Mehr Informationen: https://datenanalyse.ifkw.lmu.de/ oder auf: https://cran.r-project.org/web/packages/tidycomm/tidycomm.pdf

R Studio :: **tidycomm**

tidycomm aktivieren:

```
install.packages("tidycomm")
```

→ Sobald Sie das Package einmal installiert haben, müssen Sie in Zukunft nur noch zu Beginn der Verwendung folgenden Befehl ausführen, um das Package zu aktivieren:

```
library(tidycomm)
```

Achtung: Beim Installieren werden Anführungszeichen genutzt, beim Aktivieren nicht mehr!

Infos zu Funktionen in tidycomm:

Mit `?Befehl` erhalten Sie mehr Informationen zu den einzelnen Befehlen und Erklärungen zum Output der Befehle.

Datensatz in tidycomm: WoJ

Den Datensatz von Tidycomm können Sie mit folgendem Befehl aufrufen und nutzen:

```
WoJ <- WoJ
```

Für Informationen zum Datensatz schreiben Sie `?WoJ` (Es öffnet sich ein Fenster mit Erklärungen zu den einzelnen Variablen und weiteren Eigenschaften des Datensatzes.)

LMU-Ästhetik für ggplot2:

Tidycomm enthält die Funktion `design_lmu()`
Wenn Sie diese Funktion ausführen (Sie können um die Funktion auch `View()` schreiben, um einen überschaubareren Output zu bekommen), dann erhalten Sie Farbcodes und weitere Vorgaben, mit denen Sie Ihre Graphiken in ggplot2 an die Ästhetik der LMU anpassen können.

Skalen ändern

1. **Invertieren von Variablen** (hier: 5 wird 1 und 1 wird 5):

```
data <- data %>% reverse_scale(variable, lower_end = 1, upper_end = 5)
```

Wichtig: Die umkodierte Variable heißt jetzt variable_**rev**, außer der `overwrite`-Parameter wird genutzt.

2. **Stetige Variable zentrieren:**

```
data <- data %>% center_scale(variable)
```

Wichtig: Die umkodierte Variable heißt jetzt variable_**centered**, außer der `overwrite`-Parameter wird genutzt.

3. **Z-standardisierte Variable:**

```
data <- data %>% z_scale(variable)
```

Wichtig: Die umkodierte Variable heißt jetzt variable_**z**, außer der `overwrite`-Parameter wird genutzt.

4. **Range/Spannweite der Skala erweitern** (hier: 1 bis 5 wird 1 bis 10; die Abstände bleiben gleich groß):

```
data <- data %>% minmax_scale(variable, change_to_min = 1, change_to_max = 10)
```

Wichtig: Die umkodierte Variable heißt jetzt variable_**1to10**, außer der `overwrite`-Parameter wird genutzt.

Variablen umkodieren & Gruppen erstellen: (`select()` aus dplyr)

1. **Nominal- oder ordinalskalierte Variablen umkodieren:**
   a. String-Werte:

```
data <- data %>% select(variable) %>% recode_cat_scale(variable,
  assign = c("alter Wert1" = "neuer Wert1", "alter Wert2" = "neuer Wert2"),
  overwrite = TRUE)
```

   b. Integers (Zahlenwerte):

```
data <- data %>% recode_cat_scale(variable, assign = c('Zahl1'
  = "Bezeichnung1", 'Zahl2' = "Bezeichnung2"), overwrite = TRUE)
```

→ Wenn "alle anderen Werte" als "other" betitelt werden sollen, hängen Sie am Ende in der Klammer noch `other = "other"` an.

2. **Dummy-Variable erstellen:**

```
data <- data %>% select(variable) %>% dummify_scale(variable)
```

3. **Intervall- bzw. verhältnisskalierte Skala in eine nominal- oder ordinalskalierte Skala umkodieren und Gruppen erstellen:**

```
data <- data %>% categorize_scale(variable, lower_end = Minimum der
alten Skala, upper_end = Maximum der alten Skala, breaks = c(Werte, nach denen eine
neue Gruppe beginnt), labels = c("Name Gruppe1", "Name Gruppe2"))
```

NAs definieren:

```
data %>% setna_scale(variable, value = Wert)
```

Wichtig: Die Variable, in der der NA-Wert definiert wurde, heißt nun Variablenname_**na**, außer der `overwrite`-Parameter wird genutzt.

Häufigkeiten:

```
data %>% tab_frequencies(variable)
```

Perzentile angeben lassen (Perzentiltabelle):

```
data %>% tab_percentiles(variable)
```

Streuungs- und Lagemaße: (`mutate()` aus dplyr)

1. **Für metrische Variablen:**

```
data %>% describe(variable)
```

→ **Wichtig:** Den Interquartilsabstand berechnet R hier nicht. Durch den folgenden Befehl können Sie diesen aber mitberechnen lassen:

```
data %>% describe(variable) %>% mutate(IQR = Q75 - Q25)
```

Dasselbe mit der Varianz: Fügen Sie dafür einfach folgenden Befehl hinzu: `%>% mutate(Variance = SD^2)`

2. **Für nominal– und ordinalskalierte Variablen:**

```
data %>% describe_cat(variable)
```

Variablen als Grafiken veranschaulichen:

1. **Für nominal- oder ordinalskalierte Variablen:**
   `describe_cat() %>% visualize()`: Balkendiagramm, das absolute Häufigkeiten abbildet
   `tab_frequencies() %>% visualize()`: Balkendiagramm, das relative Häufigkeiten abbildet

2. **Für metrische Variablen:**
   `describe() %>% visualize()`: Boxplot
   `tab_frequencies() %>% visualize()`: Histogramm

Chi-Quadrat-Test: χ 2 und Cramer's V

1. Vorraussetzungen prüfen:

```
data %>% tab_frequencies(variable1, variable2)
```

2. Ausführung:

```
data %>% crosstab(unabhängige_variable, abhängige_variable, chi_square = TRUE)
```

→ **Tipp:** Wenn Sie auch die Prozentzahlen angezeigt bekommen möchten, hängen Sie am Ende innerhalb der Klammer noch `percentages = TRUE` an.

→ **Tipp:** Wenn Sie auch die Zeilensumme ausgegeben haben möchten, hängen Sie am Ende innerhalb der Klammer noch `add_total = TRUE` an.

Reliabilitätsanalyse:

1. Reliabilitätsanalyse bei Items (Cronbachs Alpha):

1.1 Index erstellen und abspeichern:

a. Mittelwertindex: `data <- data %>% add_index(index_einstellung, variable1, variable2, variable3)`
→ Der Index wird hier „index_einstellung" genannt und wurde aus Variable1 bis 3 erstellt.

b. Summenindex: `data <- data %>% add_index(index_einstellung, variable1, variable2, variable3, type = "sum")`

1.2 Interne Konsistenz/Reliabilität berechnen (Output: Cronbachs Alpha, Index-Beschreibung, deskriptive Statistiken über den Index): `data %>% get_reliability(index_einstellung)`
→ Wenn Sie nur `data %>% get_reliability()` schreiben, dann wird Ihnen die Reliabilität für alle Indizes aus Ihrem Datensatz berechnet, die Sie mit `add_index()` berechnet haben.

2. Interrater-Reliabilität/Intercoderreliabilität:

`data %>% test_icr(variable_mit_einheitenbezeichnung, variable_mit_codierendenkennung)`
Möchten Sie für alle Variablen die Intercoderreliabilität berechnen, schreiben Sie einfach `data %>% test_icr()`
→ Output: Übereinstimmung in Prozent, Reliabilitätskoeffizient nach Holsti-Schätzung und Krippendorffs Alpha. Für zusätzliche Berechnung(en) hängen Sie noch folgende(n) Parameter am Ende in der Klammer an:

```
fleiss_kappa = TRUE        # Fleiss Kappa
lotus = TRUE               # Lotus
cohens_kappa = TRUE        # Cohen's Kappa
brennan_prediger = TRUE    # Brennen & Prediger's Kappa
s_lotus = TRUE             # S-Lotus
```

Regression:

1. Einfache lineare Regression:

Voraussetzungen prüfen und ausführen: `data %>% regress(abhängige_variable, unabhängige_variable, check_independenterrors = TRUE, check_multicollinearity = TRUE, check_homoscedasticity = TRUE)`

2. Multiple lineare Regression:

Voraussetzungen prüfen und ausführen: `data %>% regress(abhängige_variable, unabhängige_variable1, unabhängige_variable2, check_independenterrors = TRUE, check_multicollinearity = TRUE, check_homoscedasticity = TRUE)`

3. Visualisierung der Regressionen:

`data %>% regress(abhängige_variable, unabhängige_variable(n)) %>% visualize()`
Es wird die automatische Grafik von Tidycomm erstellt.
oder:

```
%>% visualize(which = "correlogram")# Korrelogramm
              which = "resfit"      # Residuals vs Leverage Diagramm
              which = "pp"          # Normal probability-probability plot
              which = "scaloc"      # skaliertes Standortdiagramm
              which = "reslev"      # Residuals vs Leverage Diagramm
```

Korrelation:

1. Korrelation (Pearson, Kendall, Spearman Rho):

```
data %>% correlate(variable1, variable2, method = "pearson" ) # Pearson
                                         method = "kendall"   # Kendall
                                         method = "spearman"  # Spearman Rho
```

Hinweis: Sie können auch mehr als zwei Variablen miteinander korrelieren, wobei alle in zweier-Paaren korreliert werden. Fügen Sie dafür einfach noch weitere Variablen in der Klammer hinzu.
Hinweis: Möchten Sie die anderen Variablen mit einer speziellen Variable korrelieren, dann nutzen Sie folgenden Befehl: `data %>% correlate(variable1, variable2, with = variable_mit_der_die_anderen_korreliert_werden_sollen)`
Hinweis: Mit `data %>% correlate()` werden alle Variablen aus dem Datensatz miteinander korreliert, die geeignet sind.
Hinweis: Korrelationsmatrix erstellen: `data %>% correlate(…) %>% to_correlation_matrix()`

2. Partielle Korrelation:

`data %>% correlate(variable1, variable2, partial = drittvariable)`

3. Visualisieren:

Hängen Sie hinter jeden der vorherigen Korrelationsbefehle `%>% visualize()` an. Es wird die automatische Grafik von Tidycomm erstellt.

T-Test: (`group_by()` aus dplyr)

1. Einfacher T-Test:

a. Voraussetzungen prüfen: `data %>% describe(abhängige_variable)`
b. Ausführung: `data %>% t_test(variable, mu = Wert)` Der Wert aus der Grundgesamtheit ist `mu`.
c. Ausgabe: **Wichtig:** Tidycomm führt immer eine zweiseitige Testung durch! Bei einseitiger Testung müssen Sie p manuell halbieren **oder** Sie speichern den p-Wert in einem R-Objekt (p) und nutzen den Befehl: `p / 2`

2. Unabhängiger T-Test:

a. Voraussetzungen prüfen: `data %>% dplyr::group_by(unabhängige_variable) %>% describe(abhängige_variable)` (**Hinweis:** Der Levene-Test wird erst mit dem T-Test durchgeführt. Fällt der Test signifikant aus, dann wird automatisch eine Korrektur mit dem Welch-Test durchgeführt.)
b. Ausführung: `data %>% t_test(unabhängige_variable, abhängige_variable)` (**Tipp:** Wenn Ihre UV mehr als 2-stufig ist, dann können Sie auch speziell mit dem Argument `levels =` spezielle Ausprägungen definieren, die Sie testen möchten: `data %>% t_test(unabhängige_variable, abhängige_variable, levels = c("Ausprägung1 der UV", "Ausprägung2 der UV"))`
c. Ausgabe: **Wichtig:** Tidycomm führt immer eine zweiseitige Testung durch! Bei einseitiger Testung müssen Sie p halbieren. Ist der Levene-Test signifikant, sehen Sie die Ergebnisse des Welch-Tests.

3. Abhängiger T-Test:

a. Voraussetzungen prüfen: `data %>% dplyr::group_by(unabhängige_variable) %>% describe(abhängige_variable)`
b. Ausführung: `data %>% t_test(unabhängige_variable, abhängige_variable, paired = TRUE, case_var = variable)` Bei `case_var =` geben Sie die Variable an, die R zeigt, dass Person a in Datensatz1 Person b aus Datensatz2 ist. Das kann bspw. eine case_id sein.
c. Ausgabe: **Wichtig:** Auch hier ist der p-Wert immer zweiseitig und muss bei einseitiger Messung halbiert werden.

4. Visualisieren:

Hängen Sie am Ende des Befehls noch `%>% visualize()` an. Es wird die automatische Grafik von Tidycomm erstellt.

Einfaktorielle Varianzanalyse (ANOVA): (`group_by()` aus dplyr)

1. Voraussetzungen prüfen: `data %>% dplyr::group_by(unabhängige_variable) %>% describe(abhängige_variable)`
2. Ausführung: `data %>% unianova(unabhängige_variable, abhängige_variable, post_hoc = TRUE)`

→ **Tipp:** Wenn Sie die deskriptive Statistik erneut ausgegeben bekommen möchten, dann nutzen Sie am Ende in der Klammer noch **`descriptives = TRUE`**.

→ **Wichtig:** Auch hier ist der p-Wert immer zweiseitig und muss bei einseitiger Messung halbiert werden.

3. Visualisierung: Hängen Sie hinter der Klammer noch den Befehl **`%>% visualize()`** an. Es wird die automatische Grafik von Tidycomm erstellt.