# Wilcoxon Signed Rank Test

Joseph Oliveira

7/4/2020

### Loading data in from the `Wilcoxon_Test.Rmd` cleaning procedure

```r
faces <- faces %>%
  group_by(Group, `Participant #`, Time, Survey) %>%
  summarise(avg_resp = mean(Response)) %>%
  ungroup()
```

Wilcoxon Test function

1. Cleaned `faces` data is filtered for the Time (`Pre` or `Post`) and Group (`Experimental` or `Control`). Depends one what is being compared
2. Two data sets are created to create the vectors for comparison
3. 3 Cases need to be handled a. If the vector of response variables in data set A are longer than B b. If the vector of response variables in data set B are longer than A c. If the vector of response variables in data set A and B are equal length

```r
faces_wilcox <- function(time1, group1, time2, group2, survey, ...) {
  # Filter to data we need for comparison
  faces_ <- faces %>%
    filter(Time %in% c(time1, time2) & Group %in% c(group1, group2))

  # Create 2 datasets
  comp1 <- faces_ %>%
    filter(Time == time1, Group == group1, Survey == survey)

  comp2 <- faces_ %>%
    filter(Time == time2, Group == group2, Survey == survey)

  # 3 Cases need to be handled:
  # 1. If the lengths are unequal. Sample longer sample to obtain equal comparison
  if (length(comp1$avg_resp) > length(comp2$avg_resp)) {

    comp1_spl_resp <- sample(comp1$avg_resp, replace = T, size = length(comp2$avg_resp))

    wilcox.test(comp1_spl_resp, comp2$avg_resp)

    # Second case for unequal lengths
    } else if (length(comp1$avg_resp) < length(comp2$avg_resp)) {
```

```
    comp2_spl_resp <- sample(comp2$avg_resp, replace = T, size = length(comp1$avg_resp))

    wilcox.test(comp1$avg_resp, comp2_spl_resp, ...)

    # if they are equal, then run comparison.
    } else wilcox.test(comp1$avg_resp, comp2$avg_resp)
  }
```

### Creating the comparison groups for the function above.

```
distinct_groupings <- faces %>%
  distinct(Group, Time, Survey)

dist_grp_exp_pre <- distinct_groupings %>%
  filter(Group == 'Experimental', Time == 'Pre')

dist_grp_ctrl_pre <- distinct_groupings %>%
  filter(Group == 'Control', Time == 'Pre')

dist_grp_exp_post <- distinct_groupings %>%
  filter(Group == 'Experimental', Time == 'Post')

dist_grp_ctrl_post <- distinct_groupings %>%
  filter(Group == 'Control', Time == 'Post')

first_comp <- inner_join(dist_grp_exp_pre, dist_grp_ctrl_pre, by = c("Survey", "Time"))
secnd_comp <- inner_join(dist_grp_exp_post, dist_grp_ctrl_post, by = c("Survey", "Time"))
third_comp <- inner_join(dist_grp_exp_pre, dist_grp_exp_post, by = c("Survey", "Group"))
forth_comp <- inner_join(dist_grp_ctrl_pre, dist_grp_ctrl_post, by = c("Survey", "Group"))

group_comparison <- bind_rows(first_comp, secnd_comp) %>% mutate_all(as.character)

time_comparison <- bind_rows(third_comp, forth_comp) %>% mutate_all(as.character)
```

## Running Wilcoxon tests

test: `pmap` will map specified columns of a tibble to the arguments of a function., and will use each record of the specified columns as an argument in the function specified.

p.value: `map` is like `lapply` but for tibbles. Just like `pmap` it iterates through each record of specified column, passing it to the function call. Here the function calls are subsetting calls, just instead of `x[1]` or `x[[1]]` I am calling `'['(x)` or rater `'['( '['( x ) )`. The back-ticks make R treat the call as a `prefix`, `fn(arg1, arg2)`, instead of `infix`, `arg1 fn arg2`.

- For each test result, I'm pulling out the p.value: `test[[x]]$p.value`

```
group_wilcoxon <- group_comparison %>%
  # defined the test result
  mutate(test = pmap(list(time1 = Time, group1 = Group.x,
                          time2 = Time, group2 = Group.y,
                          survey = Survey),
                     faces_wilcox, paired = F),
         p.value = unlist(map(test, function(x) `$`(`[`(`[`(x)), 'p.value'))))

time_wilcoxon  <- time_comparison %>%
  mutate(test = pmap(list(time1 = Time.x, group1 = Group,
                          time2 = Time.y, group2 = Group,
                          survey = Survey),
                     faces_wilcox, paired = F),
         p.value = unlist(map(test, function(x) `$`(`[`(`[`(x)), 'p.value'))))
```

```
knitr::kable(select(group_wilcoxon, Time, Survey, Group.x, Group.y, p.value) %>%
               group_by(Survey) %>%
               arrange(Survey, p.value, Time))
```

| Time | Survey | Group.x | Group.y | p.value |
|---|---|---|---|---:|
| Post | AKS | Experimental | Control | 0.2442363 |
| Pre | AKS | Experimental | Control | 0.2795693 |
| Post | FACES | Experimental | Control | 0.5731549 |
| Pre | FACES | Experimental | Control | 0.6863111 |
| Post | FES | Experimental | Control | 0.3776424 |
| Pre | FES | Experimental | Control | 0.5738309 |
| Pre | FPPS | Experimental | Control | 0.1102102 |
| Post | FPPS | Experimental | Control | 0.1464892 |
| Pre | SCS | Experimental | Control | 0.6631172 |
| Post | SCS | Experimental | Control | 0.7715034 |
| Post | SEAS | Experimental | Control | 0.1030947 |
| Pre | SEAS | Experimental | Control | 0.5196412 |

```
knitr::kable(select(time_wilcoxon, Group, Survey, Time.x, Time.y, p.value) %>%
               group_by(Survey) %>%
               arrange(Survey, p.value, Group))
```

| Group | Survey | Time.x | Time.y | p.value |
|---|---|---|---|---:|
| Experimental | AKS | Pre | Post | 0.2886460 |
| Control | AKS | Pre | Post | 0.6235742 |
| Experimental | FACES | Pre | Post | 0.0567903 |
| Control | FACES | Pre | Post | 0.9357363 |
| Experimental | FES | Pre | Post | 0.0311456 |
| Control | FES | Pre | Post | 0.7479209 |
| Experimental | FPPS | Pre | Post | 0.7117698 |
| Control | FPPS | Pre | Post | 0.7715034 |
| Experimental | SCS | Pre | Post | 0.2052307 |
| Control | SCS | Pre | Post | 0.3428571 |
| Experimental | SEAS | Pre | Post | 0.1007655 |
| Control | SEAS | Pre | Post | 0.8095268 |