Code Folien

Code „Simulation“

- Brief:
  - Joachim

Code „ANOVA“

- Normal:
  - **Function:**

```
ow_a(rma_data, id = 1)
```

  - **Define needed constants :**

```
# Number of entities in one group
n_group = nrow(ow_rma_data)

# Number of factor levels
k = ncol(ow_rma_data) - 1

# Number of entities
n = (k * n_group)
```

Code „ANOVA“

– **Define basic ANOVA components:**

```
# Computation of the baseline component
  grand_mean = mean(as.matrix(ow_rma_data[,2: (k + 1)]))
  baseline_components = matrix(rep(grand_mean, times = n), nrow = n_group)

# Computation of the factor level components
  conditional_means = apply(dependent_variable, 2, mean)
  factor_level_components =
      matrix(rep(conditional_means - grand_mean, each = n_group), nrow = n_group)

 # Computation of the error components
  error_components_ANOVA =
      dependent_variable - baseline_components - factor_level_components
```

Code „ANOVA“

- **Decomposition matrix:**

```
decomposition_matrix_ANOVA = data.frame("dependent_variable" = numeric(n),
                             "baseline" = numeric(n),
                             "factor_level" = numeric(n),
                             "error" = numeric(n)
                             )

decomposition_matrix_ANOVA$dependent_variable =          as.vector(dependent_variable)
decomposition_matrix_ANOVA$baseline = as.vector(baseline_components)
decomposition_matrix_ANOVA$factor_level =       as.vector(factor_level_components)
decomposition_matrix_ANOVA$error = as.vector(error_components_ANOVA)
```

- **Compute sums of squares:**

```
ss_ANOVA = as.data.frame(t(colSums(decomposition_matrix_ANOVA^2)))
rownames(ss_ANOVA) = "sums_of_squares„
```

Code „rmANOVA“

- Detailed:
  - **Function:**

```
ow_rma(rma_data, id = 1)
```

  - **Define needed constants:**

```
# Number of entities
  n = nrow(ow_rma_data)

# Number of factor levels
  k = ncol(ow_rma_data) – 1
```

  - **Define basic rmANOVA components:**

```
grand_mean = mean(as.matrix(ow_rma_data[,2: (k + 1)]))
baseline_components = matrix(rep(grand_mean, times = k*n), nrow = n)

conditional_means = apply(dependent_variable, 2, mean)
factor_level_components = matrix(rep(conditional_means - grand_mean, each = n), nrow = n)

subject_means = apply(dependent_variable, 1, mean)
subject_components = matrix(rep(subject_means - grand_mean, times = k), nrow = n)

error_components = dependent_variable - baseline_components - factor_level_components - subject_components
```

Code „rmANOVA“

- Detailed:
  - **Decomposition matrix:**

```
decomposition_matrix = data.frame("dependent_variable" = numeric(n*k),
                                  "baseline" = numeric(n*k),
                                  "factor_level" = numeric(n*k),
                                  "subject_level" = numeric(n*k),
                                  "error" = numeric(n*k)
                                  )

decomposition_matrix$dependent_variable = as.vector(dependent_variable)
decomposition_matrix$baseline = as.vector(baseline_components)
decomposition_matrix$factor_level = as.vector(factor_level_components)
decomposition_matrix$subject_level = as.vector(subject_components)
decomposition_matrix$error = as.vector(error_components)
```

Code „rmANOVA“

– **Compute sums of squares:**

```
ss = as.data.frame(t(colSums(decomposition_matrix^2)))
rownames(ss) = "sums_of_squares"
```

– **Set degrees of freedom:**

```
dof = data.frame("dependent_variable" = n*k,
            "baseline" = 1,
            "factor_level" = k-1,
            "subject_level" = n-1,
            "error" = (n*k)-1-(k-1)-(n-1)
            )
```

– **Compute mean squares:**

```
ms = ss / dof
rownames(ms) = "mean_squares"
```

– **Compute corrected total sum of squares (variance):**

```
corrected_sst = ss$dependent_variable - ss$baseline
variance = corrected_sst / (dof$dependent_variable - dof$baseline)
```

Code „rmANOVA“

- **Compute F-values:**

```
F_value_factor = ms$factor_level / ms$error
F_value_baseline = ms$baseline / ms$subject_level
```

- **Set p-values of F distribution:**

```
p_factor = 1 - pf(F_value_factor, dof$factor_level, dof$error)
p_baseline = 1 - pf(F_value_baseline, dof$baseline, dof$subject_level)
```

Code „Between Subject SS“

- Brief:
  - Function:

```
ow_rma_sse_reduct(rma_data, id = 1, plot_type = "pie", return_anova_table = FALSE)
```

  - SSE in RM ANOVA is equal to SSE ANOVA minus SS Subjects:

```
# ANOVA-tables of rmANOVA and ANOVA without repeated measures
  ow_a_results = ow_a(ow_rma_data)[[1]]
  ow_rma_results = ow_rma(ow_rma_data)[[1]]

  sse_anova = ow_a_results[3, 2]
  ss_subject_anova = 0
# Always zero because the subject effect is not considered in an ANOVA without repeated
measures

  sse_rma = ow_rma_results[4, 2]
  ss_subject_rma = ow_rma_results[3, 2]
```

Code „CI“

- Brief:
  - **Function:**

```
rma_ci(rma_data, C_level = 0.95, id = 1, print_plot = TRUE)
```

  - **Compute CI for Anova without repeated measures:**

```
# Standard errors of the conditional means
  SE = tapply(rma_data_long$value, rma_data_long$condition, sd)/sqrt(n)

# CI for ANOVA without repeated measures
  CIdist_unadj = abs(qt((1 - C_level)/2, (n - 1))) * SE
```

  - **Compute CI for Anova with repeated measures:**

```
  # Correction factor etablished by Morey (2008)
  cf = sqrt(k/(k - 1))

  AdjVal = data.frame(Adj = (cf * ((rma_data_long$value - EEmlong$Em + Gm) - MeFlmlong$Flm)) + MeFlmlong$Flm)
  rma_data_long_adj = cbind.data.frame(rma_data_long, AdjVal)

  # Standard errors of the conditional means adjusted with the method of O'Brien and Cousineau (2014, see also Loftus & Masson;
1994)
  SE_adj = (tapply(rma_data_long_adj$Adj, rma_data_long_adj$condition, sd)/sqrt(n))
  CIdist_adj = abs(qt((1 - C_level)/2, (n - 1))) * SE_adj
```

Code „Effekt Size“

- Brief:
  - **Function:**

```
rma_eta(rma_data, id = 1, append = FALSE)
```

  - **Compute effect size measures:**

```
# Compute standard eta^2
   eta_sq = SS_Factor/SS_K_Total

# Compute partial eta^2
   eta_partial = SS_Factor/(SS_Factor + SS_Error)
```

Code „Mauchly‘s Test“

- Medium:
  - **Defining some variables:**

```
# Factor degrees of freedom
   df = k - 1

# Empirical covariance matrix
   covariance_matix = cov(dependent_variable)
```

  - **Helmert matrix required for the computation of mauchly's W:**

```
helmert = function(k) {
    H = matrix(0, k, k)
    diag(H) = (0:df) * (-((0:df) * ((0:df) + 1))^(-0.5))
    for (i in 2:k) {
       H[i, 1:(i - 1)] = ((i - 1) * (i))^(-0.5)
    }
    # H[1,] = 1/sqrt(k)
    return(H)
 }

# The first row of the helmert matrix is not required for further use in this procedure
     C = helmert(k)[-1, ]
```

Code „Mauchly‘s Test“

- Medium:
  - **Computation of mauchly's W:**

```
w = det(C %*% covariance_matix %*% t(C))/((sum(diag(C %*%
covariance_matix %*% t(C)))/df)^(df))
```

  - **Chi-Square Test:**

```
# Computing the degrees of freedom for the chi-square value
    df_w = ((k * df)/2) - 1

# Computing the chi-square value
    f = 1 - ((2 * (df^2) + df + 2)/(6 * df * (n - 1)))
    chi_sq_w = -(n - 1) * f * log(w)

# Computing the corresponding p-value
    p_w = 1 - pchisq(chi_sq_w, df_w)
```

Code „Correction of p-Values“

- Brief:
  - **The three different correction factors (epsilon):**

```
# Lower-Bound correction (Greenhouse & Geisser, 1959)
    epsilon_lb = 1/df


# Box correction (Geisser & Greenhouse, 1958)
    epsilon_gg = (sum(diag(C %*% covariance_matix %*%
t(C)))^2)/(df * sum(diag(t((C %*% covariance_matix %*% t(C)))
%*% (C %*% covariance_matix %*% t(C)))))


# Huynh-Feldt correction (Huynh & Feldt, 1976)
    epsilon_hf = min((n * df * epsilon_gg - 2)/(df * (n - 1) - (df^2 *
epsilon_gg)), 1)
```

Code „Correction of p-Values“

- Brief:
  - **Coose recomendet adjustment and add to ANOVA-table:**

```
anova_table = rma(rma_data)[[1]]

 if (p_w < 0.05) {
         if (p_factor_lb < 0.05) {
            anova_table[, "Recommended Lower-Bound corrected p-Value (Greenhouse & Geisser, 1959))"] = c(NA, p_factor_lb, NA, NA, NA, NA)
         } else {
            if (epsilon_gg < 0.75) {
              anova_table[, "Recommended Box corrected p-Value (Geisser & Greenhouse, 1958)"] = c(NA, p_factor_gg, NA, NA, NA, NA)
            } else {
              anova_table[, "Recommended Huynh-Feldt corrected p-Value (Huynh & Feldt, 1976)"] = c(NA, p_factor_hf, NA, NA, NA, NA)
            }
         }
      }
```

Code „Orthogonal Polynomial Contrasts“

- Medium:
  - **Contrast analyses:**

```
# Applying formula for linear contrasts
   weighted_dependend_variables = dependent_variable[rep(1:n, each = maxpoly), ] *
(contrast_weights)[rep(1:maxpoly, n), ]
   linear_subject_contrasts = matrix(rowSums(weighted_dependend_variables), byrow = TRUE, ncol = maxpoly)

# Computing contrast estimators for each orthogonal polynomial contrast as well as standard errors for these
estimators
   contrast_estimator = colMeans(linear_subject_contrasts)
   contrast_se = sqrt(apply(linear_subject_contrasts, 2, var))/sqrt(n)

# Computing t-values for each contrast
   contrast_t_values = contrast_estimator/contrast_se
   # contrast_F_values = contrast_t_values^2

# Computing the corresponding p-values
   contrast_p_values = 1 - pt(abs(contrast_t_values), n - 1)

# Computing sums of squares for each contrast
   contrast_ss = n * contrast_estimator^2/rowSums(contrast_weights^2)
```

Code „Displaying Trends“

- Detailed:
  - Niko
  - ggplot

Code „Package“

- Medium:
  - Joachim und Niko

Hinweise:

- Der Code ist natürlich hier noch nicht die Endfassung… wir sollten den entsprechenden finalen Code verwenden
- In der Subfunktion für die normale anova im Quantlet „SSE Reduct“ sollte auch noch das „id“ Argument eingebaut werden…
- Ich weiß, dass ist ganz schön viel Code, aber ich denke das ist gar nicht so schlecht, Sachen weglassen und überspringen kann man immer noch