Codelab R Shiny

Aline Deschamps

14 novembre 2016

1 - Rappels

Rappels sur R

R : Langage de programmation pour les statistiques / la data science

- Open source
- Gratuit
- Multiplateformes (Windows, Mac, Linux)
- Plus de 9000 packages permettant l'utilisation de méthodes statistiques dans de nombreux domaines divers et variés : analyse de données, séries temporelles, analyse de survie, machine learning, équations différentielles, économétrie, psychométrie, finances, génétique, etc…

Rappels sur Shiny

Shiny : package R (développé par RStudio) permettant de générer une application web dynamique, présentant des résultats statistiques et graphiques réalisés avec R, le tout sans avoir besoin d'aucune connaissance en HTML, CSS ou JavaScript

Liens :

- Le package "shiny" sur le CRAN
- La page "shiny" sur RStudio

2 - Pré-requis

Pré-requis : R et packages

- Installation de R version 3.3.1 minimum (et RStudio recommandé ;)
- Installation des packages R suivants :
  - *shiny*
  - *twitteR*
  - *tidyr*
  - *DT*
  - *rAmCharts*
  - *tm*
  - *wordcloud*

```
install.packages("nom_du_package", dependencies = TRUE)
```

Pré-requis : paramètres Twitter

- Avoir (créer) un compte Twitter
- Configurer connexion API : https://apps.twitter.com
  - *Cliquer sur le bouton “Create New App”*
  - *Remplir les champs “Name” (unique), “Description” (min. 10 caractères) et “Website”*
  - *Cocher la case pour valider le “Twitter Developer Agreement” et cliquer sur “Create your Twitter Application”*
- Cliquer alors sur “Keys and Access Tokens” et récupérer les éléments suivants :
  - *Consumer Key*
  - *Consumer Secret*
  - *Access Token*
  - *Access Token Secret*

3 - Objectifs du codelab

Objectifs du codelab

- Se connecter à Twitter depuis R
- Importer des données Twitter dans R
- Créer une application “Shiny” avec R :
  - *Sélection dynamique d’un hashtag à rechercher sur Twitter*
  - *Récupération dynamique des tweets correspondants au hashtag sélectionné*
  - *Affichage de résultats texte : nombre de tweets récupérés par exemple*
  - *Affichage des tweets récupérés sous forme d’un tableau manipulable*
  - *Affichage de graphiques dynamiques avec “rAmCharts” : nombre de tweets retweetés par exemple*
  - *Travail du texte des tweets récupérés et représentation sous la forme d’un “wordcloud”*

4 - R et Twitter

R et Twitter

On va d'abord voir comment récupérer des données Twitter depuis R, pour le moment on ne s'occupe pas de la partie web "Shiny".

On commence par créer un script R "Twitter_to_R.R" :

```
library(twitteR)

setup_twitter_oauth(consumer_key = "XXX",
                    consumer_secret = "XXX",
                    access_token = "XXX",
                    access_secret = "XXX")
```

=> C'est ici qu'il faut indiquer (à la place de "XXX") les identifiants récupérés plus tôt : *Consumer Key*, *Consumer Secret*, *Access Token*, *Access Token Secret*.

R et Twitter

On va à présent extraire les tweets contenant le hashtag #DataScience :

```r
tweets <- searchTwitter("#DataScience", n = 20)

tweets_df <- twListToDF(tweets)
# cette fonction va permettre de transformer la liste de tweets extraite en un "dataframe"

dim(tweets_df)[1] # nombre de tweets récupérés
```

tweets_df

Filter

	text	favorited	favoriteCount	replyToSN	created	truncated	replyToSID	id	replyToUID	statusSource
1	24 Questions to Ask when Preparing Data for Analysi...	FALSE	1	NA	2016-10-03 14:30:33	FALSE	NA	782950973874532352	NA	<a href="http://bufferapp.com" rel="nofollow">Buffe...
2	RT @KirkDBorne: Reinforcement Learning and #AI: ht...	FALSE	0	NA	2016-10-03 14:30:17	FALSE	NA	782950905964408832	NA	<a href="http://twitter.com/download/iphone" rel="...
3	closing soon #datascience for #iot course https://t...	FALSE	0	NA	2016-10-03 14:30:08	FALSE	NA	782950869016928256	NA	<a href="http://bufferapp.com" rel="nofollow">Buffe...
4	RT @dataiku: What is hardcore #DataScience - in pra...	FALSE	0	NA	2016-10-03 14:29:55	FALSE	NA	782950815086448640	NA	<a href="http://twitter.com/download/android" rel="...
5	Nice read. Thought you'd like this @grdobbin https:/...	FALSE	0	NA	2016-10-03 14:28:51	FALSE	NA	782950545636134913	NA	<a href="http://twitter.com" rel="nofollow">Twitter W...
6	Do not put your faith in what #statistics say until you...	FALSE	0	NA	2016-10-03 14:28:27	FALSE	NA	782950447585845248	NA	<a href="http://twitter.com" rel="nofollow">Twitter W...
7	RT @gp_pulipaka: Three Stages of #AI. #BigData #Ma...	FALSE	0	NA	2016-10-03 14:28:16	FALSE	NA	782950398390743040	NA	<a href="http://twitter.com/download/android" rel="...
8	How to Use Data Visualizations to Win Over Your Aud...	FALSE	0	NA	2016-10-03 14:28:04	FALSE	NA	782950350148042753	NA	<a href="http://bufferapp.com" rel="nofollow">Buffe...
9	Tom's Tutorials For Excel: Listing Column Letters Acr...	FALSE	0	NA	2016-10-03 14:28:00	FALSE	NA	782950332317962240	NA	<a href="http://www.crowdbooster.com" rel="nofoll...
10	Chief data scientist @ White House: Every #DataScie...	FALSE	1	NA	2016-10-03 14:27:04	FALSE	NA	782950096858120192	NA	<a href="http://twitter.com" rel="nofollow">Twitter W...

R et Twitter

- Paramètres de la fonction "*searchTwitter*" :
  - ***searchString*** *: chaîne de caractères qui sera contenue dans les tweets recherchés (utiliser le caractère "+" pour séparer les mots entre eux),*
  - ***n*** *: nombre maximum de tweets à récupérer,*
  - ***lang*** *: NULL par défaut, sinon permet de spécifier la langue souhaitée pour les tweets récupérés (ex: "fr", "en"),*
  - ***since*** */* ***until*** *: dates de début et de fin encadrant la recherche de tweets effectuée,*
  - ***resultType*** *: "mixed" (défaut) / "recent" / "popular".*

Voir `?searchTwitter` pour plus de détails.

R et Twitter

Tester les recherches Twitter depuis R avec d'autres mots-clés et explorer les données :

- "#BigData" :

```
# ToDo
```

- "#MachineLearning" :

```
# ToDo
```

R et Twitter

Tester les recherches Twitter depuis R avec d'autres mots-clés et explorer les données :

- "#BigData" :

```
tweets_bg <- searchTwitter("#BigData", n = 20)

tweets_bg_df <- twListToDF(tweets_bg)

View(tweets_bg_df)
```

- "#MachineLearning" :

```
tweets_ml <- searchTwitter("#MachineLearning", n = 20)

tweets_ml_df <- twListToDF(tweets_ml)

View(tweets_ml_df)
```

R et Twitter

Il est aussi possible de récupérer les tweets d'un auteur en particulier :

```
tweets_RBloggers <- userTimeline("RBloggers", n = 20)

tweets_RBloggers_df <- twListToDF(tweets_RBloggers)
```

Ou bien encore de récupérer les tendances Twitter du moment :

```
a_trends = availableTrendLocations()
head(a_trends)
a_trends[which(a_trends$country == "France"), ]

trends_nantes <- getTrends(woeid = 613858)
```

5 - Shiny

Shiny : rappels

Deux fichiers “**ui.R**” et “**server.R**” :

- “**ui.R**” : *User Interface script*, contrôle la mise en page et l’apparence de l’application
- “**server.R**” : *Server script*, contient les instructions dont l’ordinateur a besoin pour construire l’application

Shiny : initialisation “ui.R”

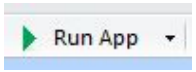
```
library(shiny)

shinyUI(fluidPage(

  # Titre général de l'application Shiny
  titlePanel("Hello Shiny !"),

  sidebarLayout(

    # Barre latérale : va contenir les éléments de contrôle / gestion des paramètres
    sidebarPanel("Barre latérale"),

    # Zone principale : va contenir les résultats / sorties, générés via la partie "server"
    mainPanel(h2("Zone principale"))

  )

))
```

Shiny : initialisation “server.R”

```
library(shiny)

shinyServer(function(input, output) {

  # sera complété ultérieurement...

})
```

Shiny : Générer l'appli

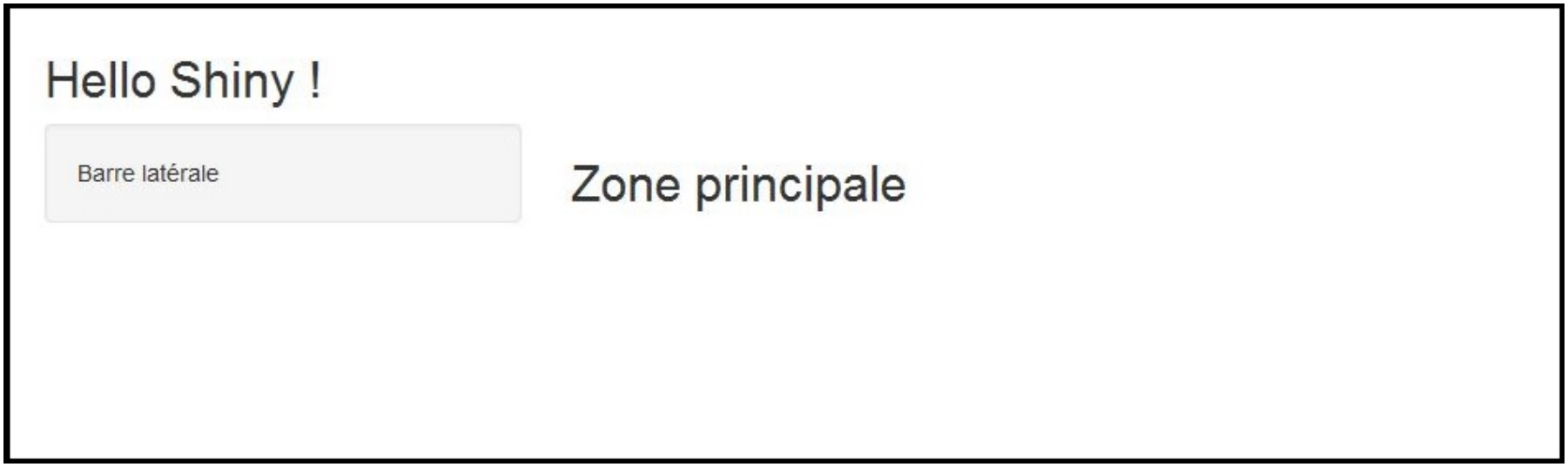
Le plus simple pour générer l'appli Shiny est de cliquer sur le bouton "Run App" en haut à droite de la zone de script dans l'IDE RStudio :



Sinon, cela peut aussi être fait avec la commande suivante ("mon_appli" étant le dossier dans lequel sont stockés les fichiers "ui.R" et "server.R") :

```
runApp("mon_chemin/mon_appli")
```

Shiny : Ma première appli



Shiny : Remarque 1

Un ensemble d'applications Shiny de démo sont chargées par défaut avec le package. Elles contiennent également le code permettant de les recréer.

Pour les voir :

```
library(shiny)

runExample()
#Valid examples are "01_hello", "02_text", "03_reactivity", "04_mpg", "05_sliders",
#"06_tabsets", "07_widgets", "08_html", "09_upload", "10_download", "11_timer"

runExample("01_hello")
```

Shiny : Remarque 2

Pour lancer votre appli avec affichage du code, vous pouvez utiliser la commande suivante :

```
runApp("mon_chemin/mon_appli", display.mode = "showcase")
```

Shiny : le fichier “global.R”

Il est également possible, en plus du fichier “ui.R” et du fichier “server.R”, de créer un fichier “**global.R**” qui sera chargé au lancement de l’appli.

Il est utilisé pour l’ensemble des actions à effectuer une seule fois au lancement de l’appli, par exemple : chargement des packages utiles, chargement d’un jeu de données statique (ex : CSV), connexion à l’API Twitter, etc.

- global.R :

```
library(twitteR)

setup_twitter_oauth(consumer_key = "XXX",
                    consumer_secret = "XXX",
                    access_token = "XXX",
                    access_secret = "XXX")
```

Shiny : Afficher un texte

On va ici afficher un texte dans la zone principale de l'application.

Ce texte contiendra notamment une valeur aléatoire entre 1 et 9, générée au lancement de l'appli.

Pour cela on va devoir effectuer quelques modifications dans nos deux fichiers "ui.R" et "server.R".

Shiny : ui.R

```
library(shiny)

shinyUI(fluidPage(

  titlePanel("Hello Shiny !"),

  sidebarLayout(

    sidebarPanel("Barre latérale"),

    mainPanel(h2("Zone principale"),

      textOutput("mon_resultat")
    )

  )

))
```

Shiny : “server.R”

```
library(shiny)

shinyServer(function(input, output) {

  output$mon_resultat <- renderText({

      ma_valeur <- sample(1:9, 1)

      paste0("Voici le chiffre généré : ", ma_valeur)

  })

})
```

Shiny : Résultat

NB : Actualiser plusieurs fois l'appli pour voir le chiffre varier.

Shiny : ouput

"*renderText*" est une fonction **reactive** qui permet d'afficher dynamiquement des sorties "texte" dans l'appli shiny.

Il existe une fonction correspondante pour chaque type de sortie souhaitée, par exemple : "*renderPlot*", "*renderTable*", …

Shiny : Récupérer des tweets

A présent, nous allons modifier uniquement la partie “server.R” : au lieu de récupérer un chiffre aléatoirement, nous allons récupérer les tweets correspondant à un hashtag en particulier et afficher en sortir le nombre de tweets récupérés.

Par exemple, nous allons prendre les tweets à propos du *#BigData*.

Shiny : “server.R” à compléter

```
library(shiny)

shinyServer(function(input, output) {

  output$mon_resultat <- renderText({

      tweets_bigdata <- (...)

      tweets_bigdata_df <- twListToDF(tweets_bigdata)

      ma_valeur <- (...)

      paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

})
```

Shiny : “server.R” correction

```
library(shiny)

shinyServer(function(input, output) {

  output$mon_resultat <- renderText({

      tweets_bigdata <- searchTwitter("#BigData", n = 50)

      tweets_bigdata_df <- twListToDF(tweets_bigdata)

      ma_valeur <- dim(tweets_bigdata_df)[1]

      paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

})
```

Shiny : Résultat

Shiny : Récupérer des tweets DYNAMIQUEMENT

On a maintenant une appli Shiny qui nous permet de récupérer les tweets associés au hashtag #BigData, mais si on veut changer le hashtag ?

On va donc ajouter dans notre appli une liste déroulante de hashtags pour récupérer **dynamiquement** les tweets associés au hashtag qui sera sélectionné par l'utilisateur.

Pour cela on va ajouter notre **liste déroulante** de sélection de hashtags dans la "barre latérale" de notre appli (partie "ui.R") et on aura juste à ajouter un simple paramètre dans la partie "server.R".

Shiny : ui.R

```
library(shiny)

shinyUI(fluidPage(

  titlePanel("Hello Shiny !"),
  sidebarLayout(
    sidebarPanel("Barre latérale",

        selectInput(inputId = "mes_hashtags",
                    label = "Sélectionner un hashtag",
                    choices = list("#BigData" = "#BigData",
                                   "#DataScience" = "#DataScience",
                                   "#MachineLearning" = "#MachineLearning"),
                    selected = 1)

    ),
    mainPanel(h2("Zone principale"),
      textOutput("mon_resultat")
    )
  )
))
```

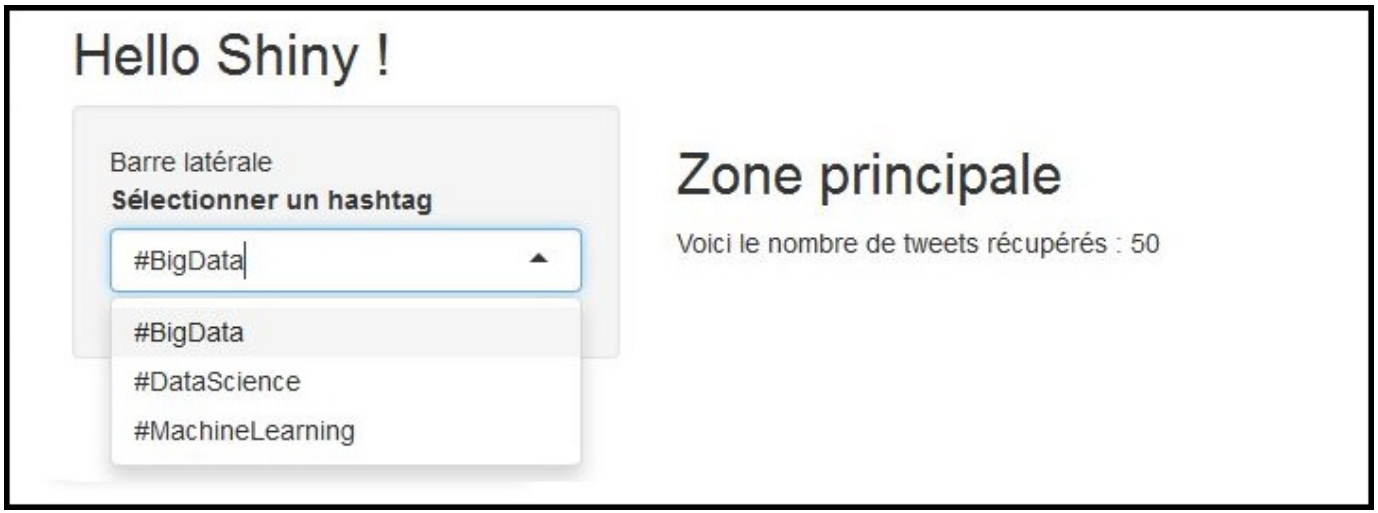
Shiny : Récupérer des tweets dynamiquement

```
selectInput(inputId = "mes_hashtags",
            label = "Sélectionner un hashtag",
            choices = list("#BigData" = "#BigData",
                           "#DataScience" = "#DataScience",
                           "#MachineLearning" = "#MachineLearning"),
            selected = 1)
```

Paramètres de la fonction “selectInput” : *(Pour plus d’options, voir ici.)*

- **inputID** : identifiant unique permettant de récupérer la valeur sélectionnée (côté server)
- **label** : libellé/titre qui sera affiché au-dessus de la liste déroulante (peut contenir de la mise en forme, ex: *h2(…)*)
- **choices** : liste (au sens R) correspondant au contenu de la liste déroulante, avec la syntaxe suivante : *list(“libellé1” = ValeurAssociee, …)*
- **selected** : élément sélectionné par défaut (peut être mis à jour conditionnellement)

Shiny : Résultat



Shiny : Récupérer des tweets dynamiquement

Affichage liste déroulante ok mais rien ne change au niveau résultat : il faut modifier le fichier “**server.R**” !

Shiny : server.R

```
library(shiny)

shinyServer(function(input, output) {

  output$mon_resultat <- renderText({

      tweets <- searchTwitter(input$mes_hashtags, n = 50) # valeur sélectionnée

      tweets_df <- twListToDF(tweets)

      ma_valeur <- dim(tweets_df)[1]

      paste0("Voici le hashtag utilisé : ", input$mes_hashtags) # valeur sélectionnée

  })

})
```

Shiny : Résultat

Shiny : Bonne pratique “server.R”

Dans la mesure où nous allons par la suite utiliser les tweets récupérés à plusieurs endroits : tableau, graphiques, etc., il est préférable des les stocker dans un élément à part et dynamique, afin que l’appel ne soit fait qu’une seule fois.

Nous allons donc les stocker dans un élément appelé “**mes_tweets**” et qui sera un élément de type *reactive*.

En savoir plus sur les éléments de type *reactive* : http://shiny.rstudio.com/tutorial/lesson6/.

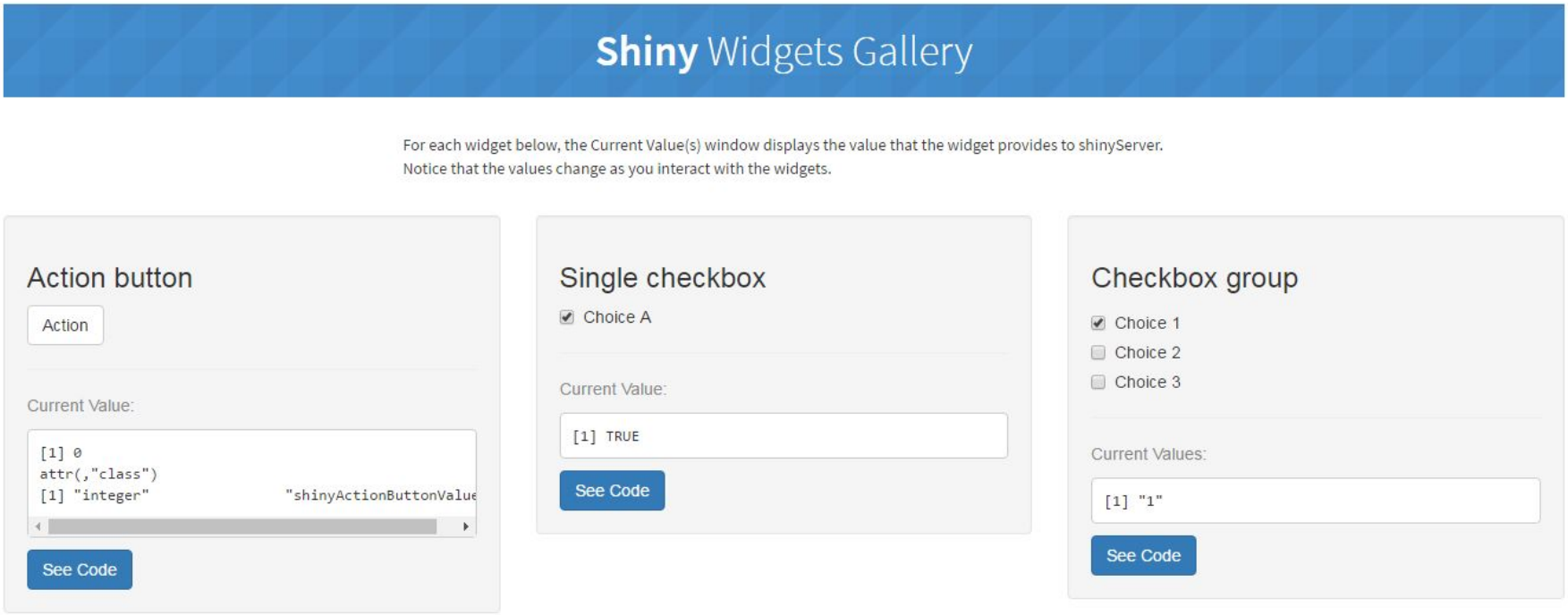
Shiny : Bonne pratique “server.R”

```
library(shiny)

shinyServer(function(input, output) {

  mes_tweets <- reactive({

    tweets <- searchTwitter(input$mes_hashtags, n = 50)
    tweets_df <- twListToDF(tweets)
    return(tweets_df)

  })

  output$mon_resultat <- renderText({

    ma_valeur <- dim(mes_tweets())[1]
    paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

})
```

Shiny : Les différents sélecteurs pour la barre latérale

On a vu ici une sélection via une liste déroulante.

Il existe un certain nombre d'autres types de *sélecteurs* disponibles dans Shiny. En voici une liste avec démonstration et code associé : Shiny Widgets Gallery.



Shiny : Exercice

Ajouter, en plus de la liste déroulante de sélection de hashtags, un élément de type “**slider**” permettant de sélectionner DYNAMIQUEMENT le nombre de tweets récupérés, avec les paramètres suivants :

- Minimum = 10
- Maximum = 60

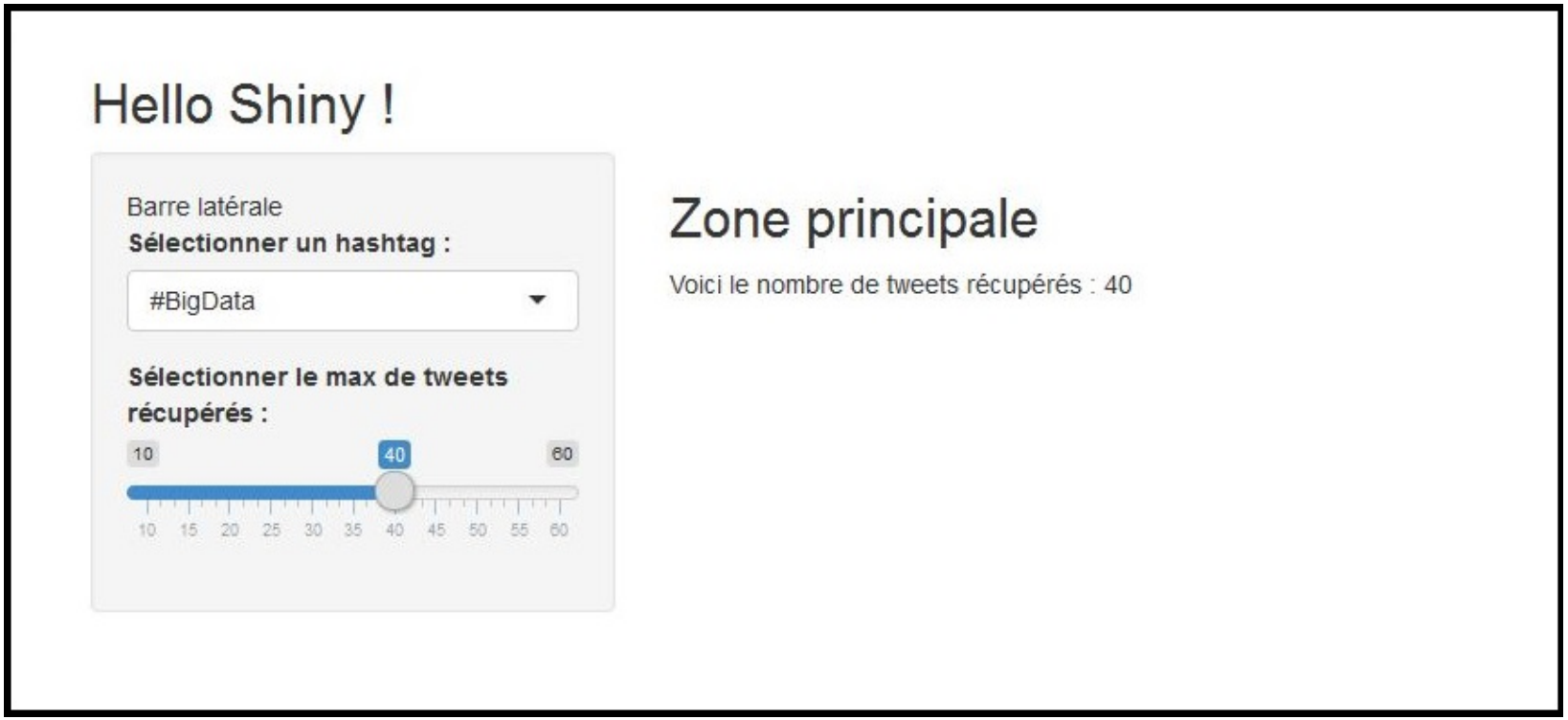
Shiny : Correction “ui.R”

```
library(shiny)

shinyUI(fluidPage(

  titlePanel("Hello Shiny !"),
  sidebarLayout(
    sidebarPanel("Barre latérale",

                 selectInput(inputId = "mes_hashtags",
                             label = "Sélectionner un hashtag :",
                             choices = list("#BigData" = "#BigData",
                                            "#DataScience" = "#DataScience",
                                            "#MachineLearning" = "#MachineLearning"),
                             selected = 1),

                 sliderInput(inputId = "nombre_tweets", label = "Sélectionner le max de tweets récupérés :",
                             min = 10, max = 60, value = 30)

    ),
    mainPanel(h2("Zone principale"),
              textOutput("mon_resultat")
    )
  )
))
```

Shiny : Correction “server.R”

```
library(shiny)

shinyServer(function(input, output) {

  mes_tweets <- reactive({

    tweets <- searchTwitter(input$mes_hashtags, n = input$nombre_tweets)
    tweets_df <- twListToDF(tweets)
    return(tweets_df)

  })

  output$mon_resultat <- renderText({

    ma_valeur <- dim(mes_tweets())[1]
    paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

})
```

Shiny : Correction “rendu”



Shiny : zone principale avec plusieurs onglets

Il peut être intéressant de séparer la “zone principale” en plusieurs parties, par exemple : texte, tableau, graphique, etc.

C’est ce que nous allons voir maintenant.

Shiny : ui.R

```
library(shiny)

shinyUI(fluidPage(

  titlePanel("Hello Shiny !"),
  sidebarLayout(
    sidebarPanel("Barre latérale",

                 selectInput(inputId = "mes_hashtags",
                             label = "Sélectionner un hashtag :",
                             choices = list("#BigData" = "#BigData",
                                            "#DataScience" = "#DataScience",
                                            "#MachineLearning" = "#MachineLearning"),
                             selected = 1),

                 sliderInput(inputId = "nombre_tweets", label = "Sélectionner le max de tweets récupérés :",
                             min = 10, max = 60, value = 30)

    ),
    mainPanel(h2("Zone principale"),
              tabsetPanel(id = "mes_onglets",
                          tabPanel("Texte", textOutput("mon_resultat")),
                          tabPanel("Tableau", ""),
                          tabPanel("Graphique", "")
              )
    )
  )
))
```

Shiny : rendu


Hello Shiny !
Barre latérale
Sélectionner un hashtag :
#BigData
Sélectionner le max de tweets récupérés :
10
30
60
10 15 20 25 30 35 40 45 50 55 60
Zone principale
Texte
Tableau
Graphique
Voici le nombre de tweets récupérés : 30

Shiny : affichage des tweets récupérés dans un tableau

On va maintenant afficher les différents tweets récupérés au sein d'un tableau (dans notre onglet "Tableau").

Pour cela, on va utiliser le package "**DT**" (qui repose sur la librairie JavaScript "DataTables").

Shiny : ui.R

Côté UI, pas grand chose de nouveau, si ce n'est qu'au lieu d'utiliser "*textOutput*" pour afficher notre sortie, on va utiliser "*dataTableOutput*".

Attention : on n'oublie pas aussi de faire appel au package au début de notre script !

Plus d'infos sur le package "DT" et son utilisation : RStudio DT Github.

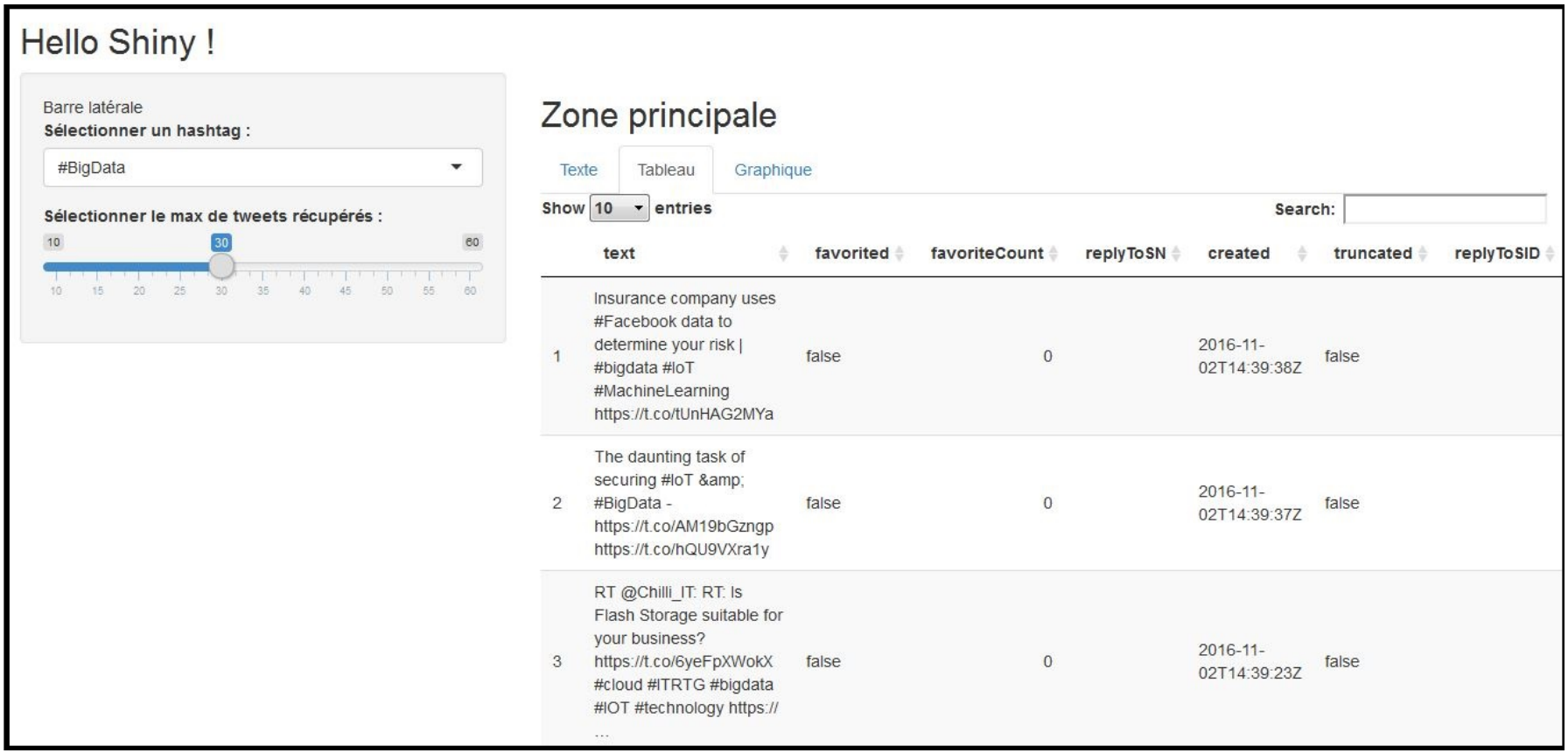
Shiny : ui.R

```
library(shiny)
library(DT)

shinyUI(fluidPage(

  titlePanel("Hello Shiny !"),
  sidebarLayout(
    sidebarPanel("Barre latérale",

                 selectInput(inputId = "mes_hashtags",
                             label = "Sélectionner un hashtag :",
                             choices = list("#BigData" = "#BigData",
                                            "#DataScience" = "#DataScience",
                                            "#MachineLearning" = "#MachineLearning"),
                             selected = 1),

                 sliderInput(inputId = "nombre_tweets", label = "Sélectionner le max de tweets récupérés :",
                             min = 10, max = 60, value = 30)

    ),
    mainPanel(h2("Zone principale"),
              tabsetPanel(id = "mes_onglets",
                          tabPanel("Texte", textOutput("mon_resultat")),
                          tabPanel("Tableau", dataTableOutput("mon_tableau")),
                          tabPanel("Graphique", "")
              )
    )
  )
))
```

Shiny : server.R

```
library(shiny)
library(DT)

shinyServer(function(input, output) {

  mes_tweets <- reactive({

    tweets <- searchTwitter(input$mes_hashtags, n = input$nombre_tweets)
    tweets_df <- twListToDF(tweets)
    return(tweets_df)

  })

  output$mon_resultat <- renderText({

    ma_valeur <- dim(mes_tweets())[1]
    paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

  output$mon_tableau <- renderDataTable({

    datatable(mes_tweets())

  })

})
```

Shiny : Résultats



=> Pas très beau visuellement… On va faire quelques modifications : sélection de variables dans nos données et ajout de paramètres de personnalisation de la fonction “datatable”.

Shiny : sélection de variables

On ne va, par exemple, afficher dans notre tableau que les variables suivantes :

- 1 = “text” : contenu du tweet
- 3 = “favoriteCount” : nombre de fois où ce tweet a été mis en favori
- 5 = “created” : date et heure de publication
- 8 = “id” : identifiant unique du compte Twitter d’où le tweet est originaire
- 12 = “retweetCount” : nombre de fois où ce tweet a été retweeté
- 13 = “isRetweet” : est-ce que ce tweet est un “retweet” (vrai / faux)

Exemple de script pour sélectionner des colonnes dans un *dataframe* en R :

```
mes_donnees[, c(1,4:6)]

mes_donnees[, c("var1", "var4")]
```

Shiny : server.R c'est à vous !

```
library(shiny)
library(DT)

shinyServer(function(input, output) {

  mes_tweets <- reactive({

    tweets <- searchTwitter(input$mes_hashtags, n = input$nombre_tweets)
    tweets_df <- twListToDF(tweets)
    return(tweets_df)

  })

  output$mon_resultat <- renderText({

    ma_valeur <- dim(mes_tweets())[1]
    paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

  output$mon_tableau <- renderDataTable({

    datatable(...) # à compléter

  })

})
```

Shiny : server.R correction

```
library(shiny)
library(DT)

shinyServer(function(input, output) {

  mes_tweets <- reactive({

    tweets <- searchTwitter(input$mes_hashtags, n = input$nombre_tweets)
    tweets_df <- twListToDF(tweets)
    return(tweets_df)

  })

  output$mon_resultat <- renderText({

    ma_valeur <- dim(mes_tweets())[1]
    paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

  output$mon_tableau <- renderDataTable({

    datatable(mes_tweets()[, c(1, 3, 5, 8, 12, 13)])

  })

})
```

Shiny : paramètres de “datatable”

Il est possible de personnaliser l’affichage en sortie de “datatable” via un certain nombre de paramètres, tels que par exemple :

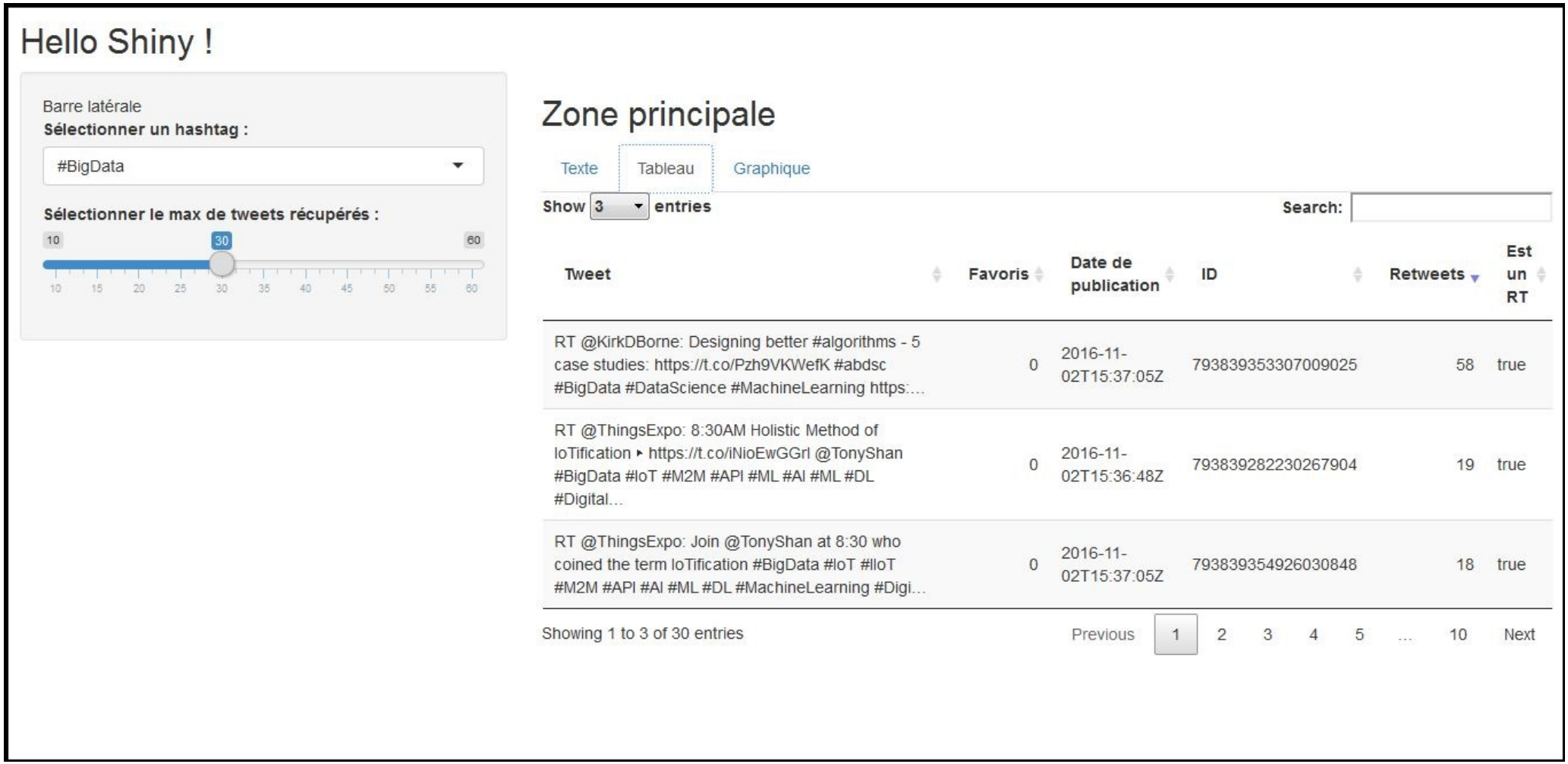
- **rownames** : mis à *FALSE* permet d’éviter l’affichage par défaut des numéros de lignes, sinon permet de renommer les lignes
- **colnames** : pour renommer les colonnes
- **class** : pour modifier le style (CSS), voir la documentation pour plus de détails
- **option = list(…)** : pour spécifier d’autres types d’options :
  - ***pageLength = 5*** *: nombre de lignes affichées par page*
  - ***lengthMenu = c(5, 10, 20)*** *: valeurs possibles pour nombre de lignes affichées par page, disponibles dans le menu déroulant*
  - ***order = list(2, “asc”)*** *: tri de la 2nde colonne par ordre croissant (décroissant : “desc”)*

Plus de détails ici.

Shiny : paramètres de “datatable” exemple server.R

```
library(shiny)
library(DT)

shinyServer(function(input, output) {

  mes_tweets <- reactive({

    tweets <- searchTwitter(input$mes_hashtags, n = input$nombre_tweets)
    tweets_df <- twListToDF(tweets)
    return(tweets_df)

  })

  output$mon_resultat <- renderText({

    ma_valeur <- dim(mes_tweets())[1]
    paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

  output$mon_tableau <- renderDataTable({

    datatable(mes_tweets()[, c(1, 3, 5, 8, 12, 13)],
              rownames = FALSE,
              colnames= c("Tweet", "Favoris", "Date de publication", "ID", "Retweets", "Est un RT"),
              options = list(
                pageLength = 3,
                order = list(4, "desc")
              ))

  })

})
```

Shiny : paramètres de “datatable” RENDU



Shiny : graphiques dynamiques

Un autre intérêt de “shiny” est la possibilité d’afficher tout un tas de graphiques différents, notamment des graphiques dynamiques (à l’aide de packages reposant sur des librairies JavaScript).

Nous allons voir ici quelques exemples de graphiques réalisés avec le package “**rAmCharts**”.

Shiny : ui.R

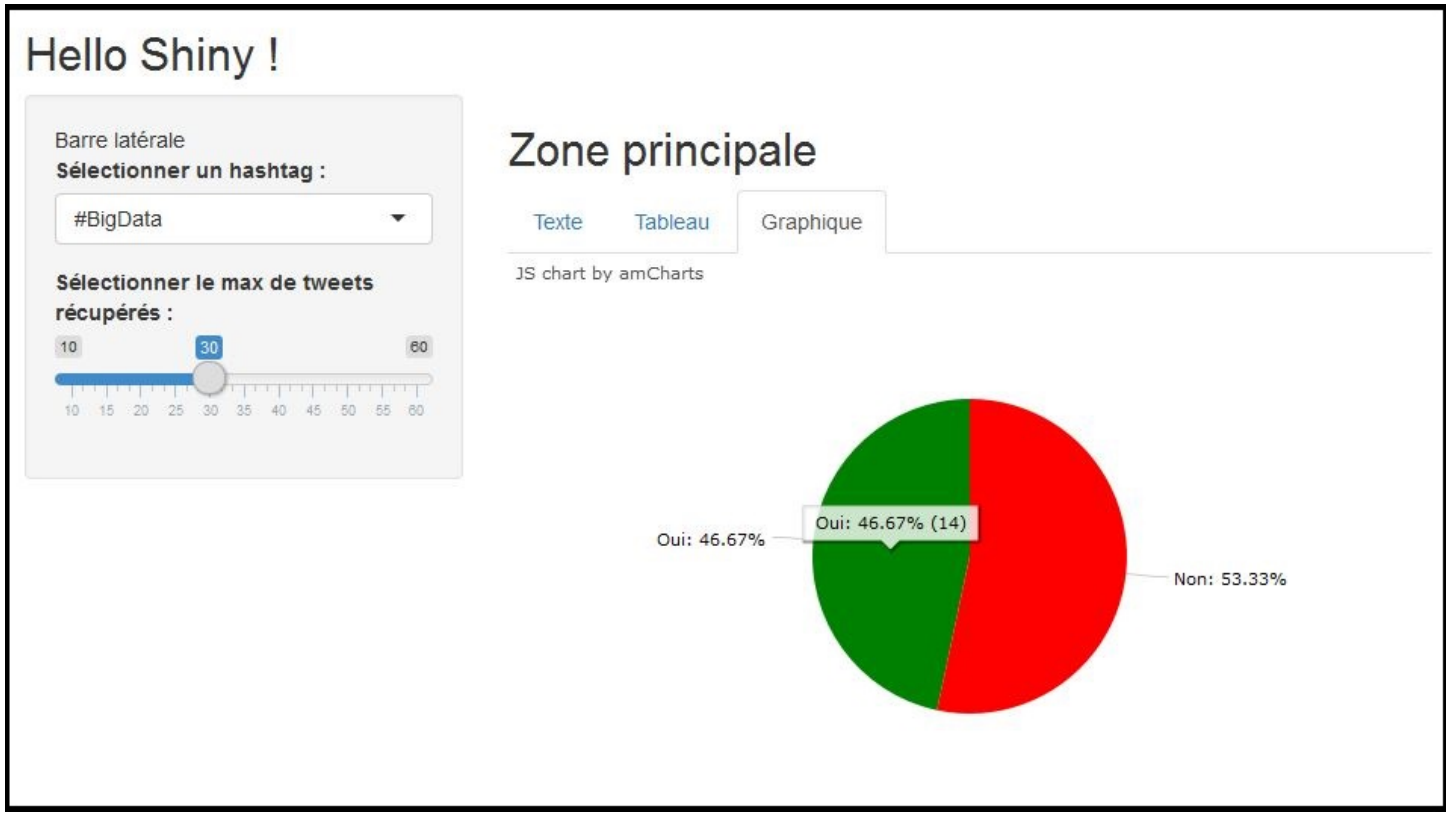
```
library(shiny)
library(DT)
library(rAmCharts)

shinyUI(fluidPage(

  titlePanel("Hello Shiny !"),
  sidebarLayout(
    sidebarPanel("Barre latérale",

                 selectInput(inputId = "mes_hashtags",
                             label = "Sélectionner un hashtag :",
                             choices = list("#BigData" = "#BigData",
                                            "#DataScience" = "#DataScience",
                                            "#MachineLearning" = "#MachineLearning"),
                             selected = 1),

                 sliderInput(inputId = "nombre_tweets", label = "Sélectionner le max de tweets récupérés :",
                             min = 10, max = 60, value = 30)

    ),
    mainPanel(h2("Zone principale"),
              tabsetPanel(id = "mes_onglets",
                          tabPanel("Texte", textOutput("mon_resultat")),
                          tabPanel("Tableau", dataTableOutput("mon_tableau")),
                          tabPanel("Graphique", amChartsOutput("mon_graphique"))
              )


    )
  )
))
```

Shiny : server.R “pie chart”

```
library(shiny)
library(DT)
library(rAmCharts)

shinyServer(function(input, output) {

  mes_tweets <- reactive({

    tweets <- searchTwitter(input$mes_hashtags, n = input$nombre_tweets)
    tweets_df <- twListToDF(tweets)
    return(tweets_df)

  })

  output$mon_resultat <- renderText({

    ma_valeur <- dim(mes_tweets())[1]
    paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

  output$mon_tableau <- renderDataTable({

    datatable(mes_tweets()[, c(1, 3, 5, 8, 12, 13)],
              rownames = FALSE,
              colnames= c("Tweet", "Favoris", "Date de publication", "ID", "Retweets", "Est un RT"),
              options = list(
                pageLength = 3,
                order = list(4, "desc")
              ))

  })

  output$mon_graphique <- renderAmCharts({
```

```
    RT_dich <- mes_tweets()[, "isRetweet"]
    RT_data <- data.frame(label = c("Non", "Oui"),
                          value = as.numeric(table(RT_dich)),
                          color = c("red", "green"))

    amPie(data = RT_data)

  })

})
```

Shiny : “pie chart” RENDU

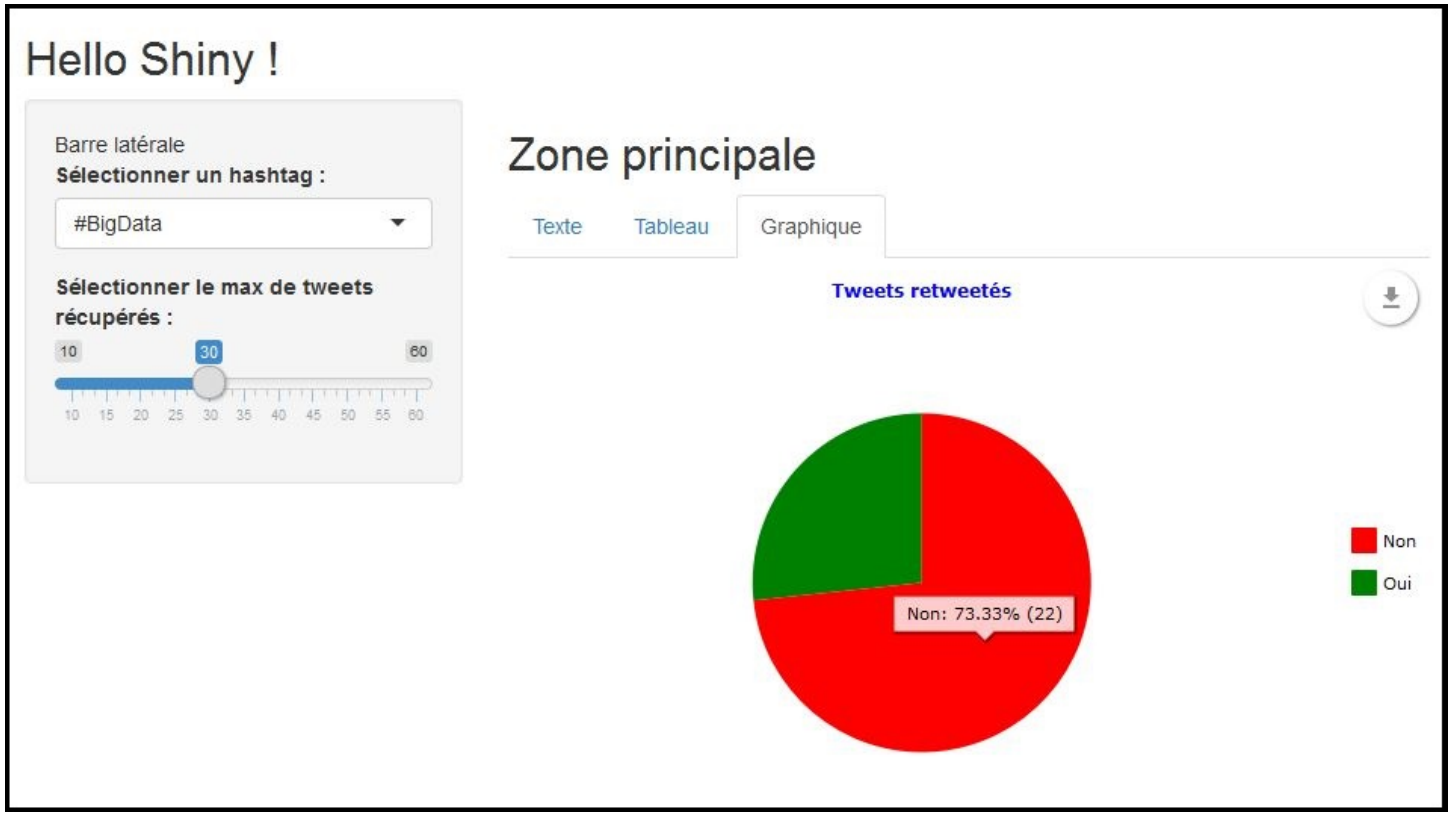


Shiny : server.R “pie chart” plus de paramètres

```
library(shiny)
library(DT)
library(rAmCharts)
library(tidyr) # ce package permet de disposer du "pipe" R : %>% , qui va nous être utile

shinyServer(function(input, output) {

  mes_tweets <- reactive({

    tweets <- searchTwitter(input$mes_hashtags, n = input$nombre_tweets)
    tweets_df <- twListToDF(tweets)
    return(tweets_df)

  })

  output$mon_resultat <- renderText({

    ma_valeur <- dim(mes_tweets())[1]
    paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

  output$mon_tableau <- renderDataTable({

    datatable(mes_tweets()[, c(1, 3, 5, 8, 12, 13)],
              rownames = FALSE,
              colnames= c("Tweet", "Favoris", "Date de publication", "ID", "Retweets", "Est un RT"),
              options = list(
                pageLength = 3,
                order = list(4, "desc")
              ))

  })
```

```
  output$mon_graphique <- renderAmCharts({

    RT_dich <- mes_tweets()[, "isRetweet"]
    RT_data <- data.frame(label = c("Non", "Oui"),
                          value = as.numeric(table(RT_dich)),
                          color = c("red", "green"))

    amPie(data = RT_data,
          export = TRUE, # active les options d'export
          creditsPosition = "bottom-left", # place des "credits" dans un des quatre coins de l'image
          show_values = FALSE, # affichage (ou non) des valeurs en dehors du survol
          legend = TRUE) %>%
      addTitle(text = "Tweets retweetés", size = 12, color = "#0000ff ") %>% # paramètres du titre
      setLegend(valueText = "", position = "right") # paramètres de la légende

  })

})
```

Shiny : “pie chart” RENDU 2



Ressources pour “rAmCharts”

- Documentation “rAmCharts”
- Github “rAmCharts”
- Lancer la commande suivante pour des exemples interactifs directement dans R :

```
runExamples()
```

- Librairie JS “AmCharts” : exemples de graphiques
- Documentation librairie JS “AmCharts”
- Interface en ligne pour créer/modifier des graphiques “AmCharts”

Shiny : exercice

A présent, on va remplacer le camembert réalisé par un autre type de graphique, par exemple : un graphique en bâtons des nombres de tweets retweetés ou non.

Pour cela, commentez les lignes relatives au camembert dans le script “server.R” (plutôt que de les supprimer).

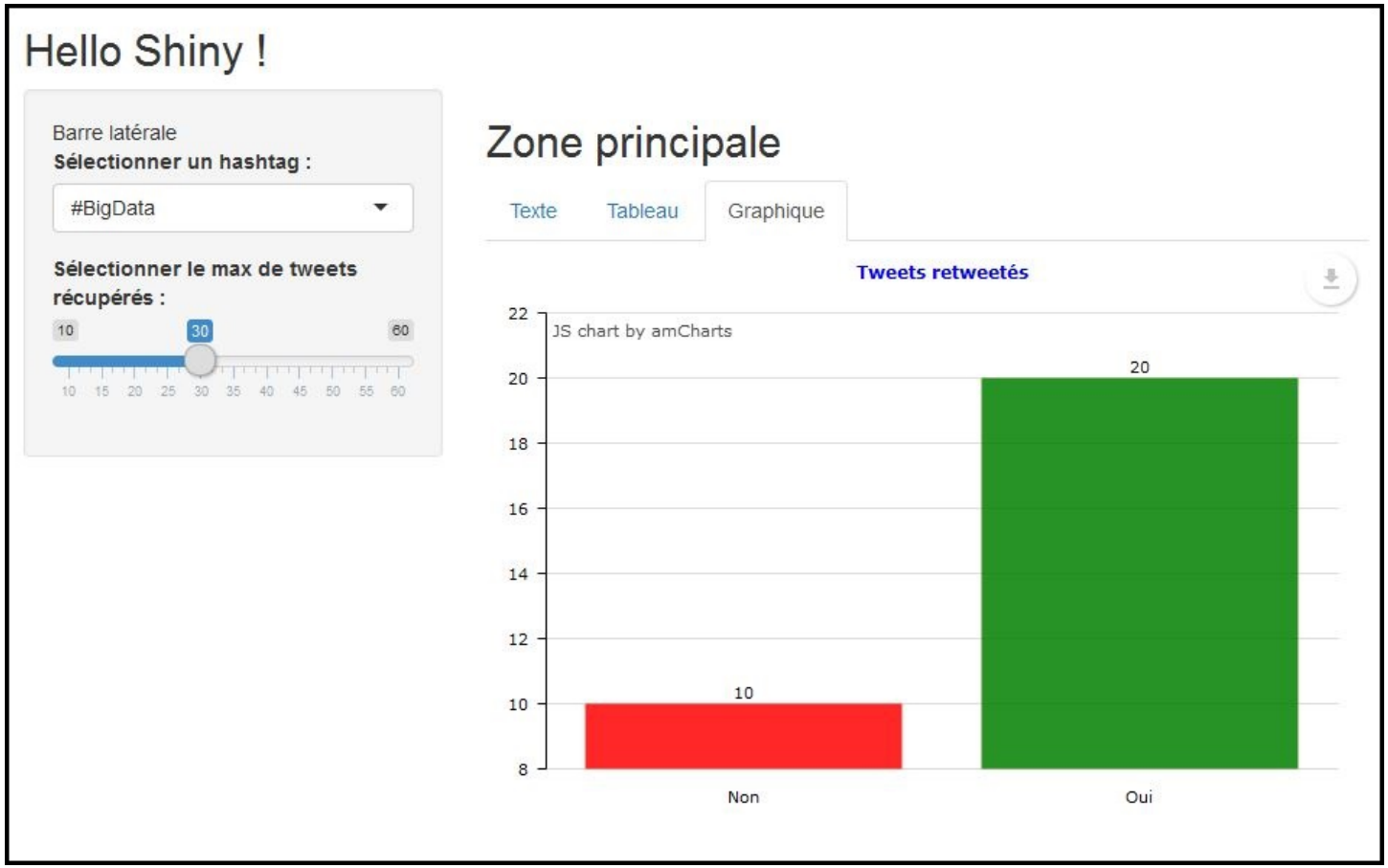
Pour voir la syntaxe “rAmCharts” d’un graphique en bâtons (ou “barplot”), c’est par exemple : ici.

Shiny : server.R correction (exemple)

```
library(shiny)
library(DT)
library(rAmCharts)
library(tidyr)

shinyServer(function(input, output) {

  mes_tweets <- reactive({

    tweets <- searchTwitter(input$mes_hashtags, n = input$nombre_tweets)
    tweets_df <- twListToDF(tweets)
    return(tweets_df)

  })

  output$mon_resultat <- renderText({

    ma_valeur <- dim(mes_tweets())[1]
    paste0("Voici le nombre de tweets récupérés : ", ma_valeur)

  })

  output$mon_tableau <- renderDataTable({

    datatable(mes_tweets()[, c(1, 3, 5, 8, 12, 13)],
              rownames = FALSE,
              colnames= c("Tweet", "Favoris", "Date de publication", "ID", "Retweets", "Est un RT"),
              options = list(
                pageLength = 3,
                order = list(4, "desc")
              ))

  })
```

```r
  output$mon_graphique <- renderAmCharts({

    RT_dich <- mes_tweets()[, "isRetweet"]
    RT_data <- data.frame(label = c("Non", "Oui"),
                          value = as.numeric(table(RT_dich)),
                          color = c("red", "green"))

    # amPie(data = RT_data,
    #       export = TRUE,
    #       creditsPosition = "bottom-left",
    #       show_values = FALSE,
    #       legend = TRUE) %>%
    #   addTitle(text = "Tweets retweetés", size = 12, color = "#0000ff ") %>%
    #   setLegend(valueText = "", position = "right")

    amBarplot(x = "label", y = "value", data = RT_data,
              show_values = TRUE,
              export = TRUE) %>%
      addTitle(text = "Tweets retweetés", size = 12, color = "#0000ff ")

  })

})
```

Shiny : correction (exemple) RENDU



Shiny : wordcloud des tweets

A présent, on va ajouter un nouvel onglet dans lequel on va afficher un “**wordcloud**” (ou “nuage de mots”) représentant les mots les plus utilisés dans les tweets que nous avons récupérés.

Pour cela, il va falloir commencer par faire un peu de nettoyage sur le texte des tweets récupérés :

- suppression des majuscules,
- suppression des “stopwords”,
- suppression de la ponctuation,
- suppression des espaces blancs.

Puis, on comptera les occurences des mots apparaissant dans ces tweets et nous représenterons les plus fréquents sous forme de “wordcloud”.

Shiny : ajout de l'onglet supplémentaire "ui.R"

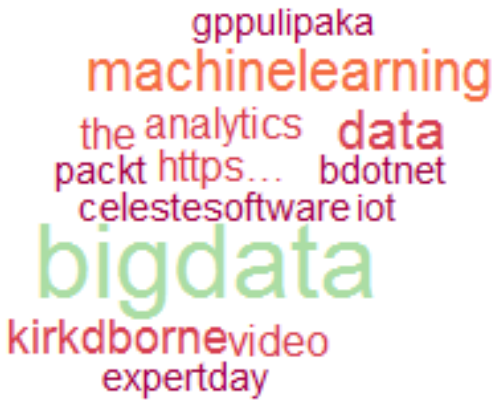
```
library(shiny)
library(DT)
library(rAmCharts)

shinyUI(fluidPage(

  titlePanel("Hello Shiny !"),
  sidebarLayout(
    sidebarPanel("Barre latérale",

                 selectInput(inputId = "mes_hashtags",
                             label = "Sélectionner un hashtag :",
                             choices = list("#BigData" = "#BigData",
                                            "#DataScience" = "#DataScience",
                                            "#MachineLearning" = "#MachineLearning"),
                             selected = 1),

                 sliderInput(inputId = "nombre_tweets", label = "Sélectionner le max de tweets récupérés :",
                             min = 10, max = 60, value = 30)

    ),
    mainPanel(h2("Zone principale"),
              tabsetPanel(id = "mes_onglets",
                          tabPanel("Texte", textOutput("mon_resultat")),
                          tabPanel("Tableau", dataTableOutput("mon_tableau")),
                          tabPanel("Graphique", amChartsOutput("mon_graphique")),
                          tabPanel("Wordcloud", plotOutput("mon_wordcloud"))
              )


    )
  )
))
```

Shiny : nettoyage du texte des tweets

```
tweets <- searchTwitter("#DataScience", n = 50)
tweets_df <- twListToDF(tweets)

library(tm)

text_corpus <- Corpus(VectorSource(tweets_df$text))

# Tout en minuscule :
text_corpus <- tm_map(text_corpus, content_transformer(tolower))
# Suppression des stopwords anglais :
text_corpus <- tm_map(text_corpus, function(x) removeWords(x,stopwords("en")))
# Suppression des stopwords français :
text_corpus <- tm_map(text_corpus, function(x) removeWords(x,stopwords("fr")))
# Suppression des caractères de ponctuation :
text_corpus <- tm_map(text_corpus, removePunctuation)
# Suppression des espaces blancs :
text_corpus <- tm_map(text_corpus, stripWhitespace)
```

Shiny : réalisation du wordcloud

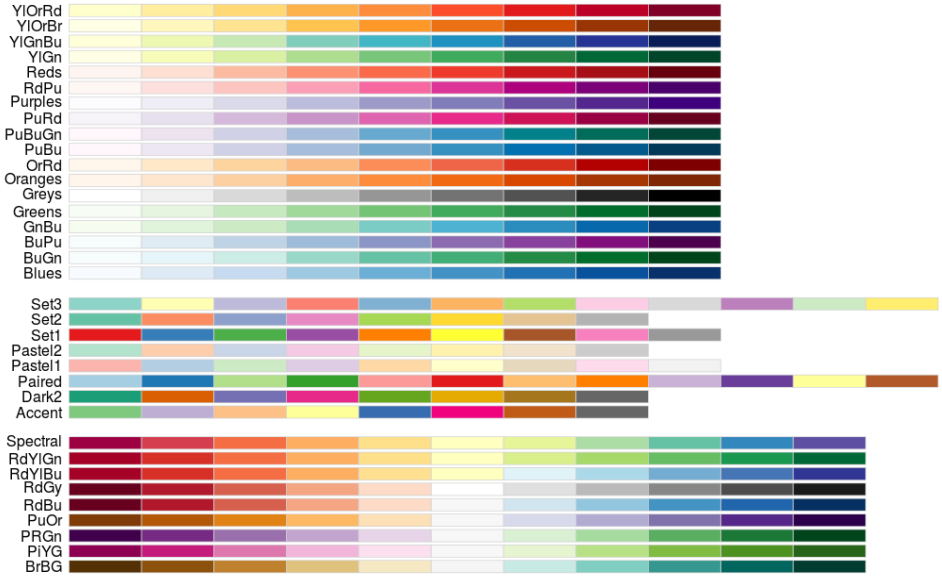
```
library(wordcloud)

wordcloud(text_corpus, max.words = 15, min.freq = 3, colors = brewer.pal(11, "Spectral")[-(5:7)])
```



Shiny : Remarques sur la sélection des couleurs du wordcloud

Pour générer un dégradé de couleurs de manière automatique, nous avons fait appel à la fonction "*brewer.pal*" du package "*RColorBrewer*" (chargé automatiquement avec le package "*wordcloud*").

Cette fonction permet de générer automatiquement une palette de couleurs parmis un certain nombre de palettes disponibles :



Shiny : mise en place dans le script “server.R”

```
output$mon_wordcloud <- renderPlot({

    tweets_rec <- mes_tweets()

    text_corpus <- Corpus(VectorSource(tweets_rec$text))

    text_corpus <- tm_map(text_corpus, content_transformer(tolower))
    text_corpus <- tm_map(text_corpus, function(x) removeWords(x,stopwords("en")))
    text_corpus <- tm_map(text_corpus, function(x) removeWords(x,stopwords("fr")))
    text_corpus <- tm_map(text_corpus, removePunctuation)
    text_corpus <- tm_map(text_corpus, stripWhitespace)

    wordcloud(text_corpus, max.words = 15, min.freq = 3, colors = brewer.pal(11, "Spectral")[-(5:7)])

})
```

Attention de ne pas oublier de spécifier les packages utilisés au début du script :

```
library(tm)
library(wordcloud)

shinyServer(function(input, output) {

(...)
```

Shiny : ajout du wordcloud sur les tweets rendus

Shiny : personnalisation du design

Dans la mesure où l'application que nous venons de créer avec "shiny" est une application web, le design peut être entièrement personnalisé à l'aide d'un script CSS, inclus dans l'application par exemple comme suit :

```
tags$head(
  tags$link(rel = "stylesheet", type = "text/css", href = "css/custom.css")
)
```

Sans avoir besoin d'aller aussi loin, il existe sinon le package "shinythemes" qui permet de changer le thème de son application "shiny" parmis une liste de thèmes prédéfinis (voir ici).

Shiny : personnalisation du design avec “shinythemes”

Pour utiliser un des thèmes présents dans le package “shinythemes”, il suffit d’y faire appel comme suit dans le fichier “ui.R” :

```
library(shiny)
library(DT)
library(rAmCharts)
library(shinythemes)

shinyUI(fluidPage(theme = shinytheme("superhero"),

  (...)
```

Déploiement

Tout ce que nous avons fait jusqu'à présent était en *local*.

Si vous souhaitez partager votre appli "shiny" avec le reste du monde, il va falloir la déployer sur le web ^^

Pour cela, **deux options** :

1/ Utiliser le service proposé par RStudio : shinyapps.io

2/ Installer soi-même un serveur Shiny

Déploiement : 1/ Shinyapps.io

- Se créer un compte sur http://www.shinyapps.io/
- Se connecter à son compte et aller dans la partie “Tokens” (menu déroulant en haut à droite)
- Récupérer ses identifiants secrets : “Show” > “Copy to clipboard”
- Exécuter dans R la commande copiée
- Puis pour déployer votre application :

```
deployApp("mon_app")
```

=> Offre gratuite jusqu’à 5 applications Shiny actives.

Documentation complète disponible : ici

Déploiement : 2/ Serveur Shiny

- Télécharger la version “Shiny Server” correspondant à votre OS ici
- Suivre les instructions d’installation fournies en fonction de votre OS (sur la même page)
- Consulter le guide d’installation et d’administration Shiny pour gérer et paramétrer votre application

Pour plus de détails, voir la documentation complète.

Ressources

Aide :

- Tutoriel officiel Shiny
- Google Group “shiny-discuss”
- Forum “Développez.com” dédié à R

Inspiration et exemples :

- Galerie Shiny
- “Shiny user showcase”
- “Show me Shiny”

Pour aller plus loin…

- Le package “shinydashboard”
- htmlwidgets for R