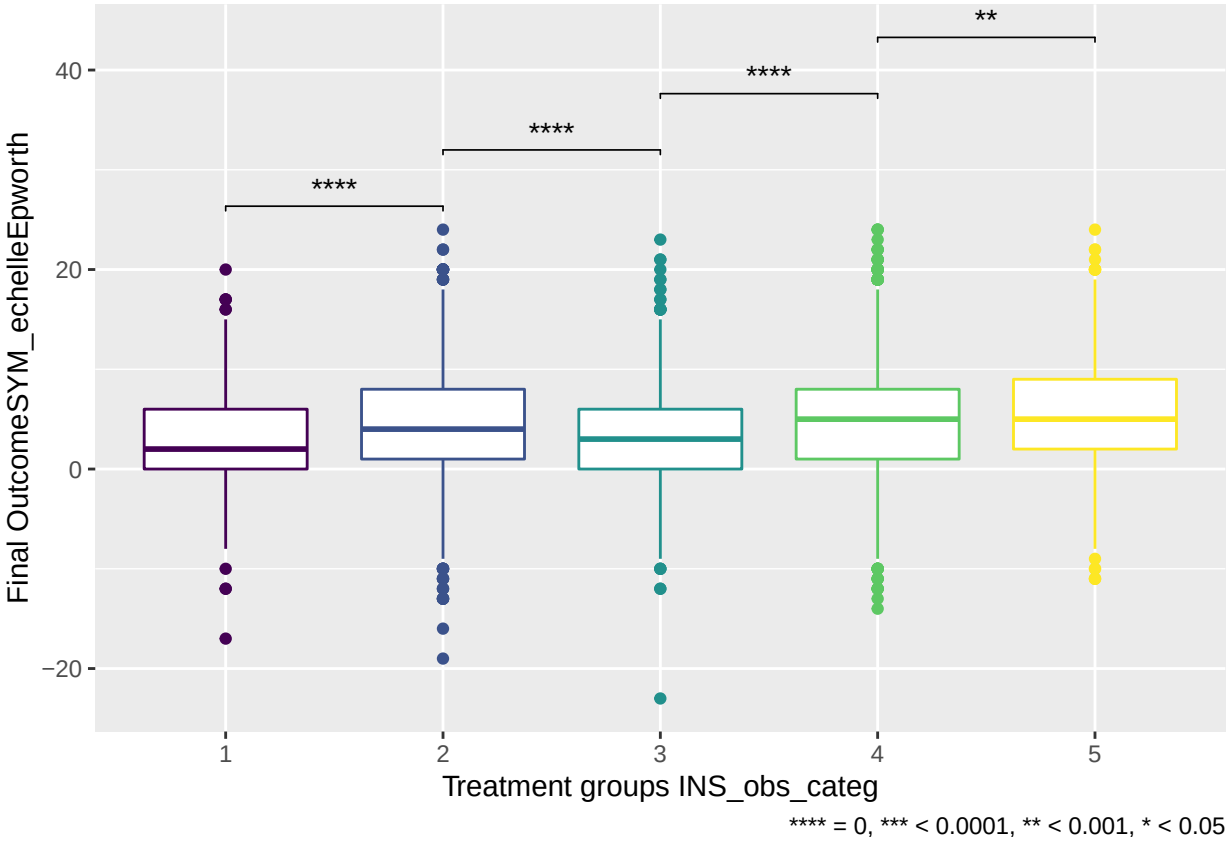
supplementary materials

Bettega Francois

22 août, 2020

Contents



Selections des variables et individus pour les modèle

Imputation density plot

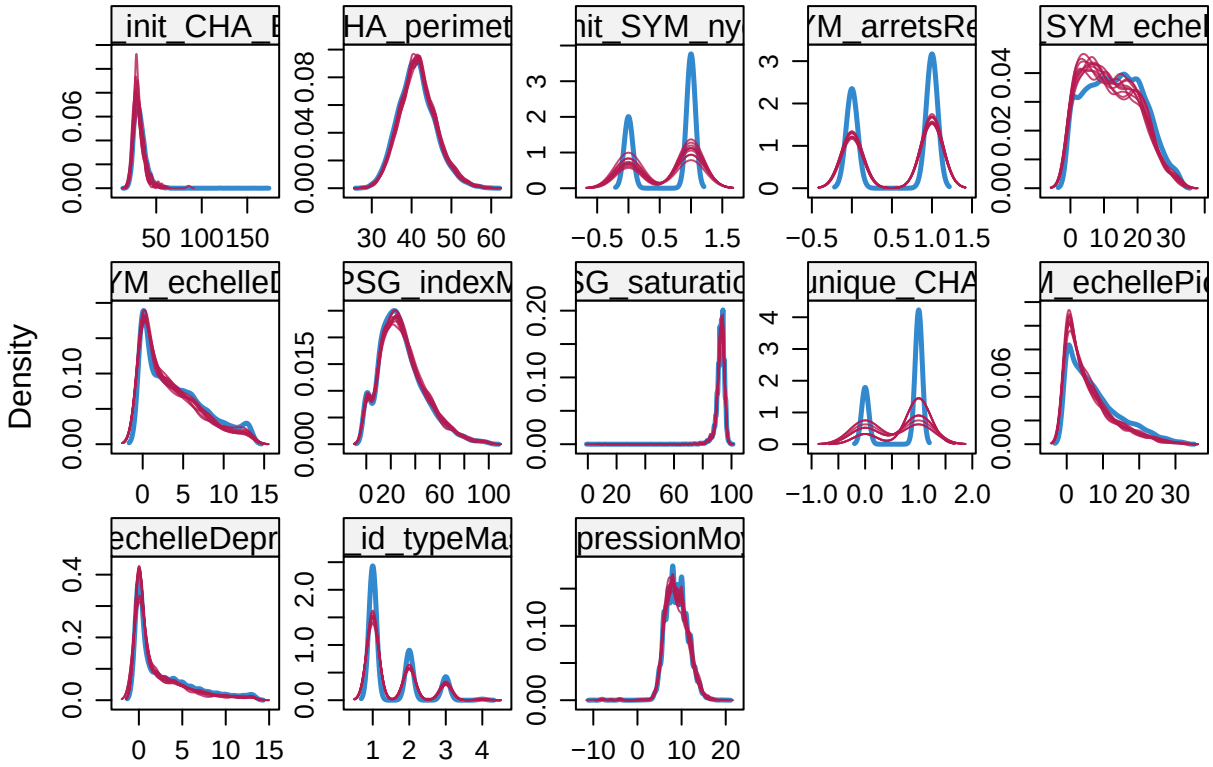


Table 1: Table des différence entre les variables Pré et Post imputation

	med	quart1	quart2	mean	sd	min	max
Vis_init_CHA_BMI	0.01	0.019	0.044	0.02	0.013	0	0
Vis_init_CHA_perimetreCervical	0	0	0	-0.114	-0.024	0	0
Vis_init_SYM_nycturie	0	0	0	1.9e-05	-6.1e-06	0	0
Vis_init_SYM_arretsRespiratoires	0	0	0	-9.9e-04	1.5e-04	0	0
Vis_init_SYM_echellePichot	0	0	0	0.134	-0.006	0	0
Vis_init_SYM_echelleDepression	0	0	1	0.066	0.02	0	0
Vis_init_PSG_indexMicroEveil	0	0	0.125	0.031	0.005	0	0
Vis_init_PSG_saturationMoyenne	0	0	0	-0.057	0.175	0	0
Mesure_unique_CHA_id_sexe	0	0	0	7.9e-05	-3.5e-05	0	0
SYM_echellePichot	0	0	0	0.199	0.062	0	0
SYM_echelleDepression	0	0	0	0.076	0.041	0	0
PPC_id_typeMasque	0	0	0	-0.003	-0.001	0	0
PPC_pressionMoyenne	0	0	0	-0.002	-2.1e-04	0	0

Estimations des poids

Table 2: Table des coeficient du modèle de poids

y.level	term	estimate	std.error	p.value
2	(Intercept)	2.178	4.8e-04	0
3	(Intercept)	0.82	6.6e-04	0
4	(Intercept)	0.773	4.4e-04	0
5	(Intercept)	0.278	7.8e-04	0
2	INS__PPC__effet__ind	0.732	0.042	8.5e-14
3	INS__PPC__effet__ind	0.921	0.042	0.052
4	INS__PPC__effet__ind	0.683	0.041	2.6e-20
5	INS__PPC__effet__ind	0.624	0.05	2.0e-21
2	INS__tmp__entre__rdv	1.001	0.047	0.991
3	INS__tmp__entre__rdv	0.939	0.053	0.229
4	INS__tmp__entre__rdv	1.067	0.046	0.157
5	INS__tmp__entre__rdv	1.162	0.047	0.001
2	PPC__id__typeMasque2	0.561	0.045	1.2e-37
3	PPC__id__typeMasque2	0.772	0.058	7.6e-06
4	PPC__id__typeMasque2	0.422	0.041	3.6e-98
5	PPC__id__typeMasque2	0.416	0.056	2.6e-54
2	PPC__id__typeMasque3	0.91	0.06	0.115
3	PPC__id__typeMasque3	0.975	0.081	0.75
4	PPC__id__typeMasque3	0.713	0.055	9.6e-10
5	PPC__id__typeMasque3	0.585	0.081	2.6e-11
2	PPC__id__typeMasque4	0.938	0.001	0
3	PPC__id__typeMasque4	1.172	0.001	0
4	PPC__id__typeMasque4	0.785	0.001	0
5	PPC__id__typeMasque4	0.678	0.001	0
2	PPC__pressionMoyenne	1.089	0.024	3.6e-04
3	PPC__pressionMoyenne	1.058	0.026	0.028
4	PPC__pressionMoyenne	1.132	0.024	1.6e-07
5	PPC__pressionMoyenne	1.16	0.026	8.3e-09
2	SYM__cephaleesMatinales	0.906	0.055	0.075
3	SYM__cephaleesMatinales	0.831	0.075	0.013
4	SYM__cephaleesMatinales	0.897	0.05	0.03
5	SYM__cephaleesMatinales	0.943	0.07	0.4
2	SYM__echelleDepression	0.993	0.024	0.773
3	SYM__echelleDepression	1.014	0.026	0.578
4	SYM__echelleDepression	0.975	0.024	0.3
5	SYM__echelleDepression	1.008	0.026	0.766
2	SYM__echellePichot	0.962	0.01	1.5e-04
3	SYM__echellePichot	0.992	0.011	0.439
4	SYM__echellePichot	0.953	0.01	2.5e-06
5	SYM__echellePichot	0.958	0.011	1.2e-04
2	SYM__fatigueMatinale	1.036	0.05	0.48
3	SYM__fatigueMatinale	1.077	0.066	0.259
4	SYM__fatigueMatinale	0.973	0.044	0.536
5	SYM__fatigueMatinale	0.911	0.061	0.124
2	SYM__somnolenceDiurne	1.409	0.05	4.6e-12
3	SYM__somnolenceDiurne	1.495	0.066	1.3e-09
4	SYM__somnolenceDiurne	1.284	0.044	1.2e-08
5	SYM__somnolenceDiurne	1.113	0.06	0.075
2	SYM__transpiNocturne	0.972	0.05	0.561
3	SYM__transpiNocturne	0.837	0.066	0.007
4	SYM__transpiNocturne	0.84	0.045	1.0e-04
5	SYM__transpiNocturne	0.748	0.063	4.2e-06
2	SYM__troubleLibido	1.298	0.048	5.1e-08
3	SYM__troubleLibido	1.12	0.065	0.08
4	SYM__troubleLibido	1.401	0.043	3.5e-15
5	SYM__troubleLibido	1.395	0.059	2.1e-08
2	Vis__init__CHA__age	1.015	0.005	0.002
3	Vis__init__CHA__age	1.006	0.005	0.235
4	Vis__init__CHA__age	1.026	0.005	1.6e-08
5	Vis__init__CHA__age	1.038	0.005	3.1e-14
2	Vis__init__CHA__BMI	0.989	0.009	0.192

Table 2: Table des coeficient du modèle de poids *(continued)*

y.level	term	estimate	std.error	p.value
3	Vis_init_CHA_BMI	0.985	0.009	0.099
4	Vis_init_CHA_BMI	0.99	0.008	0.231
5	Vis_init_CHA_BMI	0.992	0.009	0.386
2	Vis_init_CHA_perimetreCervical	0.998	0.014	0.898
3	Vis_init_CHA_perimetreCervical	1.018	0.015	0.253
4	Vis_init_CHA_perimetreCervical	1.006	0.014	0.648
5	Vis_init_CHA_perimetreCervical	1.011	0.015	0.486
2	Vis_init_GDS_gazDuSang	1.857	0.041	1.0e-51
3	Vis_init_GDS_gazDuSang	1.395	0.057	4.6e-09
4	Vis_init_GDS_gazDuSang	1.886	0.036	2.9e-68
5	Vis_init_GDS_gazDuSang	1.743	0.05	1.2e-28
2	Vis_init_INS_IAH	0.995	0.004	0.213
3	Vis_init_INS_IAH	0.993	0.004	0.099
4	Vis_init_INS_IAH	0.998	0.004	0.534
5	Vis_init_INS_IAH	1.004	0.004	0.248
2	Vis_init_PSG_delai_explo	1	1.3e-04	0.747
3	Vis_init_PSG_delai_explo	1	1.4e-04	0.737
4	Vis_init_PSG_delai_explo	1	1.3e-04	0.771
5	Vis_init_PSG_delai_explo	1	1.3e-04	0.789
2	Vis_init_PSG_indexMicroEveil	1	0.004	0.928
3	Vis_init_PSG_indexMicroEveil	1.002	0.004	0.7
4	Vis_init_PSG_indexMicroEveil	1.006	0.004	0.091
5	Vis_init_PSG_indexMicroEveil	1.004	0.004	0.273
2	Vis_init_PSG_saturationMoyenne	1.008	0.006	0.184
3	Vis_init_PSG_saturationMoyenne	1.007	0.006	0.313
4	Vis_init_PSG_saturationMoyenne	1.009	0.006	0.125
5	Vis_init_PSG_saturationMoyenne	0.993	0.006	0.288
2	Vis_init_SYM_cephaleesMatinales	1.348	0.046	1.2e-10
3	Vis_init_SYM_cephaleesMatinales	1.2	0.063	0.004
4	Vis_init_SYM_cephaleesMatinales	1.365	0.041	4.9e-14
5	Vis_init_SYM_cephaleesMatinales	1.379	0.057	1.7e-08
2	Vis_init_SYM_dyspneeDEffort	0.788	0.043	2.5e-08
3	Vis_init_SYM_dyspneeDEffort	1.008	0.058	0.893
4	Vis_init_SYM_dyspneeDEffort	0.763	0.038	1.0e-12
5	Vis_init_SYM_dyspneeDEffort	0.731	0.052	1.8e-09
2	Vis_init_SYM_echelleDepression	1.013	0.02	0.526
3	Vis_init_SYM_echelleDepression	0.988	0.021	0.581
4	Vis_init_SYM_echelleDepression	1.032	0.019	0.108
5	Vis_init_SYM_echelleDepression	1.043	0.021	0.042
2	Vis_init_SYM_nycturie	0.742	0.041	3.0e-13
3	Vis_init_SYM_nycturie	0.817	0.055	2.7e-04
4	Vis_init_SYM_nycturie	0.727	0.037	3.5e-18
5	Vis_init_SYM_nycturie	0.786	0.051	3.0e-06
2	Vis_init_SYM_somnolenceConduite	1.465	0.041	2.8e-20
3	Vis_init_SYM_somnolenceConduite	1.144	0.056	0.017
4	Vis_init_SYM_somnolenceConduite	1.507	0.037	1.0e-28
5	Vis_init_SYM_somnolenceConduite	1.369	0.051	7.5e-10

modèle final d'estimations de la somenolence

Table 4: Table des coeficient du modèle final pondéré

	Estimate	Std. Error	t value	Pr(>|t|)
Adherence group (Intercept)	3.7512507	1.4494454	2.5880594	0.0096670
Adherence group 1	-1.2397869	0.2604071	-4.7609562	0.0000020
Adherence group 2	-0.1769711	0.1071264	-1.6519837	0.0985720
Adherence group 3	-0.5433355	0.1461729	-3.7170747	0.0002027
Adherence group 5	0.0541878	0.1337646	0.4050980	0.6854149
Adherence group Vis_init_CHA_age	0.0057956	0.0038323	1.5123273	0.1304849

Table 4: Table des coeficient du modèle final pondéré *(continued)*

	Estimate	Std. Error	t value	Pr(>|t|)
Adherence group Vis_init_CHA_BMI	0.0274967	0.0077542	3.5460301	0.0003930
Adherence group Vis_init_CHA_perimetreCervical	-0.0142412	0.0133327	-1.0681399	0.2854854
Adherence group Vis_init_SYM_somnolenceConduite	1.1835988	0.0990950	11.9440814	0.0000000
Adherence group Vis_init_SYM_fatigueMatinale	0.6991004	0.1229635	5.6854308	0.0000000
Adherence group Vis_init_SYM_cephaleesMatinales	0.0908646	0.1109133	0.8192397	0.4126708
Adherence group Vis_init_SYM_troubleLibido	-0.3417607	0.1243285	-2.7488530	0.0059921
Adherence group Vis_init_SYM_dyspneeDEffort	-0.7465963	0.1012215	-7.3758638	0.0000000
Adherence group Vis_init_SYM_nycturie	0.2761769	0.0966305	2.8580722	0.0042718
Adherence group Vis_init_SYM_arretsRespiratoires	0.8196661	0.0926272	8.8490887	0.0000000
Adherence group Vis_init_SYM_echellePichot	0.2904523	0.0077179	37.6336750	0.0000000
Adherence group Vis_init_SYM_echelleDepression	-0.0192230	0.0166966	-1.1513139	0.2496330
Adherence group Vis_init_PSG_indexMicroEveil	0.0046400	0.0028666	1.6186235	0.1055625
Adherence group Vis_init_PSG_saturationMoyenne	-0.0330206	0.0132726	-2.4878818	0.0128682
Adherence group Vis_init_PSG_delai_explo	0.0000314	0.0000491	0.6396780	0.5223978
Adherence group Vis_init_INS_IAH	0.0185824	0.0028332	6.5587975	0.0000000
Adherence group Vis_init_GDS_gazDuSang	-0.5157659	0.0951764	-5.4190515	0.0000001
Adherence group Mesure_unique_CHA_id_sexe	-0.0377081	0.1228390	-0.3069715	0.7588720
Adherence group SYM_somnolenceDiurne	-0.0552797	0.1152500	-0.4796499	0.6314877
Adherence group SYM_fatigueMatinale	-0.0729479	0.1207845	-0.6039506	0.5458914
Adherence group SYM_cephaleesMatinales	-0.1843140	0.1315249	-1.4013623	0.1611394
Adherence group SYM_troubleLibido	-0.2304913	0.1363107	-1.6909260	0.0908848
Adherence group SYM_transpiNocturne	-0.4555171	0.1173715	-3.8809851	0.0001048
Adherence group SYM_echellePichot	-0.3609315	0.0091658	-39.3782734	0.0000000
Adherence group SYM_echelleDepression	-0.0063468	0.0191180	-0.3319816	0.7399107
Adherence group PPC_id_typeMasque2	0.0759211	0.1067654	0.7111021	0.4770390
Adherence group PPC_id_typeMasque3	0.4765456	0.1433043	3.3254095	0.0008863
Adherence group PPC_id_typeMasque4	0.0741935	0.6406370	0.1158120	0.9078041
Adherence group PPC_pressionMoyenne	0.0672376	0.0192669	3.4898040	0.0004856
Adherence group INS_tmp_entre_rdv	0.3180438	0.0305735	10.4026124	0.0000000
Adherence group INS_PPC_effet_ind	-0.2240908	0.0394918	-5.6743579	0.0000000

Code

```
############################################################
##                      packages                          ##
############################################################
library(data.table)
library(tidyverse)
library(stringr)
library(lubridate)
library(DiagrammeR)
library(DiagrammeRsvg)
library(rsvg)
library(ipw)
library(broom)
library(ggpubr)
library(knitr)
library(kableExtra)
library(cluster) # test découpage
library(modelr)
library(nnet)
library(xtable)
library(caret)
conflicted::conflict_prefer("filter", "dplyr")
#library(ipw)
```

Table 3: Table of final model adherence group coefficient

	Estimate	95% Confidence interval	Pr(> |t|)
(Intercept)	3.751	(0.91 ; 6.592)	0.01
INS__obs__categ1	-1.24	(-1.75 ; -0.729)	2.0e-06
INS__obs__categ2	-0.177	(-0.387 ; 0.033)	0.099
INS__obs__categ3	-0.543	(-0.83 ; -0.257)	2.0e-04
INS__obs__categ5	0.054	(-0.208 ; 0.316)	0.685
Vis__init__CHA__age	0.006	(-0.002 ; 0.013)	0.13
Vis__init__CHA__BMI	0.027	(0.012 ; 0.043)	3.9e-04
Vis__init__CHA__perimetreCervical	-0.014	(-0.04 ; 0.012)	0.285
Vis__init__SYM__somnolenceConduite	1.184	(0.989 ; 1.378)	1.2e-32
Vis__init__SYM__fatigueMatinale	0.699	(0.458 ; 0.94)	1.3e-08
Vis__init__SYM__cephaleesMatinales	0.091	(-0.127 ; 0.308)	0.413
Vis__init__SYM__troubleLibido	-0.342	(-0.585 ; -0.098)	0.006
Vis__init__SYM__dyspneeDEffort	-0.747	(-0.945 ; -0.548)	1.8e-13
Vis__init__SYM__nycturie	0.276	(0.087 ; 0.466)	0.004
Vis__init__SYM__arretsRespiratoires	0.82	(0.638 ; 1.001)	1.0e-18
Vis__init__SYM__echellePichot	0.29	(0.275 ; 0.306)	1.9e-288
Vis__init__SYM__echelleDepression	-0.019	(-0.052 ; 0.014)	0.25
Vis__init__PSG__indexMicroEveil	0.005	(-9.8e-04 ; 0.01)	0.106
Vis__init__PSG__saturationMoyenne	-0.033	(-0.059 ; -0.007)	0.013
Vis__init__PSG__delai__explo	3.1e-05	(-6.5e-05 ; 1.3e-04)	0.522
Vis__init__INS__IAH	0.019	(0.013 ; 0.024)	5.7e-11
Vis__init__GDS__gazDuSang	-0.516	(-0.702 ; -0.329)	6.1e-08
Mesure__unique__CHA__id__sexe	-0.038	(-0.278 ; 0.203)	0.759
SYM__somnolenceDiurne	-0.055	(-0.281 ; 0.171)	0.631
SYM__fatigueMatinale	-0.073	(-0.31 ; 0.164)	0.546
SYM__cephaleesMatinales	-0.184	(-0.442 ; 0.073)	0.161
SYM__troubleLibido	-0.23	(-0.498 ; 0.037)	0.091
SYM__transpiNocturne	-0.456	(-0.686 ; -0.225)	1.0e-04
SYM__echellePichot	-0.361	(-0.379 ; -0.343)	1.4e-313
SYM__echelleDepression	-0.006	(-0.044 ; 0.031)	0.74
PPC__id__typeMasque2	0.076	(-0.133 ; 0.285)	0.477
PPC__id__typeMasque3	0.477	(0.196 ; 0.757)	8.9e-04
PPC__id__typeMasque4	0.074	(-1.181 ; 1.33)	0.908
PPC__pressionMoyenne	0.067	(0.029 ; 0.105)	4.9e-04
INS__tmp__entre__rdv	0.318	(0.258 ; 0.378)	3.3e-25
INS__PPC__effet__ind	-0.224	(-0.301 ; -0.147)	1.4e-08

```r
############################################################
##                      function                          ##
############################################################
source("source_francois.R",
       encoding = "UTF-8")
source("function_spe_stage.R",
       encoding = "UTF-8")


############################################################
##                      donnee                            ##
############################################################

df <- read_rds("data/genere/data_prett.rds")
nb_core <- parallel::detectCores() - 1 # nb de coeur pouyr la parréllélisation

# nb_visite <- df %>% group_by(id_patient) %>% summarise(max = max(num_visite) ,n = n())


visite <-   df %>%
  filter(num_visite == 1) %>% #selection d'une seul visite
  group_by(Mesure_unique_INS_visite_Max) %>%
  summarise(n=n()) %>%
  t.df(.,"Mesure_unique_INS_visite_Max") %>%
  select(-key)

colonnes_ss_variance <-  df %>%
  select_if(function(col) n_distinct(col,na.rm = TRUE) <= 1) %>%
  colnames() # retire les variables visite init sans valeurs toute = a 0


df <- df %>%
  select(-Mesure_unique_INS_visite_Max) %>%
  select_if(function(col) n_distinct(col, na.rm = TRUE) > 1)

nb_pat <-  df %>%
  select(id_patient) %>%
  n_distinct()

confunding_diagram <- {"digraph R {
 rankdir=LR;
 secret_node2[height=0, width=0, margin=0,shape=point, style=invis];
 secre_right [style=invisible];
 somnolence  [label = 'Outcome: \nDaytime sleepiness' , shape = rectangle , style=filled, fillcolor='#9
 observance  [label = 'Factor of exposure: \noadherence', shape = rectangle  , style=filled, fillcolor=
 Age [label = 'Age'  , style=filled, fillcolor=red];
 Sex [label = 'Gender'  , style=filled, fillcolor=red];
 BMI [label = 'BMI'  , style=filled, fillcolor=red];
 secre_left [style=invisible];
 secre_left_BMI [style=invisible];
 secre_right_BMI [style=invisible];
 secre_left_sex [style=invisible];
 secre_right_sex [style=invisible];
```

```
 {rank=source; observance secre_left secre_left_sex secre_left_BMI} ;
 {rank=sink; somnolence secre_right secre_right_sex secre_right_BMI};
 {rank=same;secret_node2 Age Sex BMI}

 observance ->  secret_node2[arrowhead = none,penwidth = 5] ;
 secret_node2 -> somnolence [penwidth = 5];

 secre_left -> Age[style=invis];
 secre_left -> observance[style=invis];
 secre_right -> somnolence [style=invis];
 Age -> secre_right  [style=invis];
 Age -> secret_node2 [style=invis,minlen='2.9'];
 Age -> observance;
 Age ->  somnolence;

 Sex ->  Age [style=invis];
 Sex -> observance;
 Sex ->  somnolence;
 Sex -> secre_right_sex [style=invis];
 secre_left_sex -> Sex [style=invis];
 secre_left_sex -> secre_left [style=invis];


 BMI ->  Sex [style=invis];
 BMI -> observance;
 BMI ->  somnolence;
 BMI -> secre_right_BMI [style=invis];
 secre_left_BMI -> BMI [style=invis];
 secre_left_BMI -> secre_left_sex [style=invis];
}"}
grViz(confunding_diagram) %>%
  export_svg %>%
  charToRaw %>%
  rsvg_pdf("graph/cofunding_factor.pdf")
include_graphics("graph/cofunding_factor.pdf")

df <- df %>%
  replace_na(list(Vis_init_ATC_id_typeDiabete = 0,
                  ATC_id_typeDiabete = 0))

## Création des variables temps indépendant
var_mesure_unique <- df %>%
  select(contains("Mesure_unique_")) %>%
  colnames() # %>% gsub("Mesure_unique_","",.)
var_visit_init <- df %>%
  select(contains("Vis_init_")) %>%
  colnames() # %>% gsub("Mesure_unique_","",.)

var_non_visit_init <-  df %>%
  select(!contains("Vis_init_")) %>%
  colnames()
# print.vecteur(var_mesure_unique)
```

```r
## suppression de la premieres visites
df_example_mesure_uni_init <- df %>%
  select(id_patient,
         num_visite,
         Mesure_unique_CHA_id_sexe,
         contains("CHA_age"),
         contains("CHA_poids")) %>%
  filter(id_patient < 10)  %>%
  mutate_at(vars(contains("_age")),~formatC(., format = "f", digits = 1))

df_inclusion <- df %>%
  filter(num_visite != 1)


# Critère d'inclusion
visite_seuil <- 2
df_visite <- df_inclusion %>%
  filter(num_visite <= visite_seuil)

df_SAS <-  df_visite  %>%
  filter(!is.na(Vis_init_INS_IAH),
         Vis_init_INS_IAH >= 5) #INS_IAH

perte_SAS_pat <- df_visite %>%
  filter(is.na(Vis_init_INS_IAH) | Vis_init_INS_IAH < 5) %>%  #INS_IAH
  select(id_patient) %>%  n_distinct()



nb_SAS_pat <- n_distinct(df_SAS$id_patient)



df_obs <- df_SAS %>%
  group_by(id_patient)  %>%
  mutate(INS_observance = sum(is.na(PPC_observanceMoy_finale))) %>%
  filter(INS_observance == 0 ) %>%
  select(-INS_observance) # retrait de la variables instrumentale


perte_obs_pat <-  df_SAS %>%
  group_by(id_patient) %>%
  filter(num_visite != 1 ) %>%
  mutate(INS_observance = sum(is.na(PPC_observanceMoy_finale))) %>%
  filter(INS_observance != 0 ) %>%
  select(id_patient) %>%
  n_distinct()

nb_obs_pat <-  n_distinct(df_obs$id_patient)


# Choix de l'outcome
df_choix_outcome <-  df_obs %>%
```

```
  group_by(id_patient) %>%
  mutate(naEPW = sum(is.na(SYM_echelleEpworth)) == 0 ,
         nasysto =  sum(is.na(CHA_PASystolique)) == 0,
         nadiasto =  sum(is.na(CHA_PADiastolique)) == 0 ,
         PA_ok =  sum(!is.na(CHA_PASystolique),!is.na(CHA_PADiastolique))==2 ,
         PAna =  sum(is.na(CHA_PASystolique),is.na(CHA_PADiastolique))!=2) %>%
  distinct(id_patient , .keep_all = TRUE)


df_EPW <-  df_obs %>%
   group_by(id_patient) %>%
   mutate(naEPW = sum(is.na(SYM_echelleEpworth)|
                        is.na(Vis_init_SYM_echelleEpworth))) %>% # utile car permet de scale sur plus d
   filter(naEPW == 0) %>%
   select(-naEPW) %>%
   mutate(SYM_echelleEpworth = Vis_init_SYM_echelleEpworth - SYM_echelleEpworth) %>%
  select(-Vis_init_SYM_echelleEpworth)%>%
   ungroup()


#sum(is.na(df_EPW$SYM_echelleEpworth))

perte_EPW_pat <- df_obs %>%
  group_by(id_patient) %>%
   mutate(naEPW = sum(is.na(SYM_echelleEpworth)|
                        is.na(Vis_init_SYM_echelleEpworth))) %>% # utile car permet de scale sur plus d
  filter(naEPW != 0) %>% select(id_patient) %>%
  n_distinct()

nb_EPW_pat <- n_distinct(df_EPW$id_patient)

name_expo <- paste0(quote(INS_obs_categ))
name_outcome <- paste0(quote(SYM_echelleEpworth))

Observance_numeric_sav <- df_EPW
res.hc <- hclust(dist(df_EPW  %>%
                        select(PPC_observanceMoy_finale)),  method = "ward.D2")
INS_OBS_cluster <- cutree(res.hc,k = 5)

df_obs_categ_2 <- df_EPW %>%
  mutate(INS_obs_categ = INS_OBS_cluster)


cut_cluster <- df_obs_categ_2 %>%
  group_by(INS_obs_categ) %>%
  summarise(min = min(PPC_observanceMoy_finale),
            max = max(PPC_observanceMoy_finale),.groups = "drop" ) %>%
  arrange(max)

df_obs_categ_2 <- df_obs_categ_2 %>%  select(-PPC_observanceMoy_finale)

obs_cut_temp3 <- c(0,cut_cluster$max)
for (i in seq_along(cut_cluster$max)) {
```

```
  cat("-", table(df_obs_categ_2[,name_expo])[i],
      " patients avec une observance entre ",roud_hour(cut_cluster$min[i]),
      " et ", roud_hour(cut_cluster$max[i]) ,'\n')
}

eval_decoupage_cluster <- function_box_plot_expo_func(df_obs_categ_2,
                                                      name_expo,name_outcome)
eval_decoupage_cluster$plot_test

vec_cut <- obs_cut_temp3
df_obs_categ <- df_obs_categ_2

date_ddn <- df_obs_categ %>%
  ungroup() %>%
  select(which(sapply(.,is.Date)),contains("date"),contains("ddn")#,contains("INS_tmp_entre_rdv")
         ) %>%
  colnames()

id_idv <- df_obs_categ %>%
  ungroup() %>%
  select(contains("id_"),contains("idv")) %>%
  select(-id_patient,
         -Mesure_unique_ATC_id_typeDiabete,
         -PPC_id_typeMasque,
         -Mesure_unique_CHA_id_sexe) %>%
  colnames()

if(visite_seuil == 2) df_obs_categ <-  df_obs_categ %>% select(-num_visite)

df_ss_var_select_man <- df_obs_categ  %>%
  ungroup() %>%
  select(-one_of(date_ddn),
         -one_of(id_idv)) %>%
  mutate_all(as.numeric)



charecter_col <- df_ss_var_select_man  %>%
  select_if((colSums(is.na(.)) == nrow(.)) ) %>%
  colnames()

# conserver une dataframe a part de la selection de variables
df_sav_temp <- df_ss_var_select_man %>%
  mutate_all(as.numeric) %>%
  select_if(!(colSums(is.na(.)) == nrow(.))) # remove char col dont je ne sais pas quoi faire de toutes




Recherche_duplicata_column <- distinc_col(df_sav_temp )
```

```r
df_sav_temp <- Recherche_duplicata_column$df

var_non_visit_init_sav_tem <-  df_sav_temp %>%
  select(!contains("Vis_init_")) %>%
  colnames()

col_ident <- Recherche_duplicata_column$colonne_suprime

# Gestion des cas ou variables visit_init et var temps dep corrélé
temp_tt_colonne_dupli <- lapply(seq_along(col_ident), function(x) {
    court <- names(which.min(sapply(col_ident[[x]], nchar)))
    restant <- str_remove(col_ident[x] %>% unlist(), court)
      if (all(restant %in% c("Vis_init_", ""))) {
      court
    }
})


duplicate_fact_init <- temp_tt_colonne_dupli %>% compact %>% unlist



duplicate_verif_manu <-   col_ident[lapply(temp_tt_colonne_dupli,
                                           function(x) is.null(x) ) %>% unlist()]

if (!is_empty(duplicate_verif_manu)) stop("vérif manuel a faire")


# valeurs manquante
nb_val_manq_par_var <- compte_na_par_var_par_grp(df_sav_temp,
                                                 "id_patient",colnames(df_sav_temp))
seuil_na <- 0.6
frac_val_manq_par_var <- nb_val_manq_par_var %>%
  mutate_at(vars(-presence_na),list(~(./sum(.)))) %>%
  filter(presence_na == (visite_seuil - 1)) %>% select(-presence_na)
trop_manquant <- frac_val_manq_par_var %>%
  select_if(. > 1-seuil_na ) %>%
  colnames()

df_rm_na <- df_sav_temp %>% select(-all_of(trop_manquant),-id_patient)


Var_manquant_colnames <- df_rm_na %>% colnames()



# Gestions des colinéarité
seuil_cor <- 0.7
corelation_var_df <- df_rm_na  %>% select(all_of(Var_manquant_colnames))


mat_cor <- cor(corelation_var_df  ,use =  "pairwise.complete.obs")
```

```r
var_cor <- which(abs(mat_cor) > seuil_cor &
                   (row(mat_cor) != col(mat_cor)) ,arr.ind = TRUE) %>%
  as_tibble() %>%
  pivot_wider(names_from = row, values_from = col,values_fn = list(col = list)) %>%
  t.df()


var_correlle <- lapply(unique(
  lapply(
    lapply(split(var_cor, row.names(var_cor)), unlist),
    function(x) sort(
      as.numeric(
        unique(x))
      ))),
  function(x) colnames(mat_cor)[x])


# Gestion des cas ou variables visit_init et var temps dep corrélé
Var_corr_init_temp_dep <- lapply(var_correlle, function(x) {
if (length(x) == 2){
    court <- names(which.min(sapply(x, nchar)))
    long <- names(which.max(sapply(x, nchar)))
    different_part <- str_remove(long, court)
    if (different_part == "Vis_init_") {
      result <- court
    }
}}
) %>% compact %>% unlist

Var_corr_no_init_temp_dep <- lapply(var_correlle, function(x) {
 if (length(x) == 2){
    court <- names(which.min(sapply(x, nchar)))
    long <- names(which.max(sapply(x, nchar)))
    different_part <- str_remove(long, court)
    if (different_part != "Vis_init_") {
      result <- TRUE#c(court,long)
    } else {result <- FALSE}
 } else if (length(x) != 2) {
   result <- TRUE
 }
  }
) %>%
 # compact %>%
  unlist


# list of correlated variables
Var_cor_select_manu <- var_correlle[Var_corr_no_init_temp_dep]

# variable choose from list above
Select_manu_cor <- c("Vis_init_CHA_BMI")
```

```r
var_cor_retire <- c(Var_corr_init_temp_dep,
                    select_manuel_verif(Var_cor_select_manu,Select_manu_cor))

Var_cor_colnames <- Var_manquant_colnames[Var_manquant_colnames %notin% var_cor_retire]



df_ss_cor <- df_sav_temp %>%
  select(id_patient,all_of(name_outcome),all_of(name_expo),all_of(Var_cor_colnames))


not_numeric <- df_ss_cor %>% summarise_all(list(~n_distinct(.))) %>%
  t.df() %>%
  mutate(numeric = col_1 < 10) %>%
  filter(numeric) %>%
  select(key) %>%
  unlist(use.names = FALSE)

nb_valeur_facteur_deuxieme_max <- function(col){
  res <- sort(table(col),decreasing = TRUE)[2]
  return(res)
}

poucent_pat_necessaire <- 20
nb_pat_cut <- df_ss_cor %>%
  summarise_all(list(~sum(!is.na(.)) * poucent_pat_necessaire/100))  %>%
  t.df()

moda_trop_peu_pat_moda_2 <- df_ss_cor %>%
  select(all_of(not_numeric)) %>%
  summarise_all(list(~nb_valeur_facteur_deuxieme_max(.))) %>%
  t.df() %>%
  inner_join(nb_pat_cut,by = "key" ) %>%
  setNames( c("var", "second_max","cut_nb_pat")) %>%
  filter(second_max < cut_nb_pat) %>%
  select(var) %>%
  unlist(use.names = FALSE)

near_zero_var <- df_ss_cor  %>%
  select(all_of(nearZeroVar(.,names = TRUE))) %>% colnames()
moda_trop_peu_pat <- c(moda_trop_peu_pat_moda_2,near_zero_var)

Var_nb_pat_colnames <- Var_cor_colnames[Var_cor_colnames %notin% moda_trop_peu_pat]



## Selection des variables liée a l'outcome
set.seed(123)
# patients use for variables select
df_model_select_var <- df_sav_temp %>% group_by(INS_obs_categ) %>%
  slice_sample(prop= 0.2) %>% ungroup()
# other patients
df_analyse_final <- df_sav_temp %>% anti_join(df_model_select_var)
```

```r
# value for flow chart
perte_select_var <- df_model_select_var %>%
  select(id_patient) %>% n_distinct()

model_lin <- df_model_select_var %>% select(-id_patient) %>%
  select(all_of(Var_nb_pat_colnames))
model_expo <- model_lin %>%
  select(-all_of(name_outcome)) %>%
  gather(measure, value, -all_of(name_expo)) %>%
  mutate(value = as.numeric(value)) %>%
  group_by(measure) %>%
  nest() %>%
  ungroup() %>%
  mutate(fit = map(data, ~ glm(paste0(name_expo,"~ value"),
                               data = .x), family = binomial(link = 'logit'),
                   na.action = na.omit),
         tidied = map(fit, tidy)) %>%
  unnest(tidied) %>%
  filter(term != "(Intercept)") %>% # on elnlève l'intercept
  select(-data,-fit,-term)  # retire les donnée


model_outcome <- model_lin %>%
  select(-all_of(name_expo)) %>%
  gather(measure, value, -all_of(name_outcome)) %>%
   mutate(value = as.numeric(value)) %>%
  group_by(measure) %>%
  nest() %>%
  ungroup() %>%
  mutate(fit = map(data, ~ glm(paste0(name_outcome,"~ value"), data = .x),
                   family = gaussian(link = "identity"),na.action = na.omit),
         tidied = map(fit, tidy)) %>%
  unnest(tidied) %>%
  filter(term != "(Intercept)") %>% # on elnlève l'intercept
  select(-data,-fit,-term)  # retire les donnée



p_val_var <- model_expo %>%
inner_join(model_outcome, by= c("measure" = "measure")) %>%
  select(measure,contains("p.value"))


P_val_out_come_colnames <- p_val_var %>%
  filter(p.value.y < 0.2) %>%
  select(measure) %>%
  unlist(use.names = FALSE) %>%
  sort

P_val_out_come_et_expo_colnames <- p_val_var %>%
  filter(p.value.y < 0.2, p.value.x < 0.2)  %>%
  select(measure) %>%
  unlist(use.names = FALSE) %>%
```

```
  sort

Var_model_outcome <- Var_nb_pat_colnames[Var_nb_pat_colnames %in% P_val_out_come_colnames]
var_mod_pds <- Var_nb_pat_colnames[Var_nb_pat_colnames %in% P_val_out_come_et_expo_colnames]

## imputation
df__pre_imput <- df_analyse_final  %>%
  select(id_patient,
         all_of(name_outcome),
         all_of(name_expo),
         all_of(Var_model_outcome))

missing_patern <- mice::md.pattern(df__pre_imput)
col_sans_NA <- df__pre_imput %>% select_if(colSums(is.na(.)) == 0) %>% colnames()

table_resume_pre_impute <- table_resume_latex(df__pre_imput %>%
  select(-id_patient) ,name_expo, "grp_obs")




df__post_imput <- impute_si_changement(df__pre_imput,"data/genere/data_impute.rds")


table_resume_post_impute <- table_resume_latex(df__post_imput %>%
  select(-id_patient) ,name_expo,"grp_obs")


### Vérification de  l'imputation

imputation_verif <- test_imputation(df__pre_imput,df__post_imput,name_expo)

imputation_verif$T_diff %>%
  arrondie_df %>%
  kable( longtable =  TRUE,
         booktabs = TRUE,
             caption = "Table des différence entre les variables Pré et Post imputation") %>%
  kable_styling(latex_options = c("hold_position", "repeat_header"),
                font_size = 7)

df_final <- df__post_imput
# perte_na_covar_pat <- n_distinct(df_sav_temp$id_patient) - n_distinct(df_final$id_patient)

attributes(Observance_numeric_sav$id_patient) <- NULL # gestion d'un warning chiant
observanve_num <- df_final  %>%
  inner_join(Observance_numeric_sav %>% filter(num_visite == visite_seuil) ,
             by = "id_patient") %>%
  select(PPC_observanceMoy_finale)

rm(Observance_numeric_sav)

patient_par_grp_expo  <- df_final %>% select(all_of(name_expo)) %>% table
```

```r
perte_visit_pat <- nb_pat - (visite %>% unlist() %>% rev() %>% cumsum %>%
                               .[length(.) - (visite_seuil-1)])

patient_perdu <- c(perte_visit_pat,
                   perte_SAS_pat,
                   perte_obs_pat,
                   perte_EPW_pat,
                   perte_select_var)
critere <- c(paste0("Patient with less than " , visite_seuil ," visits"),
             "Patient without Sleep apnea",
             "Patient without adherence",
             "Patient without Epworth",
             "Patients use in variables selection")




grViz(flow_chart(nb_pat,critere,patient_perdu,patient_par_grp_expo)) %>%
  export_svg %>%
  charToRaw %>%
  rsvg_pdf("graph/graph_flow_chart_2_visites.pdf")

# test
## pas de NA
if (df_final %>% summarise_all(~sum(is.na(.))) %>% rowSums(.) != 0 ) {
  stop('Row avec Na')}
# test pas de tb erectile chez les femmes

if("Vis_init_SYM_troubleErection" %in% colnames(df_final)){
if ((df_final  %>% filter(Mesure_unique_CHA_id_sexe == 0) %>%
     summarise(sum(Vis_init_SYM_troubleErection))) != 0  ) {
  stop('trouble erectile chez les femmes')}
}
saveRDS(df_final, file = paste0("data/genere/data_prett_nb_vis_",visite_seuil,".rds"))


#sauvegarde des variables du model de poids
df_var_modele <- data.frame(Var_model_outcome = Var_model_outcome) %>%
  mutate(var_mod_pds = Var_model_outcome %in% var_mod_pds)


saveRDS(df_var_modele, file = paste0("data/genere/model_",visite_seuil,".rds"))

saveRDS(vec_cut, file = paste0("data/genere/obs_multinomial_en_heure.rds"))

df_modele_pds <- read_rds("data/genere/data_prett_nb_vis_2.rds")

# pour le moment facteur gérer a la main a réfléchir
donnee <- df_modele_pds %>%
  mutate_at( typages_function(df_modele_pds,10)$colonne_type$facteur,
             factor) %>%
  select(-id_patient) %>%
  #mutate(INS_obs_categ = INS_obs_categ >2) %>%
```

```r
  as.data.frame()

binaire <- calcule_pds_stage(donne = donnee,expo = INS_obs_categ,
                             covar = var_mod_pds ,out_come = SYM_echelleEpworth)


# Modèle multinomial de poids
# df_one_hot_encode <- donnee
alpha_tableau_resume <- 0.05
  tableau_resume_ex <- head(donnee,500)
  save(tableau_resume_ex,file = "tableau_resume_exemple.RData")

var_instrumental_name  <- c("INS_obs_categ" = "Adherence groups",
                            "Vis_init_INS_IAH" = "Apnea hypopnea index",
                            "INS_PPC_effet_ind" = "number of ADR types under CPAP",
                            "INS_tmp_entre_rdv" = "Duration since diagnosis (year)")

table_rename <- one_hot_fb(df__pre_imput,list_factor =  c(  "PPC_id_typeMasque"  )) %>%
  select(-id_patient) %>%  rename_variables(var_instrum_name = var_instrumental_name)


One_hot_encode_donnee <-table_rename %>% .$table_rename
table_resume_latex_ss_escape <- table_resume_latex(df_one_hot_encode_fonc =  One_hot_encode_donnee,
                                                   name_expo_fonc = "Adherence groups",
                                                   nom_grp  = "Adherence grp",
                                                   p_val = TRUE,
                                                   alpha = alpha_tableau_resume)

table_resume_html_prez <- table_resume_html(df_one_hot_encode_fonc =  One_hot_encode_donnee,
                                                   name_expo_fonc = "Adherence groups",
                                                   nom_grp  = "Adherence_grp",
                                                   p_val = TRUE,
                                                   alpha = alpha_tableau_resume)


# récupér ation du vrai nom des var du modèle depoids
label_var_mod_poids <- table_rename$complete_table %>%
  filter(str_detect(var_name, paste(var_mod_pds,collapse = '|'))) %>%
  select(Label) %>% unlist(use.names = FALSE)


model_poids_df <- tidy(binaire$res_intermediaire$regression_modele_pds$regression_temps_ind)
model_poids_df %>%
  select(-statistic) %>%
  arrondie_df %>%
  mutate(p.value = ifelse(p.value == "0.0e+00",
                          "< 0.001",p.value)) %>%
  arrange(term) %>%
  kable( longtable =  TRUE,
         booktabs = TRUE,
         caption = "Table des coeficient du modèle de poids") %>%
  kable_styling(latex_options = c("hold_position", "repeat_header"),
                font_size = 7)# %>%  landscape()
```

```r
donnee <- donnee %>% mutate_at(name_expo,list(~relevel(., ref = "4"#max(as.numeric(.)) ancienne version
                                                    )))


poids_mod_final <- binaire$res_intermediaire$poids$poids_tronc$poids_trunc_stab$`( 0.01;0.99 )`
donne_mod_fin <- donnee
model_final <- glm( SYM_echelleEpworth ~ .,
                    data = donnee,
                    family = gaussian(),
                    weight = poids_mod_final)
iR_mod_final <- confint(model_final)

summary(model_final)$coefficients %>%
  as.data.frame() %>%
  rownames_to_column %>%
  inner_join(iR_mod_final %>%
  as.data.frame()  %>% rownames_to_column() , by = "rowname") %>%
  column_to_rownames()  %>%
  arrondie_df() %>%
  rownames_to_column %>%
  rename(`Pr($>|t|$)` = `Pr(>|t|)`) %>% # gestion problème avec latex
  mutate(`95\\% Confidence interval` = paste0("( ",`2.5 %`," ; ",`97.5 %`,  " )")) %>%
  select(rowname,Estimate,`95\\% Confidence interval` ,`Pr($>|t|$)`) %>%
  column_to_rownames()  %>%
  kable(align = "c",
        label = "Table_model_final",
        booktabs = TRUE,
        escape = FALSE,
        caption = "Table of final model adherence group coefficient") %>%
  kable_styling(latex_options =  c( "striped","HOLD_position", "scale_down","repeat_header"),
                font_size = 7)


summary(model_final)$coefficients %>% as.data.frame() %>%
  rownames_to_column %>% filter(str_detect(rowname,"INS_obs_categ")) %>%
  mutate(rowname = paste0("Adherence group ",
                          str_remove_all(rowname,"INS_obs_categ"))) %>%
  column_to_rownames() %>%
  kable(longtable =  TRUE,
        booktabs = TRUE,
        caption = "Table des coeficient du modèle final pondéré") %>%
  kable_styling(latex_options = c("hold_position", "repeat_header"),
                font_size = 7)
```

Sources

```r
############################################################
##                      function                          ##
############################################################
deriv <- function(x, y) diff(y) / diff(x)
```

```r
get.elbow.points.indices <- function(x, y, threshold) {
  d1 <- deriv(x, y) # first derivative
  indices <- which(abs(d1) > threshold)
  return(indices)
}
`%notin%` <- Negate(`%in%`)



##############################################################################################
# fonction maison pour rechercher le type des données

## objectifs suivant différence entre int et float
typages_function <- function(df,nb_moda_max_fact = NULL ){
  if (is.null(nb_moda_max_fact)) {
    df %>%
      summarise_all(list(~n_distinct(na.omit(.)))) %>%
      t.df() %>% filter(col_1 > 2) %>%
      arrange(col_1) %>%
      print
    stop('Si la liste des facteurs n\'est pas fournis le nombre de modalité à partir
          duquel un facteur doit etre considéré comme un numéric avec `nb_moda_max_fact = `,
          \n pour vous aidez dans le choix du nombre de modalité la liste des variables
          avec plus de deux modalité différente est présenté au  dessus')}
  else {
    temp_moda_par_var <- df %>% summarise_all(list(~n_distinct(na.omit(.)))) %>%
      t.df() %>%
      mutate(binaire = col_1 == 2, numeric = col_1 >= nb_moda_max_fact,
             multinomial = (col_1 < nb_moda_max_fact & col_1 > 2)) %>%
      arrange(col_1)
    list_factor <- temp_moda_par_var %>%
      filter(multinomial) %>%
      select(key) %>%
      unlist(use.names = FALSE)
    liste_booleen <- temp_moda_par_var %>%
      filter(binaire) %>%
      select(key) %>%
      unlist(use.names = FALSE)
    liste_numeric <- temp_moda_par_var %>%
      filter(numeric) %>%
      select(key) %>%
      unlist(use.names = FALSE)
    res <- list(colonne_type = list(facteur = list_factor,
                                    booleen = liste_booleen,
                                    numerique = liste_numeric),
                data_frame_tot = temp_moda_par_var)
  }
  return(res)
}
##############################################################################################
# fonction maison pour le one hot encoding

one_hot_fb <- function(df, nb_moda_max_fact = NULL, list_factor = NULL){
```

```r
  if (is.null(list_factor)) {
    list_factor <- typages_function(df, nb_moda_max_fact)$colonne_type$facteur
    cat("Le nombre maximum de modalité par facteur est de ",
        nb_moda_max_fact,
        "\n pour supprimer ce warning utilisez `list_factor = ", "c( ",
        paste0("\"",list_factor,"\"",collapse = " , "),
        " )` \n au lieu de `nb_moda_max_fact = ", nb_moda_max_fact,"`")
  }



  df <- df %>% mutate_at( list_factor,as.factor)
  dmy <- dummyVars(paste0(" ~ ", paste0(list_factor,collapse = " + ")), data = df)
  trsf <- data.frame(predict(dmy, newdata = df))
  res <- df %>% select(-all_of(list_factor)) %>% cbind(trsf) # ajout all_of retirer si bug
  return(res)
  # reste a ajouter une partie qui renomme les varibles mieux
}


##############################################################################################
# function récupérant les variables pour table descriptive des variables

# df_one_hot_encode_fonc généré avec one_hot_fb

recup_var_table_res <- function(df_one_hot_encode_fonc,name_expo_fonc){
  res <- df_one_hot_encode_fonc %>%
    select(-all_of(name_expo_fonc)) %>%
    mutate_if(is.factor, ~as.numeric(as.character(.))) %>% # points litigieux j'utilise cette méthode p
    summarise_all(list(fonc_med = ~median(.,na.rm = TRUE),
                       fonc_quart1 = ~quantile(.,0.25,na.rm = TRUE),
                       fonc_quart2 = ~quantile(.,0.75,na.rm = TRUE),
                       fonc_n = ~sum(.,na.rm = TRUE),
                       fonc_pourcent = ~mean(.,na.rm = TRUE)*100,
                       fonc_nb_NA = ~sum(is.na(.))
    )
    ) %>%
    pivot_longer(cols =  everything(),
                 names_to = c(".value", "level"),
                 names_pattern = "(.*)_fonc_(.*)") %>%
    t.df(.,"level")
  return(res)}

##############################################################################################
# table descriptive des variables

# df_one_hot_encode_fonc généré avec one_hot_fb
# Version prévue pour échapper les caracter latex
table_resume_latex <- function(df_one_hot_encode_fonc,name_expo_fonc,
                               nom_grp = "clusters",
                               p_val = FALSE,
                               arrondie = TRUE, alpha = 0.05) {
```

```r
# Typage des variables a réusmer en booleen ou numeric
# Car normalement df pré_one hot encode
table_bool_var <-   df_one_hot_encode_fonc %>%
  select(-all_of(name_expo_fonc)) %>%
  summarise_all(list(~n_distinct(., na.rm = TRUE))) %>%
  t.df %>%
  rename(booleen = col_1) %>%
  {temp_verif_bool <<- .} %>%
  mutate(booleen = booleen <= 2)


if (any(temp_verif_bool$booleen < 2)) {
  print(temp_verif_bool$key[temp_verif_bool$booleen < 2])
  stop("moins de deux valeurs distinct pour une variables")}



name_all_grp <- paste0("all ",nom_grp)


all_cluster_descript_var <- df_one_hot_encode_fonc %>%
  recup_var_table_res(name_expo_fonc) %>%
  #{ifelse(arrondie ,arrondie_df(.), . )} %>% print %>%

  mutate(nb_NA = ifelse(nb_NA == 0,"",paste0("NA:", nb_NA )))


nb_grp <- df_one_hot_encode_fonc %>% select(all_of(name_expo_fonc)) %>%
  unique() %>%
  unlist(use.names = FALSE) %>%
  sort

group_cluster_descript_var <- lapply(nb_grp, function(x) {
  col_name <- paste0(nom_grp,"_",x)
  res <- df_one_hot_encode_fonc  %>%
    filter(!!sym(name_expo_fonc) == x) %>%
    recup_var_table_res(name_expo_fonc)  %>%
    mutate(nb_NA = ifelse(nb_NA == 0,"",paste0("NA:", nb_NA ))) %>%
    inner_join(table_bool_var,by = "key") %>%
    mutate({{col_name}} := ifelse( booleen
                                   , paste0(n ,"(",round(pourcent,1), "%)", nb_NA),
                                   paste0(round(med,0) ,"(",
                                          round(quart1,0),";",
                                          round(quart2,0), ")", nb_NA))) %>%
    select(all_of(col_name))
  return(res)
}
)

table_res <- all_cluster_descript_var %>%
  inner_join(table_bool_var,by = "key") %>%
  mutate( {{name_all_grp}} := ifelse(booleen
                                     , paste0(n ,"(",round(pourcent,1), "%)",  nb_NA),
```

```
                                    paste0(round(med,0) ,"(",
                                           round(quart1,0),";",
                                           round(quart2,0), ")", nb_NA) ) )  %>%
  select(key,all_of(name_all_grp)) %>%
  cbind(bind_cols(group_cluster_descript_var)) %>%
  data.frame(., row.names = 1)
 # rename_at(vars(contains(".grp")),funs(str_replace(.,"\\."," ")))


table_res <- table_res %>% rename_all(list(~str_replace_all(.,"\\."," ")))
# si choix de calculer les p-val
if (p_val) {
  # prevoir un groupe de plus pour le toutes les catégorie
  nb_group_pval <-  as.character(c(as.numeric(nb_grp),max(as.numeric(nb_grp)) + 1 ))
  # création de toutes les combinaisons de groupe a tester
  combin_grp <- nb_group_pval %>%  combn(2)
  # Création du groupe supplémentaire tout les groupes en dupliquant la dataframe avec un  groupes de
  df_pval <- df_one_hot_encode_fonc %>%
    mutate(!!sym(name_expo_fonc) := max(as.numeric(nb_group_pval))) %>%
    rbind(df_one_hot_encode_fonc)

  non_boolean_var <- table_bool_var %>% filter(!booleen) %>%
    select(key) %>% unlist(use.names = FALSE)
  boolean_var <- table_bool_var %>% filter(booleen) %>%
    select(key) %>% unlist(use.names = FALSE)
  # création  de la table avec p-value pour chaque combin et rename chaque colonnes a_b
  combin_ttest_pval <- apply(combin_grp, 2, function(x)
    df_pval %>%
      select(sym(name_expo_fonc),all_of(non_boolean_var)) %>%
      #summarise_at(vars(-(sym(name_expo_fonc))),list(~t.test(.[!!sym(name_expo_fonc) == x[1]], .[!!s
      summarise_at(vars(-(sym(name_expo_fonc))),
                   list(~t.test(.[!!sym(name_expo_fonc) == x[1]],
                                .[!!sym(name_expo_fonc)  ==  x[2]])$p.value)) %>%
      t.df %>%
      rename_at("col_1",list( ~paste0(x[1],"_",x[2])))
  )
  combin_chisq_pval <- apply(combin_grp, 2, function(x)
    df_pval %>%
      select(sym(name_expo_fonc),all_of(boolean_var)) %>%
      dplyr::summarise_at(vars(-sym(name_expo_fonc)),

                          list(~ifelse(sum(.[!!sym(name_expo_fonc) == x[1]],
                                           na.rm = TRUE) < 8|
                                         sum(.[!!sym(name_expo_fonc)  ==  x[2]],
                                             na.rm = TRUE) < 8,
                                       NA,
                                       prop.test(
                                         x = c(sum(.[!!sym(name_expo_fonc) == x[1]],
                                                   na.rm = TRUE),
                                               sum(.[!!sym(name_expo_fonc)  ==  x[2]],
                                                   na.rm = TRUE)), # compute number of success
                                         n = c(sum(!is.na(.[!!sym(name_expo_fonc) == x[1]])),
                                               sum(!is.na(.[!!sym(name_expo_fonc)  ==  x[2]])))
```

```
                                        )$p.value))
    ) %>%
    t.df %>%
    rename_at("col_1",list( ~paste0(x[1],"_",x[2])))
)


combin_total_pval <- mapply(rbind,combin_chisq_pval,combin_ttest_pval,SIMPLIFY=FALSE)

# transformation de la p-value en booléen en avec comme seuil le alpha définis en appliquant une co
result_pval <- bind_cols(combin_total_pval) %>%
  rename(key = key...1) %>%
  select(key,contains("_")) %>%
  mutate_at(vars(-key),list(~(. < (alpha / ncol(combin_grp))
  ))
  ) %>%  # hypothèse et correction de bonneferonnie
  mutate_at(vars(-key), function(x) {
    x_var <- rlang::enquo(x)
    ifelse(x , rlang::quo_name(x_var), "non") # remplacement des p-val non signif par une chaine sp
  }) %>%
  mutate_at(vars(-key), function(x) {
    x_var <- rlang::enquo(x)
    ifelse(is.na(x) , paste0(rlang::quo_name(x_var),"*"),x) # remplacement des p-val non signif par
  })

# REcherche avec une simili boucle des p-val signif pour chaque colonnnes
# on en lève la chaine spécifique de non corrélation
df_pval_final <- lapply(nb_group_pval, function(x) {
  result_pval %>% select(key,contains(x)) %>%
    mutate_at(vars(contains(x)),list(~str_remove_all(.,
                                                     paste(c("_",x),
                                                           collapse = "|")))) %>%
    unite(!!sym(paste0(nom_grp,"_",x)) ,contains(x),sep = ",")
}
) %>% bind_cols() %>%
  rename(key = key...1) %>%
  select(key,contains(nom_grp)) %>%
  rename(!!sym(name_all_grp) := paste0(nom_grp,"_",
                                       max(as.numeric(nb_group_pval)))) %>%
  mutate_all(list(~str_remove_all(.,"non,|,non"))) %>%
  mutate_all(list(~str_remove_all(.,"non")))

if(df_pval_final %>%  transmute_at(vars(-key),
                                   list(~str_detect(.,"non"))) %>%
   as.matrix() %>% any) {
  stop("il reste des p-val non traité")}

# Gestion des tables latex pour que les différences statisquement significative soit en subscript
# en échappant les underscore
table_res_pval <-  table_res %>%
  rownames_to_column() %>%
  pivot_longer(-rowname,values_to = "valeur") %>%
  inner_join((df_pval_final %>% pivot_longer(-key,values_to = "pvalue") ),
```

```
                  by = c("rowname" = "key", "name" = "name")) %>%
      mutate(combin = paste0(valeur,"\\textsubscript{",pvalue, "}")) %>%
      select(rowname,name,combin) %>%
      pivot_wider(names_from = name, values_from = combin) %>%
      column_to_rownames()

    table_res_pval <- table_res_pval %>%
      rownames_to_column() %>%
      mutate_at(vars(-rowname),list(~str_replace_all(.,"%","\\\\%"))) %>%
      column_to_rownames() %>%
      #mutate_all(funs(str_replace_all(.,"%","\\\\%"))) %>%
      select(all_of(name_all_grp),sort(tidyselect::peek_vars()))


    rownames(table_res_pval) <- str_replace_all(rownames(table_res_pval),"_","\\\\_")
    colnames(table_res_pval) <- str_replace_all(colnames(table_res_pval),"_","\\\\_")


  } else {table_res_pval <- table_res %>%
    select(all_of(name_all_grp),sort(tidyselect::peek_vars()))  }

  nb_pat_par_grp <- c(nrow(df_one_hot_encode_fonc),
                      table(df_one_hot_encode_fonc[,name_expo_fonc]))
  res <- rbind(`Number of patient` = nb_pat_par_grp,table_res_pval)
  return(res)
}

##########################################################################################
##########################################################################################
# table descriptive des variables

# df_one_hot_encode_fonc généré avec one_hot_fb
# attention la version p-val true et prevue pour etre print latex ss escape
table_resume_html <- function(df_one_hot_encode_fonc,
                              name_expo_fonc,
                              nom_grp = "clusters",
                              p_val = FALSE,
                              arrondie = TRUE,
                              alpha = 0.05) {

  # Typage des variables a réusmer en booleen ou numeric
  # Car normalement df pré_one hot encode
  table_bool_var <-   df_one_hot_encode_fonc %>%
    select(-all_of(name_expo_fonc)) %>%
    summarise_all(list(~n_distinct(., na.rm = TRUE))) %>%
    t.df %>%
    rename(booleen = col_1) %>%
    {temp_verif_bool <<- .} %>%
    mutate(booleen = booleen <= 2)


  if (any(temp_verif_bool$booleen < 2)) {
    print(temp_verif_bool$key[temp_verif_bool$booleen < 2])
```

```r
  stop("moins de deux valeurs distinct pour une variables")}



name_all_grp <- paste0("all_",nom_grp)


all_cluster_descript_var <- df_one_hot_encode_fonc %>%
  recup_var_table_res(name_expo_fonc) %>%
  #{ifelse(arrondie ,arrondie_df(.), . )} %>% print %>%
  mutate(nb_NA = ifelse(nb_NA == 0,"",paste0(" NA : ", nb_NA )))


nb_grp <- df_one_hot_encode_fonc %>% select(all_of(name_expo_fonc)) %>%
  unique() %>%
  unlist(use.names = FALSE) %>%
  sort


group_cluster_descript_var <- lapply(nb_grp, function(x) {
  col_name <- paste0(nom_grp,"_",x)
  res <- df_one_hot_encode_fonc  %>%
    filter(!!sym(name_expo_fonc) == x) %>%
    recup_var_table_res(name_expo_fonc) %>%
    inner_join(table_bool_var,by = "key") %>%
    mutate(nb_NA = ifelse(nb_NA == 0,"",paste0(" NA:", nb_NA ))) %>%
    mutate( {{col_name}} := ifelse( booleen
                                    , paste0(n ,"(",round(pourcent,1), "%)", nb_NA),
                                    paste0(round(med,2) ,"(",
                                           round(quart1,1),";",
                                           round(quart2,1), ")", nb_NA))) %>%
    select(all_of(col_name))
  return(res)
}
)

table_res <- all_cluster_descript_var %>%
  inner_join(table_bool_var,by = "key") %>%
  mutate( {{name_all_grp}} := ifelse(booleen
                                , paste0(n ,"(",round(pourcent,1), "%)",  nb_NA),
                                paste0(round(med,2) ,"(",
                                       round(quart1,1),";",
                                       round(quart2,1), ")", nb_NA) ) )  %>%
  select(key,all_of(name_all_grp)) %>%
  cbind(bind_cols(group_cluster_descript_var)) %>% data.frame(., row.names = 1)

table_res <- table_res %>% rename_all(list(~str_replace_all(.,"\\."," ")))
# si choix de calculer les p-val
if (p_val) {
  # prevoir un groupe de plus pour le toutes les catégorie
  nb_group_pval <-  as.character(c(as.numeric(nb_grp),max(as.numeric(nb_grp)) + 1 ))
  # création de toutes les combinaisons de groupe a tester
  combin_grp <- nb_group_pval %>%  combn(2)
```

```r
# Création du groupe supplémentaire tout les groupes en dupliquant la dataframe avec un  groupes de
df_pval <- df_one_hot_encode_fonc %>%
  mutate(!!sym(name_expo_fonc) := max(as.numeric(nb_group_pval))) %>%
  rbind(df_one_hot_encode_fonc)


non_boolean_var <- table_bool_var %>% filter(!booleen) %>% select(key) %>% unlist(use.names = FALSE)
boolean_var <- table_bool_var %>% filter(booleen) %>% select(key) %>% unlist(use.names = FALSE)

# création  de la table avec p-value pour chaque combin et rename chaque colonnes a_b
combin_ttest_pval <- apply(combin_grp, 2, function(x)
  df_pval %>%
    select(sym(name_expo_fonc),all_of(non_boolean_var)) %>%
    #summarise_at(vars(-(sym(name_expo_fonc))),list(~t.test(.[!!sym(name_expo_fonc) == x[1]], .[!!sy
    summarise_at(vars(-(sym(name_expo_fonc))),list(
      ~t.test(.[!!sym(name_expo_fonc) == x[1]],
              .[!!sym(name_expo_fonc)  ==  x[2]])$p.value)) %>%
    t.df %>%
    rename_at("col_1",list( ~paste0(x[1],"_",x[2])))
)
combin_chisq_pval <- apply(combin_grp, 2, function(x)
  df_pval %>%
    select(sym(name_expo_fonc),all_of(boolean_var)) %>%
    dplyr::summarise_at(vars(-sym(name_expo_fonc)),

                        list(~ifelse(sum(.[!!sym(name_expo_fonc) == x[1]],
                                         na.rm = TRUE) < 8|
                                       sum(.[!!sym(name_expo_fonc)  ==  x[2]],
                                           na.rm = TRUE) < 8,
                                     NA,
                                     prop.test(
                                x = c(sum(.[!!sym(name_expo_fonc) == x[1]],
                                          na.rm = TRUE),
                                      sum(.[!!sym(name_expo_fonc)  ==  x[2]],
                                          na.rm = TRUE)), # compute number of success
                                n = c(sum(!is.na(.[!!sym(name_expo_fonc) == x[1]])),
                                      sum(!is.na(.[!!sym(name_expo_fonc)  ==  x[2]])))
                        )$p.value))
    ) %>%
    t.df %>%
    rename_at("col_1",list( ~paste0(x[1],"_",x[2])))
)


combin_total_pval <- mapply(rbind,combin_chisq_pval,combin_ttest_pval,SIMPLIFY=FALSE)

# transformation de la p-value en booléen en avec comme seuil le alpha définis en appliquant une co
result_pval <- bind_cols(combin_total_pval) %>%
  rename(key = key...1) %>%
  select(key,contains("_")) %>%
  mutate_at(vars(-key),list(~(. < (alpha / ncol(combin_grp))
  ))
  ) %>%  # hypothèse et correction de bonneferonnie
```

```
    mutate_at(vars(-key), function(x) {
      x_var <- rlang::enquo(x)
      ifelse(x , rlang::quo_name(x_var), "non") # remplacement des p-val non signif par une chaine sp
    }) %>%
    mutate_at(vars(-key), function(x) {
      x_var <- rlang::enquo(x)
      ifelse(is.na(x) , paste0(rlang::quo_name(x_var),"*"),x) # remplacement des p-val non signif par
    })

  # REcherche avec une simili boucle des p-val signif pour chaque colonnnes
  # on en lève la chaine spécifique de non corrélation
  df_pval_final <- lapply(nb_group_pval, function(x) {
    result_pval %>% select(key,contains(x)) %>%
      mutate_at(vars(contains(x)),list(~paste0("~",str_remove_all(.,paste(c("_",x),
                                                                          collapse = "|")),
                                                 "~"))) %>%
      unite(!!sym(paste0(nom_grp,"_",x)) ,contains(x),sep = "~,~ ")
  }
  ) %>% bind_cols() %>%
    rename(key = key...1) %>%
    select(key,contains(nom_grp)) %>%
    rename(!!sym(name_all_grp) := paste0(nom_grp,"_",
                                         max(as.numeric(nb_group_pval)))) %>%
    mutate_all(list(~str_remove_all(.,"~non~~,~ |~,~~non~"))) %>%
    mutate_all(list(~str_remove_all(.,"~non~")))

  if(df_pval_final %>%
     transmute_at(vars(-key),list(~str_detect(.,"non"))) %>%
     as.matrix() %>% any) {
    stop("il reste des p-val non traité")}

  # Gestion des tables latex pour que les différences statisquement significative soit en subscript
  # en échappant les underscore
  table_res_pval <-  table_res %>%
    rownames_to_column() %>%
    pivot_longer(-rowname,values_to = "valeur") %>%
    inner_join((df_pval_final %>% pivot_longer(-key,values_to = "pvalue") ),
               by = c("rowname" = "key", "name" = "name")) %>%
    mutate(combin = paste0(valeur,pvalue)) %>%
    select(rowname,name,combin) %>%
    pivot_wider(names_from = name, values_from = combin) %>%
    column_to_rownames()

  table_res_pval <- table_res_pval %>%
    rownames_to_column() %>%

    column_to_rownames() %>%
    select(all_of(name_all_grp),sort(tidyselect::peek_vars()))


} else {table_res_pval <- table_res %>%
  select(all_of(name_all_grp),sort(tidyselect::peek_vars()))  }
```

```r
  nb_pat_par_grp <- c(nrow(df_one_hot_encode_fonc),
                      table(df_one_hot_encode_fonc[,name_expo_fonc]))
  res <- rbind(`Number of patient` = nb_pat_par_grp,table_res_pval)
  return(res)
}

##############################################################################################
function_box_plot_expo_func <- function(donne,expo,out_come){
  `%>%` <- dplyr::`%>%`
  donne <- donne %>% as.data.frame()
  exposure <- donne[,expo] %>% as.factor()
  test_equal_moy <-  donne %>% dplyr::mutate(expo_fact = as.factor(exposure))  %>%
    dplyr::select(-all_of(expo))


  # Création du box plot simple
  plot_test_res <- test_equal_moy %>%
    ggplot2::ggplot( ggplot2::aes(x = expo_fact, y = get(out_come), color = expo_fact)) +
    ggplot2::geom_boxplot()  +
    ggplot2::labs(y = paste0("Final Outcome",
                             out_come),
                  x = paste0("Treatment groups ",
                             expo) ,
                  caption = "**** = 0, *** < 0.0001, ** < 0.001, * < 0.05") +
    ggplot2::guides(colour=FALSE) +
    viridis::scale_color_viridis(discrete= TRUE,
                                 option = "D")



  if (length(levels(test_equal_moy$expo_fact)) == 2) { # Si deux modalité T test
    moy1 <- test_equal_moy %>% dplyr::filter(expo_fact == levels(expo_fact)[1]) # création var t test
    moy2 <- test_equal_moy %>% dplyr::filter(expo_fact == levels(expo_fact)[2])# création var t test
    res_testt <-  t.test(moy1[,out_come],moy2[,out_come]) # test
    sum_ttest <- summary(res_testt) # récupération résultat

    plot_test_res <- plot_test_res +
      # ajout de la significativité au box plot
      ggpubr::stat_compare_means(comparisons = list(c(1,2)), tip.length=0.01,
                                 label = "p.signif",
                                 symnum.args = list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, 1),
                                                    symbols = c("****", "***", "**", "*", "ns")))

    # préparation de la liste de sortie de la ffonction
    test_moy <- list(res_test = res_testt,
                     plot_test = plot_test_res,
                     res_annexe = list(summary_ttest = sum_ttest))

  } else if (length(levels(test_equal_moy$expo_fact)) > 2) { # si plus de 2 facteurs anova nécessaire
    # Vérifications des hypothèses abandonné kruskalwallis tout le temps la mais a l'avenir reflexion s
    # potentiellement vérif hypo anova clairement pas vérifiables

    # Anova test pour produire des résultat annexe notament test de thukey et leven  test
```

```
anova_T_temp <- aov(eval(parse(text =
                                  paste(out_come,"~","expo_fact", # récupération des paramètre de la
                                        sep = "")
)), data = test_equal_moy)

# Tukey test
tukey_anov <- TukeyHSD(anova_T_temp)
# leven test
leven_T <- car::leveneTest(eval(parse(text =
                                    paste(out_come,"~","expo_fact", # récupération des paramètr
                                          sep = "")
)), data = test_equal_moy)
# kruskal_waliis car pas vérif hypo trop compliqué
kurskal_T <- kruskal.test(eval(parse(text =
                                  paste(out_come,"~","expo_fact", # récupération des paramètre
                                        sep = "")
)), data = test_equal_moy)
sum_anova <- summary(kurskal_T)


# Tableau regroupant les moyenne par groupe de facteur d'exposition
moyenne_par_grp <- test_equal_moy %>%
  dplyr::ungroup() %>%
  dplyr::group_by(expo_fact) %>%
  dplyr::summarise_at(.vars = dplyr::vars(dplyr::all_of(out_come)),  .funs = list(moy = ~mean(.)))

# recherche de tout les comparaison effectuer par le test de tukey
liste_facteur_compare_tukey <- rownames(tukey_anov$expo_fact) %>%
  str_extract_all(".+(?=-)|(?<=-).+") %>%
  lapply(., function(x) factor(x, levels = levels(exposure)))
#   stringr::str_extract_all("[:digit:]+")

# Boolén visant a ne garder que les les comparaisons des colones au colones adjacente
# 1 avec 2 Vrai
#2 avec 5 faux
bool_facteur_adj <- lapply(liste_facteur_compare_tukey,
                           function(x) diff(as.numeric(x))) %>%
  unlist() %>% abs(.) == 1

# Récupération des comparaison vrai
temp_compare_boxplot  <- liste_facteur_compare_tukey[bool_facteur_adj] %>% lapply(., function(x) as

plot_test_res <- plot_test_res +
  ggpubr::stat_compare_means(comparisons = temp_compare_boxplot, tip.length=0.01,
                             label = "p.signif",
                             symnum.args = list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, 1),
                                                symbols = c("****", "***", "**", "*", "ns")))

test_moy <- list(res_test = kurskal_T, # les résultat de kruskalwallis
                 plot_test = plot_test_res, # les box plot
                 res_annexe = list(summary_anov = sum_anova , # regroupe les éléments liée a l'anov
                                   ano_va_ss_hypo = list(tukey = tukey_anov,
                                                         anova = anova_T_temp,
```

```
                                                          leven_test = leven_T),
                                      moy_grp = moyenne_par_grp))

  } else print(paste0("pas de test pour facteur avec ",
                      length(unique(test$INS_obs_categ))," modalitée(s)"))

  return(test_moy)
}


#exemple


# load("exemple_sebastion_df_exemple.RData") # en exemple tu as les 1000 premiere ligne de mo jeux de d
# eval_decoupage_cluster <- function_box_plot_expo_func(donne = exemple_box_plot, # data_frame
#                                                       expo = "INS_obs_categ", # ,nom entre quote de l
#                                                       out_come = "SYM_echelleEpworth") # nom entre qu
# eval_decoupage_cluster$plot_test



##############################################################################################
#print vector
print.vecteur <- function(x){
  for (name in x) {
    cat("-", name, '\n')
  }
}
##############################################################################################

# transposé dataframe
t.df <- function(df,pivot=NULL){
  if (is.null(pivot)){
    pivot <- "row_id"
    df <- df %>% mutate(row_id=paste0("col_",1:nrow(df) ))
  }
  res <- df %>% pivot_longer(cols = -!!pivot,"key","value") %>%
    pivot_wider(names_from = !!pivot,values_from = value)
  return(res)
}




hist_bins <- function(x){
  bw <- 2 * IQR(x) / length(x)^(1/3)
  return(bw)}


##############################################################################################

# compte le nombres de valeur manquante par variables avec les individus groupé par une variables
compte_na_par_var_par_grp <- function(df,group_col,colonnes){
```

```r
  df_NA_var_fonc <- df %>% select(all_of(group_col),all_of(colonnes)) %>% setDT
  nb_val_manq_par_var <- df_NA_var_fonc %>%
    .[, lapply(.SD,  function(x) sum(is.na(x))), group_col] %>%
    select(-all_of(group_col)) %>%
    gather(name,presence_na) %>%  # reshape datset
    count(name, presence_na) %>%  # count combinations
    pivot_wider(names_from = name,
                values_from = n,
                values_fill = list(n = 0))
  return(nb_val_manq_par_var)
}

##############################################################################################
# test les hypothèse d'une anova

model.line.hypo <- function(model_test_hypo){
  #shapi<-shapiro.test(model$residuals)
  shapi <- suppressWarnings(ks.test(x=model_test_hypo$residuals,y='pnorm'))
  bartletI <- bartlett.test(residuals(model_test_hypo)~
                              I(model_test_hypo$model[,2]:
                                  model_test_hypo$model[,ncol(model_test_hypo$model)])
                            )$p.value
  if (shapi$p.value > 0.05){

    res1<-paste("On ne peut pas rejeter l'hypothèse H0 de normalité des residus la p-value
              du test de shapiro étant de ",
                formatC(shapi$p.value , format = "e", digits = 2) ,
                "\nce qui est superieur au seuil alpha 5%")
  } else {
    res1 <- paste("On rejete l'hypothèse H0 de normalité des residus la p-value
                du test de shapiro étant de ",
                  formatC(shapi$p.value , format = "e", digits = 2) ,
                  "\nce qui est inférieur au seuil alpha 5%")
  }
  if (bartletI>0.05){
    bartlet <- c()
    for (i in 1:ncol(m$model[,-1])){
      bartlet <- c(bartlet,bartlett.test(m$model[,1],m$model[,i+1])$p.value)

    }
    res2 <- paste(
    "On ne peut pas rejeter l'hypothèse H0 d'homocédasticité des interactions la p-value
                du test de bartlet étant de ",
                  formatC(bartletI , format = "e", digits = 2) ,
                  "\nce qui est superieur au seuil alpha 5%")
  }  else {
    bartlet <- 0
    res2 <- paste(
    "On rejete l'hypothèse H0 d'homocédasticité des interactions la p-value du test
                de bartlet étant de ",
                  formatC(bartletI , format = "e", digits = 2) ,
                  "\nce qui est inférieur au seuil alpha 5%")
  }
```

```
  if (min(bartlet) > 0.05){res3 <- paste(
  "On ne peut pas rejeter l'hypothèse H0 d'homocédasticité des variances la plus petite
                                    p-value du test de bartlet étant de ",
                                        formatC(min(bartlet) , format = "e", digits = 2) ,"\nce qui es
  }  else {
    res3 <- paste(
    "On   rejete l'hypothèse H0 d'homocédasticité des variances la plus petite
                p-value du test de bartlet étant de ",
                  formatC(min(bartlet) , format = "e", digits = 2) ,
                  "\nce qui est inférieur au seuil alpha 5%")
  }
  if (((shapi$p.value>0.05) == TRUE) &
      ((bartletI > 0.05) == TRUE) &
      ((min(bartlet) > 0.05) == TRUE)) {
    resF <- "On accepte toutes les hypothèses du test d'annova à plus de 2 facteurs \n"}
  else {
    resF <- "On rejette au moins une hypothèse"
  }
  res <- paste(res1,res2,res3,resF,sep = '\n \n')
  return(cat(res))
}


##############################################################################################
# reset param graphique
resetPar <- function() {
  dev.new()
  op <- par(no.readonly = TRUE)
  dev.off()
  op
}
#usage
#par(resetPar())
##############################################################################################
# fait des arrondie + notation scientifique pour les nombres a virugules dans les DF en
# conservant les integer telquel
arrondie_df <- function(df_func){
  res <-   df_func %>%
    rownames_to_column() %>%
    mutate_if(is.numeric,
              # ancien avec probablement un problème dans l'ordre des ifelse
               # ~ifelse(.%%1==0,as.character(round(.,0)),ifelse((. > 10^3| 1/abs(.) > 10^3 ),
               #         formatC(., format = "e", digits = 1), # tentative de gestion des nombres
               #         as.character(round(.,3))
               #         )
               # )
              ~ifelse(.%%1==0 & . < 10^3,as.character(round(.,0)),ifelse((. > 10^3| 1/abs(.) > 10^3 ),
                                                                         formatC(., format = "e", digits = 1), # te
                                                                         as.character(round(.,3))

              )
              )
    ) %>%
    column_to_rownames()
```

```r
  return(res)
}
#############################################################################################
# convertie les heures  avec virgule en heure et minute
roud_hour <- function(decimal_hour){
  heure <- floor(decimal_hour)
  minutes <- round(60 * (decimal_hour - floor(decimal_hour)), 0)
  res <- sprintf("%02d h %02d min", heure, minutes)
  return(res)
}


#############################################################################################
# liste les  colonnes iodentiques
column_comparator <- function(col_a_compar,df_compar_func){ # recherche les collonnes identique et les
  # Attention une colonnes restante étant égale a 2 coçlonnes supprimé génère donc deux liste
  liste_colum_identique <-   lapply(seq_along(df_compar_func),function(x) {
    col_a_compar_temp <- df_compar_func[,x] %>% unlist( use.names = FALSE)
    column_compared <- (col_a_compar_temp == col_a_compar ) |
      (is.na(col_a_compar_temp) & is.na(col_a_compar ))
    matching_col <- c(names(df_compar_func[,x]),names(which(apply(column_compared,2,all))))
  })
  liste_colum_identique  <- lapply(liste_colum_identique, function(x) x[length(x) > 1]) %>% compact()
  return(liste_colum_identique)
}
#############################################################################################
# retire les colonnes iodentiques
distinc_col <- function(df_func,return_list_col_supr = TRUE){
  column_unique_df <- df_func %>% t.df() %>% distinct_at(vars(-key),.keep_all = TRUE)
  df_col_unique <- df_func %>% select(all_of(column_unique_df$key))

  if (return_list_col_supr) {
    col_supr <- colnames(df_func)[colnames(df_func) %notin% column_unique_df$key]
    df_col_supr <- df_func %>% select(all_of(col_supr))


    liste_col_supr <- column_comparator(df_col_supr,df_col_unique)

    res <- list(df = df_col_unique, colonne_suprime =  liste_col_supr)
  } else {res <- df_col_unique}
  return(res)
}
#############################################################################################
# include svg en pdf

include_svg = function(path) {
  if (knitr::is_latex_output()) {
    output = xfun::with_ext(path, 'pdf')
    # you can compare the timestamp of pdf against svg to avoid conversion if necessary
    system2('rsvg-convert', c('-f', 'pdf', '-a', '-o', shQuote(c(output, path))))
  } else {
    output = path
  }
```

```
  knitr::include_graphics(output)
}
```

```
##not in##
#############################################################################################

`%notin%` <- Negate(`%in%`)
##flow char##
#############################################################################################


flow_chart <- function(base,critere,patient_perdu,table_grp) {
  nb_crit <- length(critere)

  debut_du_plot <- paste0("digraph {
node [fontname = Helvetica, shape = rectangle]; \n",
                          paste0("base; ",paste0("critere_",letters[1:nb_crit], collapse = "; "),"; ",
                                 paste0("perte_",letters[1:nb_crit], collapse = "; "),"\n"),
                          paste0("secret_node",c(1:nb_crit) ,
                                 "[height=0, width=0, margin=0,shape=point, style=invis]; \n",collapse =
                          #paste0("secret_node",c(1:nb_crit), collapse = "; ")
  )



  elements <- paste0("base [label = '",  "Patient in OSFP base (n = ",
                     base ," )","'] \n",
                     paste0("critere_",letters[1:nb_crit], "  [label = '",
                            "Remaining Patients  (n = ",
                            base - cumsum(patient_perdu), " )" ,"' ]", collapse = " \n"),"\n",
                     paste0("perte_",letters[1:nb_crit], "  [label = '", critere ,  " (n = ",
                            patient_perdu, " )","' ]", collapse = " \n"),"\n",
                     paste0("group_obs_",letters[1:length(table_grp)], "  [label = '","Number of patien
                            table_grp, " ) \n" ,round((table_grp * 100 )/(base - sum(patient_perdu)),2)

  )

  rang <- paste0(paste0("{rank=same;","secret_node",c(1:nb_crit)," ",
                        "perte_",letters[1:nb_crit],"}",collapse = " \n"),"\n",
                 "{rank=same;",paste0("group_obs_",letters[1:length(table_grp)],collapse = "  "),"}",
                 "\n")


  lien <- paste0(
    paste0("base -> secret_node1 [arrowhead = none] \n", collapse = "") ,
    paste0("secret_node",c(1:nb_crit)," -> ","critere_",
           letters[1:nb_crit]," \n", collapse = "") ,
    paste0("critere_",letters[1:(nb_crit-1)]," -> ","secret_node",
           c(2:nb_crit)," [arrowhead = none] \n", collapse = "") ,
    paste0("secret_node",c(1:nb_crit) ," -> ", "perte_",
           letters[1:nb_crit], collapse = "\n")," \n",
    paste0("critere_",letters[nb_crit] ," -> ", "group_obs_", letters[1:length(table_grp)],"[minlen='2.
    collapse = "")
```

```
  fin_plot <- " \n }"

  res <- paste0(debut_du_plot,elements,rang,lien,fin_plot)
  grViz(res)
  return(res)
  }

##function principale du stage##
#################################################################################################
calcule_pds_stage <- function(donne,expo,covar,out_come,percentile_tronc = c(0,1,5,10,25,50)/100 ){
  tempcall <- match.call()


  fun_trunc <- function(x,.probs) {
    pmin(pmax(x, quantile(x, probs = .probs)),
         quantile(x, probs = 1-.probs))}
  exposure <- donne[,as.character(tempcall$expo)]
  mod1 <-   multinom(formula =   eval(parse(text =
                                              paste("as.numeric(",deparse(tempcall$expo),")", paste0("~
                                                    sep = "")
  )),data = donne, na.action = na.fail ,trace = FALSE)

  res <- data.frame(exposition = exposure , PS = NA) %>%
    setNames(c("exposition", "PS"))


  res <- res %>%
    group_by(exposition) %>%
    mutate(n = n(),numerator = n / nrow(.)) %>% select(-n)


  proba_tps_inv <- predict(mod1, type = "probs") %>%
    as.data.frame() %>%
    rename("1" = names(.)[1]) %>%
    mutate("0" = 1 - apply(.,1,sum)) # ajout de la colonnes 0 pour vérifier que ça somme a 1

  res$PS <- sapply(1:nrow(res),
                           function(x) proba_tps_inv[x,as.character(as.numeric(res$exposition[x]))]) #


  #SW stabilized weight
  # W eight
  #PS propencenty score
  res <- res %>% mutate(SWeight = numerator/PS, Weight = 1/PS)


  test_moy <- function_box_plot_expo_func(donne,
                                          deparse(substitute(expo)),
                                          deparse(substitute(out_come))
                                          )

  plot_propencity <- res %>%
    ggplot(aes(x = PS)) +
```

```
  geom_histogram(binwidth = hist_bins(res$PS)) +
  facet_wrap( ~ exposition,ncol = 1) + theme_bw()


# moyenne et dispersion des poids par groupe tableau a but de présentation
moyenne_pds <- res %>% group_by(exposition)  %>%
  summarise("mean_W" = mean(Weight) ,
            "min_W" = min(Weight),
            "max_W" = max(Weight),
            "sd_W" = sd(Weight),
            "mean_SW" = mean(SWeight) ,
            "sd_SW" = sd(SWeight),
            "min_SW" = min(SWeight),
            "max_SW" = max(SWeight),
            .groups = "drop"
            ) %>%
  arrondie_df %>%
  #mutate_all(list(~as.character())) %>%
  transmute(`Adherence group` = exposition,
            Mean = paste0(mean_W," / ", mean_SW),
            `Standart deviation` = paste0(sd_W," / ", sd_SW),
            Minimum = paste0(min_W," / ", min_SW),
            Maximum = paste0(max_W," / ", max_SW)
            )


# troncature des poids
poid_trunc_df <- map(percentile_tronc,function(x) fun_trunc(res$Weight,x)) %>%
  as.data.frame() %>%
  structure(names = paste0("( ",
                           as.character(percentile_tronc),
                           ";",
                           as.character(1 - percentile_tronc),
                           " )" ) )

poid_trunc_stab_df <- map(percentile_tronc,function(x) fun_trunc(res$SWeight,x)) %>%
  as.data.frame() %>%
  structure(names = paste0("( ",
                           as.character(percentile_tronc),
                           ";",
                           as.character(1 - percentile_tronc),
                           " )" ) )

tableau_pds_trunc <- poid_trunc_df %>%
  summarise_all(list(~mean(.),
                     ~sd(.),
                     ~min(.),
                     ~max(.))) %>%
  t.df() %>%
  separate(key,
           into = c("truncations",
                    "fun"),
           sep = "_") %>%
```

```
    pivot_wider(names_from = fun ,
                values_from = col_1)  %>%
    rename_all(function(x) c("truncations","mean","standard deviation","minimum","maximum"))

  tableau_pds_trunc_stab <- poid_trunc_stab_df %>%
    summarise_all(list(~mean(.),
                       ~sd(.),
                       ~min(.),
                       ~max(.))) %>%
    t.df() %>%
    separate(key,
             into = c("truncations",
                      "fun"),
             sep = "_") %>%
    pivot_wider(names_from = fun ,
                values_from = col_1)  %>%
    rename_all(function(x) c("truncations","mean","standard deviation","minimum","maximum"))

  return(list(df = res,
              res_intermediaire = list(
                poids = list(
                  moyenne_pds = moyenne_pds,
                  poids_tronc = list(poids_trunc = poid_trunc_df,
                                     poids_trunc_stab = poid_trunc_stab_df),
                  summary_pds_trunc_stab = tableau_pds_trunc_stab,
                  summary_pds_trunc = tableau_pds_trunc,
                  plot_posit = plot_propencity),
                  regression_modele_pds = list(
                  regression_temps_ind = mod1,
                  data_frame_coef = proba_tps_inv),
                test_covar = test_moy)
  )
  )
}


##Imputation si changment de données##
#########################################################################################
# a ajouter parralélisation
impute_si_changement <- function(data_frame_a_verif,path,reimputation = FALSE){
  df_impute_ex <- tryCatch(read_rds(path),
                           error = function(e) data.frame())
  colanmes_manq_func <- data_frame_a_verif %>%
    select_if(data_frame_a_verif %>%
                summarise_all(list(~sum(is.na(.)))) != 0) %>%
    colnames()

  if ((all(sort(colnames(data_frame_a_verif)) == sort(colnames(df_impute_ex)) ) &
      (nrow(data_frame_a_verif) ==  nrow(df_impute_ex))) & !reimputation) {
    res <- df_impute_ex
  } else if (any(sort(colnames(data_frame_a_verif)) != sort(colnames(df_impute_ex))) | reimputation | ni
    cat("nécessité de réimputer les données
      ça va être long + ou - une heure")
```

```
    systeme_exploitation <- Sys.info()[['sysname']]
    if (systeme_exploitation == "Windows") {
    imputed_Data_func <- mice::mice(data_frame_a_verif,
                                    m = 10,
                                    maxit = 50,
                                    method = 'pmm',
                                    printFlag = FALSE)
    } else if (systeme_exploitation == "Linux") {
    imputed_Data_func <- mice::parlmice(data_frame_a_verif,
                                    n.imp.core = 2,
                                    n.core = parallel::detectCores(),
                                    m = 10,
                                    maxit = 50,
                                    method = 'pmm',
                                    printFlag = FALSE)
    } else if (systeme_exploitation == "Darwin") {
      cat("non testé sur mac donc pas de parrallélisation")
      imputed_Data_func <- mice::mice(data_frame_a_verif,
                                      m = 10,
                                      maxit = 50,
                                      method = 'pmm',
                                      printFlag = FALSE)
    } else {
      cat("System d'exploitation n'appartenant pas a windows/linux
          donc pas de parrallélisation")
      imputed_Data_func <- mice::mice(data_frame_a_verif,
                                      m = 10,
                                      maxit = 50,
                                      method = 'pmm',
                                      printFlag = FALSE)
    }
    saveRDS(imputed_Data_func, file = paste0("data/genere/imputation_object.rds"))
    if (!is.null(imputed_Data_func$loggedEvents)) {
      print("imputation avec warning")
      print(imputed_Data_func$loggedEvents)}
    pdf(file = "graph/Imputation_plot.pdf",width = 32, height = 18 ,onefile = TRUE)
    mice::densityplot(imputed_Data_func, data = as.formula(paste0("~",paste0(colanmes_manq_func,collaps
    )
    dev.off()
    pdf(file = "graph/manquant_plot.pdf",width = 32, height = 18 ,onefile = TRUE)
    visdat::vis_miss(data_frame_a_verif %>% select(all_of(colanmes_manq_func)), cluster = TRUE)
    dev.off()
    res <- mice::complete(imputed_Data_func)
    saveRDS(res, file = paste0("data/genere/data_impute.rds"))
  } else {stop("Condition non remplie problème fonction")}
return(res)
}

##############################################################################################
# function récupérant les variables pour table descriptive des variables

# df_one_hot_encode_fonc généré avec one_hot_fb
```

```r
recup_var_table_verif_impute <- function(df_one_hot_encode_fonc,name_expo_fonc){
  res <-   res <- df_one_hot_encode_fonc %>%
    select(-all_of(name_expo_fonc)) %>%
    mutate_if(is.factor, ~as.numeric(as.character(.))) %>% # points litigieux j'utilise cette méthode p
    summarise_all(list(fonc_med = ~round(median(.,na.rm = TRUE),2),
                       fonc_quart1 = ~quantile(.,0.25,na.rm = TRUE),
                       fonc_quart2 = ~quantile(.,0.75,na.rm = TRUE),
                       fonc_mean = ~mean(.,na.rm = TRUE),
                       fonc_sd = ~sd(.,na.rm = TRUE),
                       fonc_min = ~min(.,na.rm = TRUE),
                       fonc_max = ~max(.,na.rm = TRUE))) %>%
    pivot_longer(cols = everything(),
                 names_to = c(".value", "level"),
                 names_pattern = "(.*)_fonc_(.*)") %>%
    t.df(.,"level") %>% column_to_rownames(var = "key")
  return(res)
  }

##############################################################################################
# Vérifications sommaire de l'imputations plus générations des tables

# df_one_hot_encode_fonc généré avec one_hot_fb
test_imputation <- function(df__pre_imput_func,df__post_imput_func,name_expo_fonc,check = TRUE) {
  colanmes_manq_func <- df__pre_imput_func %>%
    select_if(df__pre_imput_func %>% summarise_all(list(~sum(is.na(.)))) != 0) %>%
    colnames()

  table_pre_imput <-  df__pre_imput_func %>%
    select(all_of(colanmes_manq_func),all_of(name_expo_fonc)) %>%
    recup_var_table_verif_impute(name_expo_fonc)
  table_post_imput <-  df__post_imput_func %>%
    select(all_of(colanmes_manq_func),all_of(name_expo_fonc)) %>%
    recup_var_table_verif_impute(name_expo_fonc)

  table_diff_pre_post <- table_pre_imput - table_post_imput
  table_diff_prop_pre_post <- table_diff_pre_post / table_pre_imput
  table_diff_prop_pre_post[table_pre_imput == 0] <- 0

  if (check == TRUE) {
    if (length(colanmes_manq_func) == 0)     stop('Pas de valeur manquante dans la DF
                                                  pré imputation. Tu es sur ?
                                                  Si oui passe `check = FALSE`')
    if (any(df__post_imput_func %>% summarise_all(list(~sum(is.na(.)))) != 0)) {
      cat("les colonnes suivante de la df post imputations contiennent des manquants" ,
          df__post_imput_func %>%
            select_if(df__post_imput_func %>% summarise_all(list(~sum(is.na(.)))) != 0) %>%
            colnames() ,
          "\n pour ne pas faire de check `check = FALSE`")
      stop('Il reste des manquant dans la DF post imput')}

    table_pre_imput_col_ss_manq <- df__pre_imput_func %>%
      select(-all_of(colanmes_manq_func))  %>%
      recup_var_table_verif_impute(name_expo_fonc)
```

```
    table_post_imput_col_ss_manq <-  df__post_imput_func %>%
      select(-all_of(colanmes_manq_func)) %>%
      recup_var_table_verif_impute(name_expo_fonc)
    diff_diff_de_zero <- table_pre_imput_col_ss_manq - table_post_imput_col_ss_manq != 0

    if (any(diff_diff_de_zero)) {
      cat("les valeurs des colones sans valeurs manquantes",
          "ont était modifié par l'imputations cela concerne les colones suivante :\n",
          paste0(names(which(rowSums(diff_diff_de_zero) > 0)),collapse = "  , "),
          "\n pour ne pas faire de check `check = FALSE`")
      stop( " différence dans des colonnes non imputé")
    }
    if (any(table_diff_prop_pre_post$mean > 0.1)){
      cat("Il semble avoir eu un problème lors de l'imputations les variables :\n " ,
          rownames(table_diff_prop_pre_post[table_diff_prop_pre_post$mean > 0.1,]),
          "\n on subit une variation de lors moyennes supérieurs a 10%")

    }
  }
  res <- list(T_diff = table_diff_pre_post, T_prop_diff = table_diff_prop_pre_post)
  return(res)
  }


##############################################################################################
# Fonction qui sert quand j'ai du hard coder un choix qui vérifie que ça n'a pas changé
select_manuel_verif <- function(fuc_list_manu,choose_elem){
  # cette fonction ne sert que de vérifications
  if(length(fuc_list_manu) != length(choose_elem)) {
    print("number of element choose differ with list length")
    if(length(fuc_list_manu) > length(choose_elem)) {
      stop("not enougth manual choosen elements")
    } else { stop("to many manual choosen elements") }
  } else if(length(fuc_list_manu) == length(choose_elem)){
    # vérif que tout les éléments corresponde bien a ceux dans lequels manuellement choisir
    bool_in_list <- sapply(seq_along(choose_elem), function(x) choose_elem[x] %in% fuc_list_manu[[x]])
    if (all(bool_in_list)) {
      # selection de tout les éléements qui ne sont pas celui manuel
      res <- sapply(seq_along(choose_elem), function(x) fuc_list_manu[[x]][fuc_list_manu[[x]]  %notin% c
        compact %>% unlist
    } else {
      stop(paste0("choose elements ",choose_elem[!bool_in_list], " not in list"))
    }
  }
  return(res)
}

##############################################################################################
# Fonction qui sert a rename les colonnes depuis un fichier
rename_variables <- function(table_func, var_instrum_name = NULL, path_to_var_lab = "data/List_of_variab
  # récup fichier avec les noms
  label_var <- readxl::read_excel(path_to_var_lab) %>% select(Variable,Label)
```

```r
#On ajoute le fait que c'est le delta epworth
label_var$Label[label_var$Variable == "SYM_echelleEpworth"] <- paste0("Delta ",
                                                                      label_var$Label[label_var$Varial


# script variable name
actual_name <- data.frame(var_name = colnames(table_func)) %>%
  mutate(suffix_factor = str_extract(var_name,"\\.+\\d$"), # récupération du cas ou il y a eu one hot
         preffix_all = str_extract(var_name,"^Vis_init_|^Mesure_unique_"), # récupération de mes 3 pr
         var_ins = str_extract(var_name,"INS_"), # variables instrumentale que tu devras nommer toi m
         real_name = str_remove_all(var_name,"INS_|^Vis_init_|^Mesure_unique_|\\.+\\d$") # on enlève
  )
join_table <- actual_name %>% left_join(label_var, by = c("real_name" = "Variable")) # joionture avec
# gestions des variables instrumentales
name_need_supply <- join_table %>%
  filter(!is.na(var_ins)) %>%
  select(var_name) %>%
  distinct() %>%
  unlist(use.names = FALSE)
if(is.null(var_instrum_name)){
  cat("You must supply name for variable you create \n variables you must name are \n")
  print(name_need_supply)
  cat("\n for that ` var_instrum_name = c(\"variable_name\" = \"new name\")`\n")
  stop()
} else if (length(name_need_supply) != length(var_instrum_name)) {
  # heler pour les variables manquante ou en trop
  out_temp <- if_else(condition = length(name_need_supply) > length(var_instrum_name),
                      true = paste0("Not enought name provide, missing name for \n ",
                                    paste(name_need_supply[name_need_supply%notin%
                                                             names(var_instrum_name)],
                                          collapse = " ,")),
                      false = paste0("to many name provide, don't need name for \n ",
                                     paste(var_instrum_name[names(var_instrum_name)%notin%
                                                              name_need_supply],
                                           collapse = " ,"))
  )
  cat(out_temp)
  stop()
} else {
  instrum_name <- data.frame(var_name = names(var_instrum_name) , Label = var_instrum_name)
  complete_name_table <- join_table %>%
    left_join(instrum_name,by = "var_name") %>%
    mutate(Label = coalesce(Label.x, Label.y)) %>%
    select(-contains("Label.")) %>%
    mutate(Label = ifelse(!is.na(suffix_factor),
                          paste0(Label, suffix_factor),
                          Label),# on remet les indices de facteur
           Label = ifelse((!is.na(preffix_all)&preffix_all == "Vis_init_"),
                          paste0("Diagnostic ",Label),
                          Label)
    )
  short_name_table <- complete_name_table$var_name
  names(short_name_table) <- complete_name_table$Label
```

```
    res <- table_func %>% rename(all_of(short_name_table))
  }
  return(list(table_rename = res,
              complete_table = complete_name_table ))
}


##############################################################################################
```