

Wilcoxon Signed Rank Test

Joseph Oliveira

7/4/2020

Loading data in from the Wilcoxon_Test.Rmd cleaning procedure

```
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 on 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

```
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)  
  
  n1 <- length(comp1$avg_resp)  
  n2 <- length(comp2$avg_resp)  
  
  # 3 Cases need to be handled:  
  # 1. If the lengths are unequal. Sample longer sample to obtain equal comparison  
  if (n1 > n2) {  
    comp1_spl_resp <- sample(comp1$avg_resp, replace = T, size = n2)  
  
    x <- wilcox.test(comp1_spl_resp, comp2$avg_resp, ...)  
    x$obs <- n2 + n2  
  }
```

```

return(x)

# Second case for unequal lengths
} else if (n1 < n2) {

  comp2_spl_resp <- sample(comp2$avg_resp, replace = T, size = n1)

  x <- wilcox.test(comp1$avg_resp, comp2_spl_resp, ...)
  x$obs <- n1 + n1
  return(x)

  # if they are equal, then run comparison.
} else {

  x <- wilcox.test(comp1$avg_resp, comp2$avg_resp, ...)
  x$obs <- n1 + n2
  return(x)

}
}

```

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 rather `'['('['(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, alternative = "less"),
          p.value = unlist(map(test, function(x) `$(`['(`['(x)), 'p.value']))),
          effect_size = unlist(map(test, function(x) `$(`['(`['(x)), 'p.value']))) /
                        unlist(map(test, function(x) `$(`['(`['(x)), 'obs'])))**0.5)

time_wilcoxon <- time_comparison %>%
  mutate(test = pmap(list(time1 = Time.x, group1 = Group,
                          time2 = Time.y, group2 = Group,
                          survey = Survey),
                      faces_wilcox, paired = F, alternative = "less"),
          p.value = unlist(map(test, function(x) `$(`['(`['(x)), 'p.value']))),
          effect_size = unlist(map(test, function(x) `$(`['(`['(x)), 'p.value']))) /
                        unlist(map(test, function(x) `$(`['(`['(x)), 'obs'])))**0.5)
```

Wilcoxon Test Results

Experimental vs Control

Time	Survey	Group.x	Group.y	p.value	effect_size
Pre	AKS	Experimental	Control	0.1397847	0.0403524
Post	AKS	Experimental	Control	0.9083751	0.2622253
Pre	FACES	Experimental	Control	0.7141035	0.2061439
Post	FACES	Experimental	Control	0.7655856	0.2210055
Pre	FES	Experimental	Control	0.2869155	0.0828254
Post	FES	Experimental	Control	0.8514465	0.2457914
Post	FPPS	Experimental	Control	0.9592854	0.3391586
Pre	FPPS	Experimental	Control	0.9704641	0.3431109
Post	SCS	Experimental	Control	0.7193184	0.2543175
Pre	SCS	Experimental	Control	0.7660875	0.2708528
Pre	SEAS	Experimental	Control	0.7895536	0.2279245
Post	SEAS	Experimental	Control	0.9635164	0.2781432

Pre vs Post

Group	Survey	Time.x	Time.y	p.value	effect_size
Experimental	AKS	Pre	Post	0.1443230	0.0322716
Control	AKS	Pre	Post	0.7435691	0.2146499
Experimental	FACES	Pre	Post	0.0283952	0.0063494
Control	FACES	Pre	Post	0.5955672	0.1719254
Experimental	FES	Pre	Post	0.0155728	0.0034822
Control	FES	Pre	Post	0.3739605	0.1079531
Experimental	FPPS	Pre	Post	0.3558849	0.0889712
Control	FPPS	Pre	Post	0.3857517	0.1363838
Experimental	SCS	Pre	Post	0.1026153	0.0256538
Control	SCS	Pre	Post	0.1714286	0.0606092
Experimental	SEAS	Pre	Post	0.0503828	0.0125957
Control	SEAS	Pre	Post	0.4047634	0.1168451