# Exposé de R : Le Package gtsummary

Groupe 9 : Fatou DIOP (ISE-Math), Laurine ADOGOUN (ISE-Eco), Daniel KPEKPASSI (ISEP3)

2023-07-04

## Contents

# I- PRESENTATION

## 1- Description et utilité

Le package {gtsummary} offre un moyen élégant et flexible de créer des tableaux analytiques et récapitulatifs prêts à être publiés à l'aide du langage de programmation R. Le package {gtsummary} résume les ensembles de données, les modèles de régression, etc., en utilisant des valeurs par défaut judicieuses avec des fonctionnalités hautement personnalisables. Gtsummary permets de réaliser des tableaux statistiques combinant plusieurs variables, l'affichage des résultats pouvant dépendre du type de variables, en incluant des statistiques (moyenne, médiane,... dépendamment des besoins de l'utilisateurs) et même de combiner plusieurs tableaux.

## 2- Différences avec d'autres packages similaires

*gtsummary :* il est conçu spécifiquement pour créer des tables de résumé à partir de données de recherche en sciences sociales, biomédicales et en santé publique. Il utilise le framework "Grammar of Graphics" pour fournir des tableaux de synthèse de haute qualité et offre de nombreuses fonctionnalités de personnalisation. gtsummary est étroitement intégré à dplyr et tidyr pour faciliter la manipulation des données et est optimisé pour être utilisé avec d'autres packages de la famille tidyverse. Il fait à la fois des résumés simples, de modèles et des résumés plus complexe en joignant plusieurs tableaux.

*stargazer :* stargazer est principalement utilisé pour créer des tables de résumé pour des modèles statistiques, tels que des régressions linéaires ou des modèles de survie. Il est optimisé pour fournir des sorties formatées pour les articles académiques et offre des options de personnalisation pour formater les tableaux selon les besoins spécifiques. stargazer prend en charge une variété de modèles statistiques et peut créer des tables de résumé pour plusieurs modèles simultanément.

*Janitor :* Janitor est principalement axé sur le nettoyage et la préparation des données, en fournissant des fonctions pour la gestion des valeurs manquantes, le formatage des noms de colonnes, la conversion des types de données, etc. Il offre des fonctions pour effectuer des tâches courantes de nettoyage de données, telles que la suppression des duplications, la conversion des données factorielles en variables numériques, etc. D'autre part, Janitor est principalement utilisé pour nettoyer et préparer les données en fournissant des fonctions pour simplifier les tâches courantes de nettoyage.

## 3- Installation du package gtsummary

- Installer simplement dans R

```
install.packages("gtsummary")
```

- Installer la version de développement de {gtsummary} depuis github

```
remotes::install_github("ddsjoberg/gtsummary")
```

- appeler le package pour usagee

```
library(gtsummary)

'install.packages("rlang")
install.packages("pak")
pak::pkg_install("r-lib/rlang")'#Si requis pendant l'installation
```

# II- TABLES

Dans cette partie, nous travaillerons sur une bases de données portant sur un programme d'insertion professionnelle. Elle comporte des informations aussi bien qualitatives que quantitatives sur un ensemble d'environ 9000 inividus. Nous montrerons à partir de cette base comment produire des tables de résumé simples, complexes (combinaison de nombreuses tables), de modèles de régressions ; comment les personnaliser et de les exporter pour utilisation.

## 1- Création de Tables

### 1.1- Base

.

```
##  [1] "diplome_r" "Delai_r"   "sexe_r"    "sexe_f_r"  "StatutP_r" "StatutM_r"
##  [7] "Conseil_r" "Diff2_r"   "Exppro_r"  "Abs1_r"    "Abs2_r"
```

### 1.2- Tables de résumé simples

#### 1.2.1- Fonction tbl_summary

.

```
library(gtsummary)
## importation des données
library(readr)
BaseUtile_r <- read_csv("BaseProjet_R.csv")

##### Types de variables
binary <- BaseUtile_r[c("sexe_f_r", "Delai_r","sexe_r")]
multicoto <- BaseUtile_r[c("diplome_r", "Diff2_r", "StatutM_r", "StatutP_r", "Conseil_r")]
continu <- BaseUtile_r[c("Exppro_r", "Abs1_r", "Abs2_r")]

# Summary

binary %>% tbl_summary() #table de résumé pour les variables binaires

multicoto %>% tbl_summary() #table de résumé pour les variables multicotomiques

continu %>% tbl_summary() #table de résumé pour les variables continue
```

#### 1.2.2- Jouons avec les paramètres

.

```
BaseUtile_r %>%
dplyr::select(sexe_r, diplome_r, Diff2_r, Abs1_r, Abs2_r, Delai_r) %>%
gtsummary::tbl_summary(
  ## paramètres de tbl_summary

  by = Delai_r, ## variables qui forme les groupes

  label = list(diplome_r ~ "diplome le plus eleve",
```

```
              Diff2_r ~ "difficulte d_integration au
              debut", sexe_r ~ "sexe", Abs1_r ~ "absence aux rencontres",
              Abs2_r ~ "absence
              aux entretiens"), ## labélisation des variables dans le tableau

  percent = "column", ## Type de pourcentage affichés dans le tableau

  ## nombre de chiffre après la virgule pour les résultats des variables précisées
  digits = list(Abs1_r ~ 2, Abs2_r ~ 3),

  statistic = list(Abs1_r ~ "{mean} ({sd})",

                   all_continuous2() ~ c("{median} ({p25} - {p75}",
                                         "{mean} ({sd})", "{min} - {max}"),

                   sexe_r ~ "{n}/{N} ({p}%)",

                   ## type et formatage des statistics en forction du type de variable,
                   ## des variables précisées
                   Diff2_r ~ "{p} % ({n}/{N})"),

  ## modifier et préciser comment il faut considérer la variable en terme de type
  type = list(sexe_r ~ "categorical", Abs2_r ~ "continuous2"),

  missing = "always", ## afficher les stat sur les valeurs manquantes

  missing_text = "Missing", ## formatage et nomination de la variable "valeur manquante"

  ) %>%

  ## ajouter les statistiques sur la base totale (non par groupe)
  add_overall() %>%

  ## afficher la différence entre les groupes, le test de significativité de la différence
  add_difference() %>%

  ## afficher une colonne qui signifie les statistiques calculées et leur format
  ##d'affichage. Ex: mean (sd)
  add_stat_label()
```

### 1.2.3- Ajout d'unités aux résultats

.

```
BaseUtile_r %>% ## Base

## variables de la base utiles
dplyr::select(sexe_r, Diff2_r, Abs1_r, Abs2_r, Delai_r) %>%

gtsummary::tbl_summary(
  ## paramètres de tbl_summary

  by = Delai_r, ## formation des groupes
```

```
  label = list(Diff2_r ~ "difficulte d_integration au
              debut", sexe_r ~ "sexe", Abs1_r ~ "absence aux rencontres",
              Abs2_r ~ "absence aux entretiens"), ## intitulés dans le tableau

  ## type de proportions afficher dans le tableau
  percent = "column",

  ## appliquer aux variables continues l'affichage des pourcentages au style
  ## "style_percent" qui peut être aussi personnalisé
  digits = list(all_continuous2() ~ c(style_percent, style_pvalue, style_ratio),
                Abs1_r ~ scales::label_number(accuracy = .01, suffix = " hours",

                                              ## formatage des chiffres (nombre de
                                              ## ligne,unité...)
                                              decimal.mark = ",")),

  ## types statistiques
  statistic = list(Abs1_r ~ "{mean} ({sd})",
                   all_continuous2() ~ c("{median} ({p25} - {p75}", "{mean} ({sd})",
                                         "{min} - {max}")
                   ),

  ## spécifier le type
  type = list(sexe_r ~ "categorical", Abs2_r ~ "continuous2"),

  ## affichage de la variable "valeur manquante"
  missing_text = "Missing",
  )
```

### 1.2.4- Tableaux croisés (tbl_cross)

.

```
library(gtsummary)
library(dplyr)
BaseUtile_r |>
  gtsummary::tbl_cross(
    row =  sexe_r,
    col = Delai_r,
    percent = "row")|>
  ## ajouter les p-value en précisant le test réalisé
  add_p(source_note = TRUE)
```

| | Dificile | Facile | Total |
|---|---|---|---|
| sexe_r | | | |
| Femme | 1,180 (27%) | 3,141 (73%) | 4,321 (100%) |
| Homme | 1,065 (23%) | 3,496 (77%) | 4,561 (100%) |
| Total | 2,245 (25%) | 6,637 (75%) | 8,882 (100%) |

```
BaseUtile_r |>
  gtsummary::tbl_cross(
```

```
    row =  Diff2_r,
    col = Delai_r,
    percent =  "column" ## il y a la possibilité d'utiliser (cell) ou (row)
    ) |> add_p(source_note = TRUE)
```

| | Dificile | Facile | Total |
|---|---|---|---|
| Diff2_r | | | |
| Plutot dificile | 1,095 (49%) | 1,828 (28%) | 2,923 (33%) |
| Plutot facile | 678 (30%) | 4,039 (61%) | 4,717 (53%) |
| Tres dificile | 419 (19%) | 275 (4.1%) | 694 (7.8%) |
| Tres facile | 53 (2.4%) | 495 (7.5%) | 548 (6.2%) |
| Total | 2,245 (100%) | 6,637 (100%) | 8,882 (100%) |

### 1.2.5- Paramètres utiles

.

```
BaseUtile_r %>%
  gtsummary::tbl_summary(

    ##variables à representer dans le tableau
    include = c(sexe_r, Diff2_r, Abs1_r, Abs2_r, Delai_r),

    ## stat associée à ces variables
    statistic = c(Abs1_r, Abs2_r) ~ "{mean} ({sd})",

    by = Delai_r ## groupes
  ) %>%

  add_stat_label() %>% ##

  ## p-value et test de p-value associé à ces variables pour éviter le test
  ## par défaut associé au type de la variable concernée
  add_p(
    test = list(
      sexe_r ~ "fisher.test",
      c(Abs1_r, Abs2_r) ~ "t.test"
    ),

    ## personalisation de l'affichage de la sortie de la p-value
    pvalue_fun = scales::label_pvalue(accuracy = .0001)) %>%

  separate_p_footnotes() ## note de bas de tableau
```

### 1.2.6- Fonction tbl_continuous

La fonction tbl_continuous fournit les statistiques relatives à une variable (continue) par groupe (by=...) et suivant d'autres variables.

```
BaseUtile_r %>%
  gtsummary::tbl_continuous(
```

```
    ## variable à représenter
    variable = Abs2_r,

    ## stat à  calculer
    statistic = ~"{mean}, {sd}",

    ## variables pour lesquelles ou suivant lesquelles on calcul
    include = c(sexe_r, Diff2_r),

    by = Delai_r, ## groupe

    digits = ~1 ## formatage des chiffres
  )
```

#### 1.2.7- Paramétrage complexe

`tbl_custom_summary()` : cette fonction permet encore plus de personnaliser les options de tableau que `tbl_continuous`.

*tbl_custom_summary() & proportion_summary()*

```
BaseUtile_r %>%
  gtsummary::tbl_custom_summary(
    include = c(Diff2_r, sexe_r, Abs1_r),

    ## fonction à utiliser pour les calculs de stats. Ici, on présente
    ## les proportion du groupe difficile unique conditionnellement aux variables.
    stat_fns = ~ proportion_summary(variable = "Delai_r", value = "Dificile"),

    statistic = ~"{prop}% [{conf.low}-{conf.high}]", ## affichage des résultats

    digits = ~ scales::label_percent(accuracy = .1, decimal.mark = ",", suffix = "")
  )
```

.

### 1.3- Tables de résumé pour regression

#### 1.3.1- Fonction tbl_regression()

Cette fonction permet de sortir un tableau récapitulant des résultats d'un modèle de régression (Odds-ratio, p-value,...).

```
## Recodage de la variable Delai_f en variable binaire numérique
## pour l'adapter au modèle glm
BaseUtile_r$Delai_f <- ifelse(BaseUtile_r$Delai_r == "Facile", 0, 1)

# Estimation du modèle
modele_ <- stats::glm(
  Delai_f ~ Diff2_r * Abs2_r + Diff2_r*Abs1_r + sexe_r + diplome_r,
  data = BaseUtile_r,
  family = binomial
)
```

```r
# Resumé du modèle

## tables de résumé des résultats d'un modèle
modele_ %>% gtsummary::tbl_regression()
```

.

### 1.3.2- Choix des coeficients (variables / termes) à afficher

.

```r
modele_ %>%

  ## variables à inclure dans le tableau
  gtsummary::tbl_regression(include = c(all_continuous(),
                                        all_interaction(), sexe_r))
```

.

### 1.3.3- Quelques paramètres supplémentaires

.

```r
modele_ %>%
  gtsummary::tbl_regression(add_estimate_to_reference_rows = TRUE,
                            exponentiate = TRUE,
                 estimate_fun = scales::label_number(accuracy = .001,
                                                     decimal.mark = "."),
                 pvalue_fun = scales::label_pvalue(accuracy = .001,
                                                   decimal.mark = ".",
                                                   add_p = TRUE))%>%
  add_significance_stars(
    hide_ci = FALSE,
    hide_p = FALSE,
    hide_se = TRUE) %>%
  add_global_p(keep = TRUE)
```

.

### 1.3.4- Résumés graphiques

.

```r
# Représentation des odds ratio
modele_ %>%
  gtsummary::tbl_regression(exponentiate = TRUE) %>%
  plot()
```

. .

```r
# Paramétré
modele_ %>%
  GGally::ggcoef_model(exponentiate = F)
```

.

### 1.3.5- Ajouter des stat globales du modèle

.

```
modele_ %>%
  gtsummary::tbl_regression() %>%

  #permet de préciser les stats voulues contrairement à add_glance_table()
  add_glance_source_note(include = c("nobs", "AIC"))
```

.

### 1.3.6- tbl_uvregression()

La fonction tbl_uvregression est utile pour réaliser plusieurs régressions univariées. Il faut lui passer un tableau ne contenant que la variable à expliquer et les variables explicatives. La variable à expliquer sera indiquée avec y. L'argument method indique la fonction à utiliser pour le calcul des modèles univariés.

```
tbl_uni <- gtsummary::tbl_uvregression(
  BaseUtile_r %>% dplyr::select(Delai_f, Abs1_r, Diff2_r, sexe_r),
  method = glm,
  y = Delai_f,
  method.args = list(family = binomial),
  exponentiate = TRUE,
  hide_n = TRUE,
)
tbl_uni
```

Bonus

*Type survey*

```
library(haven)
Ehcvm <- haven::read_dta("welfare.dta")

Ehcvm %>% ## jeux de données avec la variable pondération
as_data_frame() %>%
  survey::svydesign(~1, data = ., weights = ~hhweight) %>% ## préciser la variable poids

  ## les tableaux de résumé et les paramèteres sont quasiment similaires à tbl_summary
  gtsummary::tbl_svysummary(
    include = c(region, milieu),
    by = hgender,
    percent = "row"
  ) %>%
  add_stat_label(location = "column") %>%

  add_n() %>% ## effectif

  add_overall(last = TRUE) %>%
  add_p() %>%
  separate_p_footnotes()
```

### 1.4- Tables de résumé complexes : combinaison, paramétrage complexe

.

#### 1.4.1- tbl_custom_summary() & écriture d'une fonction personnalisée

- Il est également possible, et c'est là toute la puissance de `tbl_custom_summary`, de définir une fonction personnelle et de la passer via `stat_fns` - Paramètres : `ma_fonction <- function(data, full_data, variable, by, type, stat_display){}`. - flexibilité : `ma_fonction <- function(data, ...){}`. `...` remplacent les autres paramètres - restiction : La fonction devra impérativement renvoyé un `tibble`

***Fonctions personnalisées***

```
## Calculer directement
ma_fonction <- function(data, ...) {
  Abs_rSum <- sum(data$Abs_r, na.rm = TRUE)
  Abs1_rMean <- mean(data$Abs1_r, na.rm = TRUE)
  dplyr::tibble(
    Abs_rSum = Abs_rSum,
    Abs1_rMean = Abs1_rMean
  )
}

## Fonction des variables du tableau
mean_ci <- function(data, variable, ...) {
  test <- t.test(data[[variable]])
  dplyr::tibble(
    mean = test$estimate,
    conf.low = test$conf.int[1],
    conf.high = test$conf.int[2]
  )
}

## Opérations avancées ~ indépendant des variables du tableau
diff_to_great_mean <- function(data, full_data, ...) {
  mean <- mean(data$Abs_r, na.rm = TRUE)
  great_mean <- mean(full_data$Abs_r, na.rm = TRUE)
  diff <- mean - great_mean
  dplyr::tibble(
    mean = mean,
    great_mean = great_mean,
    diff = diff,
    level = ifelse(diff > 0, "haut", "bas")
  )
}
```

***Tableaux générés à partir des fonctions personnalisées***

```
## ma_fonction
BaseUtile_r %>%
  gtsummary::tbl_custom_summary(
    include = c(Diff2_r, sexe_r, Abs1_r, Abs_r, Abs2_r, Exppro_r),
    by = Delai_r,
    type = list(.overall ~ 'continuous'),
```

```
    stat_fns = ~ma_fonction,
    statistic = ~ "Abs: {Abs_rSum} - Exp: {Abs1_rMean}",
    digits = everything() ~ c(0, 1),
    overall_row = TRUE
  ) %>%
  add_overall(last = TRUE) %>%
  modify_footnote() %>%
  bold_labels()

## mean_ci
BaseUtile_r %>%
  gtsummary::tbl_custom_summary(
    include = c( Abs1_r, Abs_r, Abs2_r),
    by = Delai_r,
    stat_fns = ~mean_ci,
    statistic = ~"{mean} [{conf.low}; {conf.high}]"
  ) %>%
  add_overall(last = TRUE) %>%
  modify_footnote(
    update = all_stat_cols() ~ "moyenne [IC 95%]"
  )

##diff_to_great_mean
BaseUtile_r %>%
  gtsummary::tbl_custom_summary(
    include = c( Abs1_r, Abs_r, Abs2_r),
    by = Delai_r,
    stat_fns = ~diff_to_great_mean,
    statistic = ~"{mean} ({level}, diff: {diff})",
    digits = ~ list(1, as.character, 1),
    overall_row = TRUE,
    missing = "no"
    ) %>%
  bold_labels()
```

### 1.4.2- Aggrégation de tables

- La fonction `tbl_stack()` permets de coller deux (ou plus) tableaux l'un au-dessus de l'autre
- La fonction `tbl_merge()` les placera côte-à-côte, en s'assurant qu'une même variable sera bien affichée sur la même ligne.

```
t1 <-
  glm(Delai_f ~ sexe_r, BaseUtile_r, family = binomial) %>%
  tbl_regression(exponentiate = TRUE)

t2 <-
  glm(Delai_f ~ sexe_r + diplome_r + Abs1_r, BaseUtile_r, family = binomial) %>%
  tbl_regression(exponentiate = TRUE)

## Empillement l'une sur l'autre

gtsummary::tbl_stack(
  list(t1, t2),
```

```
  ## intitulé des groupes de tableau associés
  group_header = c("Modèle bivarié", "Modèle multivarié")
)

## Agencement l'une à coté de l'autre

gtsummary::tbl_merge(
  list(t1, t2),

  ## intitulé des groupes de tableau associés
  tab_spanner = c("Modèle bivarié", "Modèle multivarié")
)
```

- La fonction `tbl_strata` permet de calculer un tableau gtsummary pour chaque modalité d'une variable catégorielle définie via strata, puis de combiner les tableaux entre eux.

```
tab <- BaseUtile_r %>%
  select(Delai_r, sexe_r, diplome_r, Abs2_r, Abs1_r) %>%
  gtsummary::tbl_strata(

    ## définition des différents groupes auxquels on va appliqué les type de
    ## tableau defini dans ".tbl_fun"
    strata = c(sexe_r, diplome_r),

    .tbl_fun = ## défini les types et formats de tableau
      ~ .x %>%
        tbl_summary(by = Delai_r, missing = "no") %>%
        add_n()
  )
tab
```

```
#with_gtsummary_theme(reset_gtsummary_theme(), tab, env = rlang::caller_env())

BaseUtile_r %>%
  select(Delai_r, sexe_r, diplome_r, Abs2_r, Abs1_r) %>%
  tbl_strata(
    strata = sexe_r,
    .tbl_fun =
      ~ .x %>%
        tbl_summary(by = Delai_r, missing = "no") %>%
        add_n(),

    ## préciser comment combiner les tableaux de chaque groupe. Par défaut,
    ## il combine avec "tbl_merge"
    .combine_with = "tbl_stack",
    .header = "*{strata}*"
  )
```

### 1.4.3- Combiner avec `tbl_stack()`

- Lorsqu'un tableau est trop long et qu'on souhaite le couper en plusieurs tableaux, on pourra utiliser `tbl_spit()` en indiquant le nom des variables après lesquelles le tableau doit être coupé.

```r
tab1 <- trial |>
  tbl_summary() |>

  ## coupe le long tableau après chacune des variables précisées en paramètre
  gtsummary::tbl_split(variables = c(marker, grade))

## Une manière de récupérer les différents tableaux formés et de les utiliser.
## Ici on les a mergé d'une autre manière.
gtsummary::tbl_merge(
  list(tab1[[1]], tab1[[2]],tab1[[3]]),
  tab_spanner = c("Parti 1", "Parti 2", "Parti 3")
)
```

## 2- Personnalisation

### 2.1- Thèmes

gtsummary fournit plusieurs fonctions préfixées `theme_gtsummary_*()` permettant de modifier l'affichage par défaut des tableaux. Ils nous permettent de définir au préalable les sorties (format) de tableau que nous désirons et ainsi éviter d'avoir à trop jouer avec les paramètres.

#### 2.1.1- Utilisation de thèmes existant

- La fonction `theme_gtsummary_journal` permets d'adopter les standards de certaines grandes revues scientifiques telles que *JAMA* (Journal of the American Medical Association), *The Lancet* ou encore le *NEJM* (New England Journal of Medicine).

```r
theme_gtsummary_journal(
  journal = "jama", #c("jama", "lancet", "nejm", "qjecon")
  set_theme = T
 )
```

.

#### 2.1.2- Autres thèmes : formatage des tableaux sortis

.

```r
## Eliminer les espaces
theme_gtsummary_compact(set_theme = TRUE, font_size = NULL)

## Format de la sortie
theme_gtsummary_printer(
  print_engine = "flextable",  #c("gt", "kable", "kable_extra", "flextable",
                               #"huxtable", "tibble"),
  set_theme = TRUE
)

## format général pour un type de variable donné
```

```r
theme_gtsummary_continuous2(
  statistic = "{median} ({p25, {p75})",
  set_theme = TRUE
)

## paramètre de langue
theme_gtsummary_language(
 #font_size = 13,
 language = "fr", #c("de"(allemand), "en", "es", "fr", "ja"(japonais),
                  #"kr"(coréen)....)
 decimal.mark = ",",
 big.mark = " ",
 iqr.sep = ";", # Inter Quartil Range
 ci.sep = ";"
)

##Définir toutes les variables continues sur le type récapitulatif "continuous2"
##par défaut
theme_gtsummary_mean_sd(set_theme = TRUE)
#Dans tbl_summary() afficher la médiane, la moyenne, l'IQR, l'écart-type et la
#plage par défaut

## Réétablir le thème initiale
reset_gtsummary_theme()
```

.

### 2.1.3- Définir son thème

.

```r
my_theme <-
  list(
    # round large p-values to two places
    "pkgwide-fn:pvalue_fun" = function(x) style_pvalue(x, digits = 2),
    "pkgwide-fn:prependpvalue_fun" = function(x) style_pvalue(x, digits = 2,
                                                              prepend_p = TRUE),
    # report median (IQR) and n (percent) as default stats in `tbl_summary()`
    "tbl_summary-str:continuous_stat" = "{median} ({p25} - {p75})",
    "tbl_summary-str:categorical_stat" = "{n} ({p})"
  )

check_gtsummary_theme(my_theme) #vérifier la structure du thème

set_gtsummary_theme(my_theme) #créer et appliquer le thème
```

.

### 2.1.4- Autres fonctions

.

```
##obtenir une liste nommée avec tous les éléments de thème actifs
#get_gtsummary_theme()

##évaluer une expression avec un thème défini temporairement
#with_gtsummary_theme()
```

### 2.2- Formatage du texte à l'intérieur

#### 2.2.1- Renommer : labéliser les variables et les modalités (avec les "label")

.

```
tbl_summary(trial, label = list(age ~ "Âge"),
            sort = all_categorical() ~ "frequency", by= trt) %>%

  add_overall(last = TRUE, col_label = "**Ensemble** (effectif total: {N})") %>%

  add_stat_label(location = "column")
```

. .

```
iris %>%

  labelled::set_variable_labels(
    Petal.Length = "Longueur du pétale",
    Petal.Width = "Largeur du pétale"
  ) %>%

  tbl_summary(label = Species ~ "Espèce") %>%

  add_n(
    statistic = "{n}/{N}",
    col_label = "**Effectifs** (observés / total)",
    last = TRUE,
    footnote = TRUE
  )
```

.

#### 2.2.2- Mettre en gras

.

```
#bold_labels, bold_levels, italicize_labels, italicize_levels

trial %>%
  tbl_summary(
    include = c(marker, grade, stage),
    by = trt
  ) %>%

   bold_labels() %>%
```

```
  italicize_levels()


trial %>%
  tbl_summary(
    include = c(age, marker, ttdeath),
    type = c(age, marker) ~ "continuous2",
    statistic = all_continuous2() ~ c("{median} ({p25} - {p75}",
                                      "{mean} ({sd})", "{min} - {max}"))%>%
  add_stat_label(location = "column")
```

.

### 2.2.3- Mettre les titres, mettre les notes de bas de page

.

```
#show_header_names(tab)
modify_header(tab,n_1 = "**Effectifs**")

modify_footnote(tab, all_stat_cols() ~ "median (Intrvall intrQuartill) ")

modify_spanning_header(tab, all_stat_cols() ~ "**Treatment Received**")

modify_caption(tab, "**Patient Characteristics** (N = {N})",
               text_interpret = "md") #c("md", "html"))
#separate_p_footnotes(tab)
```

## 3- Exportation

Pour l'exportation de vos tableaux, il faudra utiliser le package gt ou flextable combiné avec des sorties tibles de gtsummary.

```
library(gt)
tbl <- Ehcvm%>%tbl_summary(include=c(Female,Male,pauv),
                           by=pauv,percent = "row",
                           statistic=list(Male~"{p}%",Female~"{p}%")) %>%
               add_n() %>%
               as_gt()## necéssaire pour exporter avec l'option gt

##Exportation sous PDF
gtsave(tbl, filename ="tbl.pdf")

##Exportation sous HTML
gtsave(tbl, filename ="tbl.html")

tbl <- Ehcvm%>%tbl_summary(include=c(Female,Male,pauv),by=pauv,
                           percent = "row",
                           statistic=list(Male~"{p}%",Female~"{p}%")) %>%
               add_n() %>%
               as_flex_table()## necéssaire pour exporter avec l'option flextable
```

```
##Exportation sous Word
flextable::save_as_docx(tbl,path ="tbl.docx")
```

## 4- Application particulière

```
## Eliminer les espaces
theme_gtsummary_compact(set_theme = TRUE, font_size = NULL)

library(haven)
Ehcvm <- read_dta("welfare.dta")

##########Application############
Ehcvm$Female<-ifelse(Ehcvm$hgender==2,1,0)
Ehcvm$Male<-ifelse(Ehcvm$hgender==1,1,0)

Ehcvm$pauv<-factor(Ehcvm$pauv, levels= c(0,1),labels = c("Non pauvre","Pauvre"))

appli<-Ehcvm%>%tbl_summary(include=c(Female,Male,pauv,hage,hhsize),

                           by=pauv,percent = "column",

                           statistic=list(Male~"{p}%",
                                  Female~"{p}%", hage~"{mean}",
                                  hhsize~"{mean}")) %>% add_n() %>%

                add_difference(test=list( Female ~ "t.test", Male ~ "t.test"),
                               pvalue_fun = scales::label_pvalue(accuracy = .0001)
                               ) %>%

                add_overall()%>%

                as_flex_table()

appli
```

| **Characteristic** | **N** | **Overall**, N = 7,156[1] | **Non pauvre**, N = 4,911[1] | **Pauvre**, N = 2,245[1] | **Difference**[2] | **95% CI**[2,3] | **p-value**[2] |
|---|---|---|---|---|---|---|---|
| Female | 7,156 | 26% | 31% | 17% | 14% | 12%, 16% | <0.0001 |
| Male | 7,156 | 74% | 69% | 83% | -14% | -16%, -12% | <0.0001 |
| Age du CM | 7,156 | 51 | 51 | 52 | -0.34 | -1.0, 0.34 | 0.3282 |
| Taille menage | 7,156 | 9.2 | 7.8 | 12.4 | -4.6 | -4.9, -4.3 | <0.0001 |

[1]%; Mean

[2]Welch Two Sample t-test

[3]CI = Confidence Interval

```r
flextable::save_as_docx(appli,path = "appli.docx")
```

## Référence

Vous comprendrez peut-être mieux ici -> https://larmarange.github.io/analyse-R/gtsummary.html#tables-de-survie-avec-tbl_survfit