Data Cleaning

Mahbod Mehrvarz

Dec. 2023

Retriving the raw data

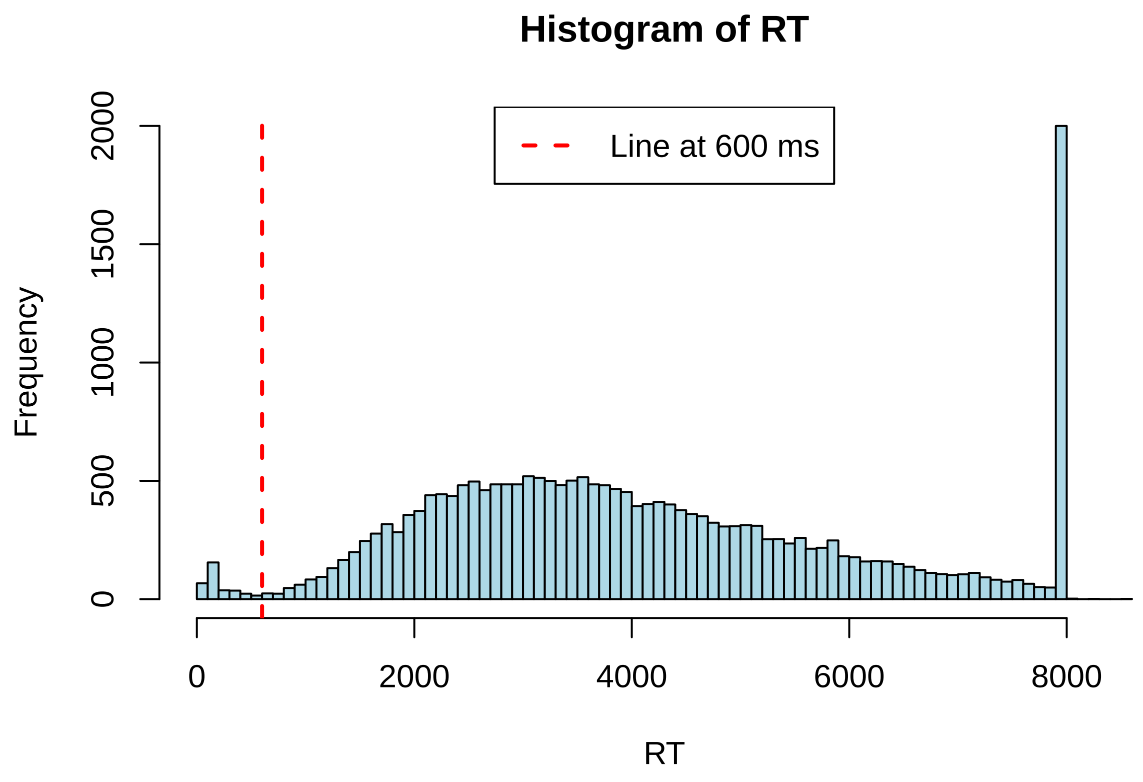
```r
processed_data = read.csv("../_data/processed-data.csv")
```

Exploring Outliers in Reaction Times

Let's first visualize the reaction time across every task and participants:

Distribution of Reaction Times (RTs)

```r
hist(processed_data$rt, breaks = 100, main = "Histogram of RT", xlab = "RT",
     ylab = "Frequency", col = "lightblue", border = "black")
abline(v = 600, col = "red", lwd = 2, lty = 2)
legend("top", legend = "Line at 600 ms", col = "red", lty = 2, lwd = 2)
```



The above histogram visualizes the distribution of RTs across all tasks and participants. The red line marks the 600 ms threshold, a commonly used benchmark in cognitive studies to identify rapid responses.

Analyzing Responses Below 600 ms

```
low_rt_ound = 600
low_rts_prop = mean(processed_data$rt < low_rt_ound)*100
bad_obs_rts = which(processed_data$rt < low_rt_ound)
```

We calculate the percentage of all responses with RTs below 600 ms to understand the prevalence of rapid responses. It's about 1.49%.

Identifying Participants with Excessive Rapid Responses

```
low_too_quick = .05
too_quick = tapply(processed_data$rt<low_rt_ound, processed_data$sub, mean)
num_partips_too_quick = sum(too_quick > low_too_quick)
bad_partips_rt = unique(processed_data$sub)[which(too_quick > low_too_quick)]
```

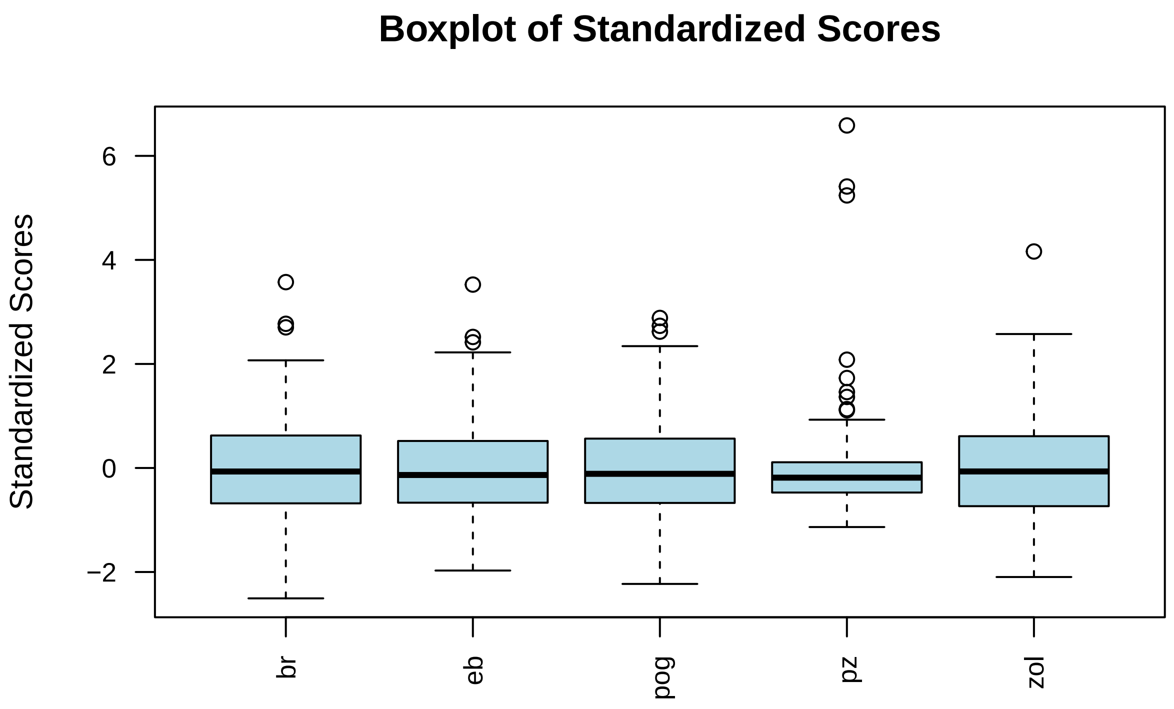
We determine the number of participants for whom more than 5% of their responses have RTs below 600 ms. This threshold helps identify participants with a consistently high rate of rapid responses. 5 participants met were below this threshold.

Cleaning Data Based on Rapid Responses

```
bad_RTs = unique(c(which(processed_data$sub %in% bad_partips_rt), bad_obs_rts))
dat1 = processed_data[-bad_RTs,]
```

Looking for Outliers: Univariate Setup

```
scores = tapply(dat1$y, list(dat1$sub,dat1$task), mean)
standardized_scores = scale(scores)
boxplot(standardized_scores,
        main = "Boxplot of Standardized Scores",
        ylab = "Standardized Scores", col = "lightblue",
        border = "black", las = 2, cex.axis = 0.8)
```



Evaluating outliers using standardized scores of participants across different tasks. The boxplot provides a visual representation of score distribution, making it easier to identify outliers. The variance of Panzo looks to be effected by the outliers.

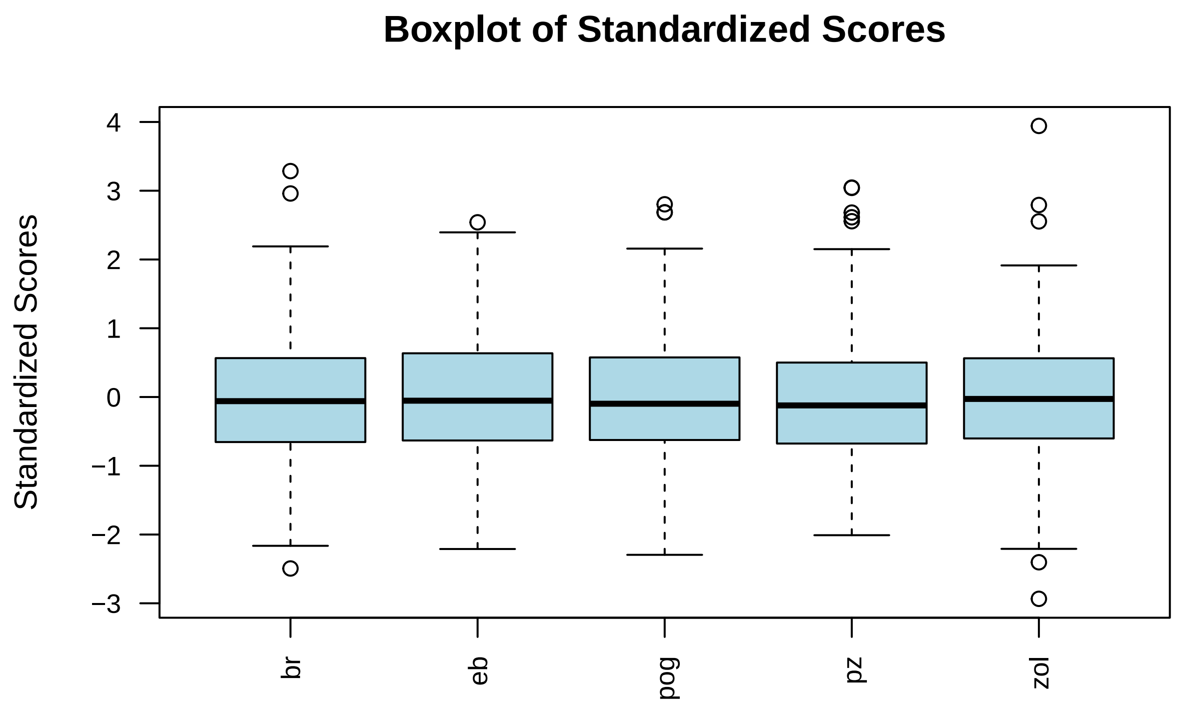
Removing Outliers in the 'Panzo' Task

```r
bad_sub_panzo = as.integer(names(which(standardized_scores[,4]>2)))
bad_sub_panzo_ind = which(dat1$sub %in% bad_sub_panzo)
num_bad_sub_panzo = length(bad_sub_panzo)
dat2 = dat1[-bad_sub_panzo_ind,]
dim(dat2)
```

```
## [1] 20846    15
```

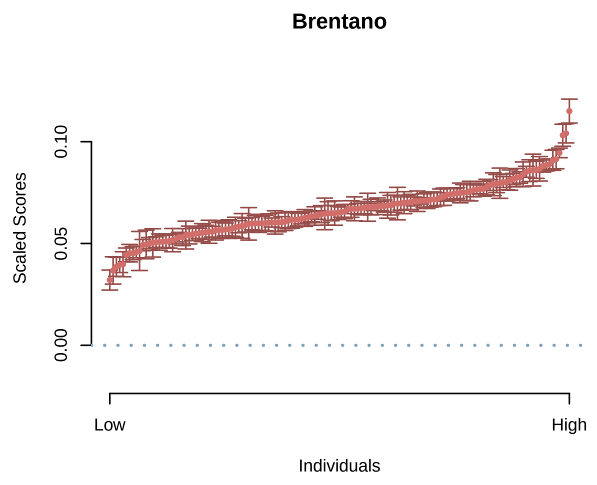
Identifying and removing participants who are outliers in the 'PZ' task based on a standardized score cutoff of 2. 4 were over this cutoff.

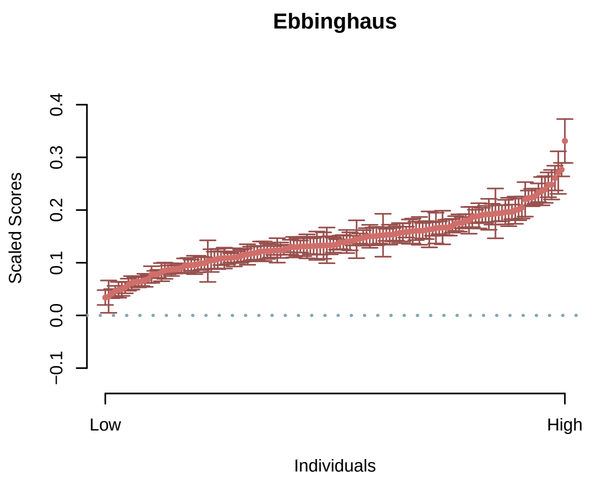
```r
scores = tapply(dat2$y, list(dat2$sub,dat2$task), median)
standardized_scores = scale(scores)
boxplot(standardized_scores,
        main = "Boxplot of Standardized Scores",
        ylab = "Standardized Scores", col = "lightblue",
        border = "black", las = 2, cex.axis = 0.8)
```

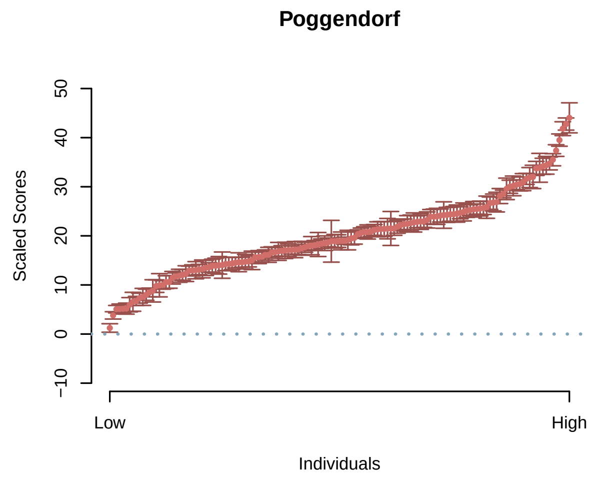


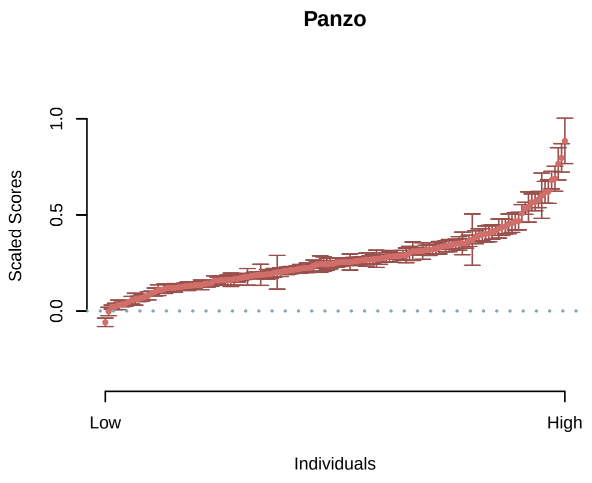
Visualizing Individual Differences Within Task

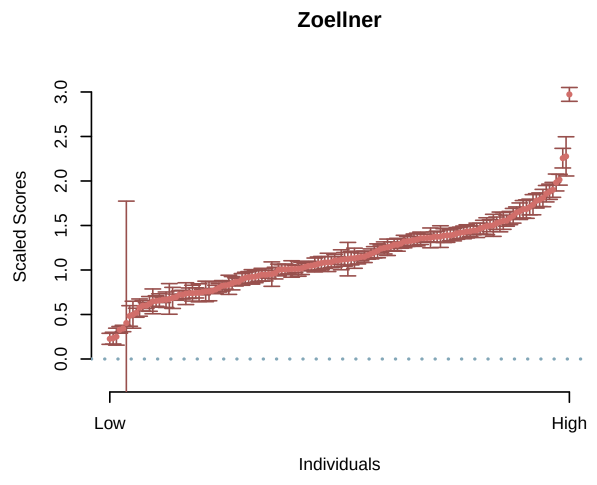
```r
scores = tapply(dat2$y, list(dat2$sub, dat2$task), mean)
stand_error = tapply(dat2$y, list(dat2$sub, dat2$task),
            function(x) sd(x) / sqrt(length(x)))
taskPlot = function(temp_scores, temp_stand_error, task_name){
  o = order(temp_scores)
  means = temp_scores[o]
  SEs = temp_stand_error[o]
  I = length(o)
  scaled_means = scale(means)
  o2 = rev(means)[1]
  o3 = rev(SEs)[1]
  ylims = c(0 - 3*o3,
              o2 + 3*o3)
  plot(NA,
       NA,
       xlim = c(1,I),
       ylim = ylims,
       axes = F,
       xlab = "Individuals",
       ylab = "Scaled Scores",
       main = task_name
  )
  axis(1, at = c(1,I), label = c("Low","High"))
  axis(2)
  arrows(1:I,
       means - SEs,
       1:I,
       means + SEs,
       angle = 90, code = 3, length = 0.05, col = "#98504D")
  points(1:I, means, pch = 16, col = "#D16E6A", cex=.8)
  abline(h=0, lty = 3, col = "#83A7B8", lwd = 2)

}
par(mfrow=c(3,2))
taskPlot(scores[,1], stand_error[,1], "Brentano")
taskPlot(scores[,2], stand_error[,2], "Ebbinghaus")
taskPlot(scores[,3], stand_error[,3], "Poggendorf")
taskPlot(scores[,4], stand_error[,4], "Panzo")
taskPlot(scores[,5], stand_error[,5], "Zoellner")
```







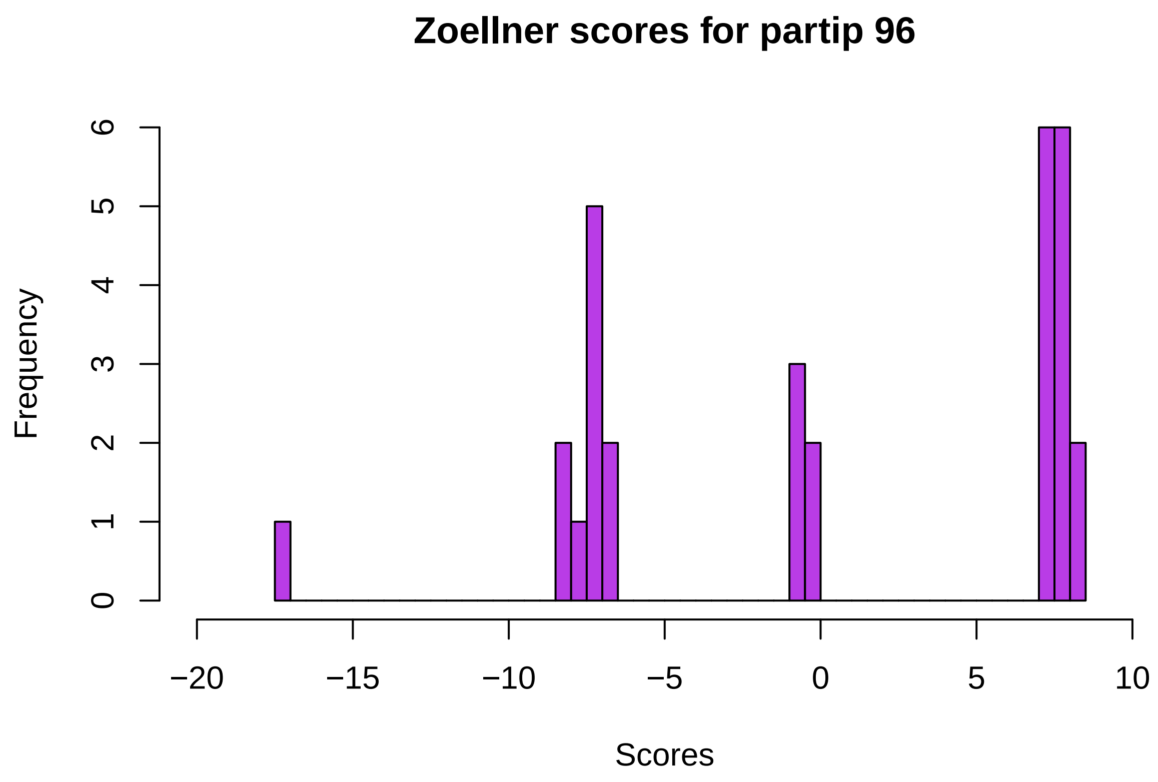




The above plots provide a detailed visual representation of individual differences within each task. By plotting median scores and their associated standard errors, we gain insight into the performance spread among participants for each task. The standard error for one of the Zoellner obeservations looks off.

Analyzing Variance in Zoellner

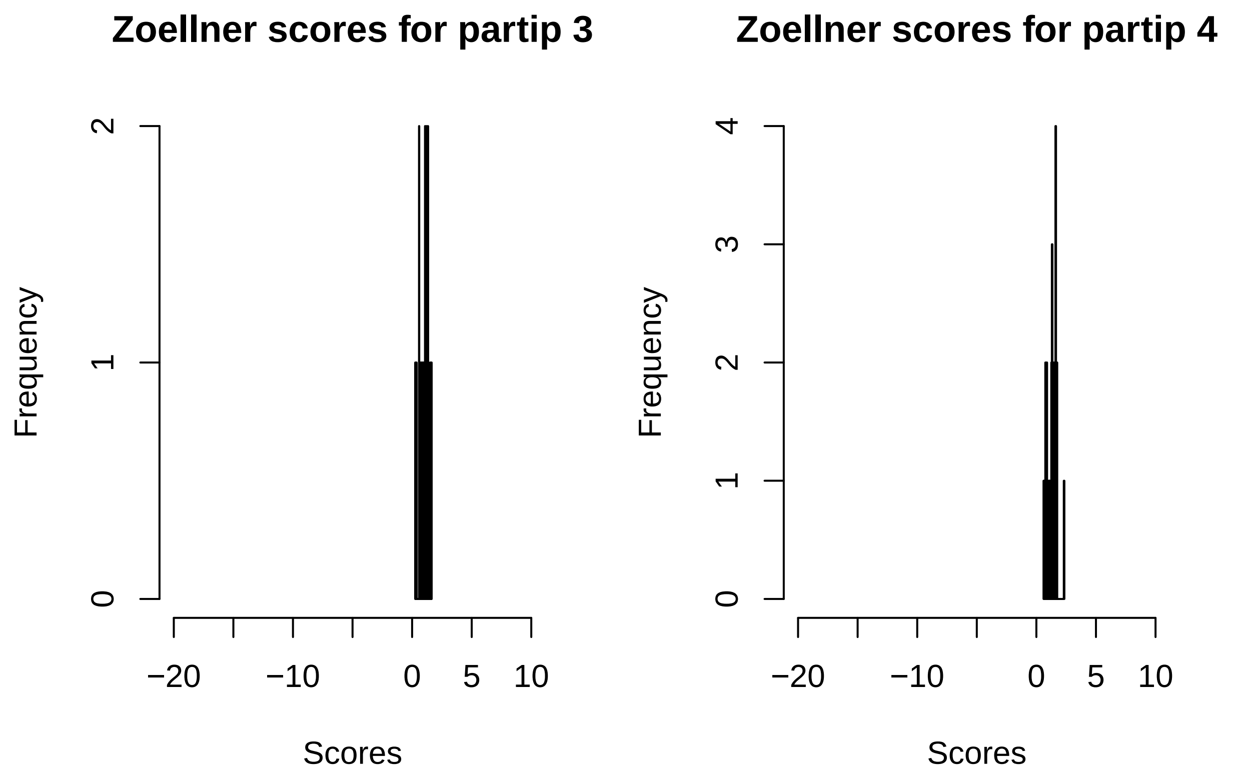
```r
bad_zol_partip = as.integer(names(which(stand_error[,5]>.8)))
temp = dat2[dat2$sub == bad_zol_partip & dat2$task == "zol",]$y
hist(temp, breaks = 50,
     main = paste0("Zoellner scores for partip ", bad_zol_partip),
     xlab = "Scores",
     ylab = "Frequency",
     col = "#B93CE6", border = "black",
     xlim = c(-20,10))
```



Visualizing the variance in scores for the bad participant in Zoellner. We use a threshold on the standard error to flag these outliers.

- Comparing with High-Performing Participants

```r
par(mfrow = c(1,2))
good_zol_partip = 3:4
for (i in good_zol_partip){
  temp = dat2[dat2$sub == i & dat2$task == "zol",]$y
  hist(temp, breaks = 50,
       main = paste0("Zoellner scores for partip ", i),
       xlab = "Scores",
       ylab = "Frequency",
       col = "#B93CE6", border = "black",
       xlim = c(-20,10))
}
```



To contextualize the performance of the flagged participant, we compare it with participants who demonstrate good performance. This comparison is crucial to understand the extent of deviation in the 'zol' task.

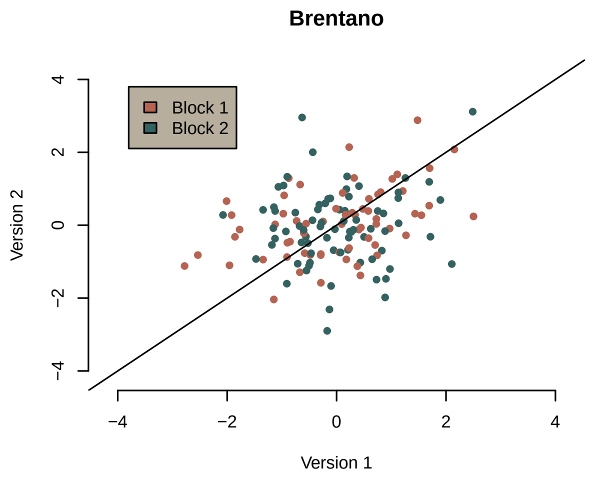
```
ind = which(dat2$sub == bad_zol_partip)
dat3 = dat2[-ind,]
```

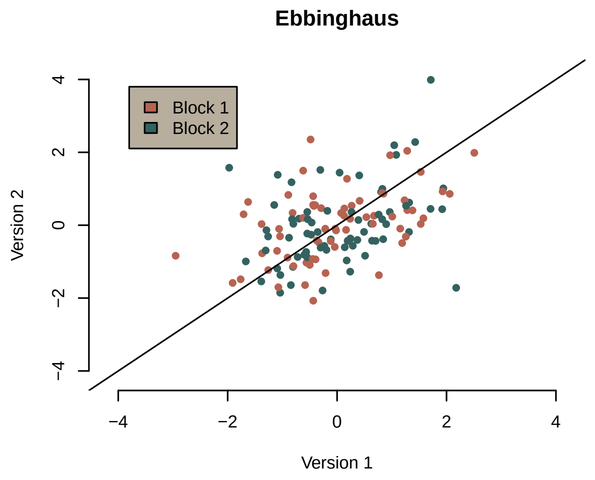
Getting rid of the said participant.

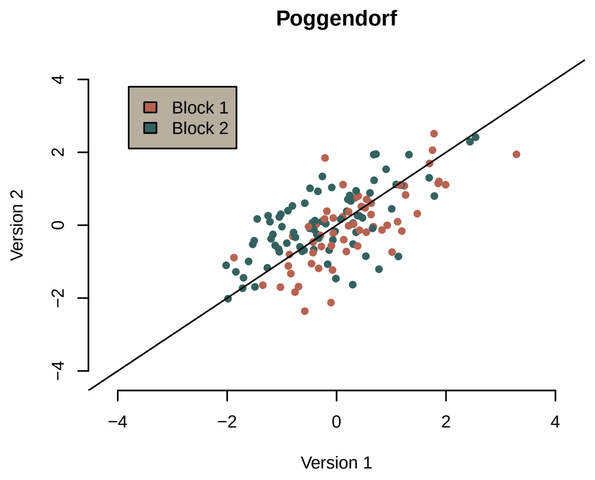
Bivariate Analysis: Examining Task Versions

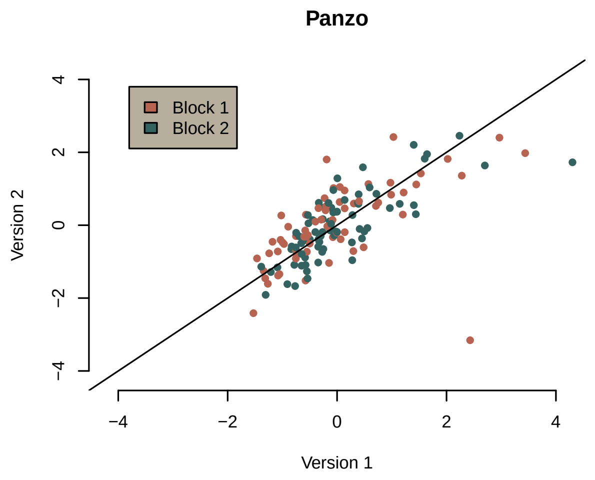
```
scores = tapply(dat3$y,
                list(dat3$sub, dat3$version, dat3$task),
                mean)
blocks = tapply(dat3$block,
                list(dat3$sub, dat3$version, dat3$task),
                mean)
```

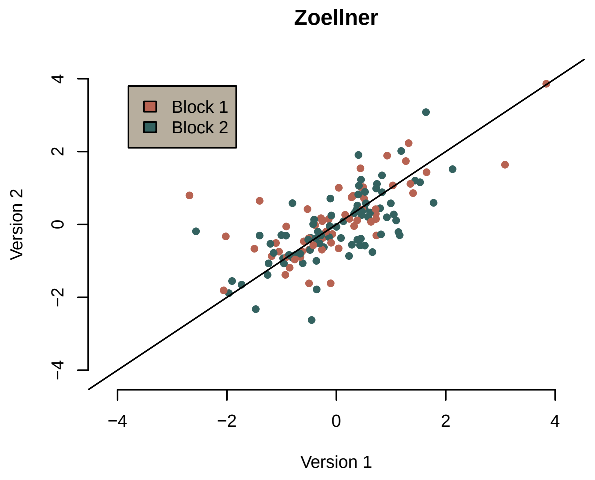
```r
plotBivariate = function(tscores, tblocks, task_name){
  tscores = scale(tscores)
  cols = c("#B76352", "#346260")
  plot(NA,
       NA,
       xlim = c(-4.2, 4.2),
       ylim = c(-4.2, 4.2),
       axes = F,
       xlab = "Version 1",
       ylab = "Version 2",
       main = task_name
  )
  axis(1)
  axis(2)
  points(tscores, col = cols[tblocks[,1]], pch = 16)
  legend(x = -3.8, y = 3.8, legend = c("Block 1", "Block 2"),
         fill = cols, bg = "#B7AE9E" )
  abline(0,1)
}
par(mfrow = c(3,2))
plotBivariate(scores[,,1], blocks[,,1], "Brentano")
plotBivariate(scores[,,2], blocks[,,2], "Ebbinghaus")
plotBivariate(scores[,,3], blocks[,,3], "Poggendorf")
plotBivariate(scores[,,4], blocks[,,4], "Panzo")
plotBivariate(scores[,,5], blocks[,,5], "Zoellner")
```







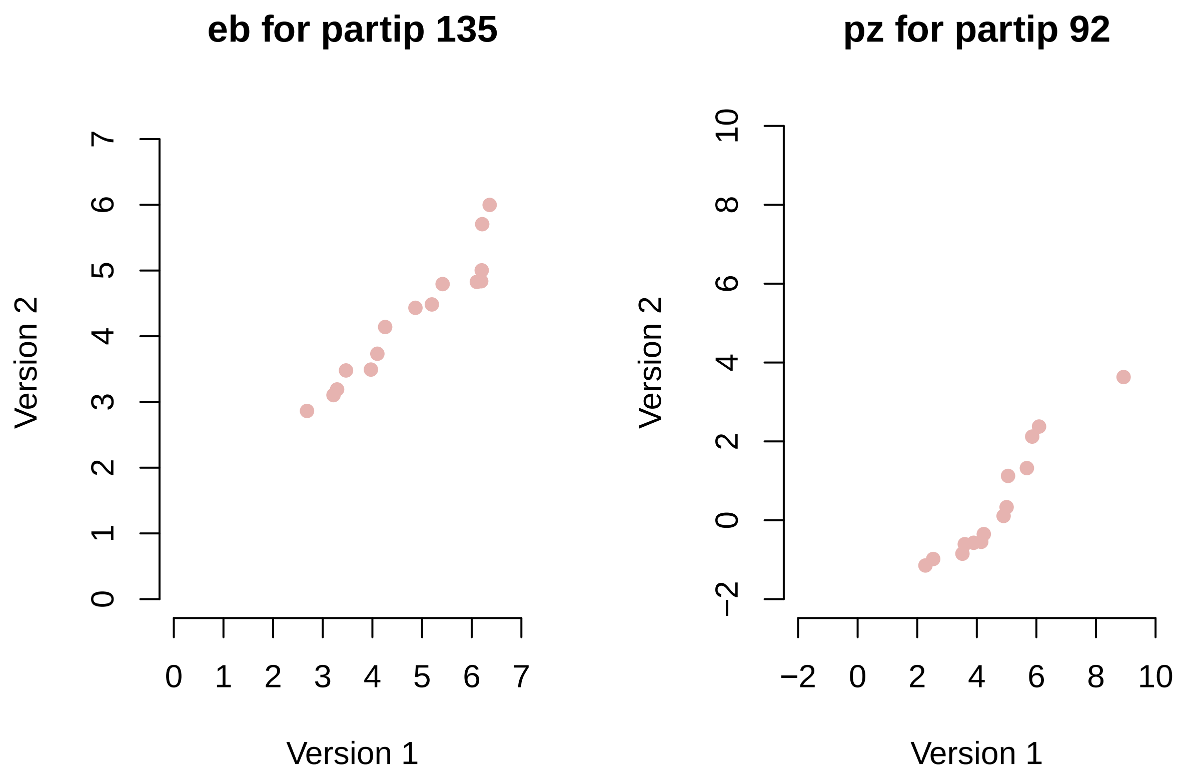




In this section, we perform a bivariate analysis to explore the relationship between different versions of a tasks and the order the participants comepelted them (Block). This approach helps us understand how variations in task design might impact participant performance.

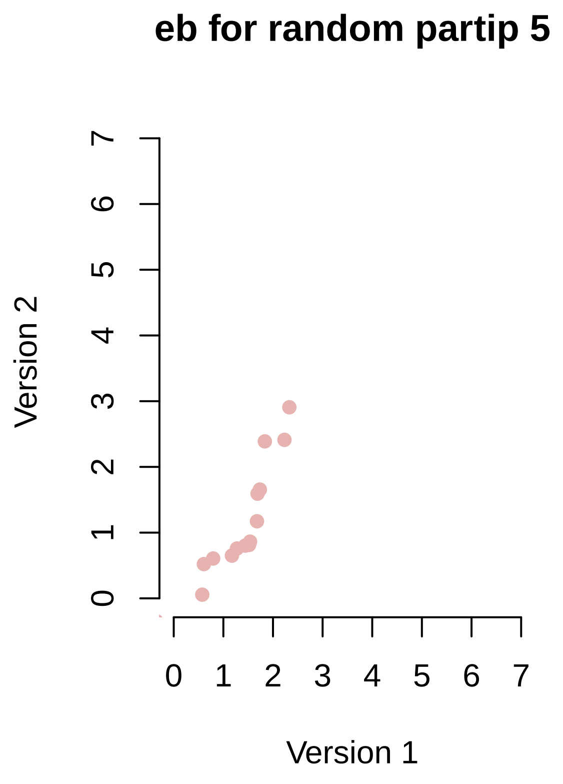
Two particular observation from the Ebbinghaus and Panzo stands out, that is the one with around 2 z-score in version 1 and -2 z-score in version two. Let's investigate this further

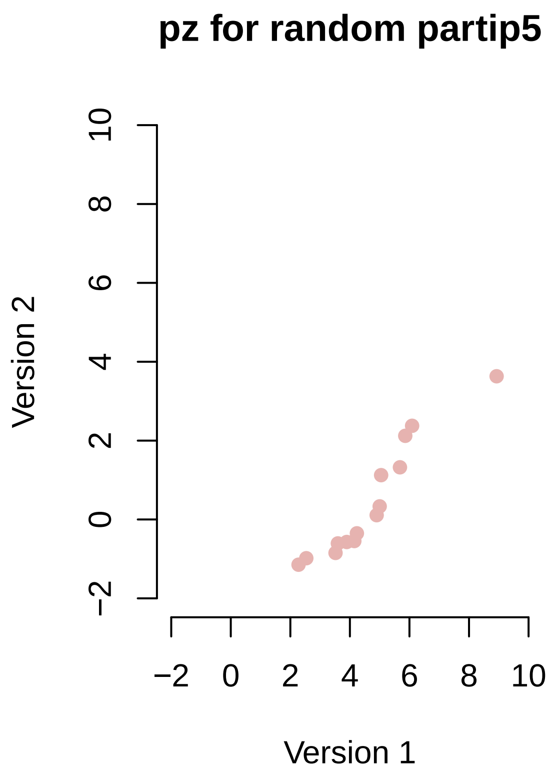
```r
pz = scale(scores[,,4])
eb = scale(scores[,,2])
s_eb = apply(eb, 1, function(x) any(x>1.8) && any(x < -1.7))
s_pz = apply(pz, 1, function(x) any(x>2) && any(x < -2))
bad_eb_sub = as.integer(rownames(eb)[which(s_eb)])
bad_pz_sub = as.integer(rownames(pz)[which(s_pz)])
temp = dat3[dat3$sub==bad_eb_sub,]
eb_temp = tapply(temp$y, list(temp$trial, temp$version), mean)
eb_temp[,1] = sort(eb_temp[,1])
eb_temp[,2] = sort(eb_temp[,2])

par(mfrow=c(1,2))
cols = "#E6B3B0"
plot(NA,
     NA,
     xlim = c(0, 7.2),
     ylim = c(0, 7.2),
     axes = F,
     xlab = "Version 1",
     ylab = "Version 2",
     main = paste0("eb for partip ", bad_eb_sub)
)
axis(1)
axis(2)
points(eb_temp, col = cols, pch = 16)


temp2 = dat3[dat3$sub==bad_pz_sub,]
pz_temp = tapply(temp2$y, list(temp2$trial, temp2$version), mean)
pz_temp[,1] = sort(pz_temp[,1])
pz_temp[,2] = sort(pz_temp[,2])
plot(NA,
     NA,
     xlim = c(-2,10),
     ylim = c(-2,10),
     axes = F,
     xlab = "Version 1",
     ylab = "Version 2",
     main = paste0("pz for partip ", bad_pz_sub)
)
axis(1)
axis(2)
points(pz_temp, col = cols, pch = 16)
```



```
good_eb_sub = 5
good_pz_sub = 5
temp = dat3[dat3$sub%in%good_eb_sub,]
eb_temp = tapply(temp$y, list(temp$trial, temp$version), mean)
eb_temp[,1] = sort(eb_temp[,1])
eb_temp[,2] = sort(eb_temp[,2])

par(mfrow=c(1,2))
cols = "#E6B3B0"
plot(NA,
     NA,
     xlim = c(0, 7.2),
     ylim = c(0, 7.2),
     axes = F,
     xlab = "Version 1",
     ylab = "Version 2",
     main = paste0("eb for random partip ", good_eb_sub)
)
axis(1)
axis(2)
points(eb_temp, col = cols, pch = 16)



temp2 = dat3[dat3$sub%in%bad_pz_sub,]
pz_temp = tapply(temp2$y, list(temp2$trial, temp2$version), mean)
```

```r
pz_temp[,1] = sort(pz_temp[,1])
pz_temp[,2] = sort(pz_temp[,2])
plot(NA,
     NA,
     xlim = c(-2,10),
     ylim = c(-2,10),
     axes = F,
     xlab = "Version 1",
     ylab = "Version 2",
     main = paste0("pz for random partip", good_pz_sub)
)
axis(1)
axis(2)
points(pz_temp, col = cols, pch = 16)
```





Getting rid of the bad Panzo Partip:

```r
ind = which(dat3$sub == bad_pz_sub)
dat4 = dat3[-ind,]
```

Next we will be Identifying High Leverage Points

DFFITS

- To calculate DFFITS, we fit the model with and without a particular observation (which might be influential) and we find the difference between the predicted values at this point according to the two models.
- If a point is influential, the difference in the two predictions should be large.
- The difference between the fitted values is also equal to the difference between residuals and deleted residuals

$$\hat{y}_i - \hat{y}_{i(i)} = (y_i - \hat{y}_{i(i)}) - (y_i - \hat{y}_i) = e_{i(i)} - e_i$$

Calculating DFFITS

- DFFITS are obtained by standardizing the differences $\hat{y}_i - \hat{y}_{i(i)}$, that is:

$$(DFFITS)_i = \frac{\hat{y}_i - \hat{y}_{i(i)}}{\sqrt{MSE_{(i)} - h_{ii}}} = t_i(\frac{h_{ii}}{1-h_{ii}})^{\frac{1}{2}}$$

- DFFITS for the i-th observation is also equal to the studentized residual, multiplied by a function of its leverage.
- If the i-th observation is an outlier in the predictors and it has a high leverage value, $\frac{h_{ii}}{1-h_{ii}}$ is greater than 1 and $(DFFITS)_i$ will be large.

Steps to Calculate DFFITS:

a) Fit Your Linear Model

```r
mod = lm(y ~ task:sub, data = dat4)
```

b) Calculate DFFITS

```r
dffits_values = dffits(mod)
```

c) Interpreting DFFITS

Rules for influential observations

- To identify influential observations we can use the following criteria:
- if the dataset is small ($n < 30$) or it has a medium number of observations: $|DFFITS_i| > 1$;
- if the dataset is large: $|DFFITS_i| > 2\sqrt{\frac{p}{n}}$
  - where p is the number of regression coefficients (and $k = p - 1$ is the number of regressors).

```r
threshold = 2 * sqrt((length(coef(mod)) - 1) / length(fitted(mod)))
# threshold = 1
influential_points = which(abs(dffits_values) > threshold)
print(length(influential_points)/nrow(dat4))
```

```
## [1] 0.06726043
```

The effect on Correlations

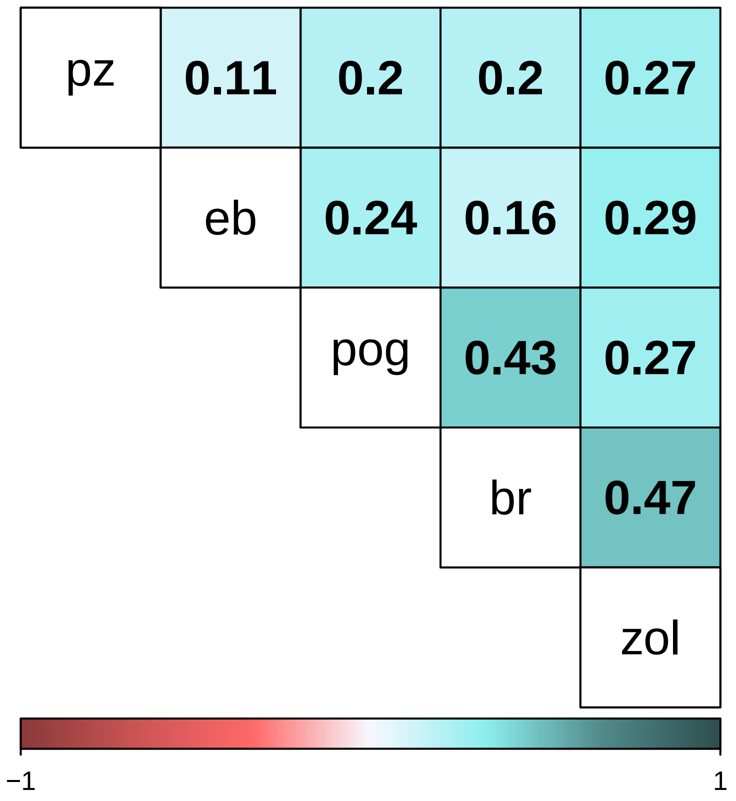
No leverage removal correlations:

```r
library(corrplot)
```

```
## corrplot 0.92 loaded
```

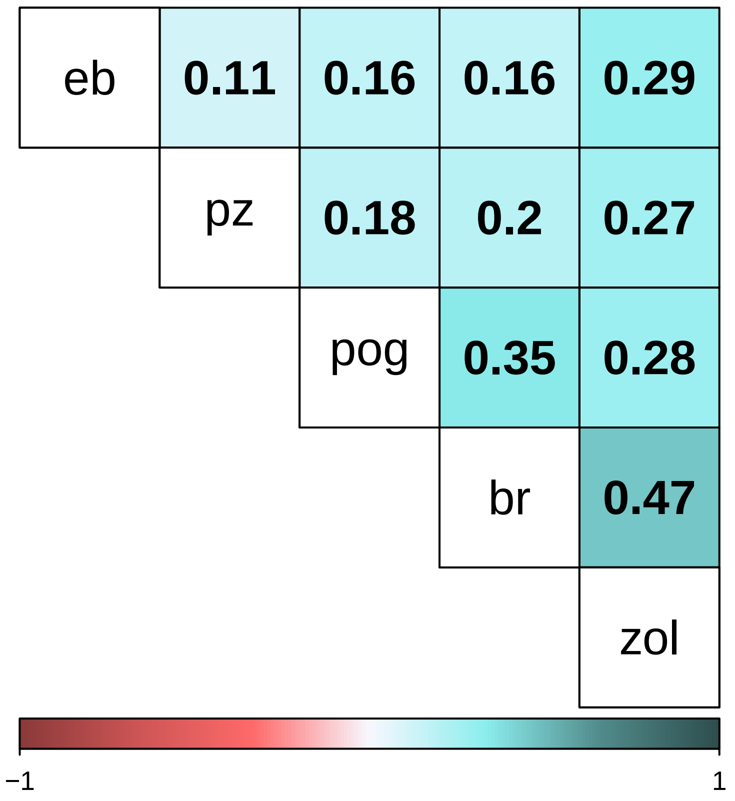
```r
plotCor = function(cors, main = " ", circ = F, clust = T, cx = 1.5){
  col = colorRampPalette(c( "indianred4","indianred3",
                            "indianred1", "ghostwhite",
                            "darkslategray2", "darkslategray4",
                            "darkslategray"))(200)
  if (clust == T){
    dist_mat = as.dist((1 - abs(cors)) / 2)
    hc = hclust(dist_mat, method = "centroid")
    cors = cors[hc$order, hc$order]
  }
  corrplot(cors,
           addCoef.col = "black",
           method = "color",
           col = col,
           cl.pos = 'b',
           tl.pos = 'd',
           tl.col = 'black',
           col.lim = c(-1, 1),
           cl.ratio = 0.2,
           cl.length = 2,
           addgrid.col = 'black',
           type = "upper",
           rect.lwd = 10,
           tl.cex = cx,
           number.cex = cx)
  if (circ != F){
    for (r in nrow(circ)){
      addCircle(circ[r, ])
    }
  }
}
```

```r
dat4$S = as.numeric(factor(dat4$sub))
dat4$J = as.numeric(factor(dat4$task))
scores = tapply(dat4$y, list(dat4$S, dat4$J), mean)
scores_cor = round(cor(scores),2)
rownames(scores_cor) = colnames(scores_cor) = sort(unique(dat4$task))
plotCor(scores_cor)
```



Leverage removal correlations:

```
tdat = dat4[-influential_points,]
scores = tapply(tdat$y, list(tdat$S, tdat$J), mean, na.rm   = T)
tdat.cor = cor(scores, use = "pairwise.complete.obs")
rownames(tdat.cor) = colnames(tdat.cor) = sort(unique(dat4$task))
plotCor(tdat.cor)
```



No effect. So we will not take out the leverage points identified by *DFFITS*

DFBETAS

- The influence of observation $i$ on coefficient β_j is: $\hat{\beta}_j - \hat{\beta}_{j(i)}$ where $\hat{\beta}_{j(i)}$ is the estimate of the β_j coefficient obtained when the model is fitted without the \$i\$0-th observation.
- The difference in β_j associated with observation $i$ is given by the $\hat{\beta}_j - \hat{\beta}_{j(i)}$ divided by its standard error:

$$(DFBETAS)_{j,i} = \frac{\hat{\beta}_j - \hat{\beta}_{j(i)}}{\sqrt{MSE_{(i)} - c_{jj}}} \qquad j = 0, 1, ..., p$$

where $c_j j$ is the $(j+1)$-th diagonal element of $(\mathrm{X}'\mathrm{X})^{-1}$, since $Var(\hat{\beta}) = \sigma^2 \cdot (\mathrm{X}'\mathrm{X})^{-1}$.

- The sign of DFBETAS indicates whether excluding an observation leads to an increase or decrease in the estimated regression coefficient.
- The magnitude of DFBETAS indicates the size of the difference relative to the estimated standard deviation of the regression coefficient.

Calculating DFBETAS

- Calculate DFBETAS

```
dfbetas_values = dfbetas(mod)
```

Rules for influential observations

- To identify influential observations we can use the following criteria:
  - if the dataset is small $(n < 30):\ |(DFBETAS)_{j,i}| > 1$;
  - if the dataset is large: $|(DFBETAS)_{j,i}| > \frac{2}{\sqrt{n}}$.

Interpreting DFBETAS

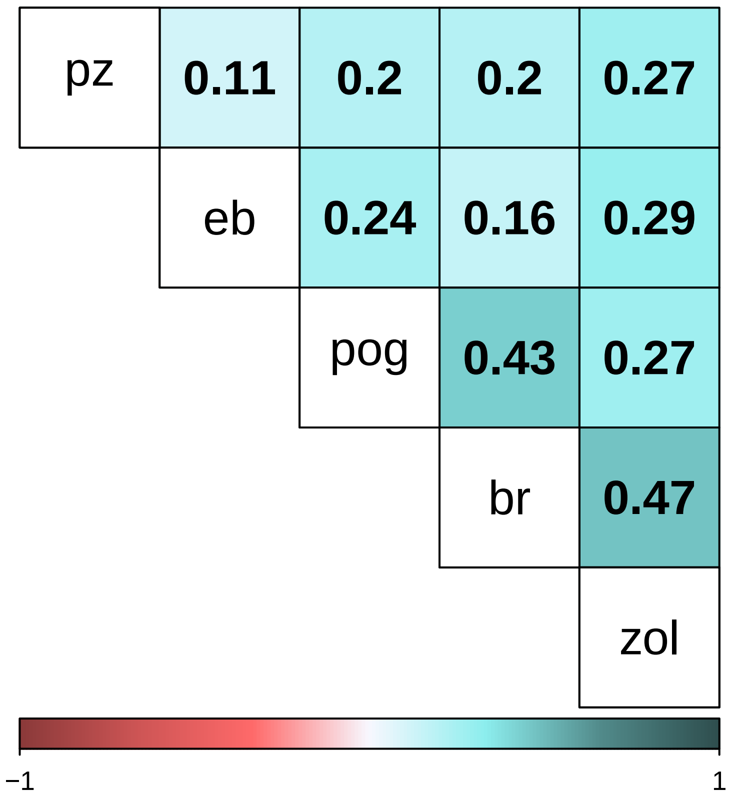
```
threshold = 2 / sqrt(nrow(dat4))
influential_points2 = which(abs(dfbetas_values) > threshold)
print(length(influential_points2)/nrow(dat4))
```

```
## [1] 0.2417385
```

The effect on Correlations

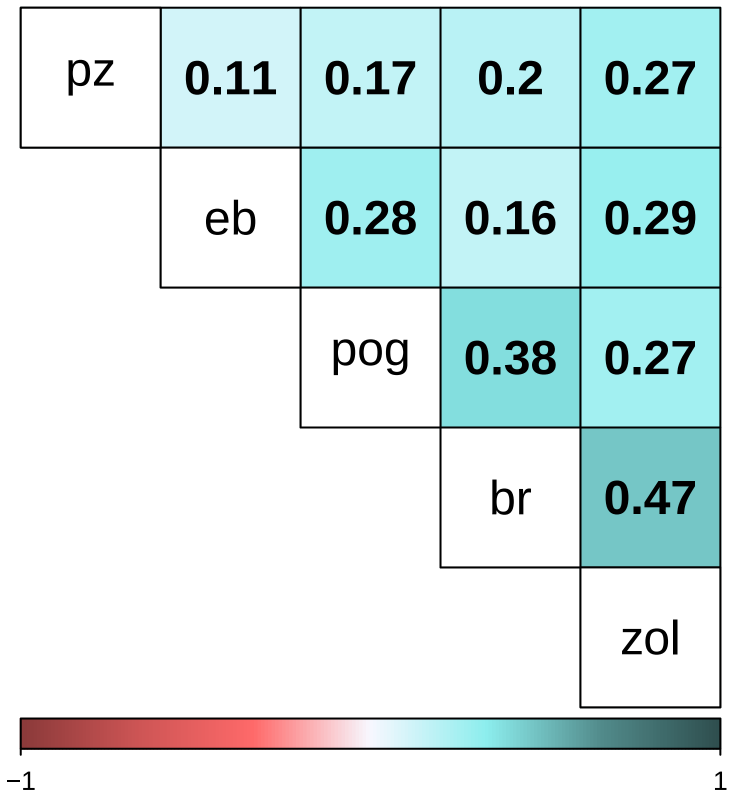
No leverage removal correlations:

```
dat4$S = as.numeric(factor(dat4$sub))
dat4$J = as.numeric(factor(dat4$task))
scores = tapply(dat4$y, list(dat4$S, dat4$J), mean)
scores_cor = round(cor(scores),2)
rownames(scores_cor) = colnames(scores_cor) = sort(unique(dat4$task))
plotCor(scores_cor)
```



Leverage removal correlations:

```
tdat = dat4[-influential_points2,]
scores = tapply(tdat$y, list(tdat$S, tdat$J), mean, na.rm   = T)
tdat.cor = cor(scores, use = "pairwise.complete.obs")
rownames(tdat.cor) = colnames(tdat.cor) = sort(unique(dat4$task))
plotCor(tdat.cor)
```



Cook's distance

- Examining DFBETAS for each coefficient can be tedious. We combine the DFBETAS for all the coefficients into a single measure called Cook's distance, denoted by:

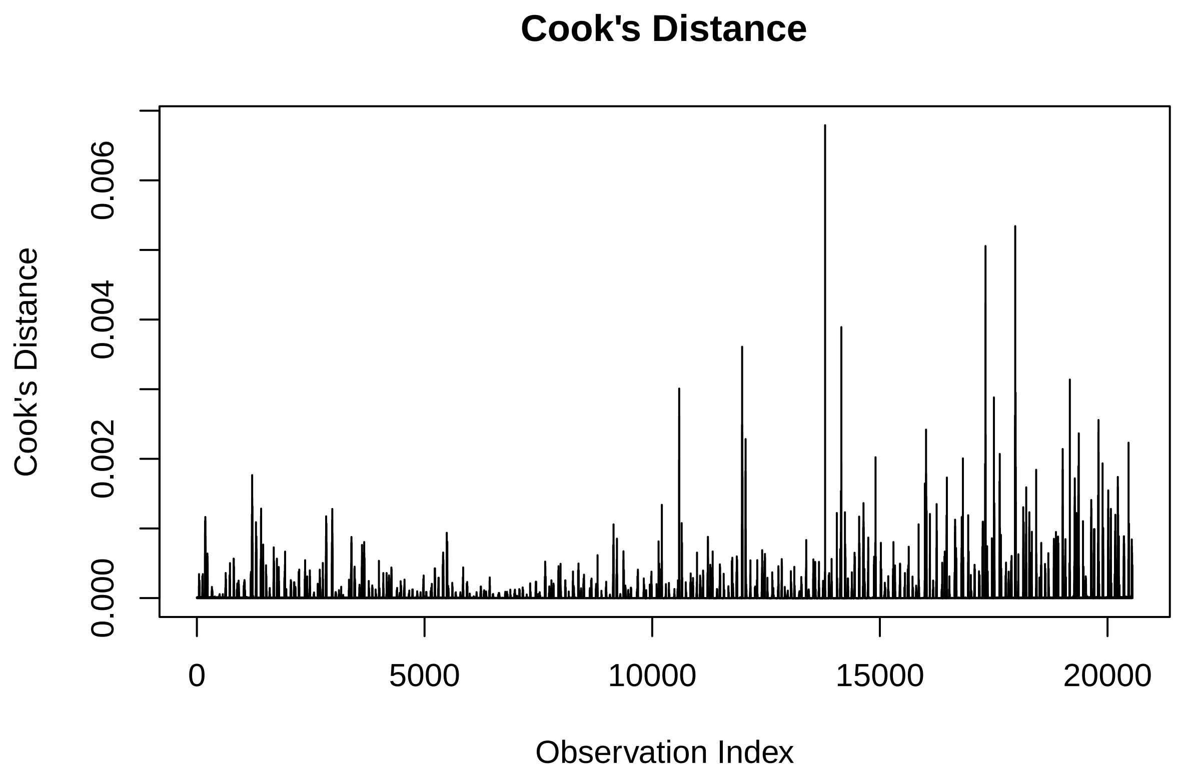
$$
\begin{aligned}
D_i &= \frac{(\hat{\beta}_j - \hat{\beta}_{j(i)})' \cdot (\mathrm{X}'\mathrm{X})^{-1} \cdot (\hat{\beta}_j - \hat{\beta}_{j(i)})}{pMSE} \\
&= \frac{e_i^2}{p \cdot MSE} \cdot \frac{h_{ii}}{(1 - h_{ii})^2}
\end{aligned}
$$

- Cook's distance examines the impact of a single observation on all the regression coefficients, hence it can also be interpreted as the impact of a single observation on all the fitted values.
- Note that D_i depends on two factors: the size of the residual e_i and the leverage $h_i i$. When either of these increase, D_i increases as well.

Calculating DFBETAS

```
cooks_dist_values = cooks.distance(mod)
p = length(coef(mod))
n = nrow(dat4)
f_threshold = qf(0.8, df1 = p, df2 = n - p)
```

```r
plot(cooks_dist_values, type = "h",
     main = "Cook's Distance",
     ylab = "Cook's Distance",
     xlab = "Observation Index")
abline(h = f_threshold, col = "red")
```



Rules for influential observations

- A large value of D_i indicates that the observation has a large influence on the results.
- **QUESTION:** How large should D_i be?
- To determine what is considered large for D_i , we use the fact that Di follows approximately a $F_{p,n-p}$ distribution.
- Hence, the i-th observation has:
  - a small influence if $D_i < F_{0.2;p,n-p}$;
  - a large influence if $D_i > F_{0.2;p,n-p}$.

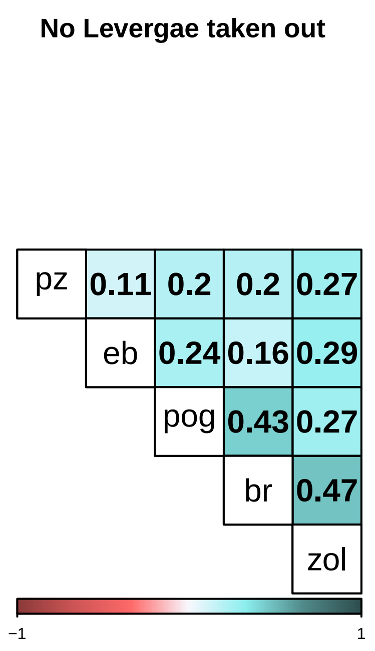
```r
influential_points3 = which(cooks_dist_values > f_threshold)
print(length(influential_points3)/nrow(dat4))
```

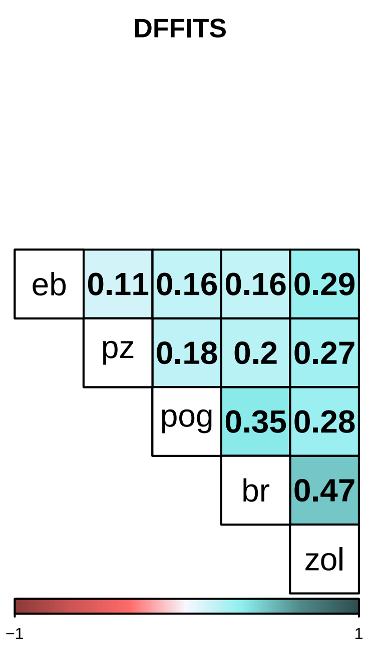
```
## [1] 0
```

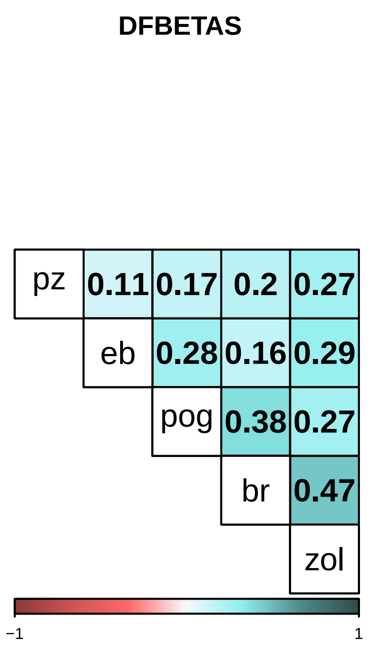
No Observation is flagged.

Visluzing the Corrlations Again

```r
par(mfrow = c(1, 3), mar = c(2.1, 4.1, 3.8, 2.1))

dat4$S = as.numeric(factor(dat4$sub))
dat4$J = as.numeric(factor(dat4$task))
scores1 = tapply(dat4$y, list(dat4$S, dat4$J), mean)
scores_cor = round(cor(scores1),2)
rownames(scores_cor) = colnames(scores_cor) = sort(unique(dat4$task))
plotCor(scores_cor)
title("No Levergae taken out", line = 3)



tdat = dat4[-influential_points,]
scores2 = tapply(tdat$y, list(tdat$S, tdat$J), mean, na.rm  = T)
tdat.cor = cor(scores2, use = "pairwise.complete.obs")
rownames(tdat.cor) = colnames(tdat.cor) = sort(unique(dat4$task))
plotCor(tdat.cor)
title("DFFITS", line = 3)


tdat = dat4[-influential_points2,]
scores3 = tapply(tdat$y, list(tdat$S, tdat$J), mean, na.rm  = T)
tdat.cor = cor(scores3, use = "pairwise.complete.obs")
rownames(tdat.cor) = colnames(tdat.cor) = sort(unique(dat4$task))
plotCor(tdat.cor)
title("DFBETAS", line = 3.1)
```







## Looking at the effect on PCA with and without high Leverage points

```
# library(extr)
pca_std_1 = stats::prcomp(scores1, scale = TRUE, center = TRUE)
scores2 = na.omit(scores2)
pca_std_2 = prcomp(scores2, scale = TRUE, center = TRUE)
scores3 = na.omit(scores3)
pca_std_3 = prcomp(scores3, scale = TRUE, center = TRUE)

print("No Levrage")
```

```
## [1] "No Levrage"
```

```
pca_std_1$sdev[1]^2/pca_std_1$sdev[3]^2
```

```
## [1] 2.479976
```

```
print("DFFITIS")
```

```
## [1] "DFFITIS"
```

```
pca_std_2$sdev[1]^2/pca_std_2$sdev[3]^2
```

```
## [1] 2.388655
```

```r
print("DFBETAS")
```

```
## [1] "DFBETAS"
```

```r
pca_std_3$sdev[1]^2/pca_std_3$sdev[3]^2
```

```
## [1] 2.45246
```