EXTRA

Preprocessing

Step prima dell’analisi dati

- Carico il dataset
- Controllo struttura, dati mancanti, dati strani
- Visualizzo

Carico pacchetti

Come primo step è buona norma caricare i pacchetti necessari (e prima installarli se non sono già installati).

```
packages=c("readr", "tidyverse", "ggplot2", "ggpubr")
sapply(packages, require, character.only = T)
```

```
readr tidyverse   ggplot2    ggpubr 
 TRUE      TRUE      TRUE      TRUE 
```

Carico dataset

Può capitare di non avere un dataset con tutti i soggetti assieme, ma di dover caricare un dataset per soggetto e poi estrarlo a posteriori. Per questo tipo di operazione è utile la funzione `map_dfr` (simile ad apply, si trova nel pacchetto **purrr**, magico mondo di **tidyverse**).

Carico dataset

`list.files` vi permette di cercare e vedere quali file hanno uno specifico pattern. Nel mio caso vado dentro la cartella e mi carico tutti i dati in formato **.csv** (utilizzo la funzione head per mostrarvi i primi 6)

```
[1] "dataRaw/GNG100_1_SD.csv" "dataRaw/GNG101_1_FS.csv"
[3] "dataRaw/GNG102_1_SD.csv" "dataRaw/GNG103_1_FS.csv"
[5] "dataRaw/GNG104_1_FS.csv" "dataRaw/GNG106_1_FS.csv"
```

Carico dataset

Applico la funzione **`map_dfr`** a tutti i file ottenuti appena prima attraverso la funzione list.files…

```
d = list.files(path = "dataRaw", pattern = "*.csv",
               full.names = TRUE)%>%
  map_dfr(~read_csv(.x))
```

```
Error in `dplyr::bind_rows()`:
! Can't combine `..37$session` <double> and `..38$session` <character>.
```

Carico dataset

Error in `dplyr::bind_rows()`: ! Can't combine `..37$session` and `..38$session`.

Per qualche ragione in qualche file l'informazione sulla sessione viene interpretata come double anzichè character.

Quindi dentro map_df ci metto anche il **`mutate`**

```
d=list.files(path = "dataRaw", pattern = "*.csv",
             full.names = TRUE) %>%
  map_dfr(~read_csv(.x) %>%
            mutate(condition = as.character(condition),
                   session = as.character(session)) %>%
            subset(RUN >= 1)) # tolgo tutte le righe in più che non mi serv
```

Guardo le variabili

Per esempio controllo che i soggetti abbiano usato gli ID indicati. La funzione **`unique()`** è molto utile poichè restituisce un vettore senza elementi doppi (nel mio dataset avevo 448 trial per soggetto e quindi ogni id è ripetuto 448 volte).

```
1 unique(d$participant)
```

```
 [1] "GN100"  "GNG101" "GNG102" "GNG103" "GNG104" "GNG106" "GNG107" "GNG15"
 [9] "GNG16"  "GNG17"  "GNG18"  "GNG19"  "GNG20"  "GNG21"  "GNG22"  "GNG23"
[17] "GNG24"  "GNG25"  "GNG26"  "GNG27"  "GNG28"  "GNG29"  "GNG33"  "GNG34"
[25] "GNG35"  "GNG36"  "GNG37"  "GNG38"  "GNG39"  "GNG40"  "GNG41"  "GNG42"
[33] "GNG43"  "GNG44"  "GNG45"  "GNG46"  "GNG47"  "GNG48"  "GNG49"  "GNG50"
[41] "GNG53"  "GNG54"  "GNG55"  "GNG56"  "GNG58"  "GNG59"  "GNG60"  "GNG61"
[49] "GNG62"  "GNG63"  "GNG64"  "GNG66"  "GNG67"  "GNG68"  "GNG69"  "GNG70"
[57] "GNG71"  "GNG73"  "GNG74"  "GNG76"  "GNG78"  "GNG79"  "GNG80"  "GNG81"
[65] "GNG82"  "GNG83"  "GNG87"  "GNG85"  "GNG86"  "GNG88"  "GNG89"  "GNG90"
[73] "GNG92"  "GNG93"  "GNG94"  "GNG95"  "GNG96"  "GNG97"  "GNG98"
```

La funzione **`grep()`** cerca un pattern dentor una stringa, in questo caso l'id del soggetto doveva essere GNG…

```
grep(pattern = "GNG", x = unique(as.factor(d$participant)))
```

```
 [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26
[26] 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51
[51] 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76
[76] 77 78 79
```

```
unique(d$participant)[-c(grep(pattern = "GNG", x = unique(d$participant)))]
```

```
[1] "GN100"
```

```
d$id = factor(case_when(d$participant == "GN100"~ "GNG100",
                        .default = d$participant))
```

Abbiano inserito correttamente la condizione sperimentale...

```
unique(d$condition)
```

```
[1] "SD"             "FS"             "sd"             "fs"
[5] "FF"             "CONDIZIONE SD" "CONDIZIONE FS"
```

```
d$C = factor(case_when(d$condition == "CONDIZIONE FS" ~ "FS",
                       d$condition == "FF" ~ "FS",
                       d$condition == "fs" ~ "FS",
                       d$condition == "sd" ~ "SD",
                       d$condition == "FS" ~ "FS",
                       d$condition == "CONDIZIONE SD" ~ "SD",
                       d$condition == "SD" ~ "SD"))
```

Sistemo il dataset

Trasformo le variabili che mi interessano a fattore, e tengo solo quelle che mi interessano per la visualizzazione ed analisi. Possiamo utilizzare **`mutate()`** per mutare/cambiare il tipo di variabile. E **`select()`** per selezionare le variabili d'interesse:

```
dat = d %>%
  mutate(P = factor(LW_P),
         target = factor(target_col),
         B = factor(B),
         rt = target_kb.rt,
         resp = target_kb.keys,
         Trial = TRIAL.thisN + 1)|>
  select(id, P, target, C, B, rt, resp, Trial)
```

Controllo

Faccio un check veloce sulla struttura del dataset

```r
str(dat)
```

```
tibble [35,392 × 8] (S3: tbl_df/tbl/data.frame)
 $ id    : Factor w/ 79 levels "GNG100","GNG101",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ P     : Factor w/ 3 levels "0.2","0.5","0.8": 1 1 1 1 1 1 1 1 1 1 ...
 $ target: Factor w/ 2 levels "go.png","nogo.png": 2 2 1 2 2 2 1 2 2 2 ...
 $ C     : Factor w/ 2 levels "FS","SD": 2 2 2 2 2 2 2 2 2 2 ...
 $ B     : Factor w/ 12 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ rt    : num [1:35392] NA NA 0.396 NA NA NA 0.377 NA NA NA ...
 $ resp  : chr [1:35392] NA NA "space" NA ...
 $ Trial : num [1:35392] 1 2 3 4 5 6 7 8 9 10 ...
```

Aggiungo variabile accuratezza

Quando ho la risposta ed il trial era Go allora giusto (1), se risposta e trial NoGo allora sbagliato (0)…

```
dat$acc= case_when(dat$target == "nogo.png" & dat$resp == "space" ~ 0,
                   dat$target == "nogo.png" &  is.na(dat$resp) ~ 1,
                   dat$target == "go.png" & dat$resp == "space" ~ 1,
                   dat$target == "go.png" & is.na(dat$resp) ~ 0,
                   TRUE ~ 3)

# check
sum(dat$acc == 3)
```

```
[1] 0
```

Escludo soggetti

A volte certi soggetti vengono esclusi a priori perchè non hanno rispettato le “regole” dell’esperimento, in questo caso potete escluderli subito

```
dat_clean=subset(dat,
                          # PSQI = 14
                          id != "GNG82" &
                          # not respect sleep schedule
                          id != "GNG16" &
                          id != "GNG21" & id != "GNG40" &
                          id != "GNG46" & id != "GNG51" &
                          id != "GNG52"& id != "GNG58" &
                          id != "GNG65" & id != "GNG72" &
                          id != "GNG75" &  id != "GNG83"&
                          id != "GNG92" & id != "GNG95"&
                          id != "GNG96" & id != "GNG105")

# potete anche crease un vettore di ID da escludere e quindi utilizzare %in
```

Escludo soggetti

Nel mio caso volevo escludere anche tutti quelli con accuratezza minore a 70%. Per calcolare le statistiche descrittive, è molto utile la funzione **`group_by`** e **`summarise`** del pacchetto **dplyr** (sempre tidy)

```
summ_acc = dat_clean %>%
  group_by(id, target)%>%
  summarise(acc_mean = round(mean(acc),2),
            n = n())

# i soggetti GNG102 GNG38  GNG59  GNG59 hanno avuto un'accuratezza molto ba
summ_acc$id[which(summ_acc$acc_mean < .7)]
```

```
[1] GNG102 GNG102 GNG38  GNG59  GNG59
79 Levels: GNG100 GNG101 GNG102 GNG103 GNG104 GNG106 GNG107 GNG15 ... GNG98
```

Quindi li escludo attraverso la negazione di %in% (che valuta la presenza di un elemento in un vettore)

```
dat_clean2=subset(dat_clean,
                  !(id %in% summ_acc$id[which(summ_acc$acc_mean < .7)]))
```

Controllo che effettivamente non ci siano gli id con accuratezza minore di .7

```
sum(dat_clean2$id %in% summ_acc$id[which(summ_acc$acc_mean < .7)])
```

```
[1] 0
```

Dopo aver pulito i dati è bene controllare quante informazioni sono rimaste, per esempio il numero di soggetti per gruppo. **`n_distinct`** conta il numero di unique combinazioni..

```
dat_clean2 %>%
  group_by(C) %>%
  summarise(n_subj = n_distinct(id))
```

```
# A tibble: 2 × 2
  C     n_subj
  <fct>  <int>
1 FS        34
2 SD        32
```

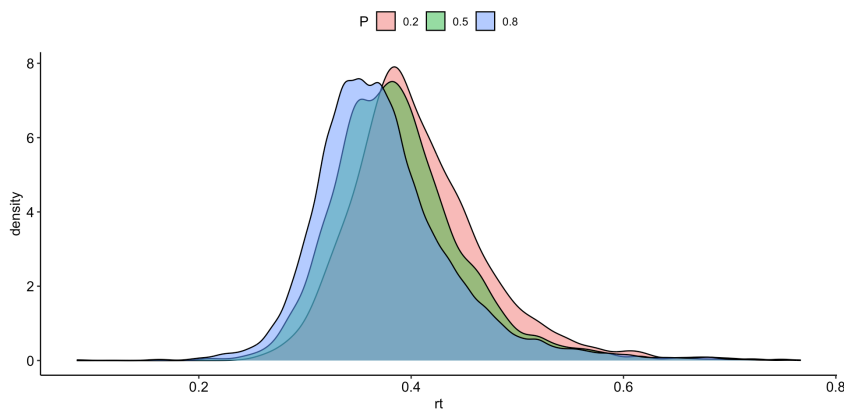
Ora visualizziamoli

Uno dei migliori pacchetti per la visualizzazione dati è **ggplot2**, potete farci praticamente tutto.

La prima cosa per cui è utile visualizzare i dati è quella di poter vedere la loro distribuzione, la presenza di valori estremi…

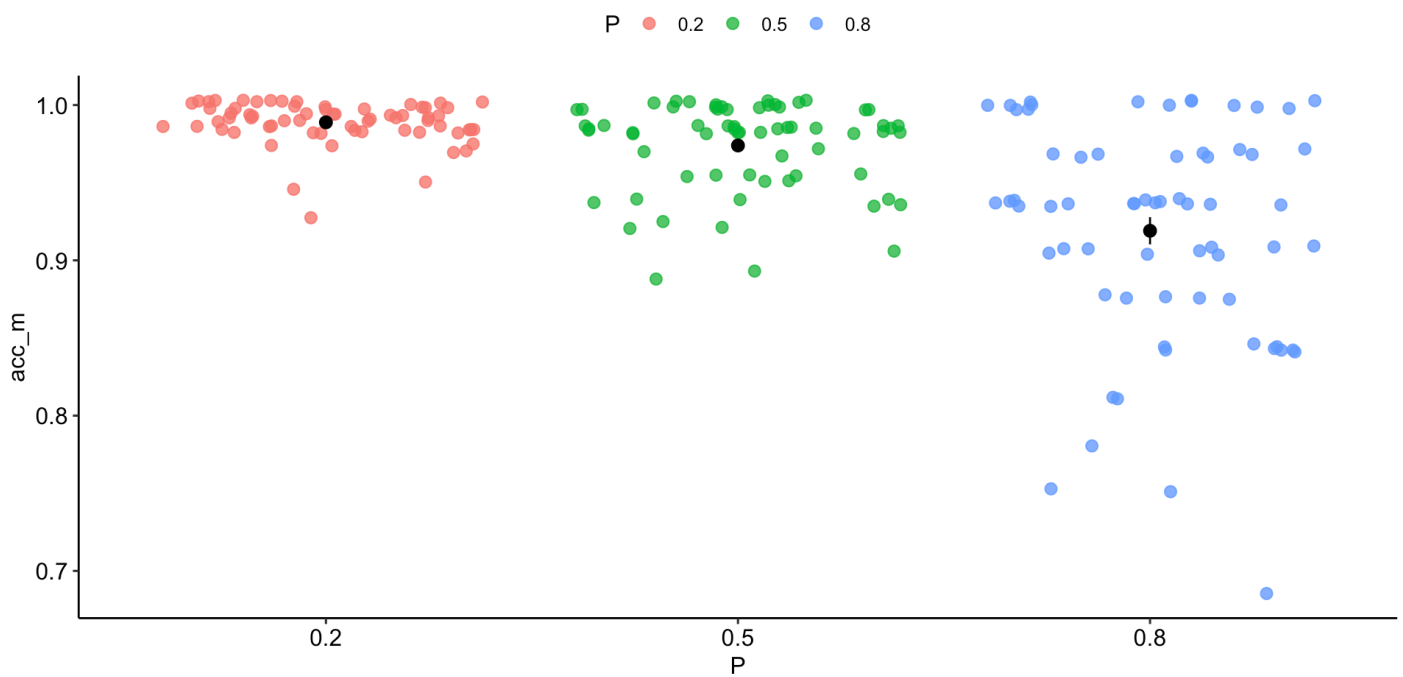
ggplot vuole come input che variabili utilizzare per la x e la y (o solo una delle due, dipende da cosa vogliamo plottare), posso specificare anche i colori, e varie cose che vedremo…

```
1  theme_set(theme_pubr())
2
3  dat_clean2 %>% subset(target == "go.png") %>%
4    ggplot(aes(x= rt, fill = P))+
5    geom_density(alpha = .5)
```



Accuratezza

```r
dat_clean2 %>% subset(target == "nogo.png") %>%
  group_by(id,P)%>% # ho raggruppato il dataset per id e P
  # ho calcolato la media di accuratezza per condizione/soggetto
  summarise(acc_m = mean(acc))%>% #
  ggplot(aes(y= acc_m, x = P, color = P))+
  geom_jitter(alpha = .8, size = 2.5)+ # puntini
  stat_summary(color = "black") # aggiungo stat mean se
```

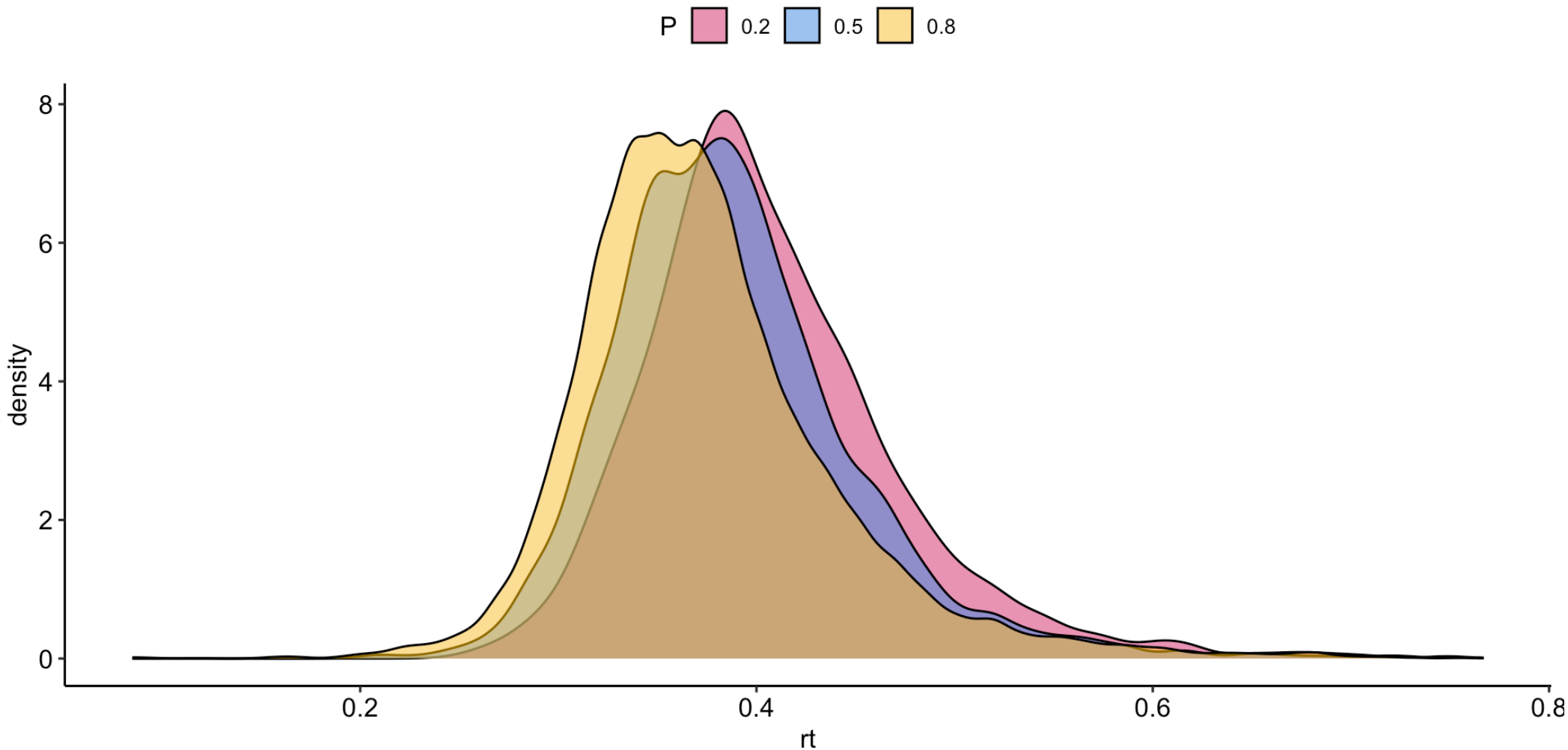


Personalizzazione

Una cosa semplice e carina da fare è crearsi la propria palette di colori da poter utilizzare quando vogliamo nei nostri plot. Potete definirla all'inizio oppure scriverla in uno script a parte e caricarla attraverso il comando **`source()`** come abbiamo visto per le funzioni. A questo link potete trovare tantissimi colori (colorblind friendly)

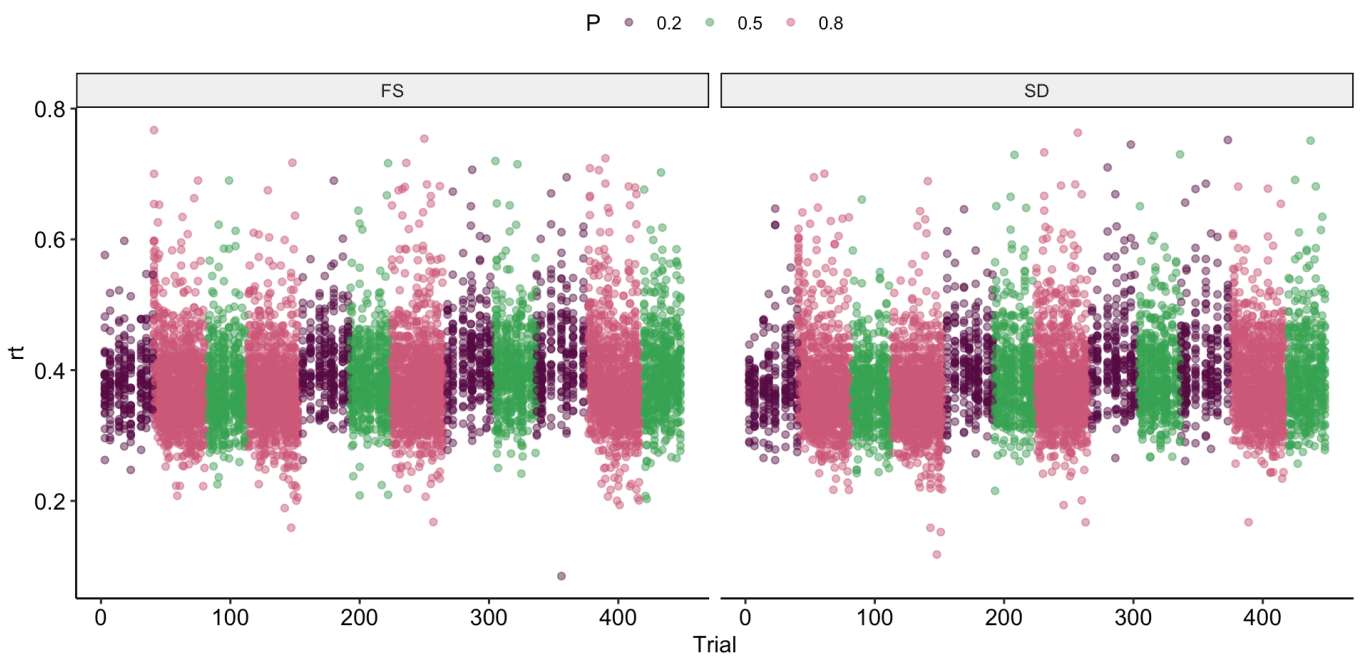
```
my_pal = list(
  pal1 = c("#D81B60","#1E88E5", "#FFC107", "#004D40"),
  pal2 = c("#580646","#3CA756", "#CE597A"),
  pal = c("#D81B60", "#004D40", "#1E88E5", "#FFC107",
          "#580646","#3CA756", "#CE597A")
)
```

```r
dat_clean2 %>% subset(target == "go.png") %>%
  ggplot(aes(x= rt, fill = P))+
  geom_density(alpha = .5)+
  scale_fill_manual(values = my_pal$pal1) # inserisco la mia palette
```

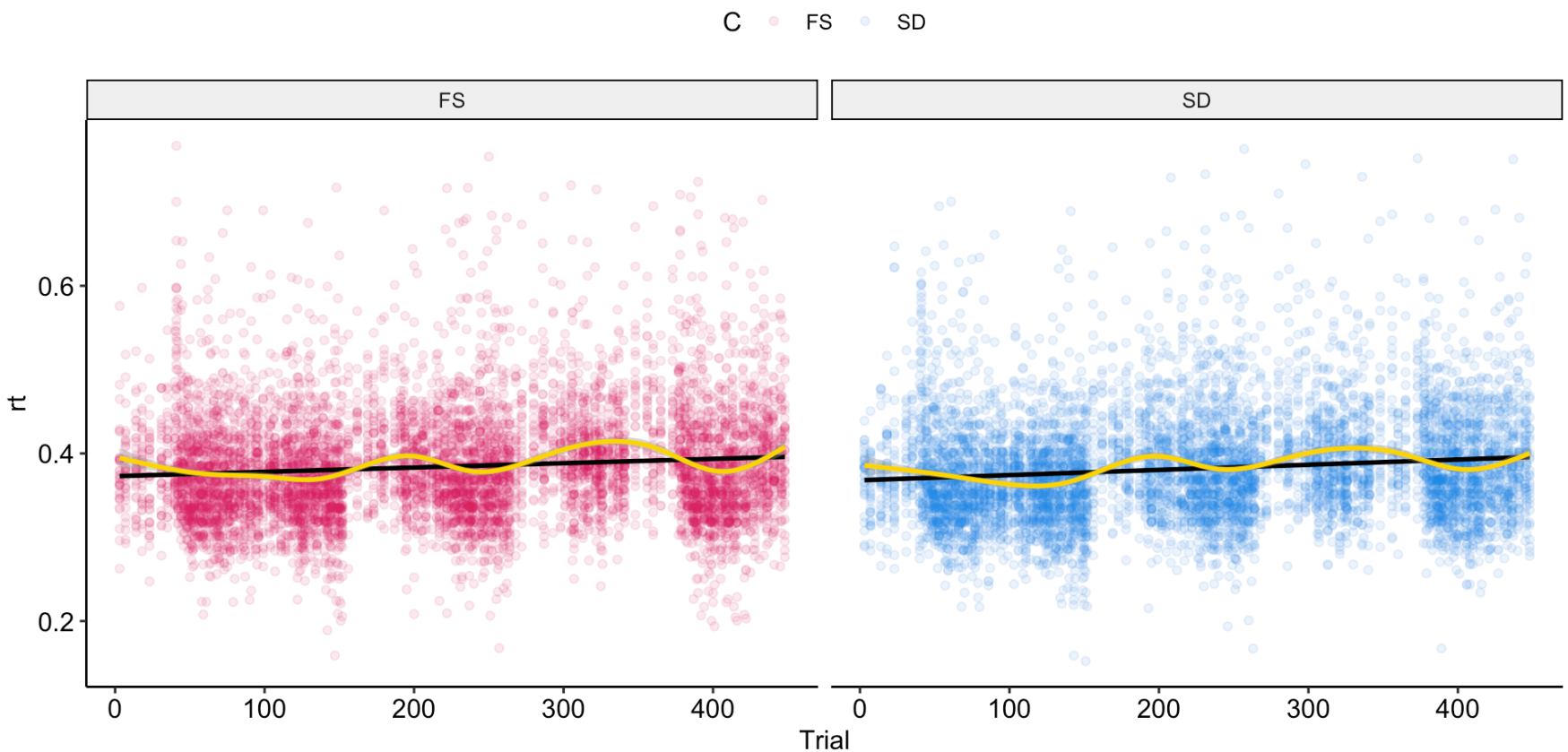


Attraverso la funzione **`facet_wrap()`** è possibile dividere il plot, per esempio mettendo un pannello per ogni gruppo nel mio caso, attraverso il comando ~ C

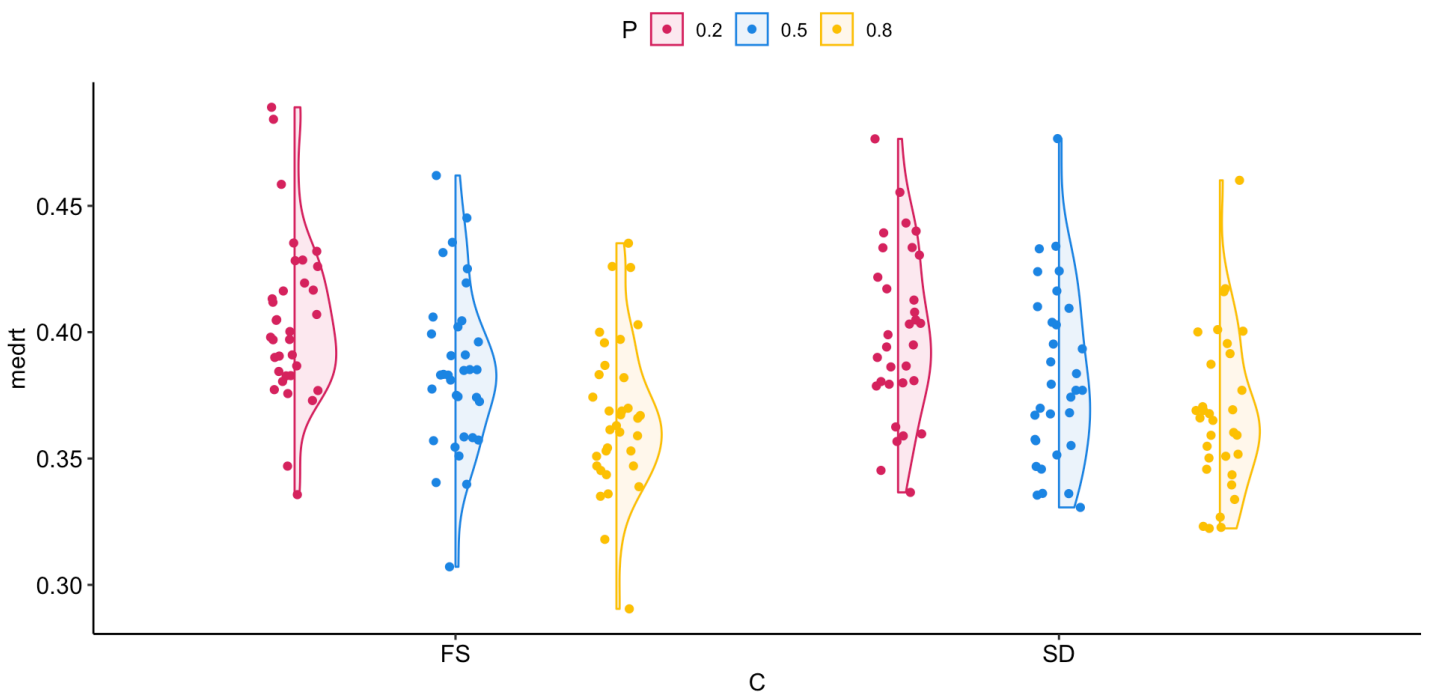
```
dat_clean2 %>% subset(target == "go.png") %>%
  ggplot(aes(y= rt, x = Trial, color = P))+
  geom_point(alpha = .5)+facet_wrap(~ C) + #facet_wrap divide in blocchi
  scale_color_manual(values = my_pal$pal2)
```



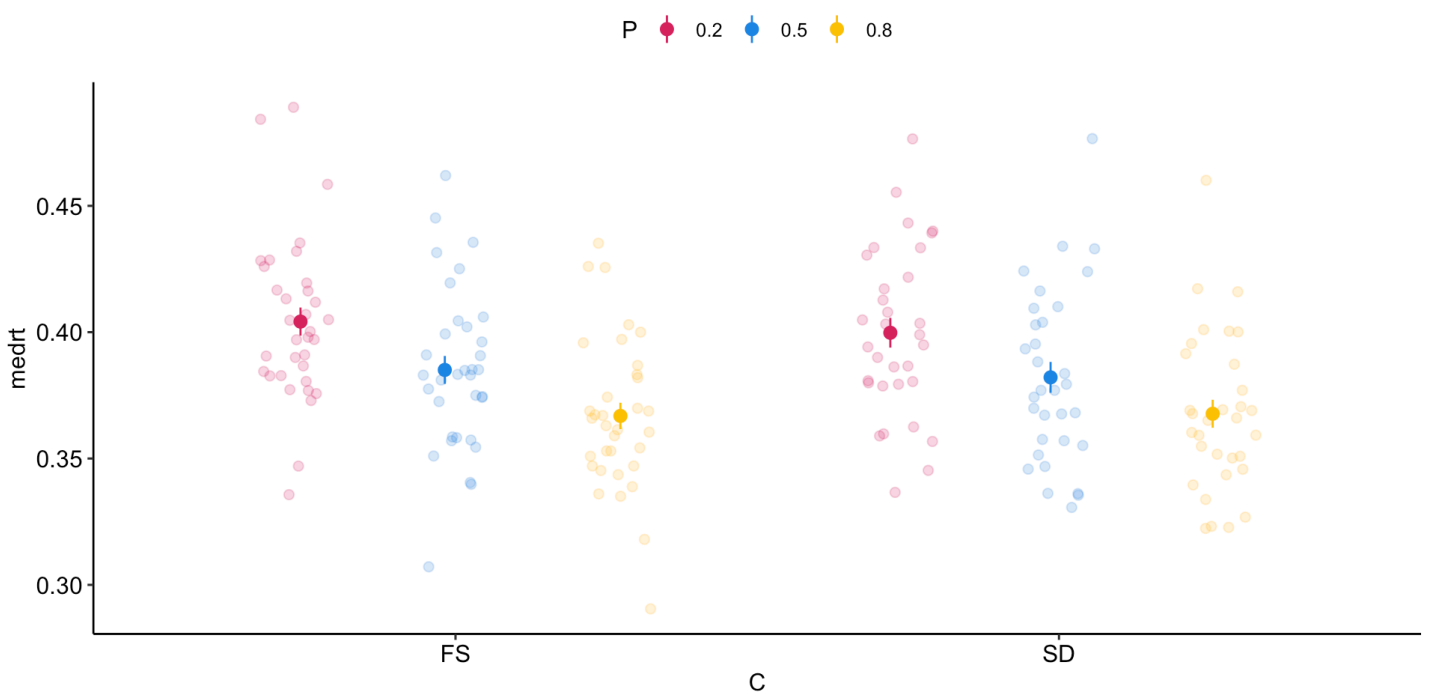
```r
dat_clean2 %>% subset(target == "go.png" & rt > .15) %>%
  ggplot(aes(y= rt, x = Trial, color = C))+
  geom_point(alpha = .1)+facet_wrap(~ C) +
  scale_color_manual(values = my_pal$pal1)+
  geom_smooth(method = "lm", color = "black", alpha = .5)+ #linear
   geom_smooth(method = "gam", #additive
               color = "gold")
```



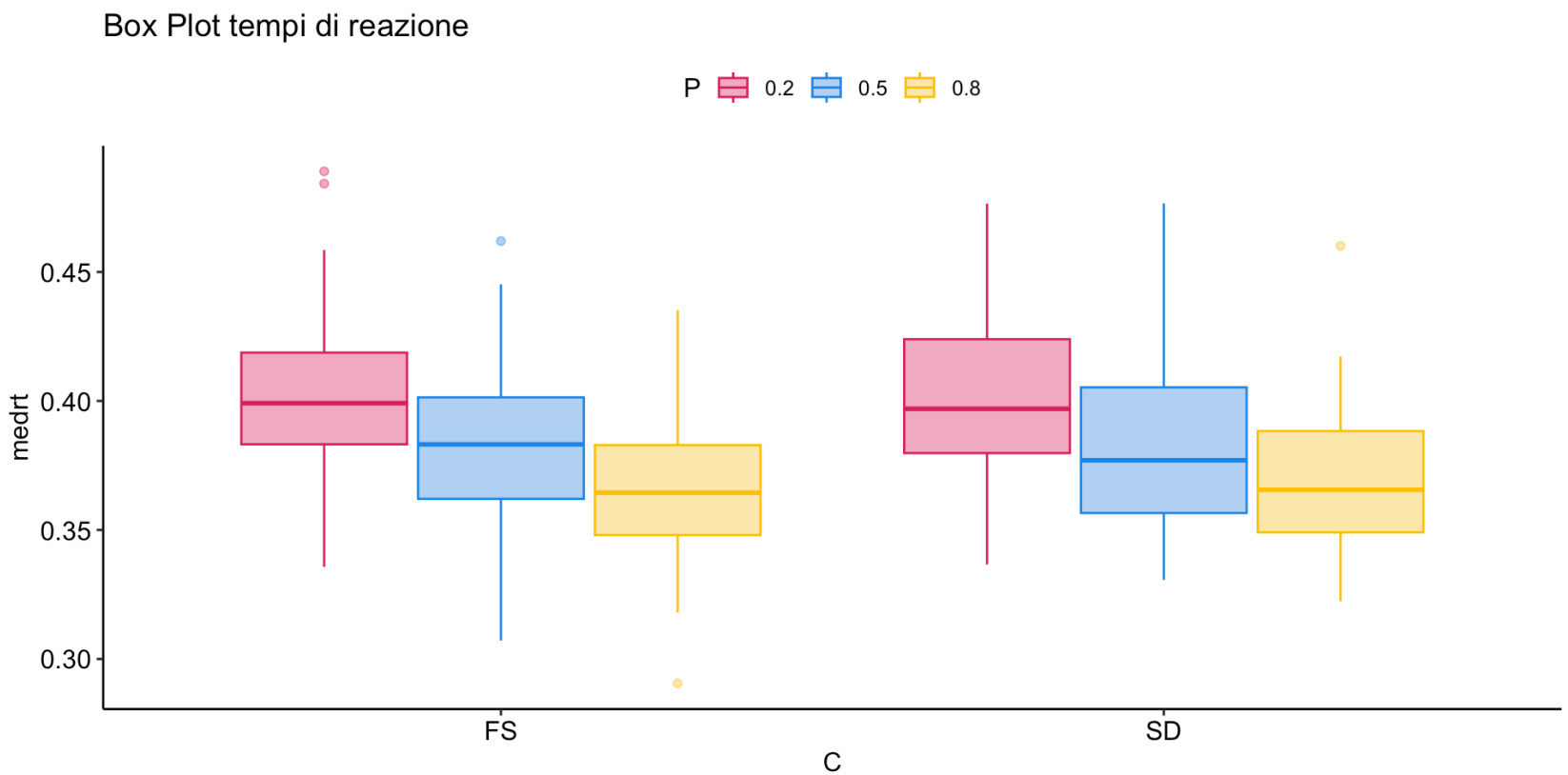
```r
dat_clean2 %>% subset(target == "go.png" & rt > .15) %>%
  group_by(id,C,P)%>%
  summarise(medrt = median(rt))%>%
  ggplot(aes(y= medrt, x = C, color = P, fill = P))+
  see::geom_violinhalf(position = position_dodge(width = .8),
                       alpha = .1)+
  geom_jitter(position = position_jitterdodge(jitter.width = .2,
                                              dodge.width = 0.8))+
  scale_color_manual(values = my_pal$pal1)+
  scale_fill_manual(values = my_pal$pal1)
```



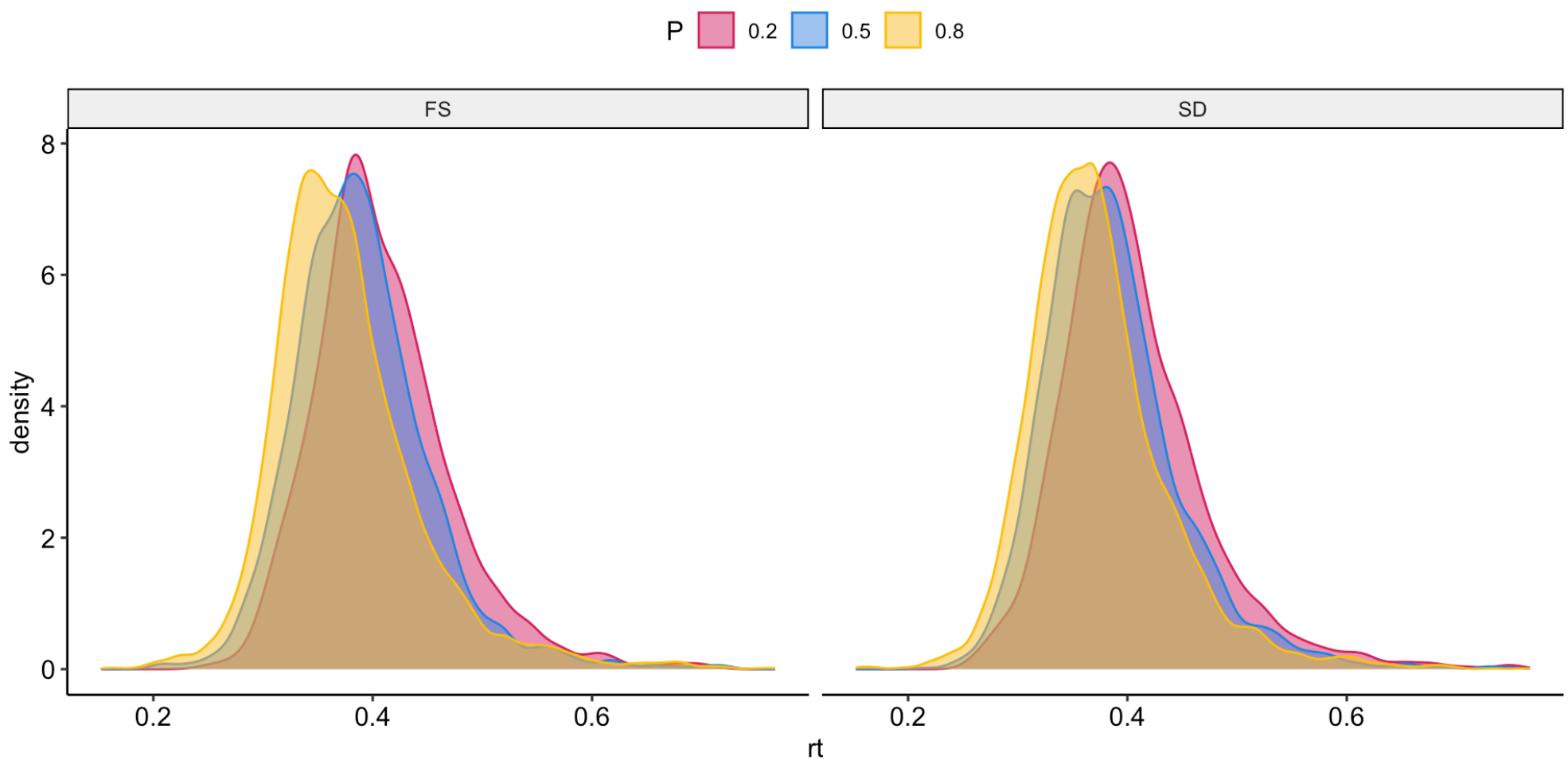
```r
dat_clean2 %>% subset(target == "go.png" & rt > .15) %>%
  group_by(id,C,P)%>%
  summarise(medrt = median(rt))%>%
  ggplot(aes(y= medrt, x = C, color = P, fill = P))+
  geom_jitter(alpha = .2, size = 2, position = position_jitterdodge(jitter.
                                                    dodge.width = 0.8))+
  stat_summary(size = .5, position =position_jitterdodge(jitter.width = .1,
                                                    dodge.width = 0.8))+
  scale_color_manual(values = my_pal$pal1)+
  scale_fill_manual(values = my_pal$pal1)
```



```r
dat_clean2 %>% subset(target == "go.png" & rt > .15) %>%
  group_by(id,C,P)%>%
  summarise(medrt = median(rt))%>%
  ggplot(aes(y= medrt, x = C, color = P, fill = P))+
  geom_boxplot(alpha = .4, position = position_dodge(width = .8))+
  scale_color_manual(values = my_pal$pal1)+
  scale_fill_manual(values = my_pal$pal1)+
  ggtitle("Box Plot tempi di reazione")
```



```r
dat_clean2 %>% subset(target == "go.png" & rt > .15) %>%
  ggplot(aes(x= rt,  color = P, fill = P))+
  geom_density(alpha = .5) +facet_wrap(~C)+
  scale_color_manual(values = my_pal$pal1)+
  scale_fill_manual(values = my_pal$pal1)
```



Salvare i plot

1. Creare un oggetto in cui c'è il vostro plot

2. Salvarlo

```
my_plot = dat_clean2 %>% subset(target == "go.png" & rt > .15) %>%
  ggplot(aes(x= rt,  color = P, fill = P))+ geom_density(alpha = .5) +
  facet_wrap(~C)+ scale_color_manual(values = my_pal$pal1)+
  scale_fill_manual(values = my_pal$pal1)+
  xlab("RTs in sec")

# creo la cartella plot
dir.create("plot")

# salvo dentro la cartella plot
ggsave("plot/my_plot.jpg", my_plot,dpi = 600, width = 18,
       height = 10, units = "cm")
```

Tabelle

Un'altra cosa che potrebbe esservi utile è quella di creare delle tabelle con per esempio le statistiche descrittive oppure l'ouput di un modello statistico. Per queste operazioni sono molto utili i pacchetti **flextable** o **kableExtra**:

```
library(flextable);library(kableExtra)
descr = dat_clean2%>%
  group_by(P,C)%>%
  summarise(medRT = round(median(rt,
                                 na.rm
            meanRT = round(mean(rt,
                                na.rm
            sd = round(sd(rt, na.rm =
```

```
descr
```

```
# A tibble: 6 × 5
# Groups:   P [3]
  P     C     medRT meanRT     sd
  <fct> <fct> <dbl>  <dbl>  <dbl>
1 0.2   FS    0.396  0.405 0.0659
2 0.2   SD    0.392  0.401 0.0728
3 0.5   FS    0.384  0.388 0.0634
4 0.5   SD    0.378  0.385 0.0639
5 0.8   FS    0.366  0.374 0.0679
6 0.8   SD    0.366  0.372 0.0646
```

Tabelle

```
tab = descr |>
    flextable() |>
    theme_vanilla() |>
    autofit() |>
    colformat_double(digits = 3)
```

```
tab
```

P	C	medRT	meanRT	sd
0.2	FS	0.396	0.405	0.066
0.2	SD	0.392	0.401	0.073
0.5	FS	0.384	0.389	0.063
0.5	SD	0.378	0.385	0.064
0.8	FS	0.366	0.374	0.068
0.8	SD	0.366	0.372	0.065

Salvo in file word

```
doc=officer::read_docx() # Crea un documento Word vuoto

doc=body_add_flextable(doc, value = tab) # Aggiungi la flextable

print(doc, target = "tabella.docx") # Salva il file
```

Fine