Aufgabe 1 in R

Aufgabe 1 – Aufgabenstellung:

a. Werten Sie alle in den Daten vorkommenden Features (Feature 1 bis Feature 6) deskriptiv aus. Bitte beachten Sie, dass sich mit den Städtenamen (Tabelle Adresse) und den Datumsangaben 6 verschiedene Gruppen bilden. Berechnen Sie den Mittelwert, den Median, die Standardabweichung, den minimalen und maximalen Wert sowie die 25% und 75% Quantile. Erstellen Sie für jedes der Features je eine Grafik mit 6 Histogrammen für die Gruppen (horizontal die 3 Städte, vertikal das Datum).

b. Gibt es fehlende Werte in dem Datensatz? Wenn ja, wie viele? Können Sie diese Daten imputieren? Falls Sie die Daten nicht imputieren können, dann entfernen Sie unvollständige Datensätze aus dem Analysebestand.

c. Visualisieren Sie die Korrelationsmatrizen.

d. Dokumentieren Sie die notwendigen Schritte zur Datenaufbereitung.

In dieser Dokumentation wird der lauffähige R-Code für die Lösung von Aufgabe 1 & 2 erklärt und an den jeweiligen Stellen werden die Analyseergebnisse besprochen. Für den gesamten Code zur Ausführung siehe R-File, das ebenfalls auf Moodle hochgeladen wurde.

<u>Im Folgenden wir der R-Code erklärt und dokumentiert:</u>

```
# Sicherstellen, dass keine alten Objekte im File sind:
rm(list=ls())

# Libraries installieren
install.packages("xlsx")
install.packages("dplyr")
install.packages("tidyr")
install.packages("corrplot")
install.packages("car")
install.packages("MASS")

# Libraries laden
library(xlsx)
library(dplyr)
library(tidyr)
library(corrplot)
library(car)
library(lattice)
library(MASS)
```

Zunächst wird sichergestellt, dass keine alten Objekte im File sind, und die notwendigen Libraries werden installiert bzw. geladen.

```
#################################################################################################
### DATA.FRAME mit allen relevanten Informationen anlegen ###
#################################################################################################

# Work Directory setzen
setwd("C:/Users/cordu/Desktop/UE4")

# Daten aus den CSV-Files einlesen
adressen <- read.csv("Adresse.csv", sep=";", dec=",", header=TRUE) # spalten: PLZ, Stadt
benutzer <- read.csv("Benutzer.csv", sep=";", dec=",", header=TRUE) # spalten: UID, PLZ
features <- read.csv("Features.csv", sep=";", dec=",", header=TRUE, stringsAsFactors = FALSE)
                                                    # spalten: UID, Datum, variable, value

# Data.Frame anlegen und Daten zusammenfuehren

 # Join von benutzer und adressen auf "PLZ"
  joined_benutzer_adressen <- benutzer %>% inner_join(adressen, by=c("PLZ"))

 # Join des obigen objekts mit features auf "UID"
  joined_all <- joined_benutzer_adressen %>% inner_join(features, by=c("UID"))

 # joined_all ist das zusammengelegte data.frame, das alle informationen enthaelt
  is.data.frame(joined_all)
  joined_all
```

Die zur Bearbeitung notwendigen Daten werden aus den CSV-Files mittels read.csv-Funktion eingelesen. Da die gewonnenen Daten in Normalformen in drei Files vorliegen, müssen die Daten erst passend zusammengeführt werden mittels Joins. Die Dataframes „benutzer“ und „adressen“ werden über die Spalte „PLZ“ zusammengeführt. Das neu entstandene Dataframe „joined_benutzer_adressen“ wird über die Spalte „UID“ zusammengeführt mit dem Dataframe „features“.

```
#################################################################################################
### AUFGABE 1 ###
#################################################################################################

# 1.a. Werten Sie alle in den Daten vorkommenden Features (Feature 1 bis Feature 6) deskriptiv aus.
#      Bitte beachten Sie, dass sich mit den Städtenamen (Tabelle Adresse) und den Datumsangaben 6 verschiedene
#      Gruppen bilden.
#      Berechnen Sie den Mittelwert, den Median, die Standardabweichung, den minimalen und maximalen Wert sowie
#      die 25% und 75% Quantile.
#      Erstellen Sie für jedes der Features je eine Grafik mit 6 Histogrammen für die Gruppen (horizontal die 3
#      Städte, vertikal das Datum).
# 1.b. Gibt es fehlende Werte in dem Datensatz? Wenn ja, wie viele? Können Sie diese Daten imputieren?
#      Falls Sie die Daten nicht imputieren können, dann entfernen Sie unvollständige Datensätze aus dem
#      Analysebestand.
# 1.c. Visualisieren Sie die Korrelationsmatrizen.


  # Daten in Feature 1 bis 6 trennen und anzahl der beobachtungen ueberpruefen

    daten_feature_1 <- joined_all %>% filter(variable == "Feature 1")
    nrow(daten_feature_1) # 299 beobachtungen

    daten_feature_2 <- joined_all %>% filter(variable == "Feature 2")
    nrow(daten_feature_2) # 300 beobachtungen

    daten_feature_3 <- joined_all %>% filter(variable == "Feature 3")
    nrow(daten_feature_3) # 299 beobachtungen

    daten_feature_4 <- joined_all %>% filter(variable == "Feature 4")
    nrow(daten_feature_4) # 300 beobachtungen

    daten_feature_5 <- joined_all %>% filter(variable == "Feature 5")
    nrow(daten_feature_5) # 300 beobachtungen

    daten_feature_6 <- joined_all %>% filter(variable == "Feature 6")
    nrow(daten_feature_6) # 300 beobachtungen

    anzahl_UID <- nrow(benutzer) # 150 benutzer
```

Die Daten werden nach Features (1 bis 6) getrennt, und die Anzahl der Beobachtungen pro Feature wird überprüft. Für jeden Benutzer (bzw. UID) wurden scheinbar 2 Beobachtungen

pro Feature gemacht – also insgesamt pro Feature 300 Beobachtungen. Es fehlt daher bei Feature 3 und bei Feature jeweils 1 Beobachtung von einer UID. Diese betroffenen UIDs müssen ermittelt werden, oder vollständig entfernt werden aus den Daten, damit keine Probleme bei der weiteren Analyse entstehen.

```
# fuer jeden benutzer (UID) wurden scheinbar 2 beobachtungen pro feature gemacht
# es fehlt aber bei feature 3 eine beobachtung und bei feature 1 eine beobachtung,
# damit es tatsaechlich 2 beobachtungen pro feature waeren pro benutzer (UID)
# die werte fuer den betroffenen benutzer (UID) kann man bis auf den value herausfinden.
# fuer den value koennte man eine schaetzung machen. es gibt hier aber keine vorgabe,
# wie man die schaetzung durchfuehren sollte.

# finde UID, deren beobachtungen unvollstaendig sind
  # bezueglich feature 1
  unvollstaendige_UID <- "a"

  for (i in 1:anzahl_UID) {
    if(sum(daten_feature_1$UID==benutzer$UID[i]) < 2){
      unvollstaendige_UID <- benutzer$UID[i]
      break
    }
  }

  sum(daten_feature_1$UID==30) # tatsaechlich gibt es nur 1 beobachtung der UID 30 in feature 1

  # bezueglich feature 3
  unvollstaendige_UID_f3 <- "a"

  for (i in 1:anzahl_UID) {
    if(sum(daten_feature_3$UID==benutzer$UID[i]) < 2){
      unvollstaendige_UID_f3 <- benutzer$UID[i]
      break
    }
  }

  sum(daten_feature_3$UID==130) # tatsaechlich gibt es nur 1 beobachtung der UID 130 in feature 3
```

Die betroffenen UIDs müssen daher geeignet ermittelt werden, wie es im obigen Code-Stück dargestellt wird. Es zeigt sich, dass in den Daten zu Feature 1 eine Beobachtung von UID 30 fehlt, und im Feature 3 eine Beobachtung von UID 130, wie auch untenstehend ersichtlich ist:

```
> unvollstaendige_UID
[1] 30
> sum(daten_feature_1$UID==30)
[1] 1
> daten_feature_1$UID
  [1]  10  10  99  99  25  25  35  35 128 128   7   7  66  66 105 105  42  42 130 130  96  96 122 122   0   0  91  91  54  54   3
 [32]   3 125 125  43  43 106 106 116 116 113 113  31  31  32  32 101 101   4   4 121 121  45  45   9   9  53  53 124 124   1   1
 [63]  20  20  60  60  68  68 139 139  84  84 143 143 142 142 126 126 104 104  51  51  46  46 129 129 135 135  71  71 123 123 131
 [94] 131  98  98 141 141  61  61 100 100  33  33  73  73   5   5  58  58  11  11  69  69 144 144  49  49  72  72  82  82 134 134
[125]   8   8 115 115  81  81 140 140 148 148  65  65  70  70  78  78 110 110 133 133  38  38  79  79 111 111  75  75  55  55 107
[156] 107  90  90  52  52  21  21   6   6  63  63  87  87  93  93  80  80  16  16 127 127  27  27  39  39 112 112 102 102 149 149
[187]  29  29  77  77 119 119  95  95  30 103 103 145 145 114 114  14  14  85  85  48  48 147 147 118 118 117 117  12  12  26  26
[218]  86  86  37  37  28  28  92  92  44  44   2   2  59  59  13  13  19  19  97  97  24  24  64  64  17  17  18  18 137 137  50
[249]  50  74  74  62  62 108 108 146 146  94  94  56  56 120 120  88  88 132 132  22  22  89  89  57  57  15  15  34  34  83  83
[280] 136 136 138 138  23  23 109 109  41  41  76  76  67  67  36  36  40  40  47  47
```

```
> daten_feature_3$UID
  [1]  10  10  99  99  25  25  35  35 128 128   7   7  66  66 105 105  42  42 130  96  96 122 122   0   0  91  91  54  54   3   3
 [32] 125 125  43  43 106 106 116 116 113 113  31  31  32  32 101 101   4   4 121 121  45  45   9   9  53  53 124 124   1   1  20
 [63]  20  60  60  68  68 139 139  84  84 143 143 142 142 126 126 104 104  51  51  46  46 129 129 135 135  71  71 123 123 131 131
 [94]  98  98 141 141  61  61 100 100  33  33  73  73   5   5  58  58  11  11  69  69 144 144  49  49  72  72  82  82 134 134   8
[125]   8 115 115  81  81 140 140 148 148  65  65  70  70  78  78 110 110 133 133  38  38  79  79 111 111  75  75  55  55 107 107
[156]  90  90  52  52  21  21   6   6  63  63  87  87  93  93  80  80  16  16 127 127  27  27  39  39 112 112 102 102 149 149  29
[187]  29  77  77 119 119  95  95  30  30 103 103 145 145 114 114  14  14  85  85  48  48 147 147 118 118 117 117  12  12  26  26
[218]  86  86  37  37  28  28  92  92  44  44   2   2  59  59  13  13  19  19  97  97  24  24  64  64  17  17  18  18 137 137  50
[249]  50  74  74  62  62 108 108 146 146  94  94  56  56 120 120  88  88 132 132  22  22  89  89  57  57  15  15  34  34  83  83
[280] 136 136 138 138  23  23 109 109  41  41  76  76  67  67  36  36  40  40  47  47
> sum(daten_feature_3$UID==130) # tatsaechlich gibt es nur 1 beobachtung der UID 130 in feature 1
[1] 1
```

```
# versuch der imputation (ANNAHME: mit wert des medians der daten)
  # fuer UID 30 in den feature 1 daten
    daten_feature_1_impute <- daten_feature_1
    info_zu_imputieren_f1 <- daten_feature_1_impute[daten_feature_1_impute$UID == unvollstaendige_UID, ]
    laenge_f1 <- nrow(daten_feature_1_impute)
    value_zu_imputieren <- median(as.numeric(as.character(daten_feature_1_impute$value)))
    info_zu_imputieren_f1$Datum <- "20180416T00:00Z"
    info_zu_imputieren_f1$value <- value_zu_imputieren
    daten_feature_1_impute <- rbind(daten_feature_1_impute, info_zu_imputieren_f1)
    daten_feature_1_impute[daten_feature_1_impute$UID == unvollstaendige_UID, ]
    rownames(daten_feature_1_impute)[300] <- "300"
    laenge_f1 <- nrow(daten_feature_1_impute)

  # fuer UID 130 in den feature 3 daten (ANNAHME: mit wert des medians der daten)
    daten_feature_3_impute <- daten_feature_3
    info_zu_imputieren_f3 <- daten_feature_3_impute[daten_feature_3_impute$UID == unvollstaendige_UID_f3, ]
    laenge_f3 <- nrow(daten_feature_3_impute)
    value_zu_imputieren_f3 <- median(as.numeric(as.character(daten_feature_3_impute$value)))
    info_zu_imputieren_f3$Datum <- "20160813T00:00Z"
    info_zu_imputieren_f3$value <- value_zu_imputieren_f3
    daten_feature_3_impute <- rbind(daten_feature_3_impute, info_zu_imputieren_f3)
    daten_feature_3_impute[daten_feature_3_impute$UID == unvollstaendige_UID_f3, ]
    rownames(daten_feature_3_impute)[300] <- "300"
    laenge_f3 <- nrow(daten_feature_3_impute)

    daten_feature_3_impute[290:300,]
```

Alle Daten außer dem “value“ der jeweiligen Beobachtungen können unter Zuhilfenahme der vorhandenen Daten ermittelt werden. Der „value“ muss geschätzt werden oder nachdem hier die Daten relativ zentriert verteilt sind, kann auch der Median innerhalt der jeweiligen Featuredaten als Wert genommen werden, wie im Code implementiert.

```
# zur sicherheit, dass keine datenverfaelschungen stattfinden durch den imputationsversuch,
# werden die unvollstaendigen UIDs aber aus den daten herausgenommen
# betroffene UIDs: 30 und 130

    liste_featuresdaten <- list(f1=daten_feature_1, f2=daten_feature_2, f3=daten_feature_3,
                                f4=daten_feature_4, f5=daten_feature_5, f6=daten_feature_6)
    str(liste_featuresdaten)

    # listenelemente von den unvollstaendigen UIDs reinigen fuer alle features
    for(i in 1:length(liste_featuresdaten)){

      daten_feature_i <- liste_featuresdaten[[i]]
      daten_feature_i_cleaned <- daten_feature_i[(daten_feature_i$UID != unvollstaendige_UID), ]
      daten_feature_i_cleaned <- daten_feature_i_cleaned[(daten_feature_i_cleaned$UID != unvollstaendige_UID_f3), ]
      liste_featuresdaten[[i]] <- daten_feature_i_cleaned
    }

    # pruefe, ob alle listenelemente nun gleiche anzahl an beobachtungen haben
    # (hier: 296 nach entfernung der 2 unvollstaendigen UIDs)
    for(i in 1:length(liste_featuresdaten)){
      print(nrow(liste_featuresdaten[[i]]))
    }
```

Da ich sicher gehen wollte, dass keine Datenverfälschung durch meine Schätzung bzw. Wertannahme ausgelöst werden, habe ich mich entschieden, den Imputationsversuch durchzuführen, um zu zeigen, dass dieser möglich ist, aber für die tatsächliche Analyse habe ich nun die betroffenen UIDs aus dem Datensatz herausgenommen. Es wird also rein auf Basis von vollständigen Beobachtungen analysiert.

```
------------------------------------------------------------------------------
GRUPPEN INNERHALB DER FEATURES BILDEN:
------------------------------------------------------------------------------
   # G1: Graz - 20180416T00:00Z
   # G2: Graz - 20160813T00:00Z
   # G3: Salzburg - 20180416T00:00Z
   # G4: Salzburg - 20160813T00:00Z
   # G5: Wien - 20180416T00:00Z
   # G6: Wien - 20160813T00:00Z

   # FUNCTION: gruppen fuer ein als parameter uebergebenes feature-dataframe ermitteln
   getGruppenDaten <- function(feature){

     # liste fuer gruppendaten des derzeitigen features anlegen
     featuregesamt <- list()

     # gruppen bilden
     featuregesamt$G1 <- liste_featuresdaten[[i]] %>% filter(Stadt=="Graz" & Datum=="20180416T00:00Z")
     featuregesamt$G2 <- liste_featuresdaten[[i]] %>% filter(Stadt=="Graz" & Datum=="20160813T00:00Z")
     featuregesamt$G3 <- liste_featuresdaten[[i]] %>% filter(Stadt=="Salzburg" & Datum=="20180416T00:00Z")
     featuregesamt$G4 <- liste_featuresdaten[[i]] %>% filter(Stadt=="Salzburg" & Datum=="20160813T00:00Z")
     featuregesamt$G5 <- liste_featuresdaten[[i]] %>% filter(Stadt=="Wien" & Datum=="20180416T00:00Z")
     featuregesamt$G6 <- liste_featuresdaten[[i]] %>% filter(Stadt=="Wien" & Datum=="20160813T00:00Z")

     # RETURN: featuregesamt (d.h. alle 6 gruppen fuer das derzeitige feature)
     featuregesamt

   }


   # DATEN FUER ALLE FEATURES ERMITTELN (GESAMTLISTE ALLER GRUPPEN FUER ALLE FEATURES)
   gruppendaten_gesamtliste <- list()

   for(i in 1:length(liste_featuresdaten)){

     gruppendaten_pro_feature <- getGruppenDaten(liste_featuresdaten[[i]])
     gruppendaten_gesamtliste[[i]] <- gruppendaten_pro_feature

   }

   names(gruppendaten_gesamtliste) <- c("Feature 1", "Feature 2", "Feature 3",
                                        "Feature 4", "Feature 5", "Feature 6")
```

Die Bildung von Gruppen erfolgt gemäß den Kombinationen von „stadt“ und „datum“, die ich in G1 bis G6 eingeteilt habe. Die Funktion „getGruppenDaten“ nimmt als Parameter „feature“, also die Daten eines Features, und bildet dafür die jeweils 6 Gruppen. Diese Funktion wird dann für alle Daten aus der gesamten Featuresdatenliste aufgerufen.

```
   # GESAMTLISTE aller gruppierten daten ueber alle features hinweg
       # gesamtliste enthaelt 6 listen (d.h. 1 fuer jedes feature)
       # jedes feature enthaelt 6 dataframes (d.h. 1 fuer jede gruppe)
       str(gruppendaten_gesamtliste)
```

Das Ergebnis der Einteilung in Gruppen ist das Objekt „gruppendaten_gesamtliste“, das eine Liste bestehend aus den 6 Features ist. Jedes Feature hat in sich 6 Dataframes, d.h. jeweils ein Dataframe pro Gruppe. Die folgende Abbildung zeigt die Struktur davon:

```
Console C:/Users/cordu/Desktop/UE4/

> str(gruppendaten_gesamtliste)
List of 6
 $ Feature 1:List of 6
  ..$ G1:'data.frame':  50 obs. of  6 variables:
  .. ..$ UID     : int [1:50] 99 66 96 91 54 53 60 68 84 51 ...
  .. ..$ PLZ     : int [1:50] 8010 8025 8022 8045 8012 8051 8045 8019 8054 8015 ...
  .. ..$ Stadt   : Factor w/ 3 levels "Graz","Salzburg",..: 1 1 1 1 1 1 1 1 1 1 ...
  .. ..$ Datum   : chr [1:50] "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z" ...
  .. ..$ variable: chr [1:50] "Feature 1" "Feature 1" "Feature 1" "Feature 1" ...
  .. ..$ value   : num [1:50] -3.05 3.53 13.4 -1.53 0.65 ...
  ..$ G2:'data.frame':  50 obs. of  6 variables:
  .. ..$ UID     : int [1:50] 99 66 96 91 54 53 60 68 84 51 ...
  .. ..$ PLZ     : int [1:50] 8010 8025 8022 8045 8012 8051 8045 8019 8054 8015 ...
  .. ..$ Stadt   : Factor w/ 3 levels "Graz","Salzburg",..: 1 1 1 1 1 1 1 1 1 1 ...
  .. ..$ Datum   : chr [1:50] "20160813T00:00Z" "20160813T00:00Z" "20160813T00:00Z" "20160813T00:00Z" ...
  .. ..$ variable: chr [1:50] "Feature 1" "Feature 1" "Feature 1" "Feature 1" ...
  .. ..$ value   : num [1:50] -3.68 -12.53 1.73 7.54 -2.74 ...
  ..$ G3:'data.frame':  49 obs. of  6 variables:
  .. ..$ UID     : int [1:49] 128 105 122 125 106 116 113 101 121 124 ...
  .. ..$ PLZ     : int [1:49] 5020 5061 5061 5023 5061 5026 5082 5082 5061 5101 ...
  .. ..$ Stadt   : Factor w/ 3 levels "Graz","Salzburg",..: 2 2 2 2 2 2 2 2 2 2 ...
  .. ..$ Datum   : chr [1:49] "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z" ...
  .. ..$ variable: chr [1:49] "Feature 1" "Feature 1" "Feature 1" "Feature 1" ...
  .. ..$ value   : num [1:49] 8.75 -8.87 -14.64 14.46 3.03 ...
  ..$ G4:'data.frame':  49 obs. of  6 variables:
  .. ..$ UID     : int [1:49] 128 105 122 125 106 116 113 101 121 124 ...
  .. ..$ PLZ     : int [1:49] 5020 5061 5061 5023 5061 5026 5082 5082 5061 5101 ...
  .. ..$ Stadt   : Factor w/ 3 levels "Graz","Salzburg",..: 2 2 2 2 2 2 2 2 2 2 ...
  .. ..$ Datum   : chr [1:49] "20160813T00:00Z" "20160813T00:00Z" "20160813T00:00Z" "20160813T00:00Z" ...
  .. ..$ variable: chr [1:49] "Feature 1" "Feature 1" "Feature 1" "Feature 1" ...
  .. ..$ value   : num [1:49] -0.154 3.453 -27.272 -17.368 5.96 ...
  ..$ G5:'data.frame':  49 obs. of  6 variables:
  .. ..$ UID     : int [1:49] 10 25 35 7 42 0 3 43 31 32 ...
  .. ..$ PLZ     : int [1:49] 1110 1140 1180 1190 1110 1020 1160 1080 1070 1160 ...
  .. ..$ Stadt   : Factor w/ 3 levels "Graz","Salzburg",..: 3 3 3 3 3 3 3 3 3 3 ...
  .. ..$ Datum   : chr [1:49] "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z" ...
  .. ..$ variable: chr [1:49] "Feature 1" "Feature 1" "Feature 1" "Feature 1" ...
  .. ..$ value   : num [1:49] 7.45 -5.3 2.55 2.28 9.26 ...
  ..$ G6:'data.frame':  49 obs. of  6 variables:
  .. ..$ UID     : int [1:49] 10 25 35 7 42 0 3 43 31 32 ...
  .. ..$ PLZ     : int [1:49] 1110 1140 1180 1190 1110 1020 1160 1080 1070 1160 ...
  .. ..$ Stadt   : Factor w/ 3 levels "Graz","Salzburg",..: 3 3 3 3 3 3 3 3 3 3 ...
  .. ..$ Datum   : chr [1:49] "20160813T00:00Z" "20160813T00:00Z" "20160813T00:00Z" "20160813T00:00Z" ...
  .. ..$ variable: chr [1:49] "Feature 1" "Feature 1" "Feature 1" "Feature 1" ...
  .. ..$ value   : num [1:49] -19.68 -6.19 -17.66 -2.16 1.16 ...
> str(gruppendaten_gesamtliste)
List of 6
 $ Feature 1:List of 6
  ..$ G1:'data.frame':  50 obs. of  6 variables:
  .. ..$ UID     : int [1:50] 99 66 96 91 54 53 60 68 84 51 ...
  .. ..$ PLZ     : int [1:50] 8010 8025 8022 8045 8012 8051 8045 8019 8054 8015 ...
  .. ..$ Stadt   : Factor w/ 3 levels "Graz","Salzburg",..: 1 1 1 1 1 1 1 1 1 1 ...
  .. ..$ Datum   : chr [1:50] "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z"
  .. ..$ variable: chr [1:50] "Feature 1" "Feature 1" "Feature 1" "Feature 1" ...
  .. ..$ value   : num [1:50] -3.05 3.53 13.4 -1.53 0.65 ...
  ..$ G2:'data.frame':  50 obs. of  6 variables:
  .. ..$ UID     : int [1:50] 99 66 96 91 54 53 60 68 84 51 ...
  .. ..$ PLZ     : int [1:50] 8010 8025 8022 8045 8012 8051 8045 8019 8054 8015 ...
  .. ..$ Stadt   : Factor w/ 3 levels "Graz","Salzburg",..: 1 1 1 1 1 1 1 1 1 1 ...
  .. ..$ Datum   : chr [1:50] "20160813T00:00Z" "20160813T00:00Z" "20160813T00:00Z" "20160813T00:00Z"
  .. ..$ variable: chr [1:50] "Feature 1" "Feature 1" "Feature 1" "Feature 1" ...
  .. ..$ value   : num [1:50] -3.68 -12.53 1.73 7.54 -2.74 ...
  ..$ G3:'data.frame':  49 obs. of  6 variables:
  .. ..$ UID     : int [1:49] 128 105 122 125 106 116 113 101 121 124 ...
  .. ..$ PLZ     : int [1:49] 5020 5061 5061 5023 5061 5026 5082 5082 5061 5101 ...
  .. ..$ Stadt   : Factor w/ 3 levels "Graz","Salzburg",..: 2 2 2 2 2 2 2 2 2 2 ...
  .. ..$ Datum   : chr [1:49] "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z" "20180416T00:00Z"
  .. ..$ variable: chr [1:49] "Feature 1" "Feature 1" "Feature 1" "Feature 1" ...
  .. ..$ value   : num [1:49] 8.75 -8.87 -14.64 14.46 3.03 ...
  ..$ G4:'data.frame':  49 obs. of  6 variables:
```

```
## ------------------------------------------------------------------------
## DESKRIPTIVE AUSWERTUNG
## ------------------------------------------------------------------------

# gesamtliste fuer ergebnisse der deskriptiven auswertung
deskriptiveStatistik_gesamtliste <- list()

# FUNCTION: berechne deskriptive statistik fuer uebergebenes feature,
#           das 6 eingeteilten gruppen enthaelt
  berechneDeskriptiveStatistik <- function(feature){

    # dataframe fuer auswertungsergebnisse anlegen
    df_deskr <- data.frame(mean=numeric(),
                           median=numeric(),
                           sd=numeric(),
                           min=numeric(),
                           max=numeric(),
                           quantile25=numeric(),
                           quantile75=numeric(),
                           stringsAsFactors=FALSE)


    # feature ist eine liste der 6 gruppen
    # d.h. fuer jede gruppe ein dataframe mit auswertung berechnen
    for(i in 1:length(feature)){

      # deskriptive kennzahlen berechnen
      mean <- mean(feature[[i]]$value) # i-te gruppe
      median <- median(feature[[i]]$value)
      sd <- sd(feature[[i]]$value)
      min <- min(feature[[i]]$value)
      max <- max(feature[[i]]$value)
      quantiles <- quantile(feature[[i]]$value, probs=c(0.25,0.75))

      vektor_gruppe <- c(mean, median, sd, min, max, quantiles[1], quantiles[2])

      # auswertung auf df_deskr zuweisen
      df_deskr <- rbind(df_deskr, vektor_gruppe)

    }
```

```
  # rownames (i.e. gruppennamen) setzen fuer df_deskr
  rownames(df_deskr) <- c("G1", "G2", "G3", "G4", "G5", "G6")
  # colnames (i.e. deskriptive kennzahlen) setzen fuer df_deskr
  colnames(df_deskr) <- c("mean", "median", "sd", "min", "max", "quantile25", "quantile75")

  # RETURN: df_deskr (i.e. dataframe mit den berechneten deskriptiven
  #         statistiken aller gruppen fuer das feature)
  df_deskr

}
```

Für die deskriptive Auswertung des Datensatzes dient die Funktion „berechneDeskriptiveStatistik“, die ein feature als Parameter übergeben bekommt. Sie berechnet pro Feature für jede Untergruppe die gefragten deskriptiven Statistikkennzahlen (mean, median, sd, min, max, quantile25, quantile75). Der Rückgabewert der Funktion ist das Dataframe „df_deskr“, das alle berechneten deskriptiven Statistiken aller Gruppen für das übergebene Feature berechnet.

```
# CHECKS ONLY (nur Feature 1): (fazit: berechnung der deskriptiven statistiken funktioniert)
  berechneDeskriptiveStatistik(gruppendaten_gesamtliste$`Feature 1`)
  mean(gruppendaten_gesamtliste$`Feature 1`$G1$value)
  median(gruppendaten_gesamtliste$`Feature 1`$G1$value)
  sd(gruppendaten_gesamtliste$`Feature 1`$G1$value)
  min(gruppendaten_gesamtliste$`Feature 1`$G1$value)
  max(gruppendaten_gesamtliste$`Feature 1`$G1$value)
  quantile(gruppendaten_gesamtliste$`Feature 1`$G1$value, probs=c(0.25,0.75))


# berechnungsfunktion fuer gesamte featuresliste durchfuehren
for(i in 1:length(gruppendaten_gesamtliste)){

  auswertungen_proFeature_alleGruppen <- berechneDeskriptiveStatistik(gruppendaten_gesamtliste[[i]])
  deskriptiveStatistik_gesamtliste[[i]] <- auswertungen_proFeature_alleGruppen

}

# namen der listenelemente mit jeweiligem featurename setzen
names(deskriptiveStatistik_gesamtliste) <- c("Feature 1", "Feature 2", "Feature 3",
                                     "Feature 4", "Feature 5", "Feature 6")


# fertige gesamtliste deskriptiver daten mit allen 6 gruppen pro feature
deskriptiveStatistik_gesamtliste
```

Die Berechnungsfunktion wird für die gesamte Featuresliste (und innerhalb pro Gruppe) ausgeführt. Dies ergibt schließlich die Gesamtliste aller deskriptiven Statistiken. Das Ergebnis sieht z.B. für das Feature 1 in der gruppen_gesamtliste folgendermaßen aus, wobei die unteren Statements zur Überprüfung ausgeführt wurden, ob die Daten tatsächlich übereinstimmen:

```
> berechneDeskriptiveStatistik(gruppendaten_gesamtliste$`Feature 1`)
         mean     median        sd       min      max quantile25 quantile75
G1  1.2448239  0.6284464 11.333374 -31.42733 21.42372  -3.296106   8.361234
G2 -1.9188120 -2.2278105 11.231719 -23.54473 21.22372  -9.888475   7.433256
G3 -1.5787335 -0.6847240  7.851082 -18.38360 14.46498  -7.137652   4.198059
G4  1.1055665  0.9502451 11.355149 -27.27196 19.68820  -5.468898   8.075091
G5 -0.8114564 -1.2018174  9.649476 -22.57135 22.13937  -5.301680   4.316624
G6 -1.2158931 -2.1567834 11.526209 -21.74028 28.60337  -8.295328   5.570597
> mean(gruppendaten_gesamtliste$`Feature 1`$G1$value)
[1] 1.244824
> median(gruppendaten_gesamtliste$`Feature 1`$G1$value)
[1] 0.6284464
> sd(gruppendaten_gesamtliste$`Feature 1`$G1$value)
[1] 11.33337
> min(gruppendaten_gesamtliste$`Feature 1`$G1$value)
[1] -31.42733
> max(gruppendaten_gesamtliste$`Feature 1`$G1$value)
[1] 21.42372
> quantile(gruppendaten_gesamtliste$`Feature 1`$G1$value, probs=c(0.25,0.75))
      25%       75%
-3.296106  8.361234
```

Für den Gesamtdatenbestand ergeben sich die folgenden Auswertungen:

Anmerkung zur Gruppenbenennung:

 # G1: Graz - 20180416T00:00Z

 # G2: Graz - 20160813T00:00Z

 # G3: Salzburg - 20180416T00:00Z

 # G4: Salzburg - 20160813T00:00Z

 # G5: Wien - 20180416T00:00Z

 # G6: Wien - 20160813T00:00Z

```
> deskriptiveStatistik_gesamtliste
$`Feature 1`
         mean     median        sd       min      max quantile25 quantile75
G1  1.2448239  0.6284464 11.333374 -31.42733 21.42372  -3.296106   8.361234
G2 -1.9188120 -2.2278105 11.231719 -23.54473 21.22372  -9.888475   7.433256
G3 -1.5787335 -0.6847240  7.851082 -18.38360 14.46498  -7.137652   4.198059
G4  1.1055665  0.9502451 11.355149 -27.27196 19.68820  -5.468898   8.075091
G5 -0.8114564 -1.2018174  9.649476 -22.57135 22.13937  -5.301680   4.316624
G6 -1.2158931 -2.1567834 11.526209 -21.74028 28.60337  -8.295328   5.570597

$`Feature 2`
         mean    median        sd        min      max quantile25 quantile75
G1 10.949135 11.333860  9.629095 -13.728317 35.42982   4.766020   16.38522
G2 12.757536 11.805260 11.197581  -8.994363 37.54291   5.090695   20.83210
G3 12.006156 12.258393 10.481430  -9.207346 34.69565   5.138037   17.88380
G4  7.730169  5.280140  9.274533 -10.649488 26.59894   1.367355   16.25316
G5  8.373801  8.682763  9.401324 -11.785145 30.59620   2.670948   11.82414
G6  8.422873 11.230767 10.947666 -25.441120 26.96638   1.590260   16.48727

$`Feature 3`
        mean   median        sd       min      max quantile25 quantile75
G1 5.177158 3.982181 10.675237 -14.83484 26.62341 -2.2787413   11.30106
G2 6.288084 8.262350 10.860415 -17.08369 29.00882 -2.0012575   12.82033
G3 4.662623 6.168371 11.286489 -22.44738 29.24919 -3.3241440   11.37979
G4 5.736515 6.623312  9.675450 -23.48863 29.46679  1.1015020   12.17431
G5 5.532993 7.243086 11.294941 -20.97305 26.91424 -3.4971835   14.17812
G6 5.786694 6.689257  7.699265 -16.76932 20.47398 -0.7331535   11.77145

$`Feature 4`
         mean    median        sd        min      max quantile25 quantile75
G1 11.218291 11.841805  8.589490  -7.414985 28.73418  7.1683793   15.33430
G2  9.857618  8.112268  9.231200  -4.644773 30.12249  3.1659512   15.74287
G3  8.836160 10.054064 11.518739 -17.150088 28.31361 -0.9187728   18.02804
G4 11.928082 12.191738 10.044963 -12.756224 33.28209  7.5920271   17.74432
G5  7.735960  8.148849  8.961904 -10.764621 29.07850  2.3723137   13.61663
G6 12.901719 13.386031 10.709557 -11.830193 40.02885  6.0616144   20.72164


$`Feature 5`
        mean    median       sd         min      max quantile25 quantile75
G1 19.03669  9.162844 47.82694 -82.810066 168.2499  -3.185744   41.82328
G2 77.72497 61.614767 59.43002  -3.457802 247.7923  36.549782  102.65103
G3 19.14947 11.695338 42.28951 -63.172848 146.3586  -5.070312   41.00319
G4 21.49126 12.163864 52.79748 -72.724722 159.1690  -3.106450   49.50729
G5 43.03583 27.076587 47.06707 -13.095907 211.7459  13.334965   63.62941
G6 39.70653 31.884432 39.77655 -54.428478 173.8591  12.381450   55.43708

$`Feature 6`
        mean   median       sd         min      max quantile25 quantile75
G1 13.07395 17.67813 50.54306 -159.836139 112.4546 -6.9960606   34.49338
G2 64.98726 58.22410 48.59245   -7.225858 182.3067 21.6831597   97.58495
G3 20.34198 10.70829 48.41708 -101.775628 194.3956 -1.6370345   36.56699
G4 19.88929 12.81350 52.48758  -96.106057 204.1449 -0.3472167   50.57108
G5 48.64403 41.76949 42.61175   -6.605393 166.9784 15.8801230   67.94203
G6 48.32187 34.23521 48.85491  -40.792507 215.9618 15.1077664   73.61527
```

```
## ------------------------------------------------------------------------------------
## VISUALISIERUNG (HISTOGRAMME FUER DIE 6 GRUPPEN)
## ------------------------------------------------------------------------------------

# FUNCTION zum erstellen der 6 subplots eines features (fixe achsen)
# parameter: feature (i.e. daten pro feature aus gruppendaten_gesamtliste)
erstelleGruppenplots <- function(feature, featurenumber){

  # plot-anzeige in 3x2 subplots teilen
  par(mfrow=c(2,3), oma=c(0,0,3,0))

  # histogramme fuer die jeweiligen gruppen des derzeitigen feature plotten
  # anordnung: horizontal staedte und vertikal datumswerte
  hist(feature$G1$value, breaks=seq(-160,250,l=15), col="darkseagreen1",
       main="Graz - 16.04.2018", xlab="value", ylab="frequency", ylim=c(0,30), xlim=c(-160,250))

  hist(feature$G3$value, breaks=seq(-160,250,l=15), col="rosybrown2",
       main="Salzburg - 16.04.2018", xlab="value", ylab="frequency",ylim=c(0,30), xlim=c(-160,250))

  hist(feature$G5$value, breaks=seq(-160,250,l=15), col="lightblue3",
       main="Wien - 16.04.2018", xlab="value", ylab="frequency",ylim=c(0,30), xlim=c(-160,250))

  hist(feature$G2$value, breaks=seq(-160,250,l=15), col="navajowhite",
       main="Graz - 13.08.2016", xlab="value", ylab="frequency",ylim=c(0,30), xlim=c(-160,250))

  hist(feature$G4$value, breaks=seq(-160,250,l=15), col="lightsalmon",
       main="Salzburg - 13.08.2016", xlab="value", ylab="frequency",ylim=c(0,30), xlim=c(-160,250))

  hist(feature$G6$value, breaks=seq(-160,250,l=15), col="khaki",
       main="Wien - 13.08.2016", xlab="value", ylab="frequency",ylim=c(0,30), xlim=c(-160,250))

  title(paste("Feature",featurenumber), outer=TRUE, cex.main=1.2)

}
```

Die Visualisierung der Histogramme für die 6 Gruppen pro Feature erfolgt mit der Funktion „erstelleGruppenplots“, die die Parameter „feature“ und „featurenumber“ erfordert. Für jede Gruppe innerhalb des Features wird ein Histogramm in passender Farbe angelegt und der gesamte Plot ist in 3x2 (d.h. Städte x Datumswerte) Subplots eingeteilt. Diese Funktion stellt die Daten basierend auf fixen Achsenwerten dar, wodurch die Daten untereinander leichter vergleichbar sind.

```
# plotfunktion "erstelleGruppenplots" (mit fixen achsenlängen) fuer gesamte featuresliste
for(i in 1:length(gruppendaten_gesamtliste)){

  x11()
  erstelleGruppenplots(gruppendaten_gesamtliste[[i]], i)

}
```

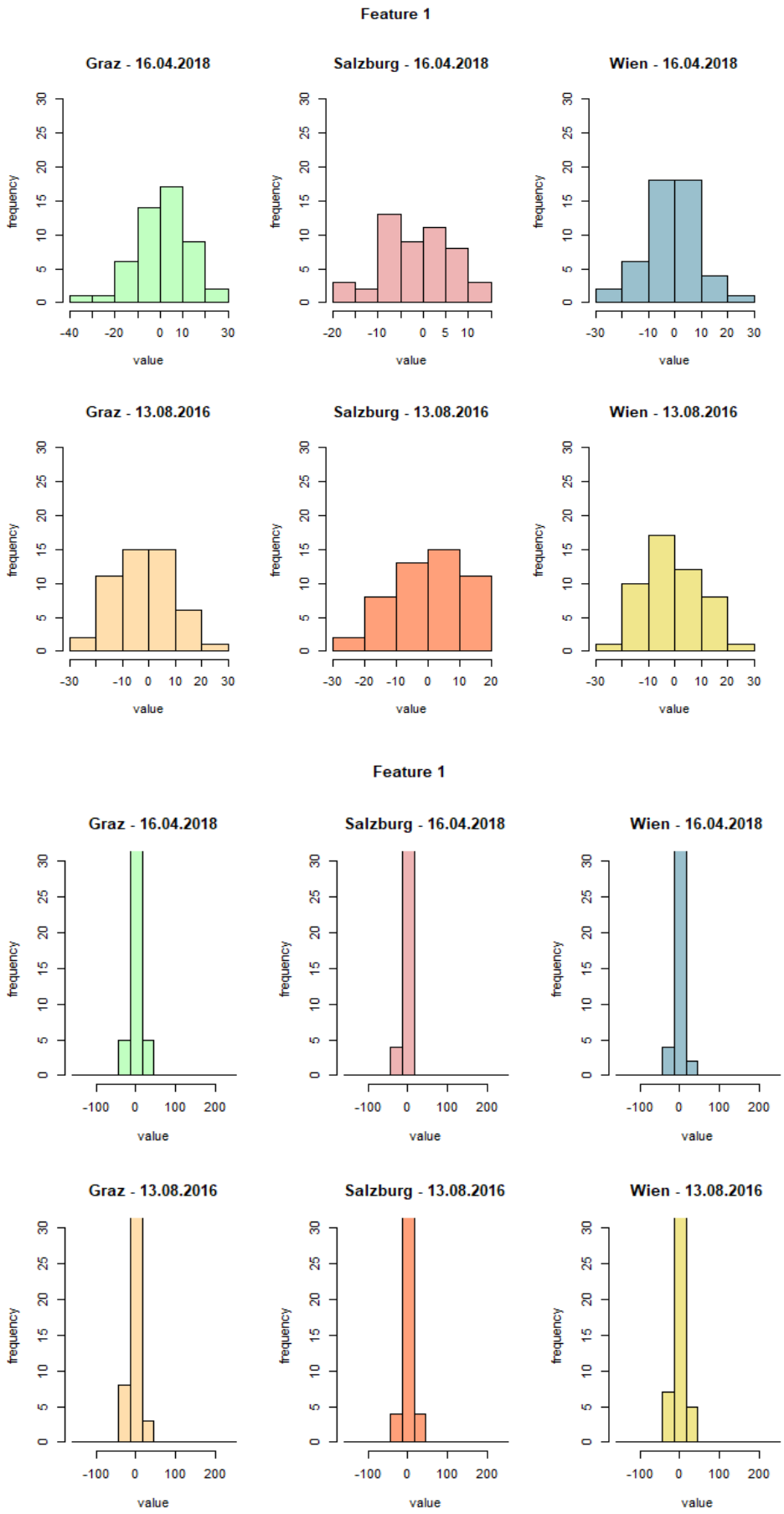
Die Plot-Funktion wird für die gruppendaten_gesamtliste aufgerufen, wodurch für alle Gruppen die passenden Plots pro Feature erstellt werden.

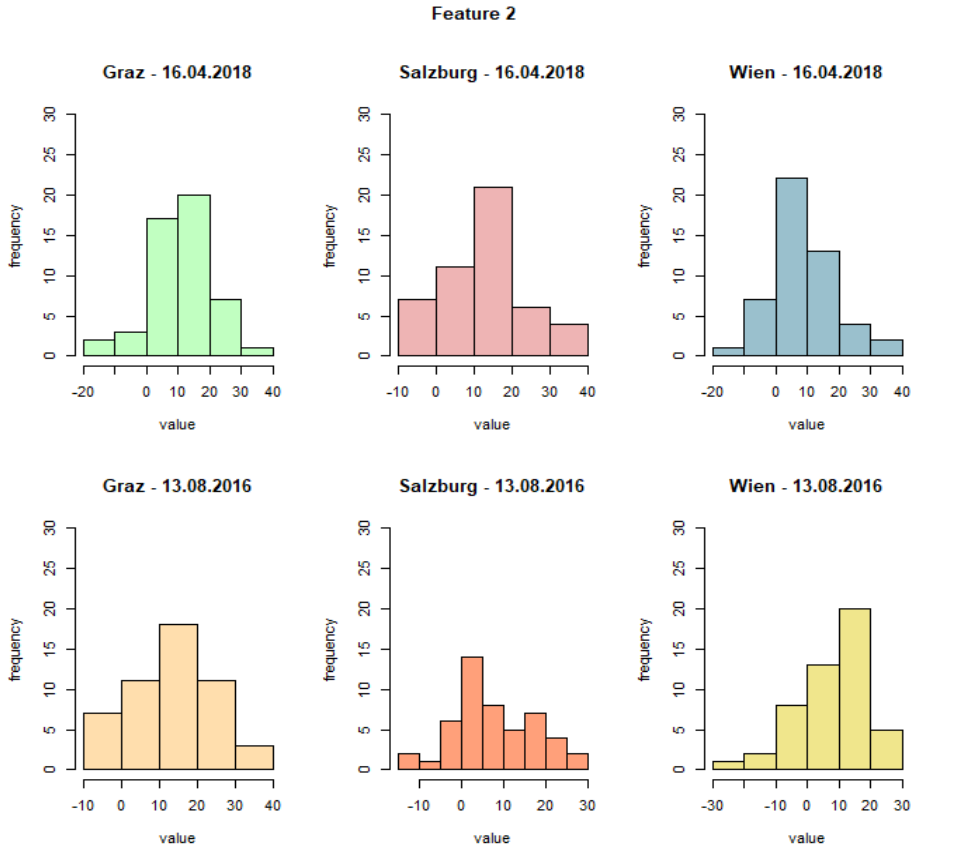
```r
# FUNCTION zum erstellen der 6 subplots eines features (an die daten angepasste x-achse)
# parameter: feature (i.e. daten pro feature aus gruppendaten_gesamtliste)
erstelleGruppenplots_adapted <- function(feature, featurenumber){

  # plot-anzeige in 3x2 subplots teilen
  par(mfrow=c(2,3), oma=c(0,0,3,0))

  # histogramme fuer die jeweiligen gruppen des derzeitigen feature plotten
  # anordnung: horizontal staedte und vertikal datumswerte
  hist(feature$G1$value, breaks=6, col="darkseagreen1",
       main="Graz - 16.04.2018", xlab="value", ylab="frequency", ylim=c(0,30))

  hist(feature$G3$value, breaks=6, col="rosybrown2",
       main="Salzburg - 16.04.2018", xlab="value", ylab="frequency",ylim=c(0,30))

  hist(feature$G5$value, breaks=6, col="lightblue3",
       main="Wien - 16.04.2018", xlab="value", ylab="frequency",ylim=c(0,30))

  hist(feature$G2$value, breaks=6, col="navajowhite",
       main="Graz - 13.08.2016", xlab="value", ylab="frequency",ylim=c(0,30))

  hist(feature$G4$value, breaks=6, col="lightsalmon",
       main="Salzburg - 13.08.2016", xlab="value", ylab="frequency",ylim=c(0,30))

  hist(feature$G6$value, breaks=6, col="khaki",
       main="Wien - 13.08.2016", xlab="value", ylab="frequency",ylim=c(0,30))

  title(paste("Feature",featurenumber), outer=TRUE)

}


# plotfunktion "erstelleGruppenplots_adapted" (mit an die daten angepassten achsenlängen)
# fuer gesamte featuresliste aufrufen
for(i in 1:length(gruppendaten_gesamtliste)){

  x11()
  erstelleGruppenplots_adapted(gruppendaten_gesamtliste[[i]], i)

}
```

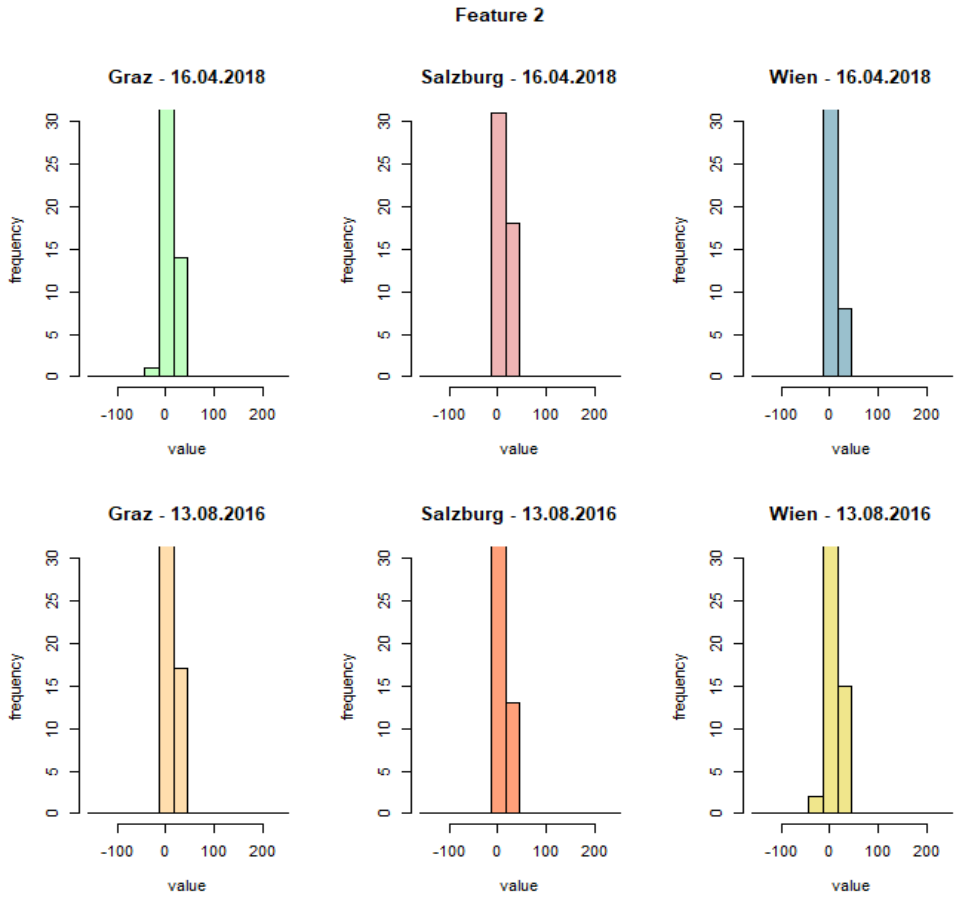
Die Funktion „erstelleGruppenplots_adapted“ erfüllt einen ähnlichen Zweck wie die vorherige Funktion. Sie stellt die Daten allerdings basierend auf an die Daten angepasste x-Achsenlängen dar, wodurch ein genauerer Blick auf die Daten gegeben werden kann.

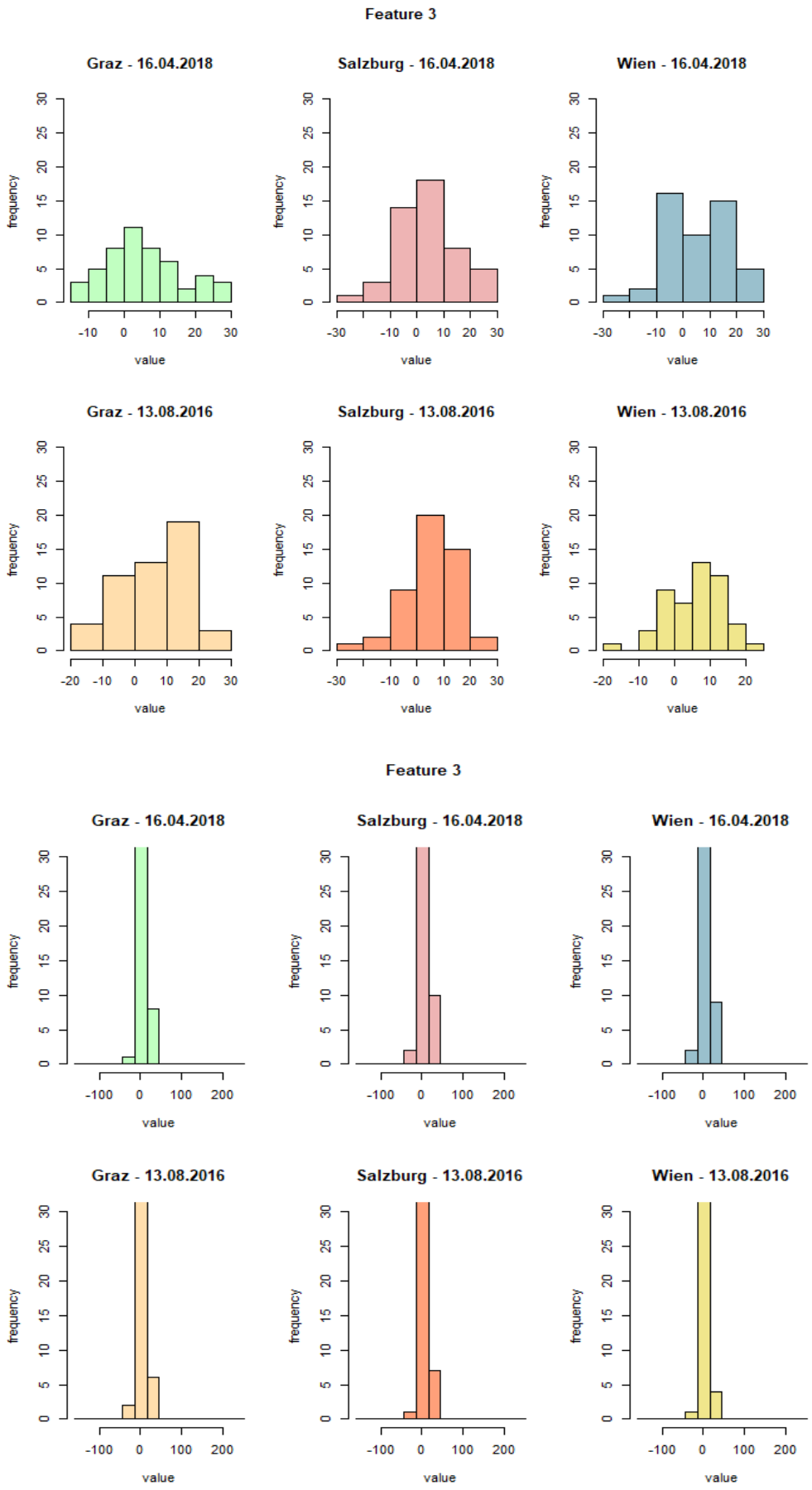
Die Subplots (Histogramme) pro Feature:

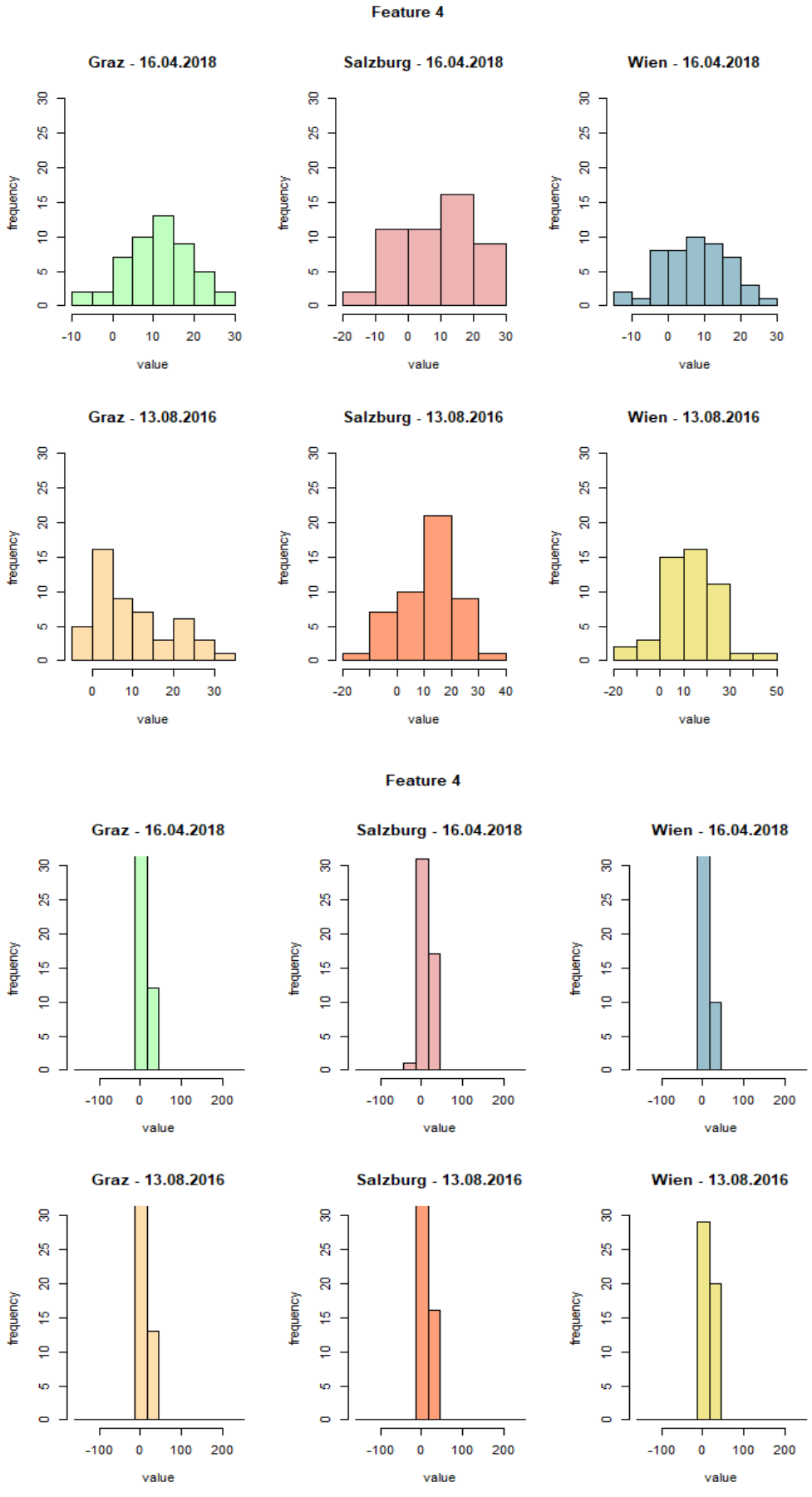
1. Output der Funktion „erstelleGruppenplots_adapted“ (Darstellung auf an die Daten angepasster x-Achsenlänge)
2. Output der Funktion „erstelleGruppenplots“ (Darstellung auf fixer Achseneinstellung)

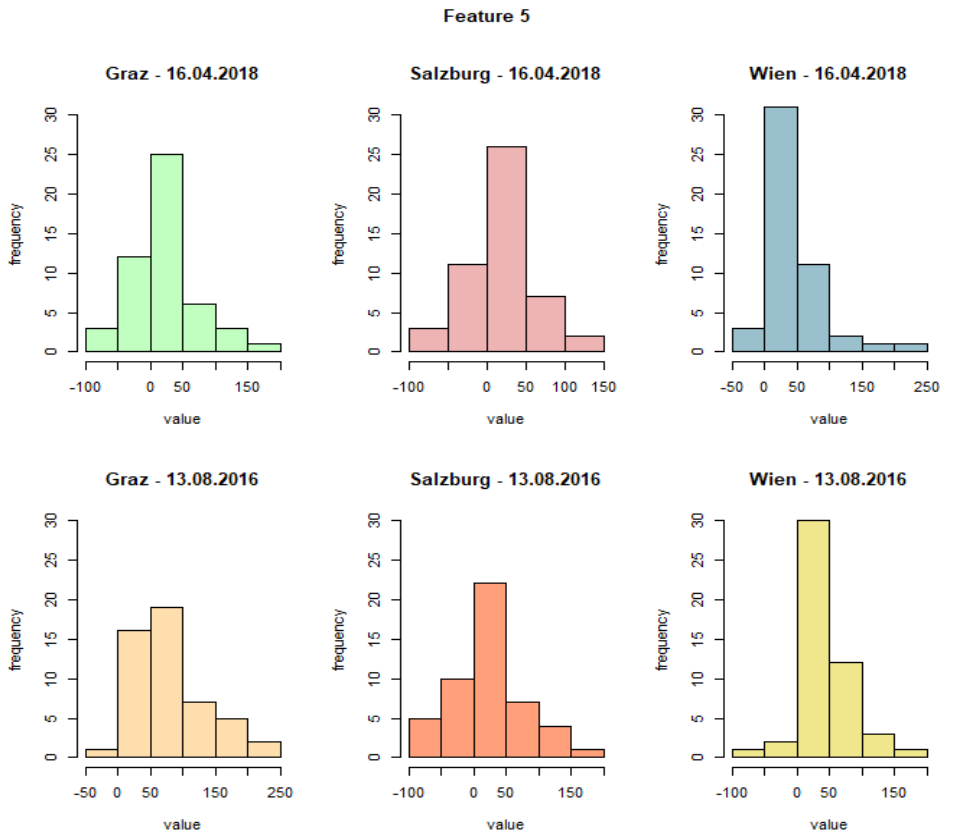

Feature 1
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
Feature 1
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
frequency
value

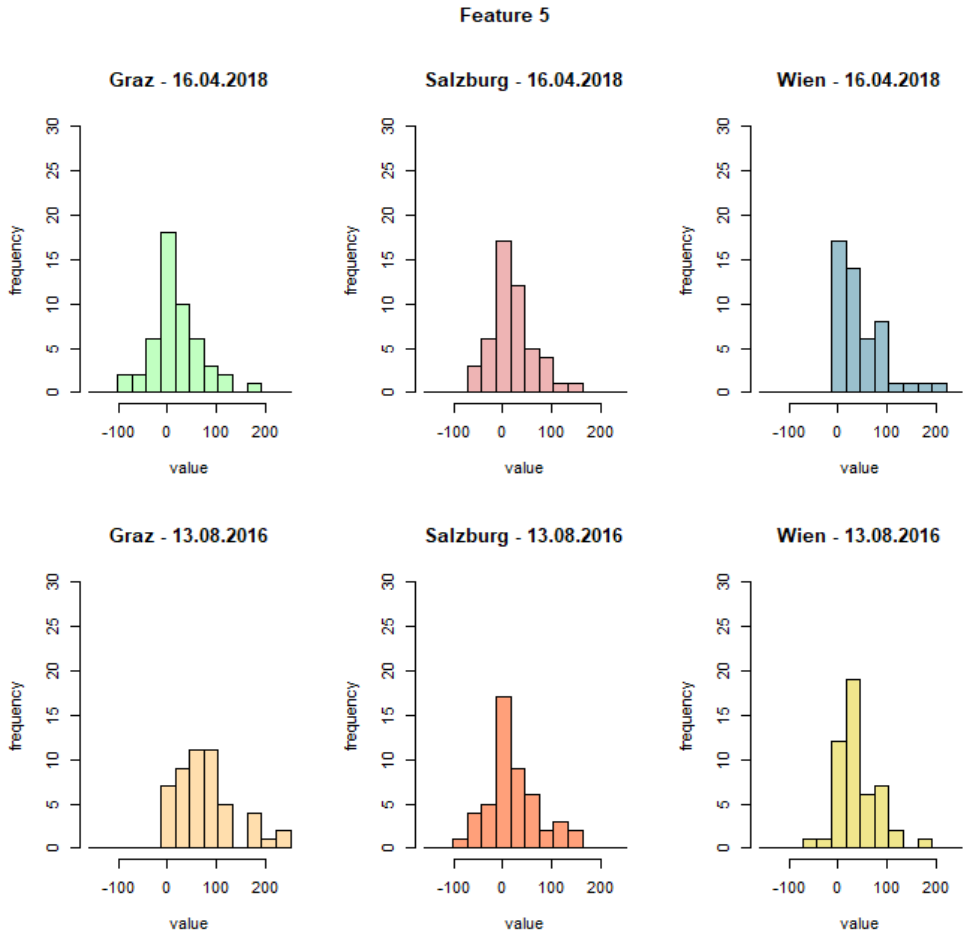

Feature 2
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
frequency
value

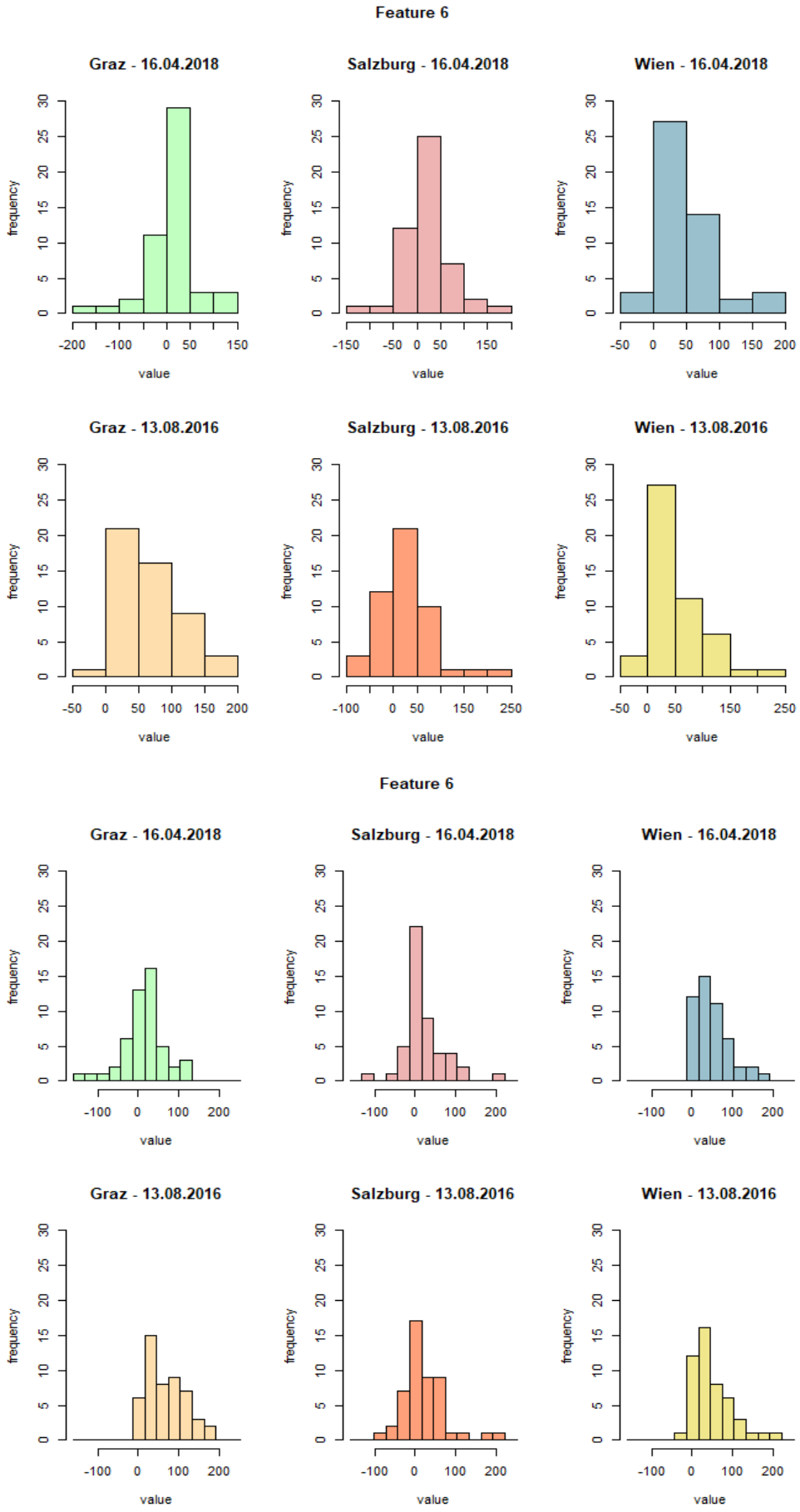

Feature 2
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
frequency
value


Feature 3
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
frequency
value
Feature 3
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
frequency
value


Feature 4
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
Feature 4
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
frequency
value


Feature 5
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
frequency
value


Feature 5
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
frequency
value


Feature 6
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
Feature 6
Graz - 16.04.2018
Salzburg - 16.04.2018
Wien - 16.04.2018
Graz - 13.08.2016
Salzburg - 13.08.2016
Wien - 13.08.2016
frequency
value

```
## ------------------------------------------------------------------------------
## VISUALISIERUNG (KORRELATIONSMATRIZEN)
## ------------------------------------------------------------------------------


### KORRELATIONSMATRIX BASIEREND AUF DEN VALUES DER FEATURES (i.e werte der variablen)

  # basisdaten
    # dataframe fuer values der features 1 bis 6 anlegen (basierend auf daten nach
    # entfernen der unvollstaendigen beobachtungen), + datum und stadt
    # datum: 1: 20160813, 2: 20180416
    # stadt: 1: graz, 2: salzburg, 3: wien
    df_features <- data.frame(f1value=liste_featuresdaten$f1$value,
                              f2value=liste_featuresdaten$f2$value,
                              f3value=liste_featuresdaten$f3$value,
                              f4value=liste_featuresdaten$f4$value,
                              f5value=liste_featuresdaten$f5$value,
                              f6value=liste_featuresdaten$f6$value,
                              datum=liste_featuresdaten$f1$Datum,
                              stadt=liste_featuresdaten$f1$Stadt,
                              stringsAsFactors=FALSE)

  str(df_features)
  df_features$datum <- as.numeric(as.factor(df_features$datum))
  df_features$stadt <- as.numeric(as.factor(df_features$stadt))


  # korrelationsmatrix berechnen
  korrelationsmatrix_feature_values <- cor(df_features)

  # korrelationsmatrix visualisieren (verschiedene typen von visiualisierung)
  # rot: negative korrelation
  # blau: positive korrelation
```

Zur Visualisierung der Korrelationsdaten werden die Daten für die jeweiligen Features sowie Datum und Stadt verwendet. Die Korrelationsmatrix wird mittels der im R schon vorhandenen Funktion „corr“ berechnet. Für die Visualisierung wird die Library „corrplot“ verwendet und daraus die Funktion „corrplot“. Letztere wird in verschiedenen Versionen basierend auf denselben Daten geplottet.

```r
  x11()

  par(mfrow=c(2,2), oma=c(0,0,3,0))

  corrplot(korrelationsmatrix_feature_values, method="color")
  corrplot(korrelationsmatrix_feature_values, method="number")
  corrplot(korrelationsmatrix_feature_values, type="upper", order="hclust")
  corrplot(korrelationsmatrix_feature_values, method = "circle")


  title("Korrelationen von Features 1-6, Daten, Stadt", outer=TRUE)



### KORRELATIONSMATRIX BASIEREND AUF DEN GESAMTDATEN (joined_all)

  # basisdaten
  basisdaten <- joined_all
  str(basisdaten)

  # cols als factor und dann numeric umwandeln, damit man cor() anwenden kann
  basisdaten$Datum <- as.numeric(as.factor(basisdaten$Datum))
  basisdaten$variable <- as.numeric(as.factor(basisdaten$variable))
  basisdaten$Stadt <- as.numeric(as.factor(basisdaten$Stadt))

  basisdaten_ohneUID_ohnePLZ <- basisdaten[ , 3:ncol(basisdaten)]
  str(basisdaten_ohneUID_ohnePLZ)

  # korrelationsmatrix berechnen
  korrelationsmatrix <- cor(basisdaten_ohneUID_ohnePLZ)

  # korrelationsmatrix visualisieren (verschiedene typen von visiualisierung)
    # rot: negative korrelation
    # blau: positive korrelation
  corrplot(korrelationsmatrix, method="color")

  corrplot(korrelationsmatrix, method="number")

  corrplot(korrelationsmatrix, type="upper", order="hclust")
```

```
### KORRELATIONSMATRIX BASIEREND AUF DEN IN DIE JEWEILIGEN FEATURES GETRENNTEN DATEN

    # basisdaten
    f1 <- daten_feature_1
    f2 <- daten_feature_2
    f3 <- daten_feature_3
    f4 <- daten_feature_4
    f5 <- daten_feature_5
    f6 <- daten_feature_6
    features_corrBasis <- list(f1=f1, f2=f2, f3=f3, f4=f4, f5=f5, f6=f6)
    str(features_corrBasis)

    # FUNCTION: fuer plot der correlation matrizen pro feature
    plotCorrelationMatrix <- function(featuredaten, numberOfFeature){

      # cols als factor und dann numeric umwandeln, damit man cor() anwenden kann
      featuredaten$Datum <- as.numeric(as.factor(featuredaten$Datum))
      featuredaten$variable <- as.numeric(as.factor(featuredaten$variable))
      featuredaten$Stadt <- as.numeric(as.factor(featuredaten$Stadt))

      featuresdaten_ohneUID_ohnePLZ <- basisdaten[ , 3:ncol(featuredaten)]
      str(featuresdaten_ohneUID_ohnePLZ)

      # korrelationsmatrix berechnen
      korrelationsmatrix_feature <- cor(featuresdaten_ohneUID_ohnePLZ)

      # korrelationsmatrix visualisieren (verschiedene typen von visiualisierung)
      # rot: negative korrelation
      # blau: positive korrelation

      # plot-anzeige in 3x2 subplots teilen

      x11()

      par(mfrow=c(2,2), oma=c(0,0,3,0))

      corrplot(korrelationsmatrix_feature, method="color")
      corrplot(korrelationsmatrix_feature, method="number")
      corrplot(korrelationsmatrix_feature, type="upper", order="hclust")
      corrplot(korrelationsmatrix_feature, method = "circle")

      title(paste("Feature",i), outer=TRUE)

    }
```

```
    # function plotCorrelationMatrix fuer die features_corrBasis liste aufrufen
    for(i in 1:length(features_corrBasis)){

      plotCorrelationMatrix(features_corrBasis[[i]], i)

    }
```

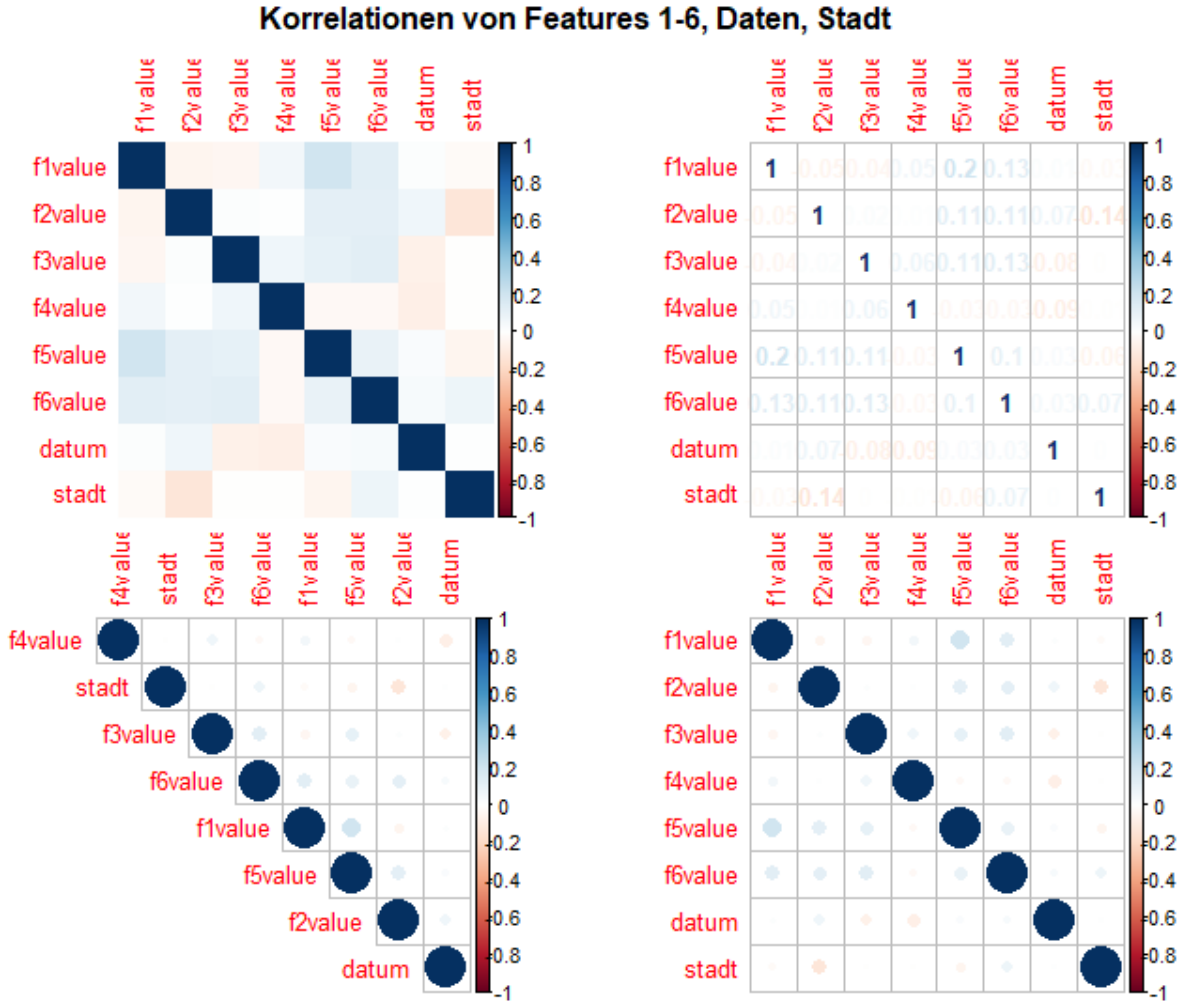
Berechnete Korrelationsmatrix:

```
> korrelationsmatrix_feature_values <- cor(df_features)
> korrelationsmatrix_feature_values
            f1value      f2value      f3value      f4value     f5value     f6value       datum        stadt
f1value  1.00000000 -0.050644099 -0.040476379  0.054269208  0.19942885  0.12990795  0.01488272 -0.026118816
f2value -0.05064410  1.000000000  0.016815921  0.005453269  0.11292580  0.11218931  0.06640479 -0.137658867
f3value -0.04047638  0.016815921  1.000000000  0.060480123  0.10582701  0.12746992 -0.07835098 -0.003039945
f4value  0.05426921  0.005453269  0.060480123  1.000000000 -0.03485945 -0.03346037 -0.08803051 -0.009018593
f5value  0.19942885  0.112925801  0.105827014 -0.034859447  1.00000000  0.09924771  0.02585653 -0.055884119
f6value  0.12990795  0.112189310  0.127469925 -0.033460366  0.09924771  1.00000000  0.03394240  0.073305816
datum    0.01488272  0.066404792 -0.078350980 -0.088030513  0.02585653  0.03394240  1.00000000  0.000000000
stadt   -0.02611882 -0.137658867 -0.003039945 -0.009018593 -0.05588412  0.07330582  0.00000000  1.000000000
```

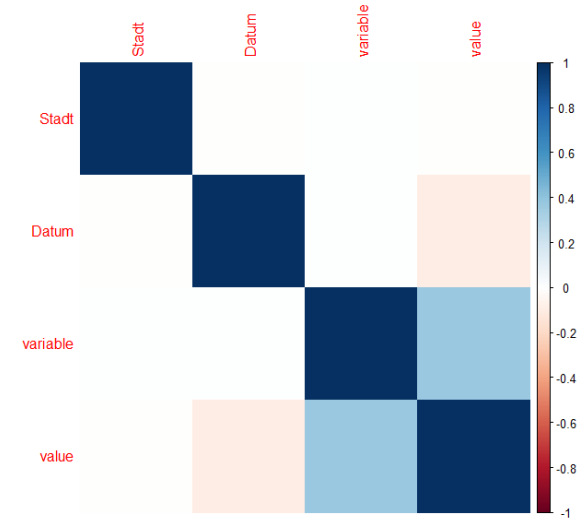
```
> str(df_features)
'data.frame':   296 obs. of  8 variables:
 $ f1value: num  -19.68 7.45 -3.68 -3.05 -5.3 ...
 $ f2value: num  3.55 -11 14.56 -8.44 16.49 ...
 $ f3value: num  7.85 -2.88 10.38 10.99 -16.77 ...
 $ f4value: num  6.06 -3.58 13.07 24.81 13.39 ...
 $ f5value: num  44.8 -54.43 5.05 57.47 95.96 ...
 $ f6value: num  41.77 -5.41 2.88 21.18 19.91 ...
 $ datum  : num  1 2 1 2 2 1 1 2 1 2 ...
 $ stadt  : num  3 3 1 1 3 3 3 2 2 ...
```

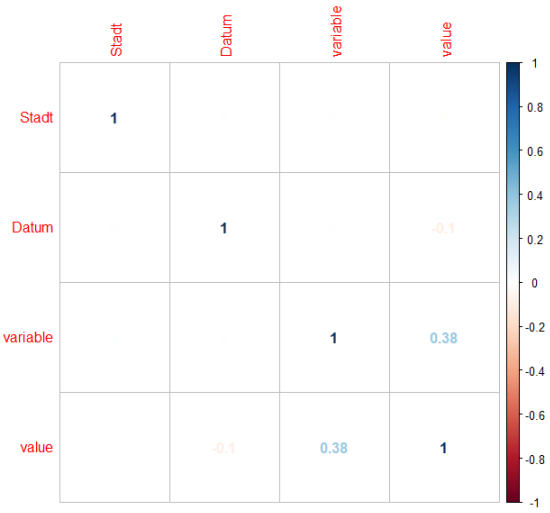
Visualisierung der Korrelationen:

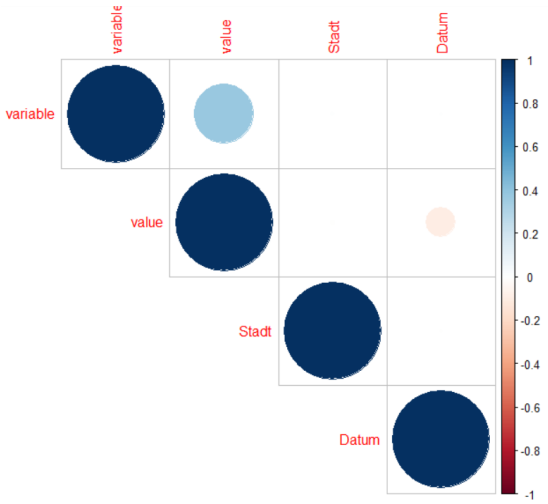
- Rot: negative Korrelation
- Blau: positive Korrelation



Korrelationen zwischen Stadt, Datum, variable und value:

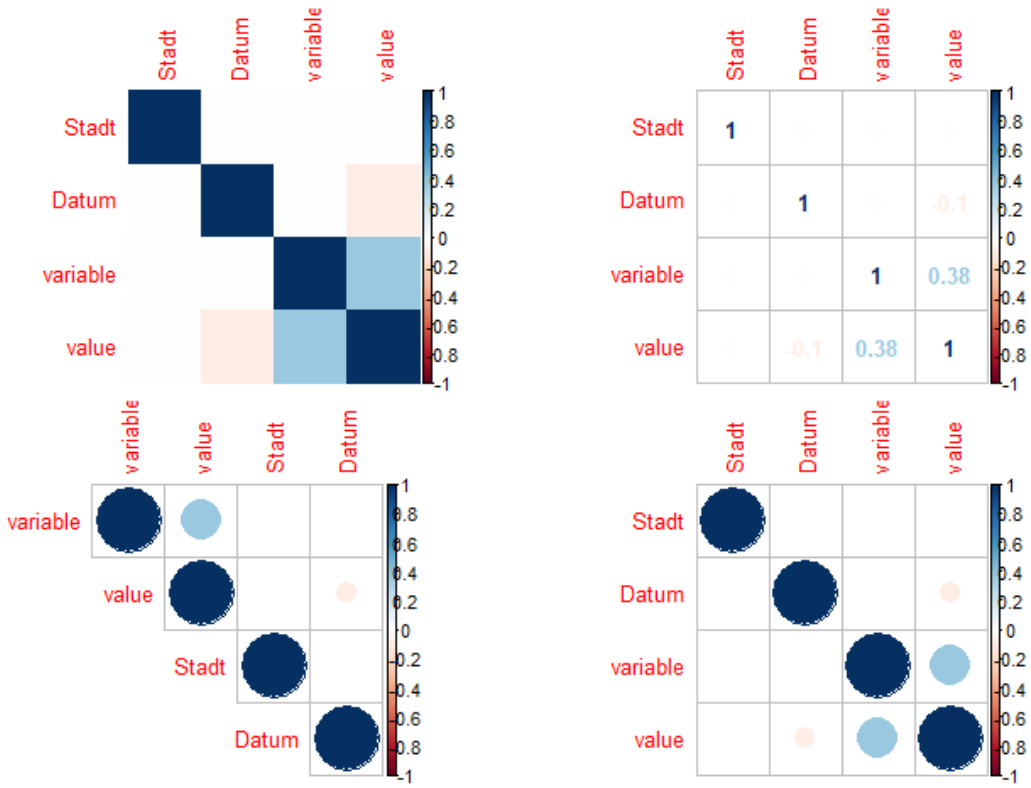




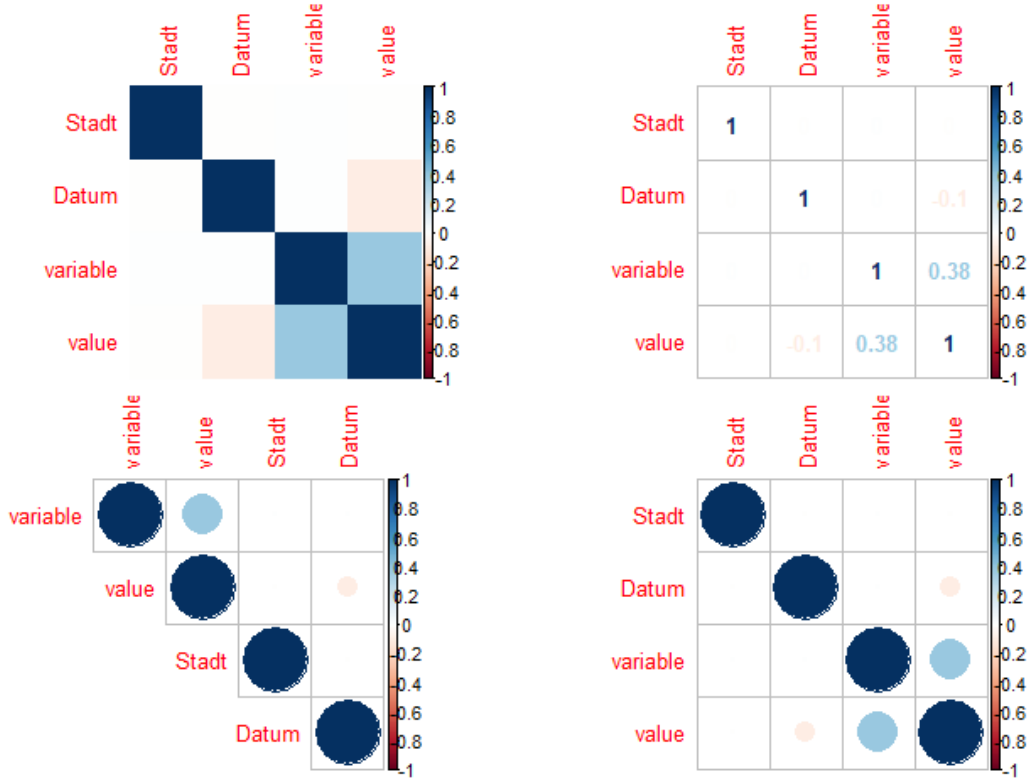


Korrelationen pro Feature (UID und PLZ Spalte weggelassen):

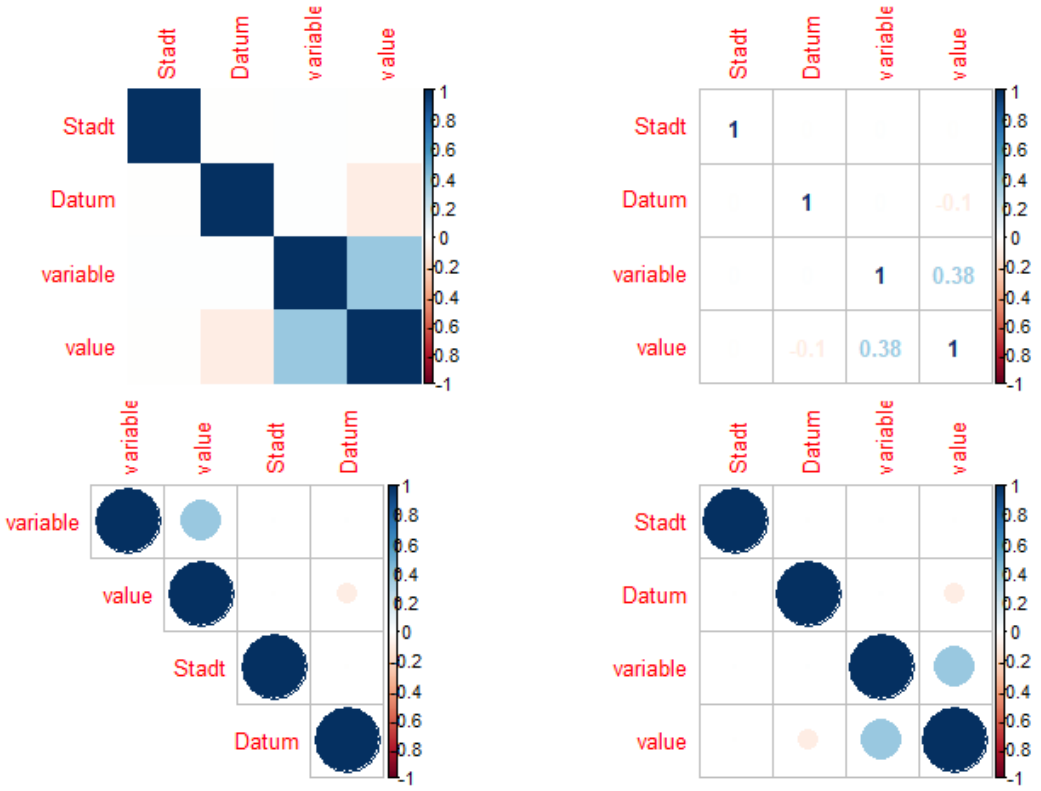
Feature 1



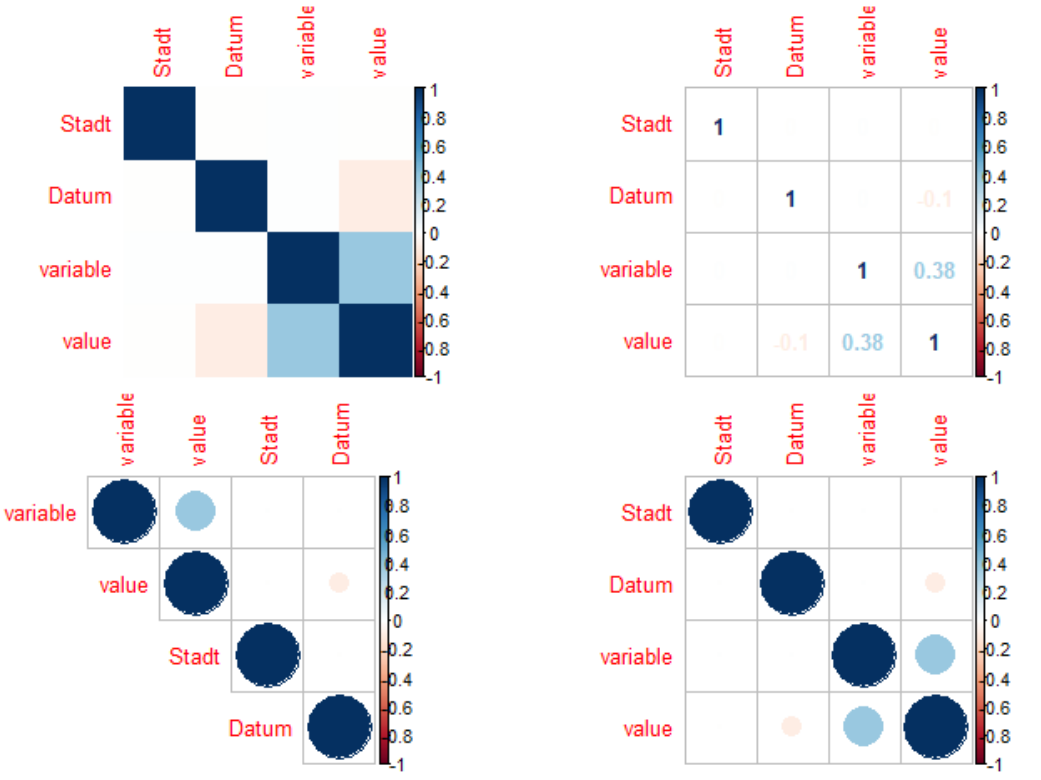
Feature 2



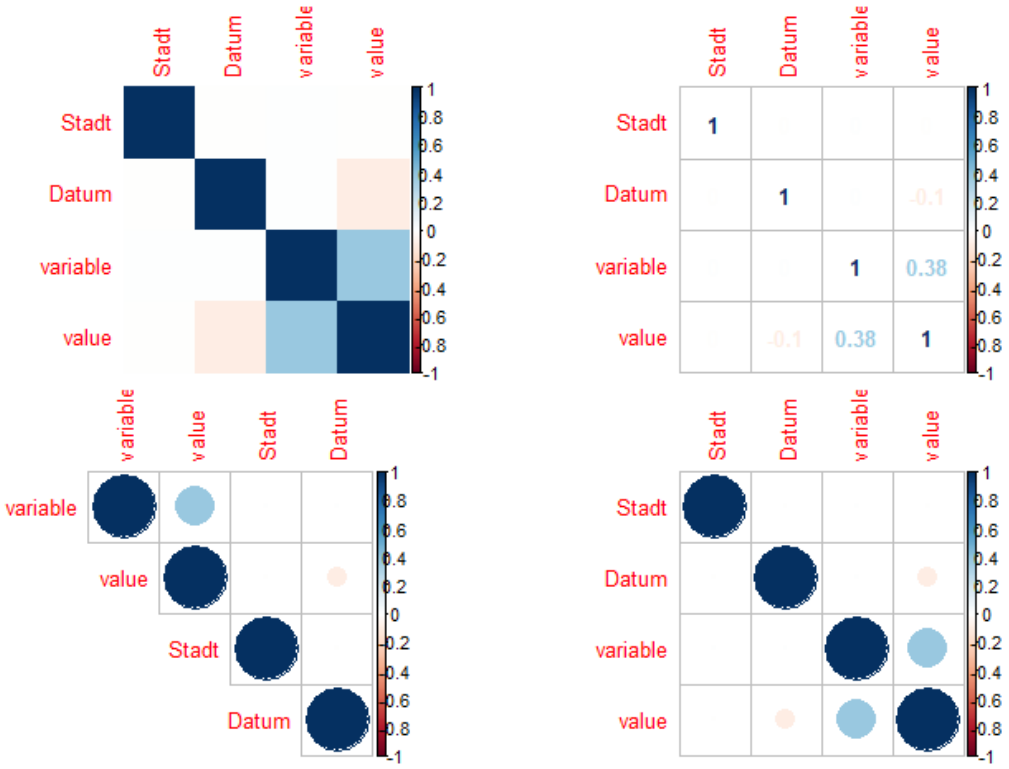
Feature 3



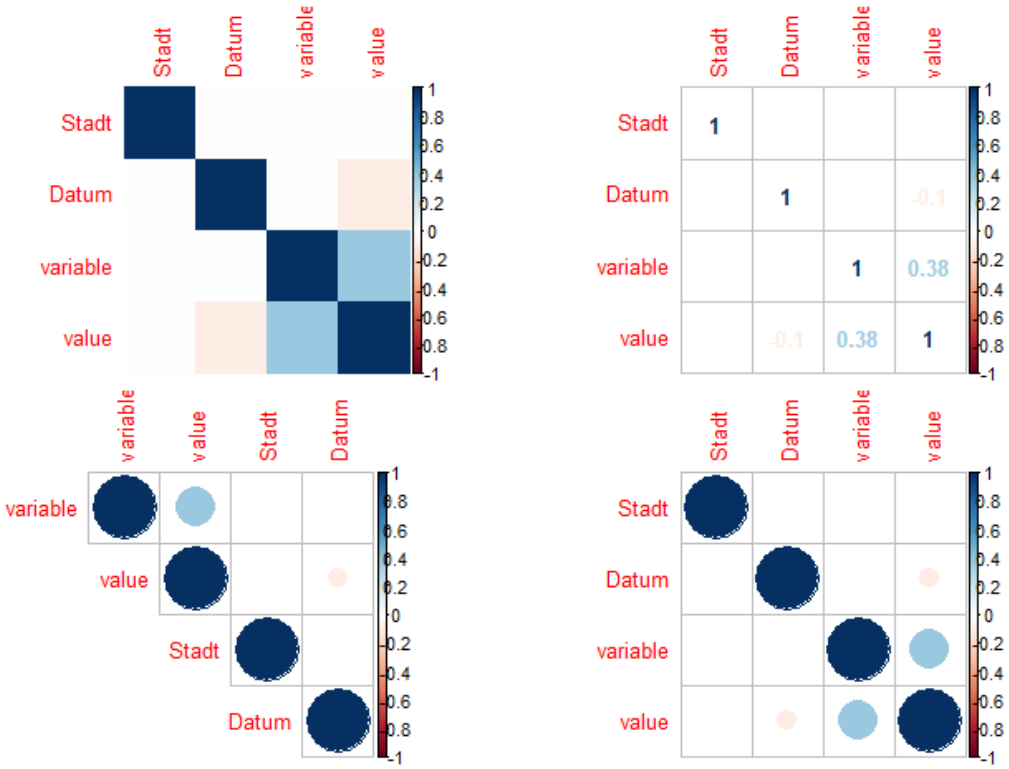
Feature 4



Feature 5



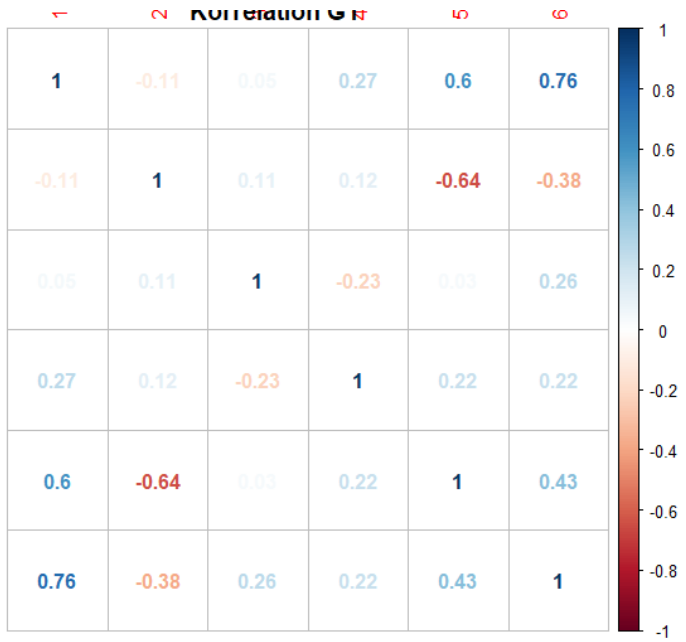
Feature 6



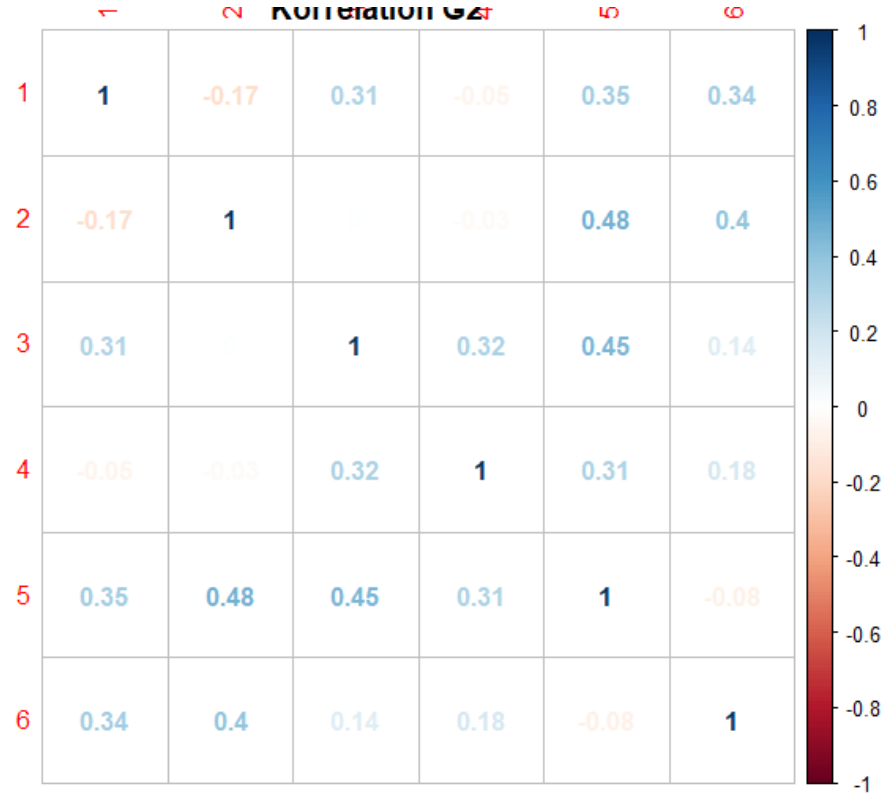
Korrelation basierend auf den Gruppen pro jeweiligem Feature:

```
## KORRELATIONSMATRIZEN PRO GRUPPE:
    # FUNCTION zum erstellen 6 gruppenplots
    # parameter: gesamtdatenliste (enthält alle features und jeweils gruppeninfos)
    # parameter: gruppenNr (nummer der betrachteten gruppe)
    erstelleGruppenplots_correlation <- function(gesamtdatenliste, gruppenName, gruppenNr){

      par(mfrow=c(1,1), oma=c(0,0,0,0), cex.main=0.7)

      # features trennen
      feature1 <- gruppendaten_gesamtliste[[1]]
      feature2 <- gruppendaten_gesamtliste[[2]]
      feature3 <- gruppendaten_gesamtliste[[3]]
      feature4 <- gruppendaten_gesamtliste[[4]]
      feature5 <- gruppendaten_gesamtliste[[5]]
      feature6 <- gruppendaten_gesamtliste[[6]]


      # correlationsmatrix fuer die gerade betrachtete gruppe berechnen
      f1_gr <- feature1[[gruppenNr]]
      f2_gr <- feature2[[gruppenNr]]
      f3_gr <- feature3[[gruppenNr]]
      f4_gr <- feature4[[gruppenNr]]
      f5_gr <- feature5[[gruppenNr]]
      f6_gr <- feature6[[gruppenNr]]


      korrelationsmatrix_gruppe <- cor(cbind(f1_gr$value, f2_gr$value, f3_gr$value,
                                             f4_gr$value, f5_gr$value, f6_gr$value)
                                       )

      # correlation plot
      x11()
      corrplot(korrelationsmatrix_gruppe, method="number", main=paste("Korrelation", gruppenName))
      # title(paste("Korrelation", gruppenName), outer=TRUE)

    }


    # plotfunktion "erstelleGruppenplots_correlation" fuer gesamte featuresliste aufrufen
    n_gruppen <- 6
    gruppen <- c("G1", "G2", "G3", "G4", "G5", "G6")
    for(i in 1:n_gruppen){

      erstelleGruppenplots_correlation(gruppendaten_gesamtliste, gruppen[i], i)

    }
```

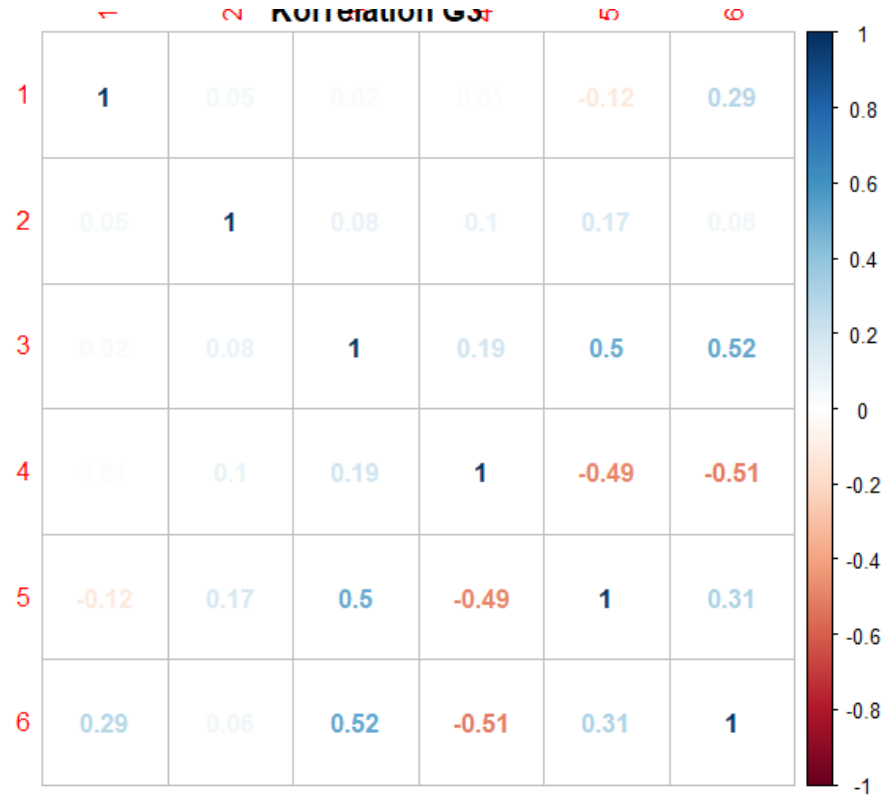
G1:



G2:


1 2 3 4 5 6
1 1 -0.17 0.31 -0.05 0.35 0.34
2 -0.17 1 0.03 -0.03 0.48 0.4
3 0.31 0.03 1 0.32 0.45 0.14
4 -0.05 -0.03 0.32 1 0.31 0.18
5 0.35 0.48 0.45 0.31 1 -0.08
6 0.34 0.4 0.14 0.18 -0.08 1
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

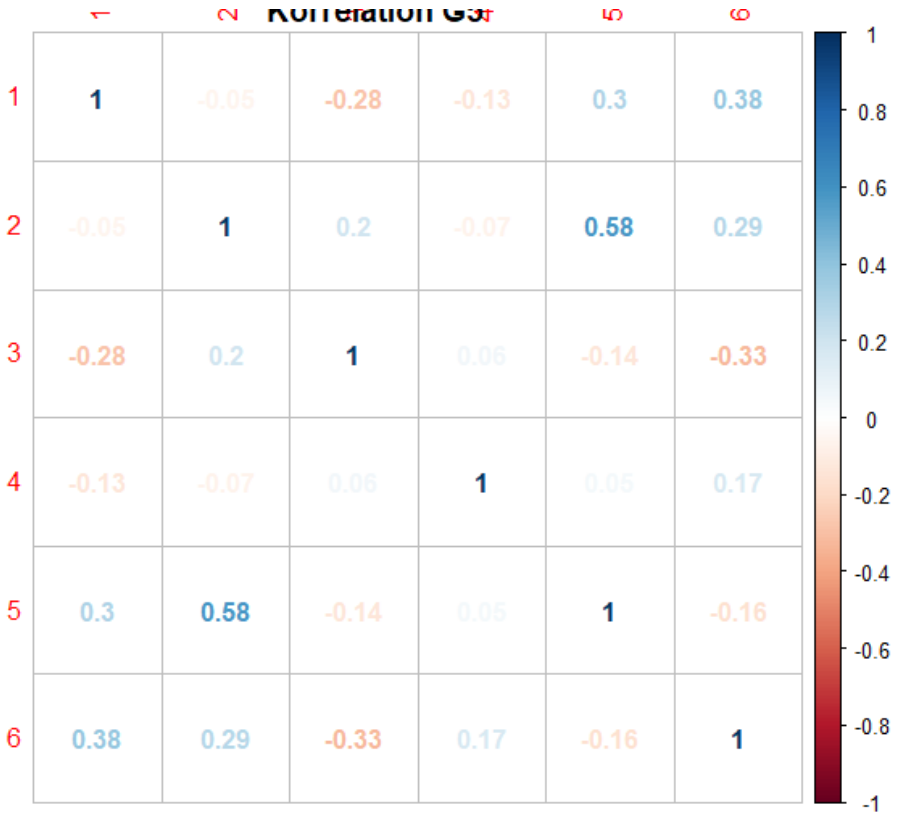
G3:


1 2 3 4 5 6
1 1 0.05 0.02 0.01 -0.12 0.29
2 0.05 1 0.08 0.1 0.17 0.06
3 0.02 0.08 1 0.19 0.5 0.52
4 0.01 0.1 0.19 1 -0.49 -0.51
5 -0.12 0.17 0.5 -0.49 1 0.31
6 0.29 0.06 0.52 -0.51 0.31 1
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

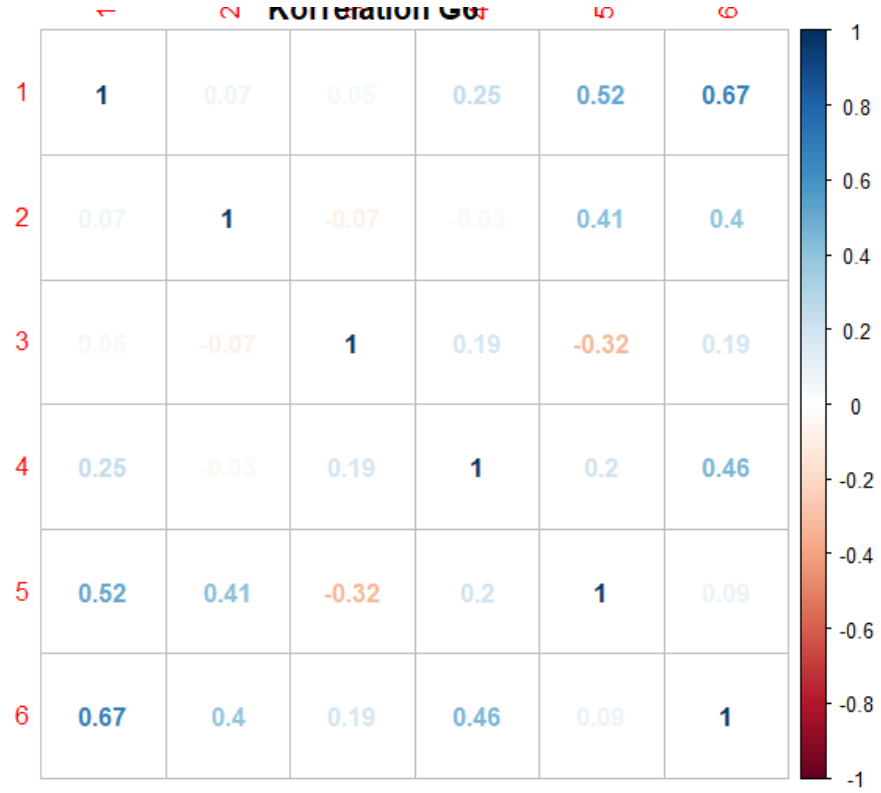
G4:


1 2 3 4 5 6
1 1 0.14 0 0.57 0.58
2 0.14 1 -0.27 0.26 0.13 0.1
3 0 -0.27 1 -0.14 0.34 0.38
4 0.26 -0.14 1 -0.47 -0.45
5 0.57 0.13 0.34 -0.47 1 0.38
6 0.58 0.1 0.38 -0.45 0.38 1
1 0.8 0.6 0.4 0.2 0 -0.2 -0.4 -0.6 -0.8 -1

G5:


1 2 3 4 5 6
1 1 -0.05 -0.28 -0.13 0.3 0.38
2 -0.05 1 0.2 -0.07 0.58 0.29
3 -0.28 0.2 1 0.06 -0.14 -0.33
4 -0.13 -0.07 0.06 1 0.05 0.17
5 0.3 0.58 -0.14 0.05 1 -0.16
6 0.38 0.29 -0.33 0.17 -0.16 1
1 0.8 0.6 0.4 0.2 0 -0.2 -0.4 -0.6 -0.8 -1

G6:



Anmerkung:

Man sieht aus der Visualisierung der Korrelationsmatrizen der Feature-Werte innerhalb der 6 Gruppen, dass die Korrelationen der verschiedenen Variablen je nach Gruppen variieren.

Aufgabe 2 in R

Aufgabe 2 – Aufgabenstellung:

a. Erzeugen Sie eine abgeleitete Variable aus der Summe von Feature 5 und Feature 6

b. Gibt es Korrelationen zwischen den verbleibenden Variablen und der neuen abgeleiteten Variable?

c. Modellieren Sie die abgeleitete Variable mit einem linearen Modell.

d. Welche Variablen sind im Modell sinnvoll, wie gehen Sie mit den kategoriellen Variablen um?

e. Beschreiben Sie Ihre Modellierungsergebnisse und erzeugen Sie Grafiken um Ihre Ergebnisse zu dokumentieren.

f. Welche Modellierungsmethode verwenden Sie und warum haben Sie sich für dieses Modell entschieden?

```
#########################################################################################################
### AUFGABE 2 ###
#########################################################################################################

# 2.a. Erzeugen Sie eine abgeleitete Variable aus der Summe von Feature 5 und Feature 6

      # basisdaten
      # dataframe fuer values der features 1 bis 6 anlegen (basierend auf daten nach
      # entfernen der unvollstaendigen beobachtungen)
      df_features_basis <- data.frame(f1value=liste_featuresdaten$f1$value,
                                      f2value=liste_featuresdaten$f2$value,
                                      f3value=liste_featuresdaten$f3$value,
                                      f4value=liste_featuresdaten$f4$value,
                                      f5value=liste_featuresdaten$f5$value,
                                      f6value=liste_featuresdaten$f6$value,
                                      stringsAsFactors=FALSE)

      # neue variable "f_sum56" berechnen (und f5 sowie f6 weglassen, f1 bis f4 behalten)
      df_features_inklSum56 <- df_features_basis[ ,1:4]
      df_features_inklSum56$f_sum56 <- (df_features_basis$f5value + df_features_basis$f6value)

      # stadt und datum informationen einfuegen
      df_features_inklSum56$datum <- liste_featuresdaten$f1$Datum
      df_features_inklSum56$stadt <- liste_featuresdaten$f1$Stadt

      str(df_features_inklSum56)

      # col datum als factor und umwandeln
      df_features_inklSum56$datum <- as.factor(df_features_inklSum56$datum)
```

Aus den Summen von Feature 5 und Feature 6 wird in den gesamten Featuredaten eine neue Variable erstellt und an das Dataframe angehängt. Die Features 5 und 6 werden weggelassen. Auch Datum und Stadt werden hinzugenommen.

Das neu entstandene Dataframe sieht folgendermaßen aus (siehe untenstehender Ausschnitt):

```
> df_features_inklSum56
         f1value      f2value      f3value      f4value      f_sum56           datum    stadt
1   -19.68214247   3.55493617   7.84698810   6.06161436   86.5737337 20160813T00:00Z     Wien
2     7.45469925 -10.99864901  -2.87834574  -3.58051249  -59.8416642 20180416T00:00Z     Wien
3    -3.67649927  14.55576307  10.38206434  13.07458453    7.9305051 20160813T00:00Z     Graz
4    -3.05179755  -8.43542982  10.98891880  24.80510099   78.6530788 20180416T00:00Z     Graz
5    -5.30167981  16.48727019 -16.76932412  13.38603087  115.8687989 20180416T00:00Z     Wien
6    -6.18525240   0.45185190   2.21705758  17.14407640   64.0475463 20160813T00:00Z     Wien
7   -17.65603396  30.59619671   6.37864806  17.47223119  183.8047180 20160813T00:00Z     Wien
8     2.54664363  15.30399160  12.73780839   3.49515161   63.9895971 20180416T00:00Z     Wien
9    -0.15401186  11.14712630  -7.36734782  28.31361219  -28.9704204 20160813T00:00Z Salzburg
10    8.75238577   1.84417916   8.55328351   8.62629897  -18.4818465 20180416T00:00Z Salzburg
11    2.27744401   2.11490074  -1.06459512   4.58862895   74.7090095 20180416T00:00Z     Wien
12   -2.15678339  -3.50145062  10.64524087  20.42514207   37.5811942 20160813T00:00Z     Wien
13    3.53370607   2.13443334   1.10835960   4.71133790   -4.3705144 20180416T00:00Z     Graz
14  -12.52508016  25.41931699 -15.53068918  12.30141895  -20.4826533 20160813T00:00Z     Graz
15    3.45253559   5.28014026  -2.92541475  10.05406441  -70.2218506 20160813T00:00Z Salzburg
16   -8.87050482   1.64869027 -10.60642370  22.05116320  -22.1874994 20180416T00:00Z Salzburg
```

```
> str(df_features_inklSum56)
'data.frame':   296 obs. of  7 variables:
 $ f1value: num  -19.68 7.45 -3.68 -3.05 -5.3 ...
 $ f2value: num  3.55 -11 14.56 -8.44 16.49 ...
 $ f3value: num  7.85 -2.88 10.38 10.99 -16.77 ...
 $ f4value: num  6.06 -3.58 13.07 24.81 13.39 ...
 $ f_sum56: num  86.57 -59.84 7.93 78.65 115.87 ...
 $ datum  : Factor w/ 2 levels "20160813T00:00Z",..: 1 2 1 2 2 1 1 2 1 2 ...
 $ stadt  : Factor w/ 3 levels "Graz","Salzburg",..: 3 3 1 1 3 3 3 3 2 2 ...
```

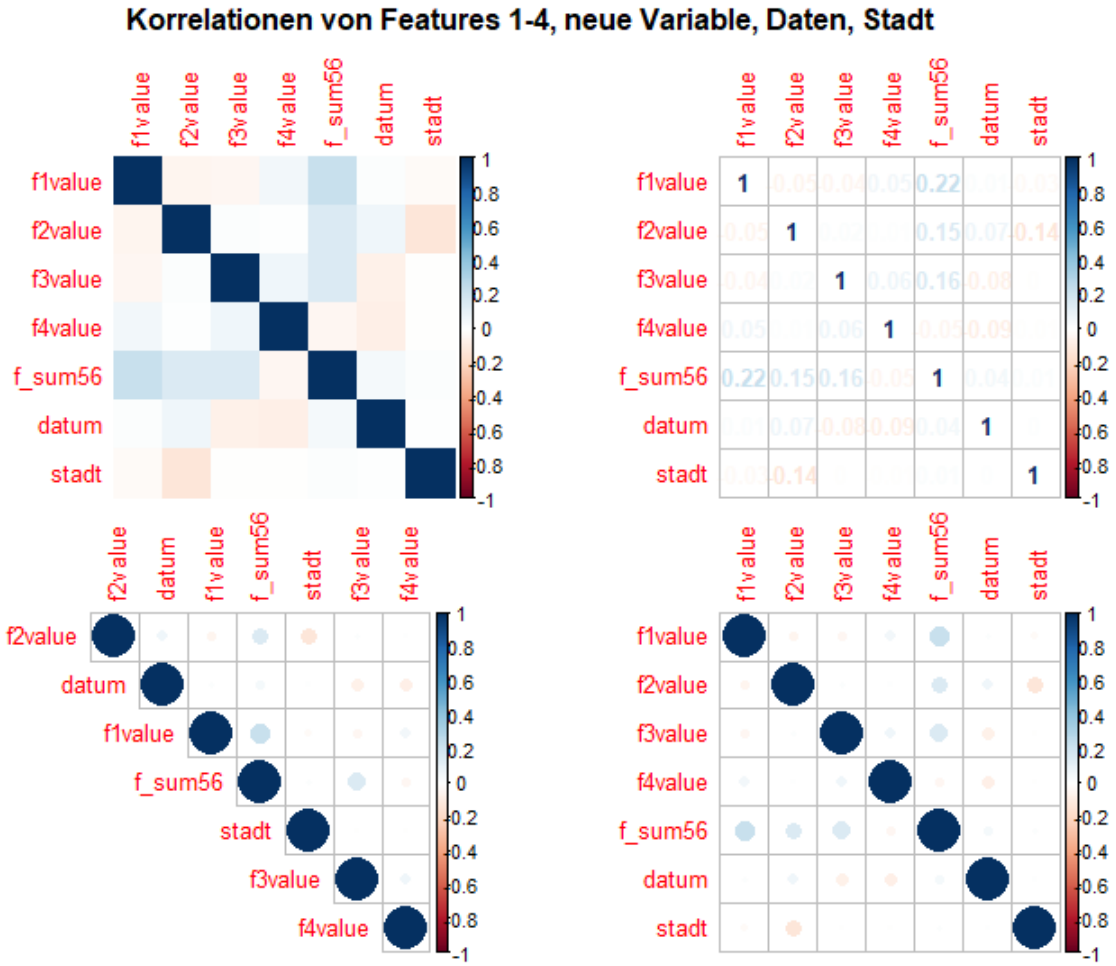
```
# 2.b. Gibt es Korrelationen zwischen den verbleibenden Variablen und der neuen abgeleiteten Variable?

    # datum: 1: 20160813, 2: 20180416
    # stadt: 1: graz, 2: salzburg, 3: wien
    df_features_inklSum56$datum <- as.numeric(as.factor(df_features_inklSum56$datum))
    df_features_inklSum56$stadt <- as.numeric(as.factor(df_features_inklSum56$stadt))


    # korrelationsmatrix berechnen
    korrelationsmatrix_inklSum56 <- cor(df_features_inklSum56)

    # korrelationsmatrix visualisieren (verschiedene typen von visiualisierung)
    # rot: negative korrelation
    # blau: positive korrelation

    # plot-anzeige in 2x2 subplots teilen

    x11()

    par(mfrow=c(2,2), oma=c(0,0,3,0))

    corrplot(korrelationsmatrix_inklSum56, method="color")
    corrplot(korrelationsmatrix_inklSum56, method="number")
    corrplot(korrelationsmatrix_inklSum56, type="upper", order="hclust")
    corrplot(korrelationsmatrix_inklSum56, method = "circle")


    title("Korrelationen von Features 1-4, neue Variable, Daten, Stadt", outer=TRUE)
```

Für die Untersuchung der Korrelation wird die Korrelationsmatrix für das neue Dataframe df_features_inklSum56 gebildet. Basierend darauf wird corrplot aufgerufen, und die Korrelation wird auf verschiedene Arten visualisiert.

Die Korrelationsmatrix:

```
> korrelationsmatrix_inklSum56 <- cor(df_features_inklSum56)
> korrelationsmatrix_inklSum56
            f1value      f2value      f3value      f4value     f_sum56       datum        stadt
f1value  1.00000000 -0.050644099 -0.040476379  0.054269208  0.22240239  0.01488272 -0.026118816
f2value -0.05064410  1.000000000  0.016815921  0.005453269  0.15182519  0.06640479 -0.137658867
f3value -0.04047638  0.016815921  1.000000000  0.060480123  0.15724946 -0.07835098 -0.003039945
f4value  0.05426921  0.005453269  0.060480123  1.000000000 -0.04608210 -0.08803051 -0.009018593
f_sum56  0.22240239  0.151825188  0.157249464 -0.046082098  1.00000000  0.04029573  0.011208573
datum    0.01488272  0.066404792 -0.078350980 -0.088030513  0.04029573  1.00000000  0.000000000
stadt   -0.02611882 -0.137658867 -0.003039945 -0.009018593  0.01120857  0.00000000  1.000000000
```

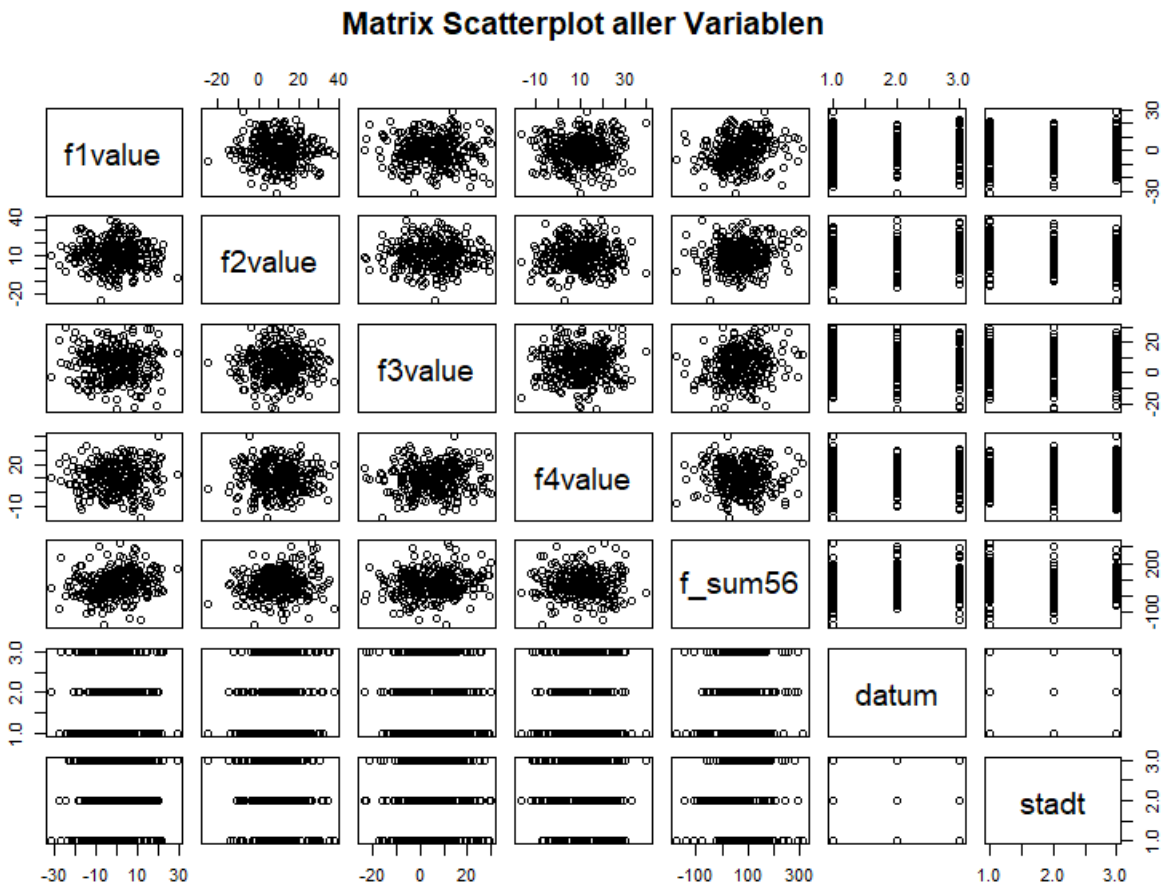
Die Visualisierung der Korrelation:



Lineares Modell – Versuch 1 (additiver Zusammenhang):

```
# 2.c. Modellieren Sie die abgeleitete Variable mit einem linearen Modell.
# 2.d. Welche Variablen sind im Modell sinnvoll, wie gehen Sie mit den kategoriellen Variablen um?
# 2.e. Beschreiben Sie Ihre Modellierungsergebnisse und erzeugen Sie Grafiken um Ihre Ergebnisse zu dokumentieren.
# 2.f. Welche Modellierungsmethode verwenden Sie und warum haben Sie sich für dieses Modell entschieden?

      # dummy-codierung bzw. treatment-codierung der kategoriellen variablen (je nach anzahl der factorlevels)
      # datum:
      df_features_inklSum56$datum <- as.factor(df_features_inklSum56$datum)
      contr.treatment(2) # contrastmatrix fuer 2 levels des factor
      contrasts(df_features_inklSum56$datum) = contr.treatment(2) # treatmentcontrast zuweisen

      # stadt:
      df_features_inklSum56$stadt <- as.factor(df_features_inklSum56$stadt)
      contr.treatment(3) # contrastmatrix fuer 3 levels des factor
      contrasts(df_features_inklSum56$stadt) = contr.treatment(3) # treatmentcontrast zuweisen

      str(df_features_inklSum56)

      # MATRIX-SCATTERPLOT ALLER DATEN VOR REGRESSION:
      # (ueberblick ueber die paarweise relation der daten erhalten)
      plot(df_features_inklSum56, pch=1, col="black", main="Matrix Scatterplot aller Variablen")


      # VERSUCH 1: zunaechst normales lineares modell (additiv die variablen beruecksichtigt)
      limod_results <- lm(f_sum56 ~ f1value+f2value+f3value+f4value+datum+stadt, data = df_features_inklSum56)
      summary(limod_results)
```

Das Datum und die Stadt werden zunächst als Factor verwendet und mittels Treatment-Codierung werden die Kontraste gesetzt. Danach werden Versuche von linearen Modellen gemacht. Ein erster Matrix-Scatterplot der Daten gibt einen Überblick über die Datenlage, wie untenstehend zu sehen ist:

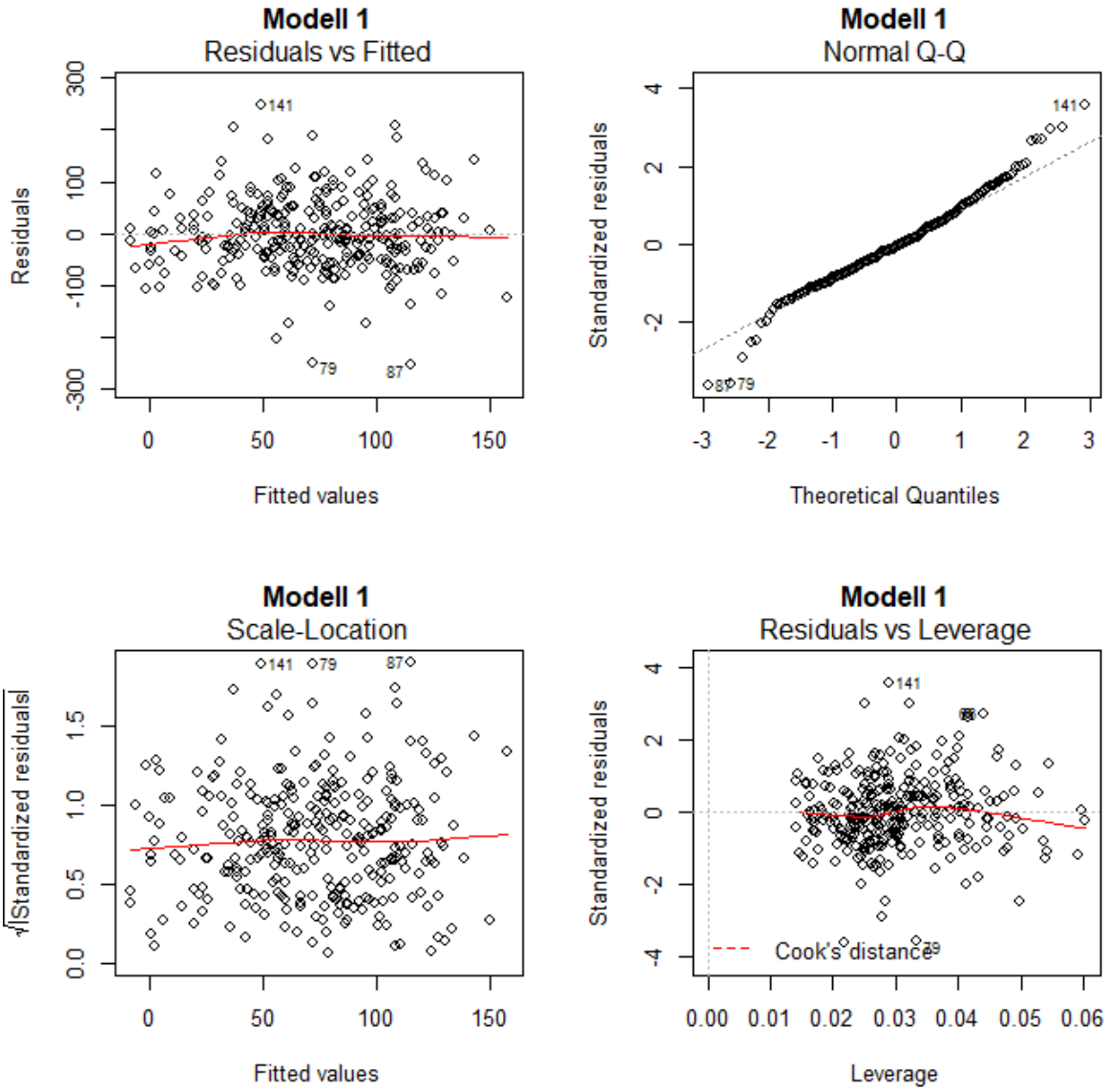


```
# diagnose der regressionsresultate
# x11()
par(mfrow = c(2,2))
plot(limod_results, main="Modell 1")
# allgemeine anmerkung zu den plots:
# 1. Residuals vs. Fitted:
#    Scatterplot zwischen Residuen und Predicted Values. Dieser sollte moeglichst "zufaellig aussehen".
# 2. Normal Q-Q:
#    Wenn die Fehler normalverteilt sind, liegen die Punkte auf der eingezeichneten Gerade.
# 3. Scale-Location:
#    Sollte "zufaellig" ausschauen und keine Muster aufweisen. Kann in etwa die Annahme der Homoskedastie
#    beurteilen - Annahme weitgehend erfuellt wenn horizontale Linie und random verstreute Punkte.
# 4. Residuals vs. Leverage:
#    Plot hilft, "einflussreiche" Outlier ("leverage points") zu identifizieren.Punkte außerhalb der Cook's Distanz
#    haben Einfluss auf die Regressionsergebnisse.

# influenceplot dafuer
par(mfrow = c(1,1))
influence.measures(limod_results)
influencePlot(limod_results2, main="Influence Plot",
              sub="Anmerkung: Kreisgröße proportional zu Cook's Distance" )
# outliertest
outlierTest(limod_results)
```

Mit der Funktion lm() wurde ein erster Versuch für ein lineares Modell gemacht, in dem die Regressoren additiv betrachtet wurden. Die adj. erklärte Varianz (adj. R-Squared) liegt bei 17.37%. Signifikante Ergebnisse liegen bei f1value und stadt2 (i.e. Gruppe Salzburg) vor. Die Werte für f2value und f3value sind ebenfalls signifikant. Bei Datum, f4value und stadt3 (i.e. Wien) liegen keine signifikanten Ergebnisse vor. Die gesamte Auswertung des linearen Modells sieht folgendermaßen aus:

```
Call:
lm(formula = f_sum56 ~ f1value + f2value + f3value + f4value +
    datum + stadt, data = df_features_inklSum56)

Residuals:
     Min       1Q   Median       3Q      Max
-252.941  -43.428   -4.279   39.213  247.994

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  69.5159    10.6707   6.515 3.26e-10 ***
f1value       1.8128     0.3899   4.649 5.08e-06 ***
f2value       1.2162     0.4046   3.006  0.00288 **
f3value       1.2459     0.4025   3.096  0.00216 **
f4value      -0.5290     0.4147  -1.275  0.20318
datum1        4.2228    10.0895   0.419  0.67587
datum2        5.9339    10.1072   0.587  0.55760
stadt2      -44.2046    10.0508  -4.398 1.54e-05 ***
stadt3        7.8379    10.1148   0.775  0.43904
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 70.43 on 287 degrees of freedom
Multiple R-squared:  0.1962,    Adjusted R-squared:  0.1737
F-statistic: 8.754 on 8 and 287 DF,  p-value: 1.042e-10
```

Zur Diagnose der Regression wurden Residuals vs. Fitted Values, Normal Q-Q Plot, Scale-Location Plot und Residuals vs. Leverage betrachtet:



Residuals vs. Fitted zeigt einen Scatterplot zwischen Residuen und Predicted Values. Dieser sollte möglichst „zufällig“ aussehen. Potenziell kann man in einem solchen Plot erkennen, ob ein nicht-linearer Zusammenhang besteht.

Im vorliegenden Fall sieht der Plot „zufällig“ aus und es ist kein nicht linearer ersichtlich. Die Wahl der linearen Modellierung ist also in Ordnung.
Der Normal Q-Q Plot dient zu Analyse, ob die Fehler normalverteilt sind. Wenn die Punkte auf der eingezeichneten Gerade liegen, sind die Fehler normalverteilt. Falls nicht, ist die Annahme der normalverteilten Fehler nicht erfüllt.

Im vorliegenden Fall liegen die liegen die Punkte in der Mitte direkt auf der Gerade, aber an den Enden weichen sie stärker davon ab. Die Normalverteilungsannahme der Fehler ist daher nicht voll erfüllt.

Der Scale-Location Plot sollte möglichst „zufällig“ ausschauen und keine Muster aufweisen. Damit kann man in etwa die Annahme der Homoskedastizität (Varianzhomogenität bzgl.

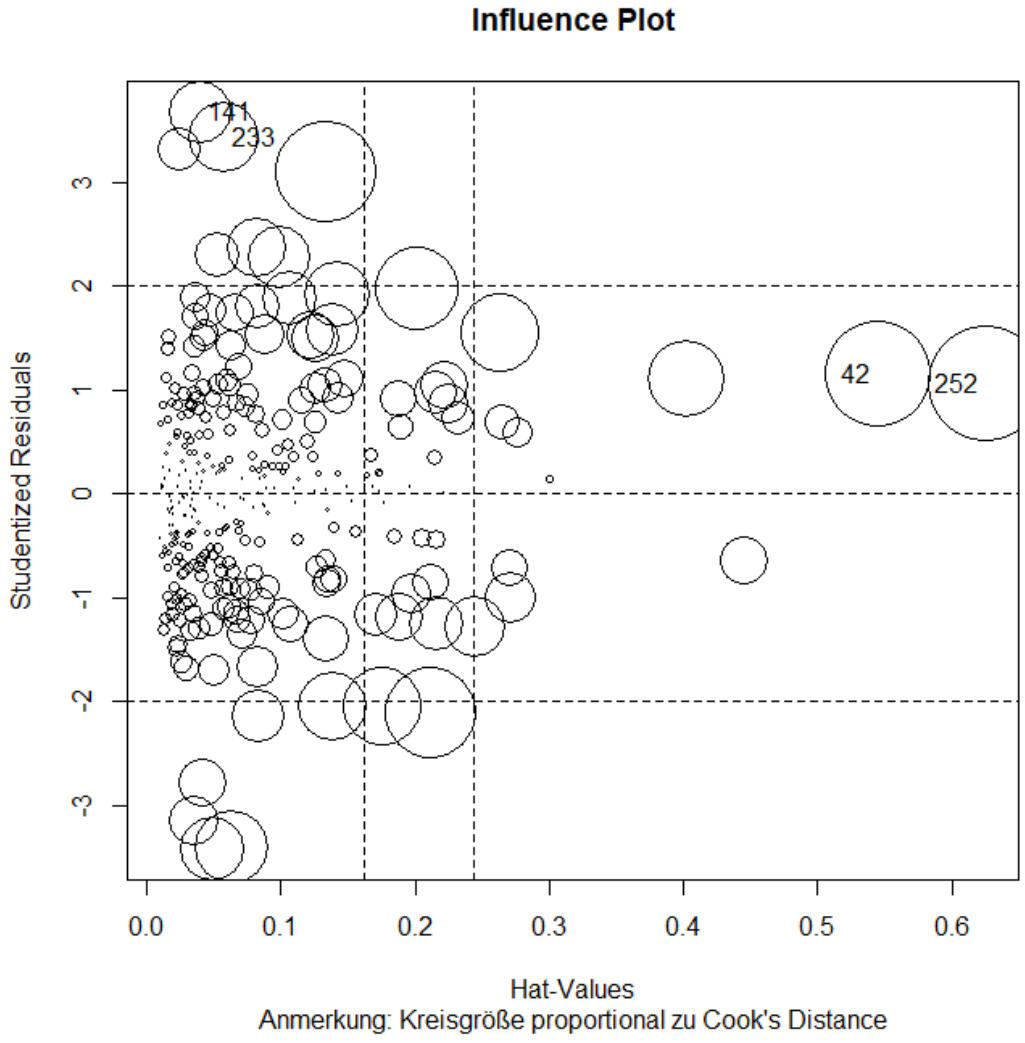
Fehlertermen) beurteilen. Die Annahme ist weitgehend erfüllt, wenn der Plot eine annähernd horizontale Linie und „zufällig“ verstreute Punkte aufweist.

Der Plot sieht im obigen Modell „zufällig“ aus und es gibt keine starken einseitigen Abweichungen und Muster im Plot.

Der Residuals vs. Leverage Plot hilft, “einflussreiche Outlier“ (leverage points) zu identifizieren. Diese liegen außerhalb der Cook’s Distanz und haben Einfluss auf die Regressionsergebnisse.

Hier sind keine Punkte außerhalb der Cooks Distance Linie ersichtlich. Daher kann davon ausgehen, dass keine starke Beeinträchtigung der Regression durch einzelne influence points stattfindet. Punkt 141 und die umliegenden Punkte sollte jedoch genauer angesehen werden.

Untenstehend wird der Influence Plot gezeigt, der den Einfluss der Beobachtungen veranschaulicht. Die Beobachtungen 141, 233, 42 und 252 scheinen unter anderem bezüglich Influence größere Bedeutung zu haben.



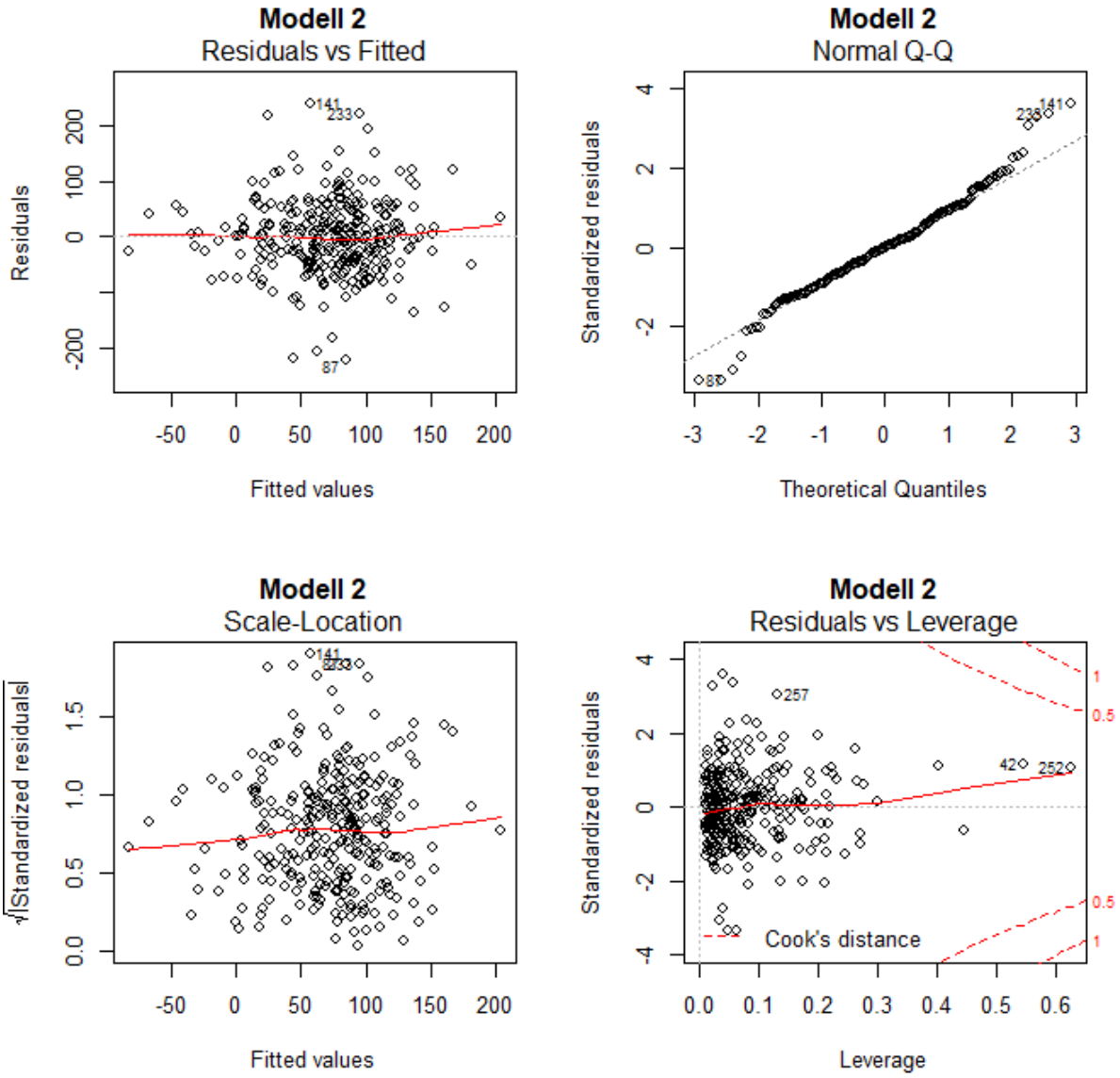
Anmerkung: Kreisgröße proportional zu Cook's Distance

Lineares Modell – Versuch 2 (Zusammenhang mit Interaktion):

```
# VERSUCH 2: lineares modell (mit interaktion, R-squared verbessert sich)
limod_results2 <- lm(f_sum56 ~ f1value*f2value*f3value*stadt, data = df_features_inklSum56)
summary(limod_results2)

# diagnose der regressionsresultate
x11()
par(mfrow = c(2,2))
plot(limod_results2, main="Modell 2")

  # influenceplot dafuer
  par(mfrow = c(1,1))
  influence.measures(limod_results2)
  influencePlot(limod_results2, main="Influence Plot",
                sub="Anmerkung: Kreisgröße proportional zu Cook's Distance" )
  # outliertest
  outlierTest(limod_results2)

  # plot dataframe
  plot(df_features_inklSum56)
```

Im Versuch 2 wird ein lineares Modell mit Interaktion ausgewertet. Der Wert von adjusted R-Squared verbessert sich auf 22.59%. Die Interaktion von f1value:f2value:f3value und von f1value:f3value sind signifikant. Leicht signifikant sind auch die Interaktion von f2value:stadt3, f1value:f3value:stadt2, f1value:f3value:stadt3 und f1value:f2value:f3value:stadt3.

```
Call:
lm(formula = f_sum56 ~ f1value * f2value * f3value * stadt, data = df_features_inklSum56)

Residuals:
     Min       1Q   Median       3Q      Max
-223.018  -40.378   -1.227   38.826  240.527

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                 87.096633  12.363158   7.045 1.52e-11 ***
f1value                      1.372282   1.198342   1.145 0.253153
f2value                     -0.157455   0.756011  -0.208 0.835174
f3value                     -0.269381   1.147104  -0.235 0.814513
stadt2                     -59.516704  16.689210  -3.566 0.000428 ***
stadt3                     -13.736153  15.945988  -0.861 0.389767
f1value:f2value             -0.003137   0.080496  -0.039 0.968943
f1value:f3value              0.356546   0.114835   3.105 0.002105 **
f2value:f3value              0.067021   0.076992   0.870 0.384796
f1value:stadt2               0.762650   1.853618   0.411 0.681075
f1value:stadt3               0.656512   1.569887   0.418 0.676138
f2value:stadt2               0.021524   1.138127   0.019 0.984926
f2value:stadt3               2.413214   1.136420   2.124 0.034613 *
f3value:stadt2               2.267038   1.576735   1.438 0.151639
f3value:stadt3              -0.665050   1.507967  -0.441 0.659546
f1value:f2value:f3value     -0.028925   0.008130  -3.558 0.000441 ***
f1value:f2value:stadt2       0.086354   0.135416   0.638 0.524212
f1value:f2value:stadt3      -0.001207   0.128618  -0.009 0.992522
f1value:f3value:stadt2      -0.302832   0.145289  -2.084 0.038062 *
f1value:f3value:stadt3      -0.336126   0.141638  -2.373 0.018333 *
f2value:f3value:stadt2       0.010639   0.115470   0.092 0.926658
f2value:f3value:stadt3       0.012753   0.112533   0.113 0.909857
f1value:f2value:f3value:stadt2  0.012856  0.011438  1.124 0.262033
f1value:f2value:f3value:stadt3  0.026636  0.011931  2.233 0.026392 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 68.18 on 272 degrees of freedom
Multiple R-squared:  0.2862,    Adjusted R-squared:  0.2259
F-statistic: 4.743 on 23 and 272 DF,  p-value: 9.604e-11
```



Hinsichtlich Residuals vs. Fitted Values ist im Vergleich zum ersten Modell eine stärkere Dichte der Punkte in der Mitte zu verzeichnen. Ein nichtlinearer Zusammenhang ist nicht direkt ersichtlich.

Im Normal-Q-Q-Plot liegen die Punkte zwischen -2 und 2 annähernd auf der Gerade, aber außerhalb dieses Intervalls weichen sie stark ab von der Gerade. Das lässt darauf schließen, dass die Fehler nicht normalverteilt sind.

Der Scale-Location-Plot weist keine Muster auf.

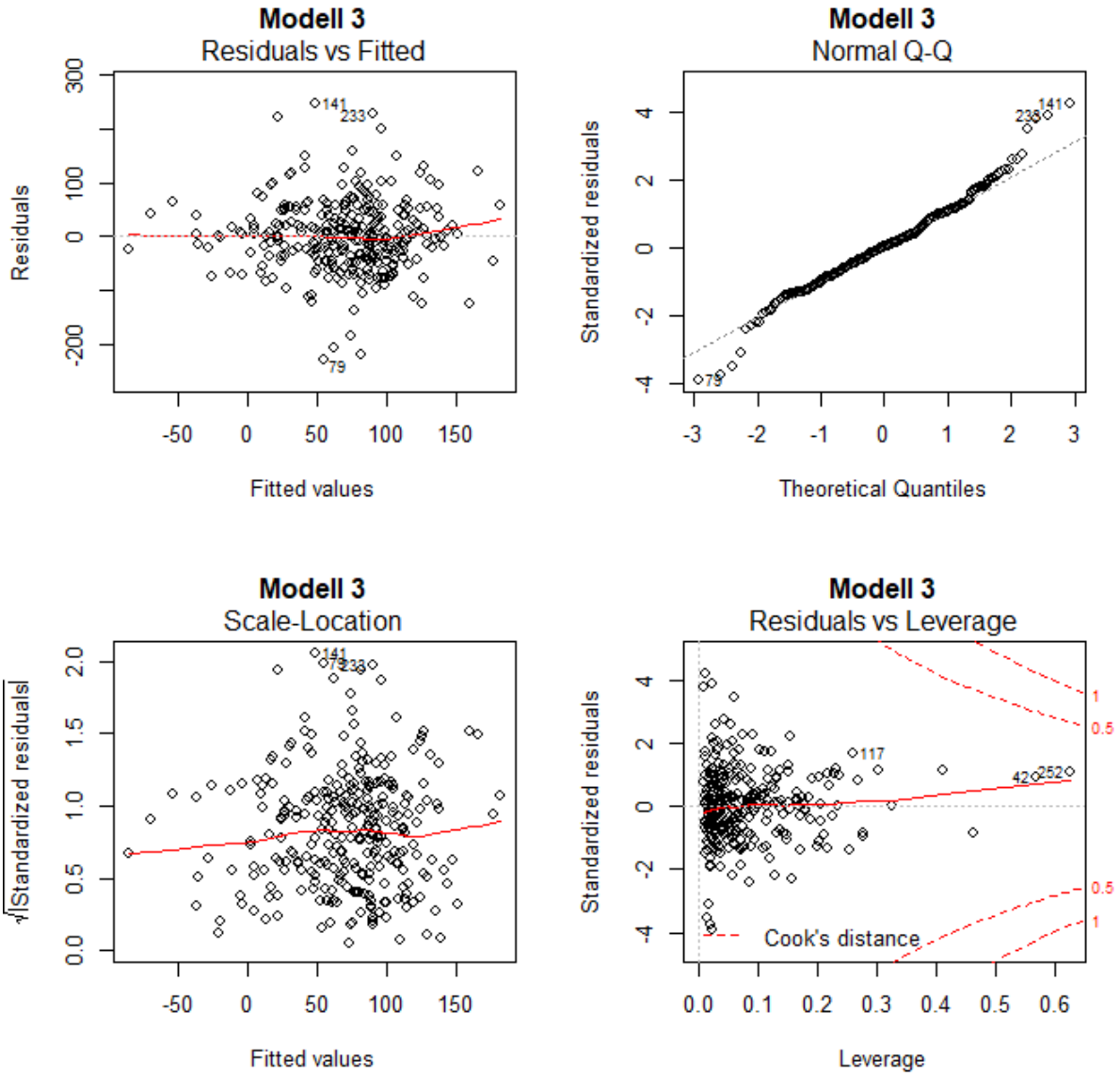
Der Residuals vs. Leverage Plot zeigt eine starke Dichte der Punkte am linken mittigen Rand. Die Punkte 42 und 252 liegen nahe an der Cooks Distance Linie. Daher sind sie als einflussreichere unter den Punkten des Plots zu beurteilen.

Lineares Modell – Versuch 3 (robustes Modell nach Huber-Ansatz):

```
# VERSUCH 3: robuste regression (mit huber estimator; residual standard error reduziert vs. versuch 2)
#            anmerkung:
#            mittels robuster regression wird darauf geachtet, dass die einzelne beobachtung nicht so
#            stark ins gewicht faellt, d.h. es werden "iterated re-weighted least squares" eingesetzt.
#            die default-belegung ist die gewichtung nach huber. bringt gut resultate, falls es einfluss-
#            reiche outlier gibt.
limod_results3 <- rlm(f_sum56 ~ f1value*f2value*f3value*stadt, data = df_features_inklSum56)
summary(limod_results3)

# diagnose der regressionsresultate
x11()
par(mfrow = c(2,2))
plot(limod_results3,main="Modell 3")
# anmerkung zu den plots:
# 1. Residuals vs. Fitted
# 2. Normal Q-Q
# 3. Scale-Location
# 4. Residuals vs. Leverage

# influenceplot dafuer
par(mfrow = c(1,1))
influence.measures(limod_results3)
influencePlot(limod_results3, main="Influence Plot",
              sub="Anmerkung: Kreisgröße proportional zu Cook's Distance" )
# outliertest
outlierTest(limod_results3)
```

Für das robuste lineare Modell wurde die Funktion „rlm“ verwendet, die als default-Belegung den Huber-Ansatz wählt. Der im Versuch 3 errechnete Residual standard error ist im Vergleich zu den beiden vorhergehenden Modellen geringer. Ein robustes Modell, wie z.B. nach dem Ansatz von Huber misst einzelnen Punkte nicht so große Bedeutung bei, sodass (einflussreiche) Outlier nicht so stark ins Gewicht fallen.

```
Call: rlm(formula = f_sum56 ~ f1value * f2value * f3value * stadt,
    data = df_features_inklSum56)
Residuals:
      Min        1Q    Median        3Q       Max
-229.3935  -37.9687    0.6619   40.4146  248.0411

Coefficients:
                            Value    Std. Error t value
(Intercept)                  88.7482  11.0882     8.0039
f1value                       1.1140   1.0748     1.0365
f2value                      -0.3851   0.6780    -0.5680
f3value                      -0.0591   1.0288    -0.0575
stadt2                      -63.4468  14.9681    -4.2388
stadt3                      -15.4888  14.3015    -1.0830
f1value:f2value               0.0075   0.0722     0.1034
f1value:f3value               0.3226   0.1030     3.1320
f2value:f3value               0.0526   0.0691     0.7616
f1value:stadt2                1.3886   1.6625     0.8353
f1value:stadt3                1.3305   1.4080     0.9450
f2value:stadt2                0.2221   1.0208     0.2176
f2value:stadt3                2.5000   1.0192     2.4528
f3value:stadt2                2.1783   1.4141     1.5404
f3value:stadt3               -0.8221   1.3525    -0.6078
f1value:f2value:f3value      -0.0256   0.0073    -3.5139
f1value:f2value:stadt2        0.0623   0.1215     0.5131
f1value:f2value:stadt3       -0.0518   0.1154    -0.4494
f1value:f3value:stadt2       -0.2799   0.1303    -2.1480
f1value:f3value:stadt3       -0.3216   0.1270    -2.5314
f2value:f3value:stadt2        0.0061   0.1036     0.0590
f2value:f3value:stadt3        0.0305   0.1009     0.3018
f1value:f2value:f3value:stadt2   0.0109   0.0103     1.0674
f1value:f2value:f3value:stadt3   0.0258   0.0107     2.4126

Residual standard error: 59.13 on 272 degrees of freedom
```



Hinsichtlich Residuals vs. Fitted Values ist im Vergleich zum ersten Modell eine stärkere Dichte der Punkte in der Mitte zu verzeichnen. Ein nichtlinearer Zusammenhang ist nicht direkt ersichtlich.

Im Normal-Q-Q-Plot liegen die Punkte zwischen -2 und 2 annähernd auf der Gerade, aber außerhalb dieses Intervalls weichen sie stark ab von der Gerade. Das lässt darauf schließen, dass die Fehler nicht normalverteilt sind.

Der Scale-Location-Plot weist keine Muster auf.

Der Residuals vs. Leverage Plot zeigt eine starke Dichte der Punkte am linken mittigen Rand. Die Punkte 42 und 252 liegen nahe an der Cooks Distance Linie. Daher sind sie als einflussreichere unter den Punkten des Plots zu beurteilen.

Vergleich verschiedener linearer Modelle:

```
# VERGLEICHE VERSCHIEDENE MODELLE
# anmerkung: untersucht, ob zunahme je einer weiteren variable signifikant an information beitragen,
# nachdem die vorherigen variablen bereits beruecksichtigt wurden im modell
anova_res <- anova( lm(f_sum56 ~ f1value, data = df_features_inklSum56),
                    lm(f_sum56 ~ f1value+f2value, data = df_features_inklSum56),
                    lm(f_sum56 ~ f1value+f2value+f3value, data = df_features_inklSum56),
                    lm(f_sum56 ~ f1value+f2value+f3value+f4value, data = df_features_inklSum56),
                    lm(f_sum56 ~ f1value+f2value+f3value+f4value+datum, data = df_features_inklSum56),
                    lm(f_sum56 ~ f1value+f2value+f3value+f4value+datum+stadt, data = df_features_inklSum56)
                  )
anova_res
summary(anova_res)
```

Mittels der Funktion „anova“ wird die Varianzanalyse-Tabelle für 6 Modelle erstellt. Es wird untersucht, ob die Zunahme je einer weiteren Variable signifikant an Information beiträgt, nachdem die vorherigen Modellvariablen bereits berücksichtigt wurden. Der Ergebnis der ANOVA ist, dass die Variablen f1value, f2value und f3value jeweils neue Informationen bringen und im Modell sinnvoll sind. Die Variablen f4value und datum tragen nicht signifikant zu einer Verbesserung der Modellerklärung bei, wenn f1value, f2value und f3value bereits berücksichtigt wurden. Die Variable stadt bring zusätzlich noch Informationen und ist im Modell sinnvoll.

```
> anova_res
Analysis of Variance Table

Model 1: f_sum56 ~ f1value
Model 2: f_sum56 ~ f1value + f2value
Model 3: f_sum56 ~ f1value + f2value + f3value
Model 4: f_sum56 ~ f1value + f2value + f3value + f4value
Model 5: f_sum56 ~ f1value + f2value + f3value + f4value + datum
Model 6: f_sum56 ~ f1value + f2value + f3value + f4value + datum + stadt
  Res.Df     RSS Df Sum of Sq       F    Pr(>F)
1    294 1683612
2    293 1636380  1     47232  9.5208  0.002230 **
3    292 1588748  1     47633  9.6016  0.002137 **
4    291 1580068  1      8680  1.7497  0.186973
5    289 1578103  2      1965  0.1980  0.820461
6    287 1423785  2    154318 15.5533 3.862e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Versuch der automatischen Modellauswahl mit step():

```
step_result <- step(lm(f_sum56 ~ f1value+f2value+f3value+f4value+datum+stadt, data = df_features_inklSum56))
step_result
summary(step_result)
# interpretation:
# empfehlung, die auch durch anova betrachtung vorgeschlagen wurde, wird auch durch step ermittelt
```

Mittels der step-Funktion wird ein Versuch gemacht, ein möglichst gutes Modell zu wählen nach dem AIC-Kriterium, d.h. einer Kennzahl, die die Modellqualität und Anzahl der notwendigen Schätzer berücksichtig. Es wird hier auch Acht gegeben, dass nicht zu viele Variablen in das Modell genommen werden. Man startet mit dem vollen Modell und es werden diejenigen Variablen mit nicht-signifikanten Beiträgen aussortiert. Im vorliegenden Beispiel (für das lineare Modell mit additiven Zusammenhängen) werden also f1value, f2value, f3value und stadt2 für das Modell genommen. Die anderen Variablen werden aussortiert.

```
> summary(step_result)

Call:
lm(formula = f_sum56 ~ f1value + f2value + f3value + stadt, data = df_features_inklSum56)

Residuals:
     Min       1Q   Median       3Q      Max
-254.592  -42.116   -3.659   37.471  255.689

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  66.5640     8.7812   7.580 4.68e-13 ***
f1value       1.7883     0.3886   4.602 6.27e-06 ***
f2value       1.2329     0.4026   3.062   0.0024 **
f3value       1.1926     0.3998   2.983   0.0031 **
stadt2      -44.0718    10.0309  -4.394 1.57e-05 ***
stadt3        7.9992    10.0988   0.792   0.4290
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 70.33 on 290 degrees of freedom
Multiple R-squared:  0.1901,    Adjusted R-squared:  0.1761
F-statistic: 13.61 on 5 and 290 DF,  p-value: 6.079e-12
```