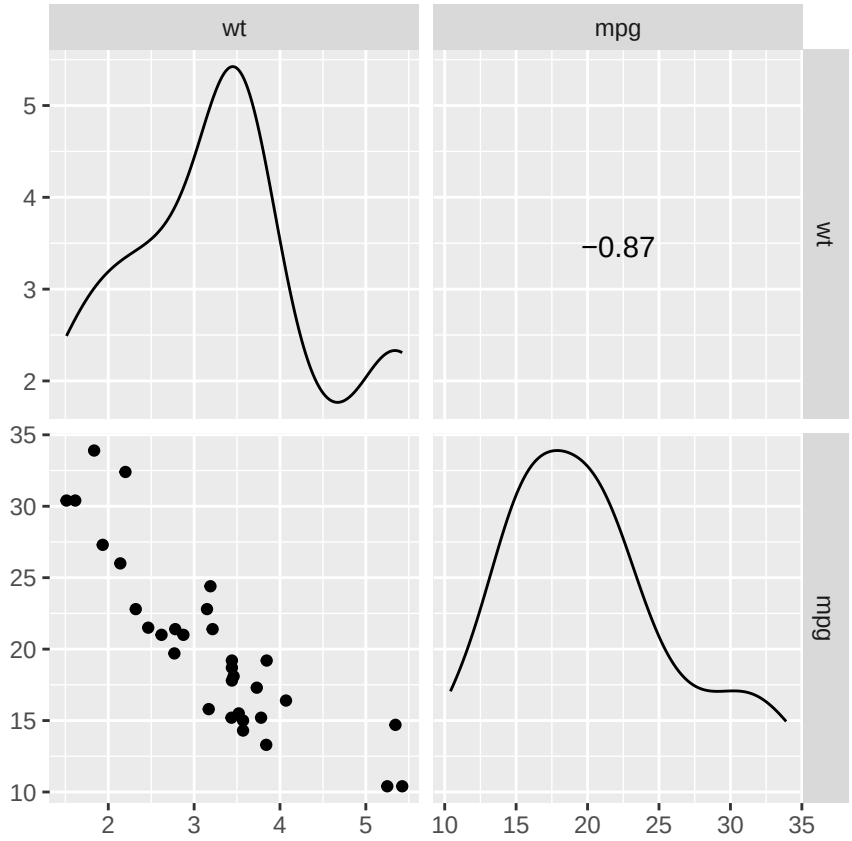
```
data("mtcars")
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(dplyr)
```

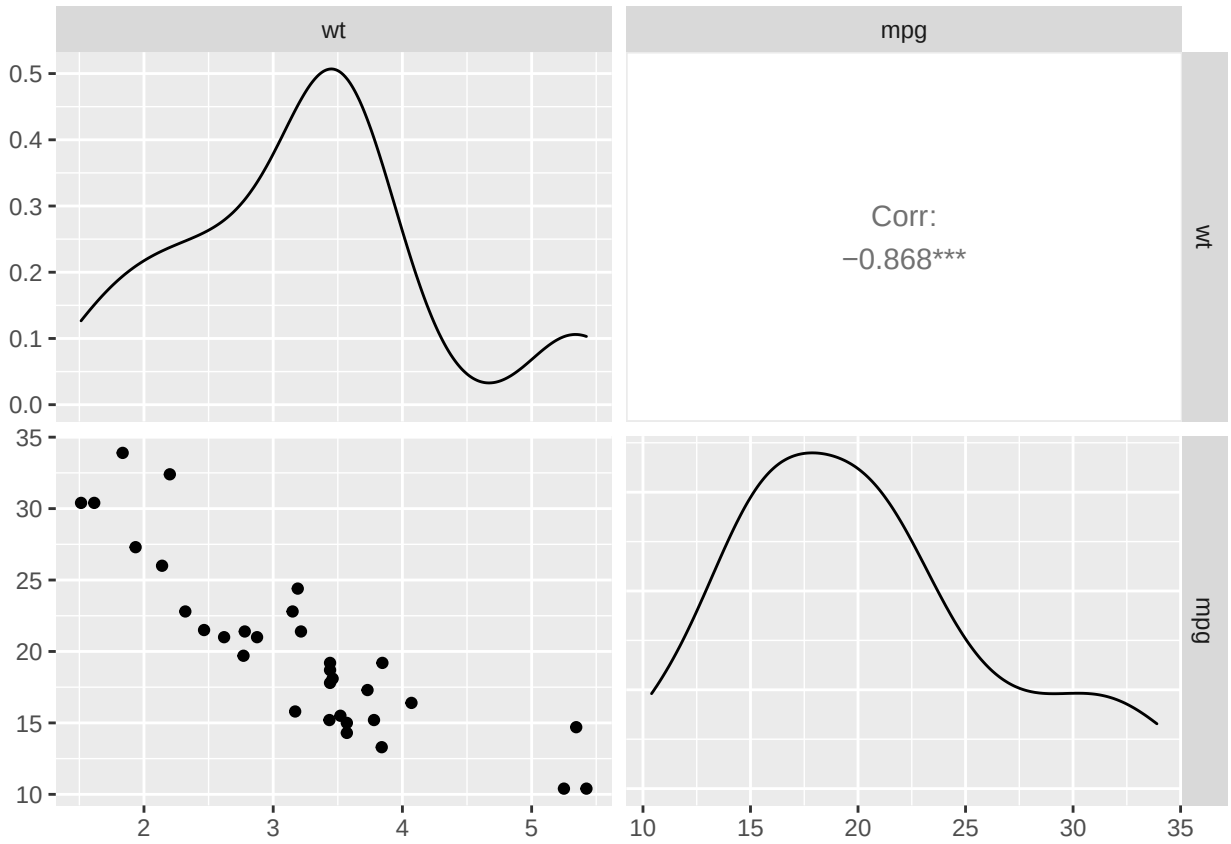
```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
mtcars %>% select(wt, mpg) %>%
ggscatmat()
```



Si nous souhaitons quelque chose de plus précis nous pourrons alors
changer ggscatmat() par ggpairs(), mais la fonction est cependant plus lente à utiliser.

```
mtcars %>% select(wt, mpg) %>% ggpairs()
```



J'ai également trouvé un package assez drôle *ggbernie*, bien que n'ayant pas réussi à l'installer sur R studio. **Par ici**.

FLEXDASHBOARD

Auteurs: Marko ARSIC
lien Github

J'ai été curieuse de découvrir ce package.
En effet travaillant plutôt sur des outils de type Power BI, il me paraissait intéressant de voir ce que R permet de faire.
Ce travail permet de comprendre de façon très simple comprend créer une structure de dashboard sur R.

Installons le package En chargeant d'abord le package puis la librairie. Je ne pouvais pas installer la package directement sur mon R studio. J'ai donc choisi une option via chooseCRANmirror(), en entrant le code correspondant à Lyon 1 (30) Voir ci-dessous :

```
#chooseCRANmirror()
#install.packages(c("flexdashboard"))
#library(flexdashboard)
```

Je créée ensuite selon les instructions du travail de Marko, un nouveau fichier rmd, from template. Cela m'ouvre un Dashboard vierge dans un nouveau fichier.

Cela se fait très rapidement, et en cliquant 'knit', l'aperçu est quasi immédiat.

La première question que je me suis posée en créant ce Dashboard, est celle de la flexibilité/de l'interactivité permise.

Les blocs visuels sont conçus de façon à remplir la page dans la hauteur du navigateur. Une option existe néanmoins pour créer un premier défilement via vertical-layout.(Nous remplaçons le fill par scroll). La barre de défilement apparait alors de façon quasi immédiate.

Une organisation en onglet, nous permet d'y voir encore plus clair. Pour cela il nous suffit d'ajouter la mention {.tabset} dans l'entête de section. Dans mon cas nous voyons que les deux lignes du bas s'organisent en onglet.

J'ai fait un premier test en insérant une représentation graphique utilisée avec prophet() un peu plus bas. Là encore chaque section possède ses propres chunks de code, ce qui permet d'insérer des graphes très rapidement bien que l'ajustement à la taille du cadre ne soit pas immédiat.

J'ai beaucoup aimé les jauges qui s'apparentent pour moi à la visualisation sous forme de carte sous Power BI. Flexdhabord permet de créer des jauges en : - en titrant la jauge
- agrégant les données via compute - en choisissant d'insérer ses données dans la jauge avec valuebox - en choisissant l'icône adaptée, via icon, 2nd argument de value box

Pour des questions de taille nous comprenons pourquoi le dashboard s'ouvre dans un nouveau fichier, néanmoins, nous regrettons quelque part de ne pas pouvoir avoir d'aperçu même rapide dans notre fichier rmd source.

Combinaison avec Shiny La combinaison avec Shiny me semble être un point intéressant. Il suffit pour cela d'ajouter la mention runtime:shiny tout en haut de votre section.

J'ai personnellement trouvé ce travail très didactique. Il m'a été très facile, en suivant les étapes de créer mon propre Dashboard. J'ai beaucoup aimé les représentations graphiques, qui accompagnent le lecteur tout au long de la création de son premier dashboard. Un travail très clair, agréable à lire et à dérouler.

Evaluation : 16/20

SHINY

Auteur: Benjamin GUIGON
lien Github

Shiny est un package bien utile à mon sens, il vient apporter de l'interactivité là ou flexdahboard propose une vision plus statique. Il permet la création de pages web interactives qui permettent en fait de réaliser toutes les analyses déjà possibles avec R. C'est un outil très plébiscité et qui ne nécessite pas toujours de connaissances dans d'autres langages de programmation (web notamment) lorsqu'on souhaite effectuer quelques realisations simples.

Il se compose à la manière d'une application classique de 2 blocs
- Un bloc « UI » : User Interface script qui permet de contrôler la mise en page et l'apparence de l'application.
- Un bloc « server » : Server script qui abrite les instructions dont l'ordinateur a besoin pour construire l'application.

Afin de pouvoir comprendre comment Shiny marche, j'ai décidé de procéder par étapes, en m'intéressant d'abord à chacun des blocs puis au module shinydahboard.

Bloc UI
A partir du tutoriel Shiny je créée l'interface.
Title Panel permet de définir le titre de l'interface générée.

Dans ce premier bloc je definis l'interface comme ci-dessous:

```
# Définition de l'interface utilisateur de notre application
#ui <- shinyUI(fluidPage(
```

Le titre de votre application

```
#titlePanel("Aperçu d'un dataset"),
```

Indiquer le 'layout' de votre application : autrement dit le squelette visuel de l'application

```
#sidebarLayout(
```

Composants de la région gauche de l'application

```
#sidebarPanel(
```

Ici, nous insérons un champ pour entrer un chiffre, ainsi #qu'un menu déroulant

```
# textInput(inputId = "lignes",
                #  label = "Combien de lignes voulez-vous voir ? ",
                 # value = 10),
     ## selectInput(inputId = "labs",
      #            label = "Dataset",
      #            choice = c("cars","rock","beaver1", "sleep")
 #                 )
   # ),
```

Ici, nous indiquons l'élement qui sera présent dans la fenêtre #principale : l'element "dataset".

```
#    mainPanel(
#      tableOutput("dataset")
#    )
#  )))
```

Dans le 2eme bloc, je vais finalement aller coder la partie qui se situe derrière mon interface :

```
#server <- shinyServer(function(input, output) {
  #On retrouve ici l'élement "dataset", qui communique avec ui par #'output'.
#  output$dataset <- renderTable({
```

Nous imprimons les éléments d'après les données en entrée : ces derniers sont appellés avec `input` puis le nom de la composante de ui (ici 'labs' et 'lignes').

```
#   if(input$labs == "cars"){
#    print(head(cars, input$lignes))
#   } else if(input$labs == "rock"){
#     print(head(rock, input$lignes))
#  } else if(input$labs == "sleep"){
#     print(head(sleep, input$lignes))
#  } else {
#     print(head(beaver1, input$lignes))
#   }
#  })
#})
```

Je souhaite ensuite avoir une idée de ce à quoi ressemble mon application en local. Je demande donc à shiny de charger la partie UI (front) et le server derrière.

```
#shinyApp(ui, server)
```

La création de Dashboard interactifs via Shiny passe par plusieurs étapes tout d'avoir via l'interface UI.
- une entête via DashBoard Header.
- Une première interactivité via dashboardSidebard.
- la création du cœur du tableau de bord avec dashboardBody.
- Puis la création de pages avec l'instruction tabItem.

Du cote server : - On va définir les représentations graphiques qui vont aller dans chaque section du tableau de bord. Chaque section sera matérialisée par un output que l'on pourra nommer comme l'exemple ci-dessous : output$Tableau_6

Le package semble facile à prendre en main, néanmoins la customisation de tableaux interactifs via Shiny semble toutefois demander une maitrise un peu plus poussée du sujet.

Un travail assez pousse et bien fourni sur Shiny.
En ce qui me concerne j'ai trouvé que bien que très abouti, le travail pouvait manquer d'aspects didactiques pour des débutants en R comme moi.
J'ai dû effectuer plusieurs recherches complémentaires enfin d'en comprendre le fonctionnement global. Une mention spéciale pour l'originalité du package choisi.

Appréciation globale :15/20.

liens utiles:
lien utile1.
lien utile2.
lien utile3.
lien utile4.

LUBRIDATE

Auteur:Gaspard PALAY **lien Github**

Pourquoi avoir choisi ce package ? C'est un package bien pratique pour manipuler des dates et j'ai moi-même été gênée de nombreuses fois par des cellules avec des dates entrées sous des formats peu pratiques.
Lubridate appartient à la collection Tidyverse, super utile pour manipuler des données.
Je vous recommande donc de télécharger la totalité du package si vous ne l'avez pas encore fait.

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
dmy_hm("1 Jan 2017 23:00")
```

```
## [1] "2017-01-01 23:00:00 UTC"
```

On constate ici qu'il reconnait immédiatement le format. A noter on choisit la fonction selon le format d'entrée (le format de date que nous avons plutôt que le format de sortie).

```
dmy_h("1 Jan 2017 23")
```

```
## [1] "2017-01-01 23:00:00 UTC"
```

Regardons si nous avons la meme chose avec la fonction

```
ydm("2017 17, March")
```

```
## [1] "2017-03-17"
```

```
duration("0 hours 30 min")
```

```
## [1] "1800s (~30 minutes)"
```

```
duration("1.5 hours")
```

```
## [1] "5400s (~1.5 hours)"
```

Les résultats affiches semblent étranges, on comprend ici difficilement l'affichage.

Lubridate est un package à garder absolument sous le coude.

Appréciation. Le travail m'a paru très didactique, clair et synthetique, illustre d'exemples tout du long.

Il permet d'appliquer soi-même très rapidement les différentes méthodes proposées.
La Cheat Sheet aurait pu être integrée directement dans le corps du document. On notera l'absence de bibliographie, que l'on constate souvent sur les descriptions de packages R, les miens y compris. Note : 15/20

Par ici vers la **cheat sheet**

Liens utiles :
**lien utile1*.
**lien utile2*.
**lien utile3*

Une Confusion Matrix (matrice de confusion) ou tableau de contingence est un outil permettant de mesurer les performances d'un modèle de Machine Learning en vérifiant notamment à quelle fréquence ses prédictions sont exactes par rapport à la réalité dans des problèmes de classification.

PROPHETE

Auteur: Lucas
lien Github

J'ai choisi Prophete car il permet de faire des prédictions sur des séries temporelles, et combine donc agréablement code R et mathématiques. Ce package CRAN a été développé par Facebook.

Autre **lien intéressant** pour se procurer des jeux de données :

Installons le afin de voir de quoi il s'agit. Et chargeons également sa librairie. Chargeons ensuite le jeu de données présent dans R "AirPassengers". Je n'arrivais pas à utiliser les données Airpassengers telles quelles, en attendant je me suis donc amusée à les analyser. Cela m'a permis de comprendre que les données étaient déjà organisées de façon très propre en séries temporelles.

```
library(prophet)
```

```
## Loading required package: Rcpp
```

```
## Loading required package: rlang
```

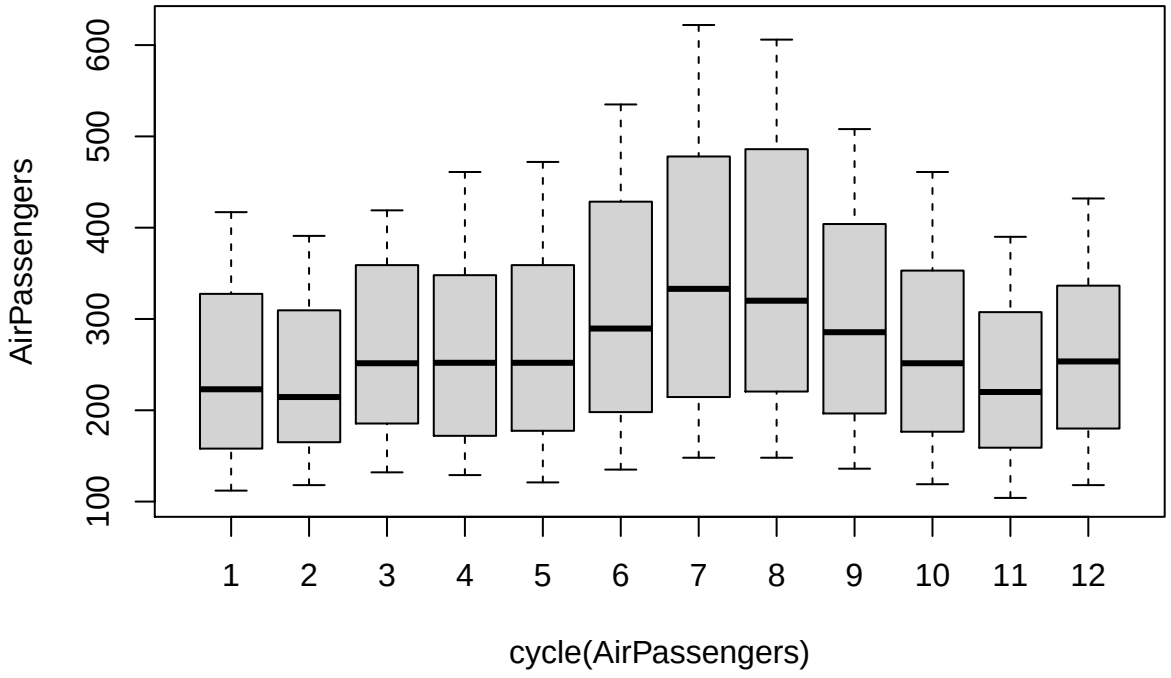
```
data("AirPassengers")
summary(AirPassengers)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   104.0   180.0   265.5   280.3   360.5   622.0
```

```
cycle(AirPassengers)
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949   1   2   3   4   5   6   7   8   9  10  11  12
## 1950   1   2   3   4   5   6   7   8   9  10  11  12
## 1951   1   2   3   4   5   6   7   8   9  10  11  12
## 1952   1   2   3   4   5   6   7   8   9  10  11  12
## 1953   1   2   3   4   5   6   7   8   9  10  11  12
## 1954   1   2   3   4   5   6   7   8   9  10  11  12
## 1955   1   2   3   4   5   6   7   8   9  10  11  12
## 1956   1   2   3   4   5   6   7   8   9  10  11  12
## 1957   1   2   3   4   5   6   7   8   9  10  11  12
## 1958   1   2   3   4   5   6   7   8   9  10  11  12
## 1959   1   2   3   4   5   6   7   8   9  10  11  12
## 1960   1   2   3   4   5   6   7   8   9  10  11  12
```

```
boxplot(AirPassengers~cycle(AirPassengers))
```

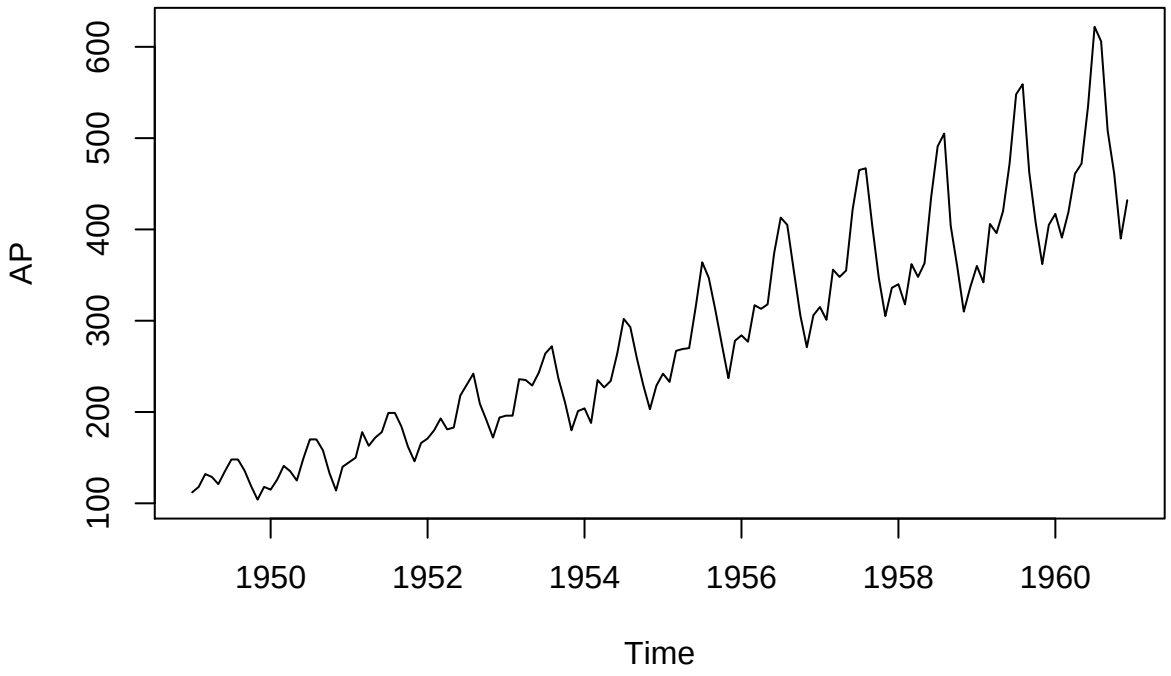


Il me fallait changer le format des données pour pouvoir les utiliser avec prophet(). J'ai pour cela reçu un petit coup de pouce d'un camarade, qui m'a néanmoins permis de comprendre comment les données doivent être organisées pour pouvoir être utilisées avec prophet. Elles doivent impérativement comporter une colonne ds, qui est en fait la colonne date. Afin de transformer le format des données, j'ai utilisé le package zoo. vous y trouverez plus de détails **ici*

```
AP <- (AirPassengers)
AP
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
## 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

```
plot(AP)
```




```
DF <- window(AirPassengers, end = c(1958, 12))
test  <- window(AirPassengers, start = c(1959, 1), end = c(1960,12))
```

```
#install.packages("zoo")
#library(zoo)
#airp <- data.frame(ds = as.Date(as.yearmon(time(DF))), #y=as.matrix(DF))
```

```
#airp
#View(airp)
#DF2 <- prophet(airp,weekly.seasonality=TRUE, daily.seasonality=TRUE)
#DF2
```

Nous avons rétabli ici le format des données pour avoir des dates affichées sous le format année, mois, jour et respecter le format requis décrit par Lucas: - ds reflects the date (YYYY-MM-DD, or YYYY-MM-DD HH:MM:SS)
- y represents the output in this date (numerical values)

Prophete peut se résumer par l'équation suivante : y(t) = g(t) + s(t) + h(t) + e(t) - g(t) la tendance
- s(t) la saisonnalité - h(t) les irrégularités du modèle
- e(t) les erreurs

```
#plot(AP, ylab= "Passengers (1000's)")
#plot(decompose(AirPassengers))
```

Pour une raison qui m'est inconnue la manipulation avec zoo n'a plu marché.
Je n'ai donc pas pu tester prophete sur mon jeu de données. En revanche j'ai noté 2 avantages a ce package, il combine : - prédiction - possibilité de représentation graphique, là où j'ai eu l'impression que d'autres comme stats ont des possibilités plus limitées.

Il s'agit d'un bon travail, très didactique à mon sens (avec une description étape par étape depuis l'installation. On note un bon travail de vulgarisation des concepts mathematiques
J'ai beaucoup aimé les représentations graphiques qui accompagnent le travail.
Il s'agit d'un des rares travaux avec une bibliographie.
Un parallèle avec d'autres packages type stats ou Rminer ou encore une comparaison avec des modèles Arima aurait été appréciée.

Evaluation : 15/20

lien utile

DERIVEES

Auteur : Corentin Bretonniere.
lien Github

Il s'agit ici d'un concept mathématique assez simple. Il est toutefois intéressant de pouvoir l'utiliser sous R Utiliser du code R Les « highlights » R : - Utilisation de la fonction D, qui nous permet de dériver une fonction - Une fonction plus simple à utiliser pour moi que deriv

```
f=expression(x^2+3*x)
D(f,'x')
```

```
## 2 * x + 3
```

```
a<-expression(x^2+3*x+5*y^6)
D(a,'x')
```

```
## 2 * x + 3
```

```
D(a,'y')
```

```
## 5 * (6 * y^5)
```

On voit qu'on obtient ainsi : - La dérivée par rapport à x - La dérivée par rapport à y

On notera qu'il nous faut définir notre fonction au préalable. Nous avons également la possibilité de créer notre propre fonction.

```
monExpression <- deriv(~4*x^2 + sqrt(x), "x")
maDerivee <- function(x){
    monExpression
    r <- eval(monExpression);
    r <- attr(r, 'gradient');
   }; monExpression
```

```
## expression({
##     .value <- 4 * x^2 + sqrt(x)
##     .grad <- array(0, c(length(.value), 1L), list(NULL, c("x")))
##     .grad[, "x"] <- 4 * (2 * x) + 0.5 * x^-0.5
##     attr(.value, "gradient") <- .grad
##     .value
## })
```

Le rappel de Mon Expression à la fin permet d'afficher la valeur. Comme en Python nous retrouvons sqrt(x) pour la racine carrée. On notera l'usage du ~ indispensable au bon fonctionnement de la fonction.

```
valeurDunPoint <- maDerivee(2); valeurDunPoint
```

```
##             x
## [1,] 16.35355
```

On obtient donc la valeur de la dérivée en un point précis. Je ne trouve le calcul des dérivées si pratique sur R.

On retrouve plusieurs fonctionnalités associées au calcul de dérivées dans le package stats. Installons-le.

```
library(stats)
```

```
x <- c(1,2,3,4)
y <- c(4, 15, 32, 54)
p0 <- 3
p1 <- 2
numericDeriv(quote( p0 * x^p1 ), c("p0", "p1"))
```

```
## [1]  3 12 27 48
## attr(,"gradient")
##      [,1]      [,2]
## [1,]    1  0.000000
## [2,]    4  8.317766
## [3,]    9 29.662532
## [4,]   16 66.542130
```

A noter : en entrée de numericDeriv nous devons avoir un vecteur exprimé sous forme numérique. En second argument nous plaçons un vecteur contenant les noms de variables. Analyse du résultat :

3 12 27 48 correspond en fait à : 3 = 3 * 1^2 12 = 3 * 2^2 27 = 3 * 3^2 48 = 3 * 4^2 En sortie nous observons qu'en première colonne à gauche nous avons les valeurs correspondant à l'expression (quote dans laquelle nous avons remplace les valeurs de x et à droite. les dérivées correspondant aux éléments des variables nommées en 2nd argument).

Splinefun Splinefun semble être une extension intéressante à considérer.

```
x<-1:10
y<-x^2

fff<-splinefun(x,y)
fff(3,deriv=0)
```

```
## [1] 9
```

En ayant sur d'autres packages et fonctions, je ne trouve pas que le calcul de dérivé es ait été particulièrement simplifié sur R. Appréciation J'ai trouvé le travail assez didactique et facile à comprendre autant d'un point de vue mathématique que du point de vue du code R Pas de bibliographie comme dans beaucoup de travaux sur les packages R. Une distinction entre les différents type de dérivées (premières secondes etc) aurait pu être ajoutée.

lien utile1.
lien utile2
lien utile3

Appreciation : 13/20

UNE APPROCHE DEEP LEARNING POUR LA CLASSIFICATION DE MODELES BIM

Auteur: Thomas MASSE
lien Github.
J'ai choisi ce travail car j'entends beaucoup parler du BIM autour de moi sans en saisir tous les concepts encore.

Derrière le terme BIM (Building Information Modelling) se cache la modélisation de maquettes 3D numériques.

Le BIM repose principalement sur des environnements de travail en 3D. Un objet BIM, qui est en fait la réalisation virtuelle d'un ouvrage, se compose d'une géométrie 3D et de ses attributs.

Ce travail explique comment un modèle de Deep Learning peut aider à la réalisation du BIM.

J'ai repris les étapes telles que décrites dans le papier afin de pouvoir mieux les comprendre.

1 Choisir un jeu de données d'entraînement et de tests appropriés.
2 Traiter les modèles 3D et représenter ses attributs sous forme de matrice.
3 Traiter cette donnée comme un auto-encodeur model.
4 La sortie d'une couche cachée du modèle auto-encodé est utilisée comme entrée de la couche.
5 Initialiser les paramètres pour chaque couche de l'étape 4.
6 La sortie de la dernière couche cachée est définie comme entrée d'une couche supervisée.
7 Ajuster tout les paramètres du réseaux neuronal en fonction des règles de supervision.
Ajouter tout les modèles auto-encodés pour former un stacked auto-encoder model.

Un auto-encodeur est un réseau de neurones artificiels utilisé dans le cadre d'un apprentissage non supervisé, il est composé d'un décodeur et d'un encodeur.
On se réfèrera au travail sur le GAN afin de se rappeler ce qu'est un réseau de neurones.
L'apprentissage non supervisé : l'apprentissage dans le cadre ou les données ne sont pas étiquetées, c'est à dire ou l'algorithme/la machine va regrouper les données en clusters. La machine "n'apprend" pas à prédire selon un étiquetage qui lui serait fourni, elle le fait de façon indépendante.

Ici la couche d'entrée et de sortie possèdent le même nombre de nœuds, contrairement à ce qu'on a pu voir sur du GAN. On cherche en effet à reconstituer de l'information, non à prédire.

L'encodeur va chercher à adapter/diminuer les dimensions des données afin de pouvoir les reproduire dans un espace aux nouvelles dimensions.

Bien que ne comprenant pas entièrement la notation. Je comprends l'intuition derrière. On comprend que la première formule encode, avec F en sortie, tandis que la 2eme décode, avec F en entrée.

encodeur:
$\phi : X \rightarrow$

décodeur :
$\psi : F \rightarrow X$

On voit que derrière cette formule se cache Argmin. L'argument minimum ou Argmin d'une fonction représente la valeur de la variable pour laquelle la valeur de la fonction concernée atteint son minimum. On a donc ici l'idée de minimisation des dimensions, sans saisir pour autant toute la formule.

$phi, \psi = arg = min(\phi, \psi) \mid\mid X - (\phi \circ \psi)X \mid\mid$

Il s'agit d'un bon travail.
On sent que l'auteur maitrise le sujet du BIM.
J'ai beaucoup apprecie l'illustration graphique et l'actualité du sujet.
J'ai moins reussi m'approprier les concepts mathématiques en revanche.
Points à noter : l'absence de bibliographie.

lien utile et un autre

LES RESEAUX DE NEURONES ANTAGONISTES GENERATIFS.

Auteur : Jeremy Sayag.
lien Github

Quand on parle de réseau de neurones, on pense tout de suite à l'intelligence artificielle.

Un bref rappel sur ce qu'est un réseau de neurones :
- Une couche d'entrée
- Une couche de sortie.
- Une couche cachée entre les deux premières.

NB : A partir de plus de 3 couches on commence généralement à parler de Deep learning (apprentissage profond). Dans ces réseaux de Deep Learning chaque couche de nœuds s'entraine sur des données qui proviennent de la sortie de la couche précédente. Plus le nombre de couches est important, plus la capacité à reconnaître des informations complexes est grande.
Le réseau prend des décisions en fonction de la valeur de l'information attribuée à un nœud individuel, aussi appelée « poids». Le poids c'est finalement la valeur d'utilité que le réseau donne à cette information, c'est-à-dire à quel point elle lui est utile pour classer l'information. Si le poids est trop faible, l'information ne sera pas transmise à la couche suivante. Le poids permet donc en quelque sorte de « filtrer » ce qui sort des nœuds. Revenons à nos neurones antagonistes. Pourquoi antagonistes ? Parce qu'on parle en fait de réseaux antagonistes.
- Un réseau *Générateur* qui produit un contenu dit « réaliste »
- Un second réseau qu'on appelle *Discriminateur* qui est finalement là pour policer la sortie du réseau générateur.
Il se charge d'estimer si le contenu en sortie du générateur semble réaliste ou non.

Ces réseaux collaborateurs dans le but d'atteindre un équilibre.
Cet équilibre est finalement atteint lorsque le générateur arrive à créer un mirage tel que le discriminateur ne parvient plus à distinguer l'image artificielle produite d'une image réelle.

Analyse des concepts mathématiques : 1ère formule à insérer.
En revenant à la définition d'un réseau de neurone et à la notion de poids, nous ne sommes pas surpris de voir apparaitre des probabilités ici.
De ce que je comprends.
Le Générateur cherche à maximiser le poids que le Discriminateur va attribuer à l'information soit vraie c'est-à-dire la probabilité que l'information z soit vraie : D(G(z)).
Formule du discriminateur optimal lorsque le générateur est fixe : Formule à insérer.
Là encore on comprend que plus pg(x) est grande plus le poids fourni par le générateur va diminuer.
Si pg(x) devenait nulle, alors DG(x) serait maximale (1).
Là encore l'intuition nous laisse penser qu'au maximum pg = pdata et qu'alors Dg(x) est optimal lorsqu'elle vaut 1/2 (le même poids est alors donné à l'information qu'elle soit réelle ou irréelle).
C'est à ce moment que le discriminateur deviendrait incapable de faire la distinction

Le travail me parait intéressant car il « vulgarise » un concept assez compliqué et propose d'aller soi-même tester des applications concrètes comme sur playground.tensorflow.org J'ai beaucoup apprécié le schéma qui facilite la compréhension. L'aspect didactique est pour moi complètement respecté. On aurait peut-être aimé pour les novices un petit rappel sur les réseaux de neurones

Evaluation: 16/20

EVALUATION DE MES TRAVAUX :

ALGEBRE DE GROUPE EN CARACTERISTIQUE 1

lien Github

Il n'est jamais évident d'évaluer son propre travail. Mon travail s'est probablement plus concentre sur l'algèbre tropicale et ses notions que la géométrie qui en découle et les idempotents. Selon moi un bon effort de vulgarisation des concepts a été fait, notamment à l'aide de graphiques. J'ai été limitée par ma compréhension des concepts mathématiques. J'ai toutefois tenté de m'approprier les concepts d'algebre. On note l'effort fait sur la manipulation de LaTex, via l'éditeur overleaf, très pratique. Une prochaine étape serait de faire la meme chose sous R.

Evaluation : 14/20

TRAVAIL SUR JANITOR

lien Github

Il s'agit d'un travail portant sur un package R relativement simple : JANITOR.
Sa fonction principale est de nettoyer des jeux de données en temps réduit. Il permet ainsi grâce à ses différentes fonctions d'optimiser le temps de traitement d'un jeu de données.
J'ai indiqué dans le travail les fonctions qui m'ont paru les plus intéressantes.
Ce travail aurait sûrement mérité qu'on lui ajoute des liens supplémentaires vers d'autres packages ainsi qu'une correspondance vers Python.
Il reflète à mon sens le niveau de R que j'avais au moment où il a été rédigé.
Ce n'est pas le package le plus original, mais plutôt un package à avoir dans son tool kit R.

Evaluation : 12/20