Plot Tutorial

Asperger Cieslik

15/12/2021

```
if("tidyverse" %in% rownames(installed.packages()) == FALSE)
{
  install.packages("tidyverse")
}
if("Hmisc" %in% rownames(installed.packages()) == FALSE)
{
  install.packages("Hmisc")
}
if("ggsignif" %in% rownames(installed.packages()) == FALSE)
{
  install.packages("ggsignif")
}

library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(ggsignif)
```

```
data <- read.table(
    "csv/expression.csv",
    header = TRUE,
    quote = '"',
    stringsAsFactors = FALSE,
    sep = ","
  )

data_ttest <- data.frame(Cell.Line  = character(), Protein = character(), p.val = numeric(), stringsAsF
data_ratios <- data.frame(Cell.Line = character(), Protein = character(), Ratio = numeric(), stringsAsF
i <- 1
j <- 1
for(Cell.Line in unique(data$Cell.Line)){
  for(Protein in unique(data$Protein)){
    subset <- data[which(data$Cell.Line == Cell.Line & data$Protein == Protein), ]
    subset_EVC <- subset[which(subset$Vector == "EVC"), ]
```

```
    subset_PGRMC1 <- subset[which(subset$Vector == "PGRMC1"), ]
    data_ttest[i, "Cell.Line"] <- Cell.Line
    data_ttest[i,  "Protein"] <- Protein
    data_ttest[i, "p.val"] <- t.test(subset_EVC$Expression, subset_PGRMC1$Expression, alternative ="two
    i = i+1

    j_end <- j+length(subset_PGRMC1[,1])-1
    data_ratios[j:j_end, "Cell.Line"] <- Cell.Line
    data_ratios[j:j_end, "Protein"] <- Protein
    data_ratios[j:j_end, "Ratio"] <- subset_PGRMC1$Expression / mean(subset_EVC$Expression)

    j = j_end+1
  }
}
data_ttest$q.val <- p.adjust(data_ttest$p.val, "fdr")

data_ratios$Name <- paste(data_ratios$Protein, data_ratios$Cell.Line, sep="\n")

data_summary <- data %>% group_by(Cell.Line, Protein, Vector) %>%
  summarize(mean = mean(Expression), sd = sd(Expression))
```

```
## `summarise()` has grouped output by 'Cell.Line', 'Protein'. You can override using the `.groups` argu
```

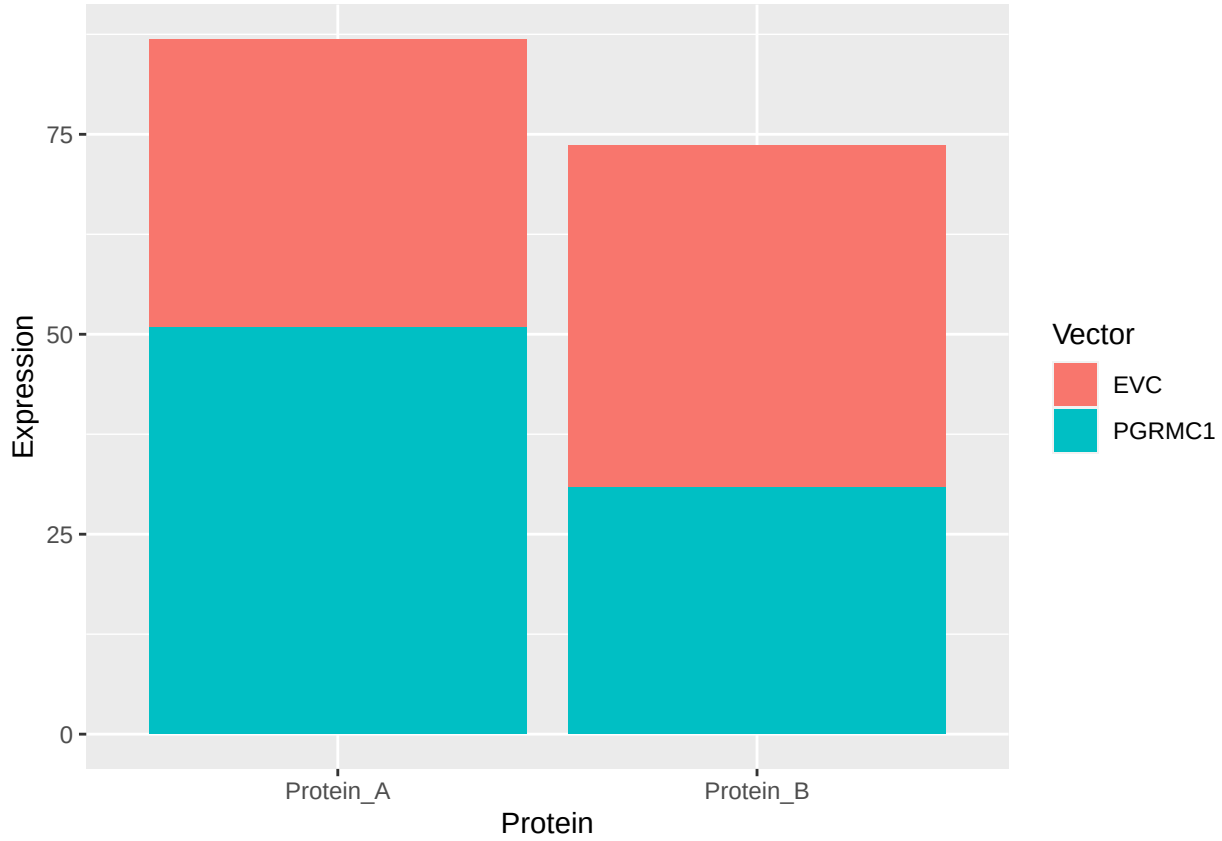
```
rm("i", "subset", "subset_EVC", "subset_PGRMC1", "Cell.Line", "Protein", "j", "j_end")
```

The function ggplot() creates the empty canvas and you can set some aesthetics parameters like axes and color/fill options.

```
p <- ggplot(data, aes(x = Protein, y = Expression, fill = Vector))
```

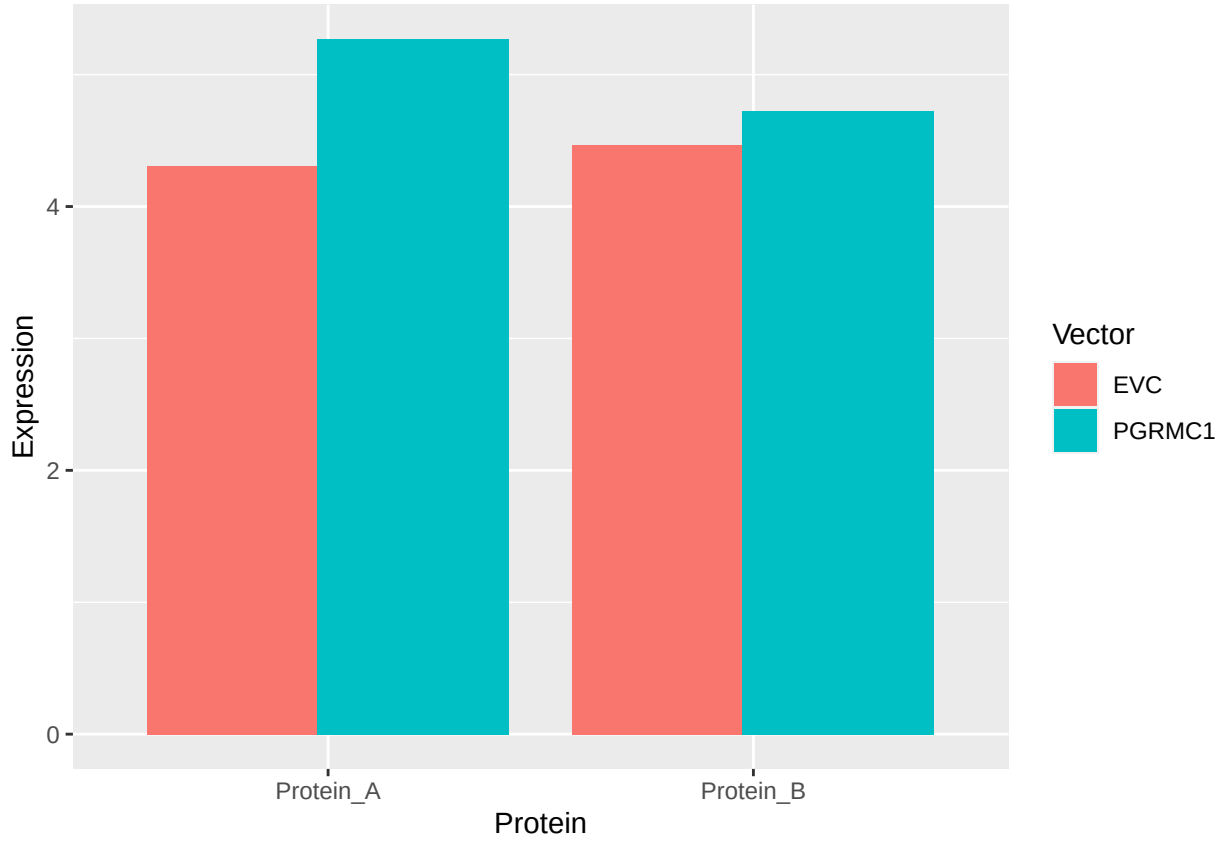
To create the plot itself you need to "add" further functions. The geom_bar functions adds a bar plot to the empty canvas.

```
p <- ggplot(data , aes(x = Protein, y = Expression, fill = Vector))
p <-p + geom_bar(stat="identity")
print(p)
```



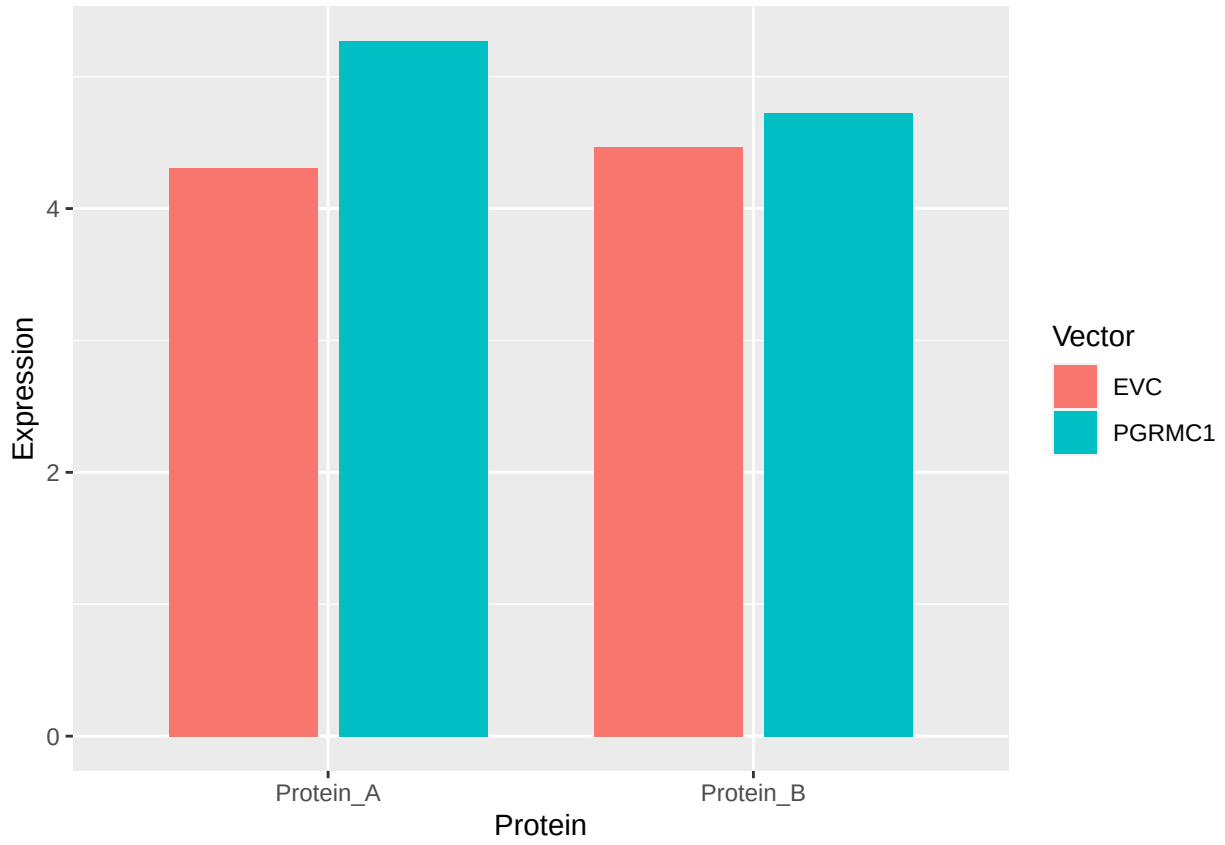
The barplot can then be modified. For instance changing it from a stacked bar plot to a side by side plot.

```
p <- ggplot(data , aes(x = Protein, y = Expression, fill = Vector))
p <-p + geom_bar(position = position_dodge(width = 0.8) ,
                 stat = "identity")
print(p)
```



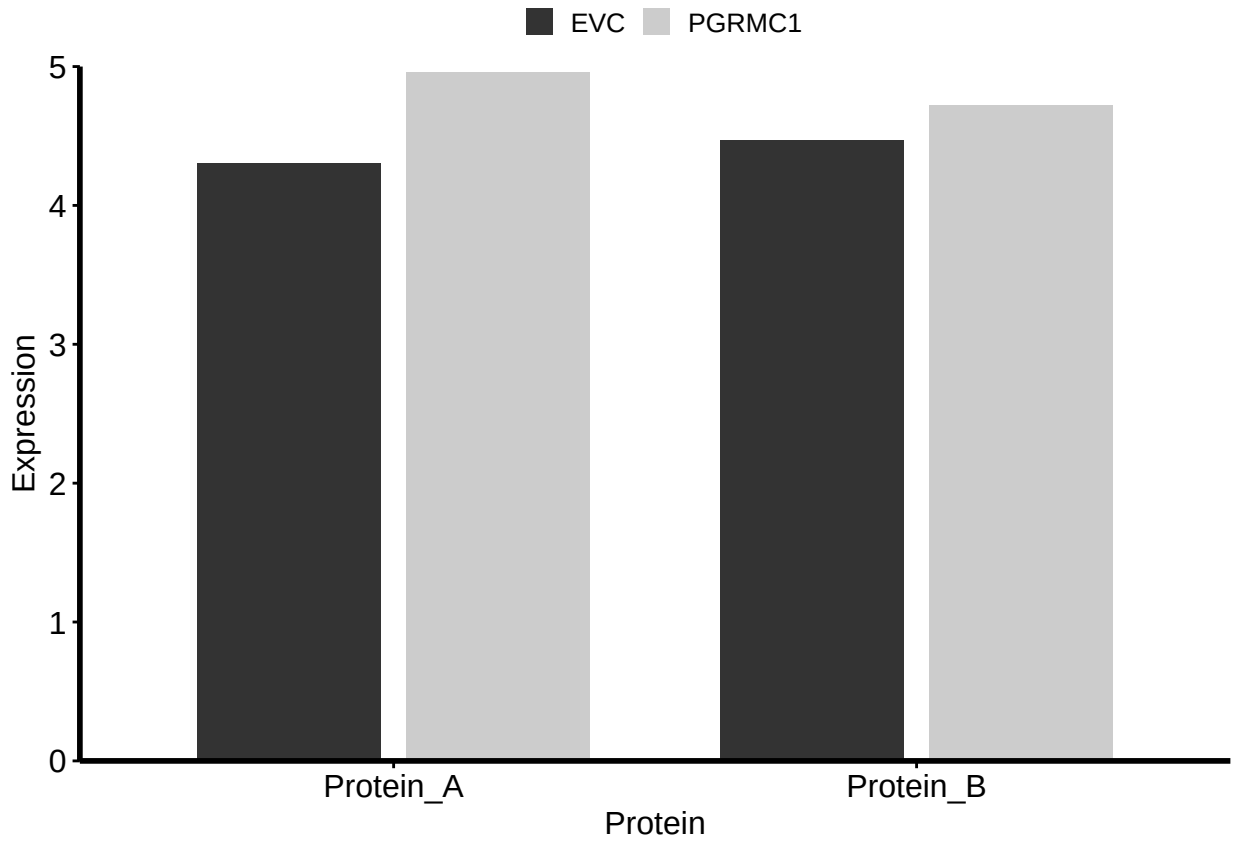
Then we can change the distance between the bars.

```
p <- ggplot(data , aes(x = Protein, y = Expression, fill = Vector))
p <-p + geom_bar(width = 0.7, position = position_dodge(width = 0.8) ,
                 stat = "identity")
print(p)
```



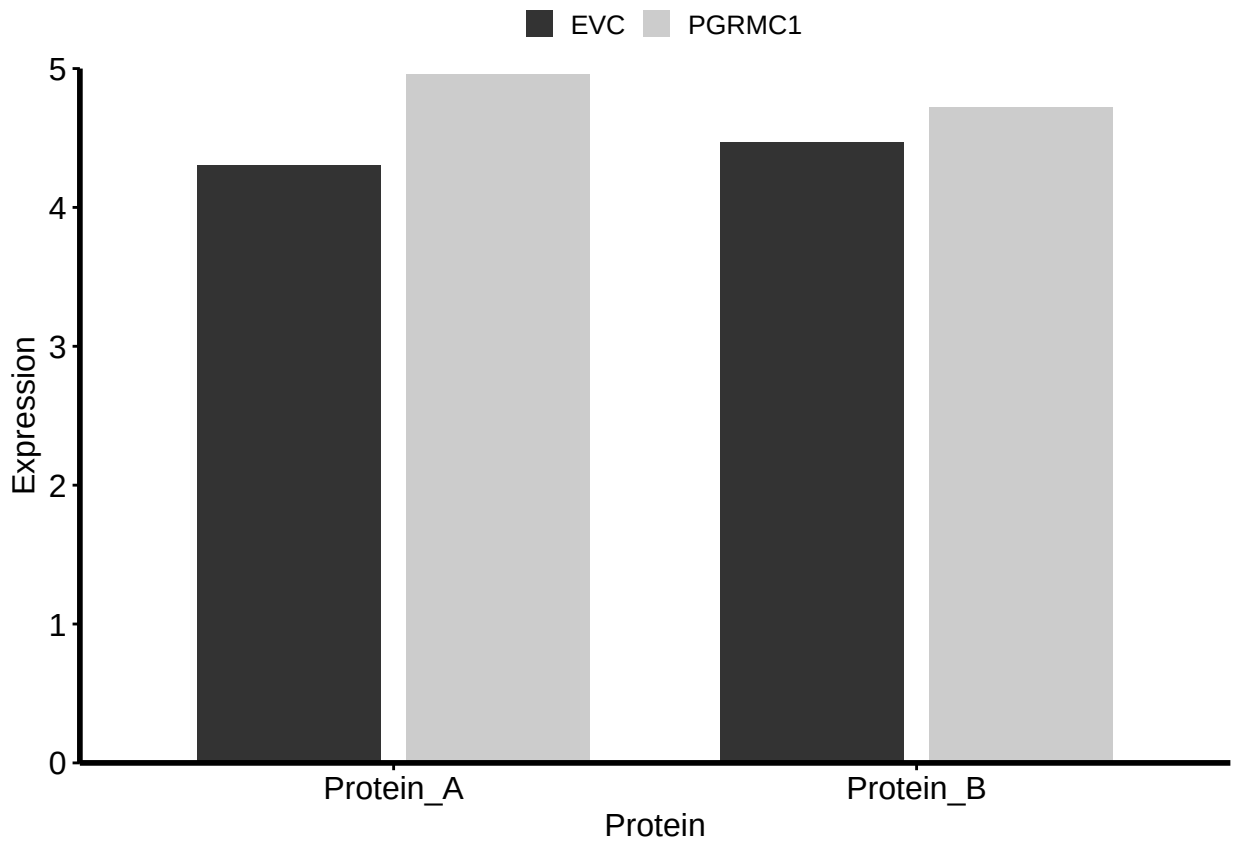

```
p <- ggplot(data , aes(x = Protein, y = Expression, fill = Vector))
p <-p + geom_bar(width = 0.7, position = position_dodge(width = 0.8) ,
                 stat = "identity")

p <- p + scale_fill_grey(start = 0.2, end = 0.8)
p <- p + scale_y_continuous(limits = c(0, 5), expand = c(0, 0))

p <- p + theme_classic()

p <- p + theme(legend.position = "top")
p <- p + theme(text = element_text(size=12, color="black"))
p <- p + theme(axis.text.y = element_text(size=12, color="black"))
p <- p + theme(axis.text.x = element_text(size=12, color="black"))
p <- p + theme(axis.line = element_line(
  colour = "black", size = 1, linetype = "solid", lineend = 'butt'))
p <- p + theme(legend.key.size = unit(0.8,"line"))
p <- p + theme(axis.ticks.x = element_line(color = "black"))
p <- p + theme(axis.ticks.y = element_line(color = "black"))
p <- p + theme(legend.title = element_blank())
p <- p + theme(legend.margin=margin(t = 0, unit='cm'),
               legend.key = element_blank(),
               legend.background=element_blank())
print(p)
```

```
## Warning: Removed 3 rows containing missing values (geom_bar).
```

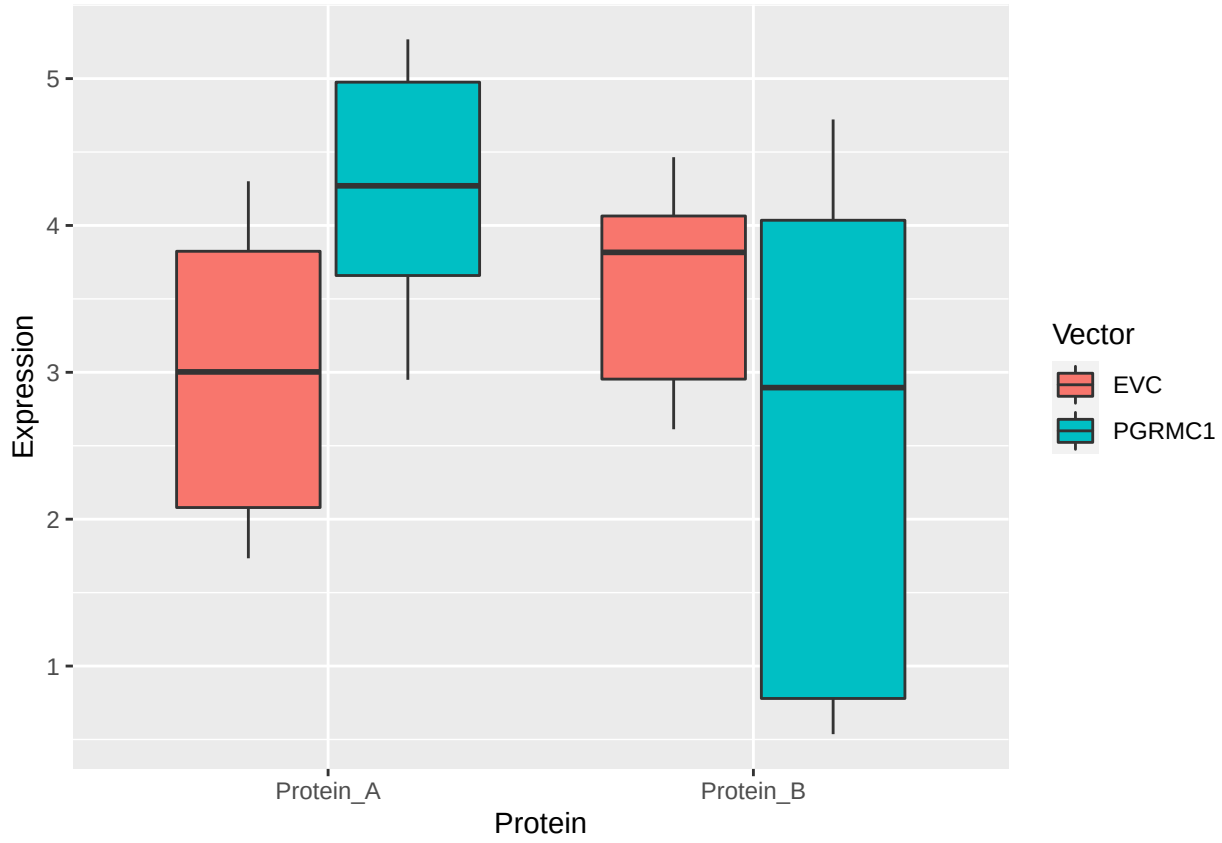



```
p <- ggplot(data , aes(x = Protein, y = Expression, fill = Vector))
p <-p + geom_bar(width = 0.7, position = position_dodge(width = 0.8) ,
                 stat = "identity")

p <- p + scale_fill_grey(start = 0.2, end = 0.8)
p <- p + scale_y_continuous(limits = c(0, 5), expand = c(0, 0))

source("theme.R")
print(p)
```

```
## Warning: Removed 3 rows containing missing values (geom_bar).
```




```
p <- ggplot(data, aes(x=Protein, y=Expression, fill = Vector)) + geom_boxplot(outlier.colour="red", outl
                outlier.size=4)
print(p)
```



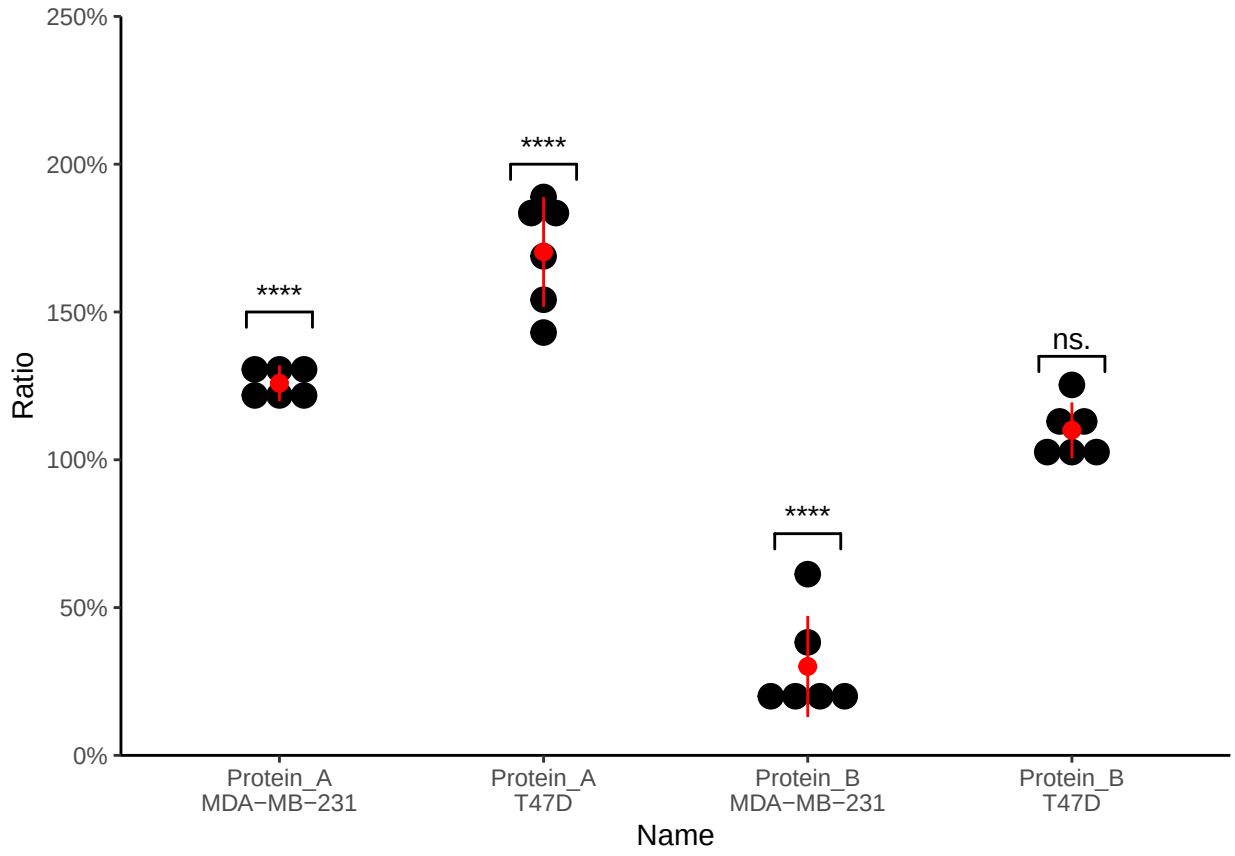

```
p <- ggplot(data_ratios, aes(x=Name, y=Ratio))
#p <- p + geom_violin(trim = FALSE)
p <- p + geom_dotplot(binaxis='y', stackdir='center')

p <- p + scale_y_continuous(limits = c(0, 2.5), expand = c(0, 0),
                            labels = scales::percent_format(accuracy = 1))
#p <- p + coord_flip()
p <- p + stat_summary(fun=mean, geom="point", shape=18, size=3, color="red")
#p + stat_summary(fun.data="mean_sdl", fun.args = list(mult=1),
#                 geom="crossbar", width=0.3)
p <- p + stat_summary(fun.data=mean_sdl, fun.args = list(mult=1),
                  geom="pointrange", color="red", width=0.5)
```

```
## Warning: Ignoring unknown parameters: width
```

```
p <- p + theme_classic()


data_significance <- data.frame("signif_x" = numeric(),
                                "signif_xend" = numeric(),
                                "signif_y" = numeric(),
                                "signif_annotation" = character(),
                                stringsAsFactors = FALSE)

sig_length <- 0.125
data_significance[1, ]<- c(1-sig_length, 1+sig_length, 1.5, '****')
data_significance[2, ]<- c(2-sig_length, 2+sig_length, 2, ' **** ')
```

```
data_significance[3, ]<- c(3-sig_length, 3+sig_length, 0.75, '  ****  ')
data_significance[4, ]<- c(4-sig_length, 4+sig_length, 1.35, 'ns.')

data_significance$signif_x <- as.numeric(data_significance$signif_x)
data_significance$signif_xend <- as.numeric(data_significance$signif_xend)
data_significance$signif_y <- as.numeric(data_significance$signif_y)

p <- p + geom_signif(xmin = data_significance$signif_x,
                xmax  = data_significance$signif_xend,
                y_position = data_significance$signif_y,
                annotation= data_significance$signif_annotation)

print(p)
```

```
## Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.
```




```
ggsave(
  plot = p,
  width = 10,
  height = 7.5,
  dpi = 300,
  filename = paste('export/dotplot.pdf', sep = ''),
  units = "cm"
)
```

```
## Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.
```