Aufgabe 1 & 2 in Python

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). Hinweis: verwenden Sie die subplots Funktion aus der matplotlib Library, Beispiel: https://matplotlib.org/examples/pylab_examples/subplots_demo.html

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 (z.B. unter Verwendung der matshow Funktion aus der matplotlib Bibliothek).

d. Dokumentieren Sie die notwendigen Schritte zur Datenaufbereitung.

In dieser Dokumentation wird der lauffähige Python-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 Python-File, das ebenfalls auf Moodle hochgeladen wurde.

Im Folgenden wir der Python-Code erklärt und dokumentiert:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
import statsmodels.formula.api as smf
import statsmodels.api as sm
from patsy.contrasts import Treatment


# ------------------------------------------------------------------------
# DATEN AUS CSV-FILES EINLESEN & DATAFRAMES ANLEGEN
# ------------------------------------------------------------------------
# adressen (spalten: PLZ, stadt)
adressen = pd.read_csv("C:/Users/cordu/Desktop/UE4/Adresse.csv", sep=";", decimal=",")
adressen_df = pd.DataFrame(adressen)

# benutzer (spalten: UID, PLZ)
benutzer = pd.read_csv("C:/Users/cordu/Desktop/UE4/Benutzer.csv", sep=";", decimal=",")
benutzer_df = pd.DataFrame(benutzer)

# features (spalten: UID, datum, variable, value)
features = pd.read_csv("C:/Users/cordu/Desktop/UE4/Features.csv", sep=";", decimal=",")
features_df = pd.DataFrame(features)
```

Zunächst werden die notwendigen Libraries (*pandas*, *numpy*, *matplotlib*, *statsmodels*) importiert. Dann werden die Daten aus den 3 csv-Files mittels der pd.read_csv-Funktion von *pandas* eingelesen. Die eingelesenen Daten werden als Dataframe gespeichert.

```
# ------------------------------------------------------------------------
# DATEN ZUSAMMENFUEHREN
# ------------------------------------------------------------------------

# Join von benutzer und adressen auf "PLZ"
joined_benutzer_adressen = pd.merge(benutzer, adressen, how="inner",
                                    on= "PLZ")

# Join von joined_benutzer_adressen mit features auf "UID"
joined_all = pd.merge(joined_benutzer_adressen, features, how="inner",
                                    on= "UID")

# CHECK DATA TYPES OF DATAFRAME (DF)
joined_all.dtypes

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

Dann werden die Daten der 3 obigen Dataframes „benutzer_df“, „features_df“ und „adressen_df“ zusammengeführt mit Hilfe von pd.merge aus pandas, sodass sich das Dataframe „joined_all“ ergibt, das folgendermaßen aussieht:

```
In [451]: print(joined_all)
      UID   PLZ Stadt            Datum   variable       value
0      10  1110  Wien  20180416T00:00Z  Feature 6   41.769487
1      10  1110  Wien  20160813T00:00Z  Feature 3    7.846988
2      10  1110  Wien  20180416T00:00Z  Feature 2    3.554936
3      10  1110  Wien  20180416T00:00Z  Feature 5   44.804247
4      10  1110  Wien  20160813T00:00Z  Feature 2  -10.998649
5      10  1110  Wien  20180416T00:00Z  Feature 3   -2.878346
6      10  1110  Wien  20160813T00:00Z  Feature 4    6.061614
7      10  1110  Wien  20180416T00:00Z  Feature 4   -3.580512
8      10  1110  Wien  20160813T00:00Z  Feature 5  -54.428479
9      10  1110  Wien  20160813T00:00Z  Feature 1  -19.682142
10     10  1110  Wien  20160813T00:00Z  Feature 6   -5.413186
```

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

 # DATEN IN FEATURE 1 BIS 6 TRENNEN und anzahl der beobachtungen ueberpruefen

daten_feature_1 = joined_all[ joined_all["variable"] == "Feature 1"]
daten_feature_1.UID.count() # 299 beobachtungen (bzw. zeilen)

daten_feature_2 = joined_all[ joined_all["variable"] == "Feature 2"]
daten_feature_2.UID.count() # 300 beobachtungen (bzw. zeilen)

daten_feature_3 = joined_all[ joined_all["variable"] == "Feature 3"]
daten_feature_3.UID.count() # 299 beobachtungen (bzw. zeilen)

daten_feature_4 = joined_all[ joined_all["variable"] == "Feature 4"]
daten_feature_4.UID.count() # 300 beobachtungen (bzw. zeilen)

daten_feature_5 = joined_all[ joined_all["variable"] == "Feature 5"]
daten_feature_5.UID.count() # 300 beobachtungen (bzw. zeilen)

daten_feature_6 = joined_all[ joined_all["variable"] == "Feature 6"]
daten_feature_6.UID.count() # 300 beobachtungen (bzw. zeilen)

anzahl_UID = daten_feature_6.UID.count()
```

Nun werden die Daten anhand der Spalte „variable“ des Dataframe *joined_all* in die einzelnen Features (1-6) getrennt.

Wie schon aus den Kommentaren zu der Anzahl der Beobachtungen erkennbar, wurde hier festgestellt, dass scheinbar für jeden Benutzer (UID) 2 Beobachtungen pro Feature vorhanden sein sollte. Allerdings ist dies nicht der Fall, denn Feature 1 und Feature 3 haben jeweils nur 1 Beobachtung bei bestimmten UIDs. Diese UIDs gilt es zu ermitteln.

Der Umgang mit den fehlenden Werte kann entweder insofern erfolgen, dass man den „value“ beispielsweise durch den Median der Daten bzw. der Featuresdaten schätzt. Alle Daten außer dem „value“ können aus den restlichen Daten des betroffenen Benutzers übernommen werden. Andererseits kann man auch die betroffenen UIDs ganz weglassen aus allen Features, d.h. die Beobachtungen werden einfach aus dem Dataframe insgesamt herausgenommen.

```
# UNVOLLSTAENDIGE BEOBACHTUNGEN:
# finde UID, deren beobachtungen unvollstaendig sind
    # bezueglich feature 1
unvollstaendige_UID = "a"
for i in range(0,anzahl_UID):
    if( (daten_feature_1["UID"] == benutzer["UID"][i]).sum() < 2):
        unvollstaendige_UID = benutzer["UID"][i]
        break
# resultat: UID 30 ist unvollstaendig bzgl. feature 1,
#           weil es gibt nur 1 beobachtung davon
(daten_feature_1["UID"]==30).sum()

    # bezueglich feature 3
unvollstaendige_UID_f3 = "a"
for i in range(0,anzahl_UID):
    if( (daten_feature_3["UID"] == benutzer["UID"][i]).sum() < 2):
        unvollstaendige_UID_f3 = benutzer["UID"][i]
        break
# resultat: UID 130 ist unvollstaendig bzgl. feature 3,
#           weil es gibt nur 1 beobachtung davon
(daten_feature_3["UID"]==130).sum()


# LISTE VON DATEN_FEATURE DFs
liste_featuresdaten = [daten_feature_1, daten_feature_2, daten_feature_3,
                       daten_feature_4, daten_feature_5, daten_feature_6]
```

Mittels der oben angeführten Code-Zeilen können die UIDs der unvollständigen Beobachtungen ermittelt werden. Es kann festgestellt werden, dass dies UID 30 bei Feature 1 ist, und UID 130 bei Feature 3.

```
# UMGANG MIT UNVOLLSTAENDIGEN BEOBACHTUNGEN:
# zur sicherheit, dass keine datenverfaelschungen entstehen,
# werden die beobachtungen UID 30 und 130 weggelassen

# FUNKTION: entferne unvollstaendige beobachtungen
def entferneUnvollstaendigeBeobachtungen(liste_featuresdaten):
    """
    funktion entfernt unvollstaendige beobachtungen
    param: liste_featuresdaten
    returnwert: daten_feature_cleaned
    """

    for i in range(0, len(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


    return liste_featuresdaten


# aufruf der funktion entferneUnvollstaendigeBeobachtungen für die gesamtliste
    liste_featuresdaten = entferneUnvollstaendigeBeobachtungen(liste_featuresdaten)


# CHECK: ob alle listenelemente nun gleiche anzahl an beobachtungen haben
# (hier sollten es 296 beob. pro daten_feature nach entfernung der 2 unvollstaendigen UIDs sein)

for i in range(0,len(liste_featuresdaten)):
    print(liste_featuresdaten[i]["UID"].count())
    # ANMERKUNG: ja, alle haben nun 296 beobachtungen


# unvollstaendige beobachtungen aus joined_all entfernen (gesamtdaten ohne featurestrennung)
joined_all = joined_all[ joined_all["UID"] != unvollstaendige_UID ]
joined_all = joined_all[ joined_all["UID"] != unvollstaendige_UID_f3 ]

# CHECK, ob 296*6 elemente:
len(joined_all) == 296*6  # ja, ist korrekt
```

Die Funktion „*entferneUnvollstaendigeBeobachtungen*“ entfernt für eine übergebene Liste von Featuresdaten die unvollständigen Beobachtungen anhand der unvollständigen UID Informationen. Dann wird überprüft, ob tatsächlich nach Entfernen alle Features 296 Beobachtungen haben – dies wurde bestätigt.

```python
# AGGREGIERTE STATISTIKEN BERECHNEN
# aggregation definieren:
# Quelle: https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/
aggregierte_deskriptiv = {
    'value': { # spalte "value"
        'Mean': 'mean',
        'Median': 'median',
        'Standard Deviation': 'std', # sample standard deviation
        'Min': 'min',
        'Max': 'max',
        'Quantile25': lambda q: q.quantile([0.25]),
        'Quantile75': lambda q: q.quantile([0.75]),
    }
}


# DATEN GRUPPIEREN: (nach variable, Stadt und Datum in 6 gruppen pro feature)
    # GRUPPEN:
    # G1: Graz - 20160813T00:00Z
    # G2: Graz - 20180416T00:00Z
    # G3: Salzburg - 20160813T00:00Z
    # G4: Salzburg - 20180416T00:00Z
    # G5: Wien - 20160813T00:00Z
    # G6: Wien - 20180416T00:00Z
deskriptiveStatistik_aggregiert = joined_all.groupby(['variable', 'Stadt', 'Datum']).agg(aggregierte_deskriptiv)
print(deskriptiveStatistik_aggregiert)
```

Für die Berechnung der deskriptiven Statistiken (Mittelwert, Median, Standardabweichung, Minimum, Maximum, 25% Quantil und 75% Quantil) pro Gruppe wird zuerst die Variable „*aggregierte_deskriptiv*“ angelegt. Dann wird auf den mittels „*groupby*“ gruppierten Daten basierend auf „*joined_all*“ die aggregierte Berechnung der deskriptiven Statistiken durchgeführt.

Output der berechneten deskriptiven Statistiken pro Gruppe:

```
In [454]: deskriptiveStatistik_aggregiert
Out[454]:
                                      value
                                       Mean     Median Standard Deviation
variable  Stadt     Datum
Feature 1 Graz      20160813T00:00Z  -1.918812  -2.227810          11.231719
                    20180416T00:00Z   1.244824   0.628446          11.333374
          Salzburg  20160813T00:00Z   1.105566   0.950245          11.355149
                    20180416T00:00Z  -1.578734  -0.684724           7.851082
          Wien      20160813T00:00Z  -1.215893  -2.156783          11.526209
                    20180416T00:00Z  -0.811456  -1.201817           9.649476
Feature 2 Graz      20160813T00:00Z  12.757536  11.805260          11.197581
                    20180416T00:00Z  10.949135  11.333860           9.629095
          Salzburg  20160813T00:00Z   7.730169   5.280140           9.274533
                    20180416T00:00Z  12.006156  12.258393          10.481430
          Wien      20160813T00:00Z   8.422873  11.230767          10.947666
                    20180416T00:00Z   8.373801   8.682763           9.401324
Feature 3 Graz      20160813T00:00Z   6.288084   8.262350          10.860415
                    20180416T00:00Z   5.177158   3.982181          10.675237
          Salzburg  20160813T00:00Z   5.736515   6.623312           9.675450
                    20180416T00:00Z   4.662623   6.168371          11.286489
          Wien      20160813T00:00Z   5.786694   6.689257           7.699265
                    20180416T00:00Z   5.532993   7.243086          11.294941
Feature 4 Graz      20160813T00:00Z   9.857618   8.112268           9.231200
                    20180416T00:00Z  11.218291  11.841805           8.589490
          Salzburg  20160813T00:00Z  11.928082  12.191738          10.044963
                    20180416T00:00Z   8.836160  10.054064          11.518739
          Wien      20160813T00:00Z  12.901719  13.386031          10.709557
                    20180416T00:00Z   7.735960   8.148849           8.961904
Feature 5 Graz      20160813T00:00Z  77.724969  61.614767          59.430024
                    20180416T00:00Z  19.036694   9.162844          47.826942
          Salzburg  20160813T00:00Z  21.491261  12.163864          52.797477
                    20180416T00:00Z  19.149466  11.695338          42.289512
          Wien      20160813T00:00Z  39.706531  31.884432          39.776546
                    20180416T00:00Z  43.035827  27.076587          47.067071
Feature 6 Graz      20160813T00:00Z  64.987259  58.224097          48.592447
                    20180416T00:00Z  13.073950  17.678125          50.543056
          Salzburg  20160813T00:00Z  19.889293  12.813501          52.487582
                    20180416T00:00Z  20.341982  10.708293          48.417079
          Wien      20160813T00:00Z  48.321870  34.235214          48.854913
                    20180416T00:00Z  48.644033  41.769487          42.611750
```

variable	Stadt	Datum	Min	Max	Quantile25
Feature 1	Graz	20160813T00:00Z	-23.544733	21.223720	-9.888475
		20180416T00:00Z	-31.427334	21.423720	-3.296106
	Salzburg	20160813T00:00Z	-27.271964	19.688204	-5.468898
		20180416T00:00Z	-18.383603	14.464980	-7.137652
	Wien	20160813T00:00Z	-21.740285	28.603370	-8.295328
		20180416T00:00Z	-22.571353	22.139371	-5.301680
Feature 2	Graz	20160813T00:00Z	-8.994363	37.542909	5.090695
		20180416T00:00Z	-13.728317	35.429819	4.766020
	Salzburg	20160813T00:00Z	-10.649488	26.598943	1.367355
		20180416T00:00Z	-9.207346	34.695654	5.138037
	Wien	20160813T00:00Z	-25.441120	26.966376	1.590260
		20180416T00:00Z	-11.785145	30.596197	2.670948
Feature 3	Graz	20160813T00:00Z	-17.083691	29.008824	-2.001258
		20180416T00:00Z	-14.834836	26.623408	-2.278741
	Salzburg	20160813T00:00Z	-23.488626	29.466789	1.101502
		20180416T00:00Z	-22.447385	29.249193	-3.324144
	Wien	20160813T00:00Z	-16.769324	20.473976	-0.733153
		20180416T00:00Z	-20.973053	26.914239	-3.497183
Feature 4	Graz	20160813T00:00Z	-4.644773	30.122488	3.165951
		20180416T00:00Z	-7.414985	28.734182	7.168379
	Salzburg	20160813T00:00Z	-12.756224	33.282090	7.592027
		20180416T00:00Z	-17.150088	28.313612	-0.918773
	Wien	20160813T00:00Z	-11.830193	40.028853	6.061614
		20180416T00:00Z	-10.764621	29.078502	2.372314
Feature 5	Graz	20160813T00:00Z	-3.457802	247.792345	36.549782
		20180416T00:00Z	-82.810066	168.249879	-3.185744
	Salzburg	20160813T00:00Z	-72.724722	159.169014	-3.106450
		20180416T00:00Z	-63.172848	146.358611	-5.070312
	Wien	20160813T00:00Z	-54.428479	173.859125	12.381450
		20180416T00:00Z	-13.095907	211.745919	13.334965
Feature 6	Graz	20160813T00:00Z	-7.225858	182.306703	21.683160
		20180416T00:00Z	-159.836139	112.454572	-6.996061
	Salzburg	20160813T00:00Z	-96.106057	204.144890	-0.347217
		20180416T00:00Z	-101.775628	194.395596	-1.637034
	Wien	20160813T00:00Z	-40.792507	215.961778	15.107766
		20180416T00:00Z	-6.605393	166.978449	15.880123

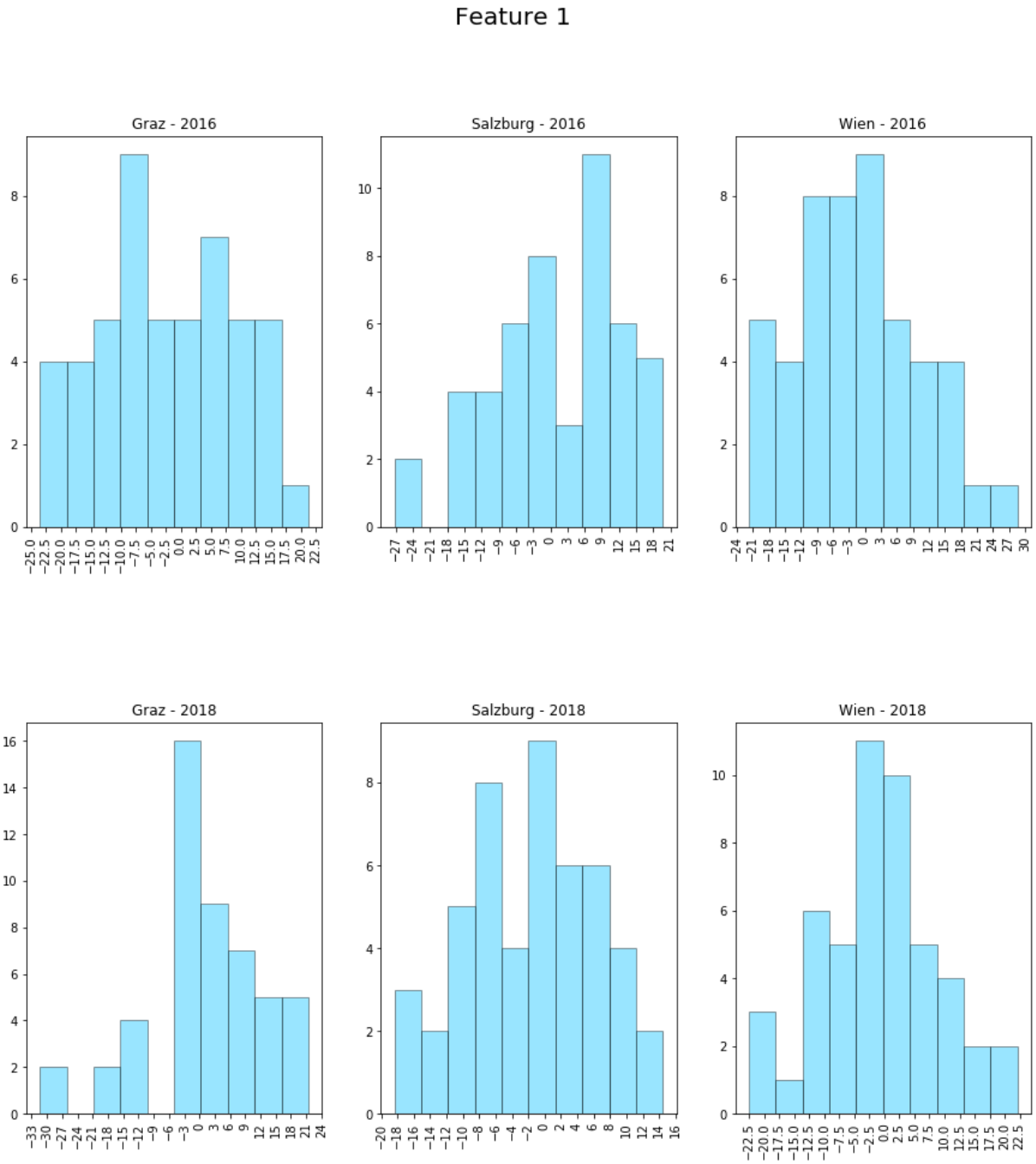
variable	Stadt	Datum	Quantile75
Feature 1	Graz	20160813T00:00Z	7.433256
		20180416T00:00Z	8.361234
	Salzburg	20160813T00:00Z	8.075091
		20180416T00:00Z	4.198059
	Wien	20160813T00:00Z	5.570597
		20180416T00:00Z	4.316624
Feature 2	Graz	20160813T00:00Z	20.832104
		20180416T00:00Z	16.385222
	Salzburg	20160813T00:00Z	16.253165
		20180416T00:00Z	17.883799
	Wien	20160813T00:00Z	16.487270
		20180416T00:00Z	11.824135
Feature 3	Graz	20160813T00:00Z	12.820326
		20180416T00:00Z	11.301063
	Salzburg	20160813T00:00Z	12.174305
		20180416T00:00Z	11.379790
	Wien	20160813T00:00Z	11.771451
		20180416T00:00Z	14.178119
Feature 4	Graz	20160813T00:00Z	15.742874
		20180416T00:00Z	15.334297
	Salzburg	20160813T00:00Z	17.744320
		20180416T00:00Z	18.028042
	Wien	20160813T00:00Z	20.721639
		20180416T00:00Z	13.616628
Feature 5	Graz	20160813T00:00Z	102.651026
		20180416T00:00Z	41.823283
	Salzburg	20160813T00:00Z	49.507287
		20180416T00:00Z	41.003190
	Wien	20160813T00:00Z	55.437084
		20180416T00:00Z	63.629411
Feature 6	Graz	20160813T00:00Z	97.584946
		20180416T00:00Z	34.493382
	Salzburg	20160813T00:00Z	50.571077
		20180416T00:00Z	36.566994
	Wien	20160813T00:00Z	73.615271
		20180416T00:00Z	67.942029

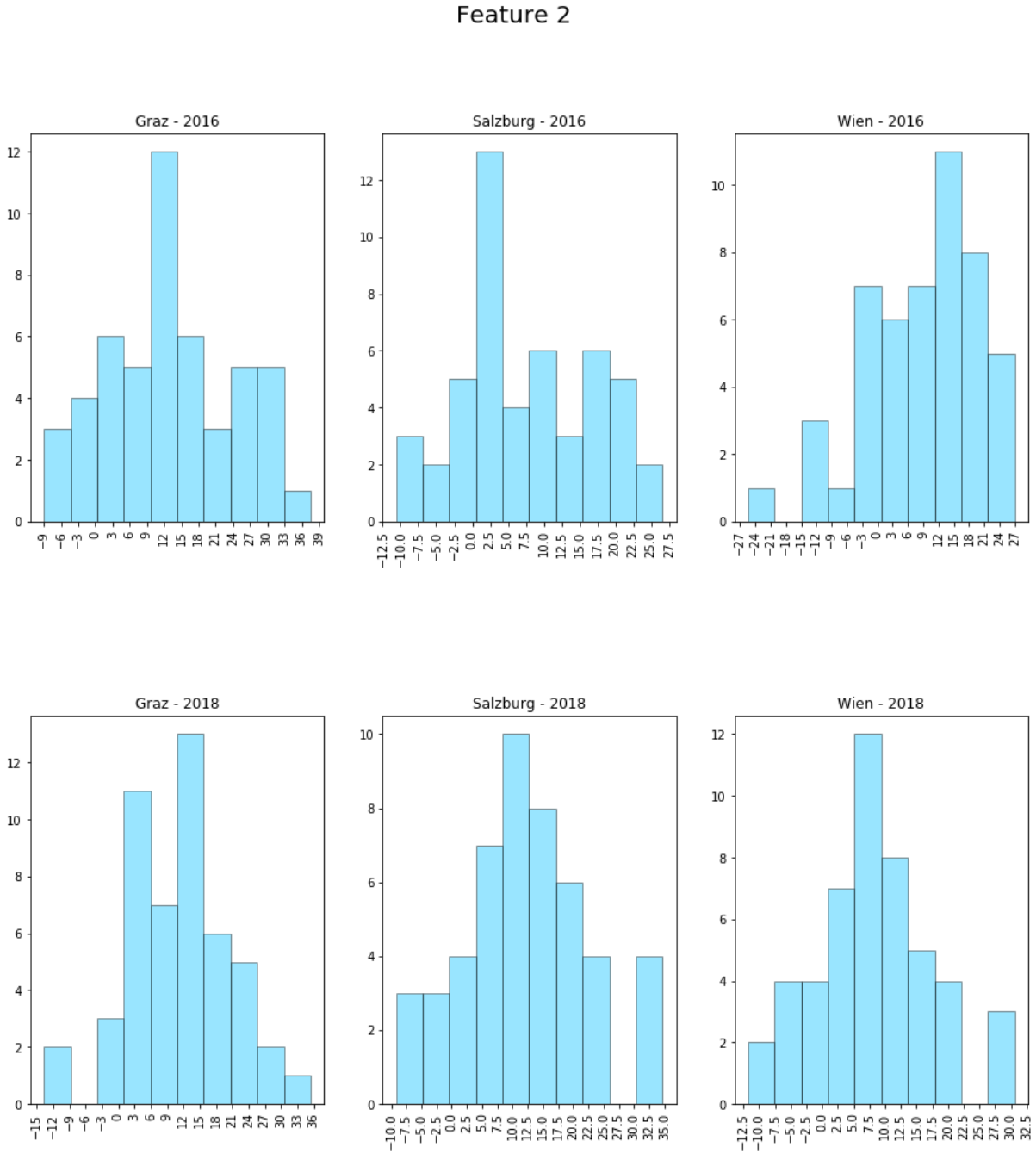
```
## ------------------------------------------------------------------------------
## VISUALISIERUNG (HISTOGRAMME FUER DIE 6 GRUPPEN)
## ------------------------------------------------------------------------------

# basisdaten (liste mit jeweiligen featuresdaten)
liste_featuresdaten

# FUNKTION: plot fuer alle gruppen eines features machen
def plotGroupsPerFeature(daten_feature, featurenumber):
    """
    funktion erstellt fuer das uebergebene daten_feature die gruppenplots
    param: daten_feature
    param: featurenumber
    """
    # gruppen bilden
    gruppe1 = daten_feature[ (daten_feature["Stadt"]=="Graz") & (daten_feature["Datum"]=="20180416T00:00Z") ]
    gruppe2 = daten_feature[ (daten_feature["Stadt"]=="Graz") & (daten_feature["Datum"]=="20160813T00:00Z") ]
    gruppe3 = daten_feature[ (daten_feature["Stadt"]=="Salzburg") & (daten_feature["Datum"]=="20180416T00:00Z") ]
    gruppe4 = daten_feature[ (daten_feature["Stadt"]=="Salzburg") & (daten_feature["Datum"]=="20160813T00:00Z") ]
    gruppe5 = daten_feature[ (daten_feature["Stadt"]=="Wien") & (daten_feature["Datum"]=="20180416T00:00Z") ]
    gruppe6 = daten_feature[ (daten_feature["Stadt"]=="Wien") & (daten_feature["Datum"]=="20160813T00:00Z") ]

    liste_fuerPlot = [gruppe2, gruppe4, gruppe6, gruppe1, gruppe3, gruppe5]
    gruppennamen_fuerPlot = ["Graz - 2016", "Salzburg - 2016", "Wien - 2016",
                             "Graz - 2018", "Salzburg - 2018", "Wien - 2018"]
    # reihenfolge der gruppen in plot: 2 4 6 1 3 5


    # subplots anfertigen
    # Quelle: https://jakevdp.github.io/PythonDataScienceHandbook/04.08-multiple-subplots.html
    fig=plt.figure(figsize=(15, 15))
    fig.suptitle("Feature " + str(featurenumber+1), fontsize=20)
    for i in range(1, 7):
        ax = fig.add_subplot(2, 3, i)
        plt.xticks(rotation=90)
        max_xticks = 20
        xloc = plt.MaxNLocator(max_xticks)
        ax.xaxis.set_major_locator(xloc)
        plt.hist(liste_fuerPlot[i-1]["value"], color = "deepskyblue", alpha=0.4, histtype='bar', ec='black')
        plt.subplot(2, 3, i).set_title(gruppennamen_fuerPlot[i-1])
        plt.subplots_adjust(hspace=0.5)

# CHECK: fuer daten_feature_2, ob funktion korrektes ergebnis liefert
plotGroupsPerFeature(daten_feature_4, 3)
```

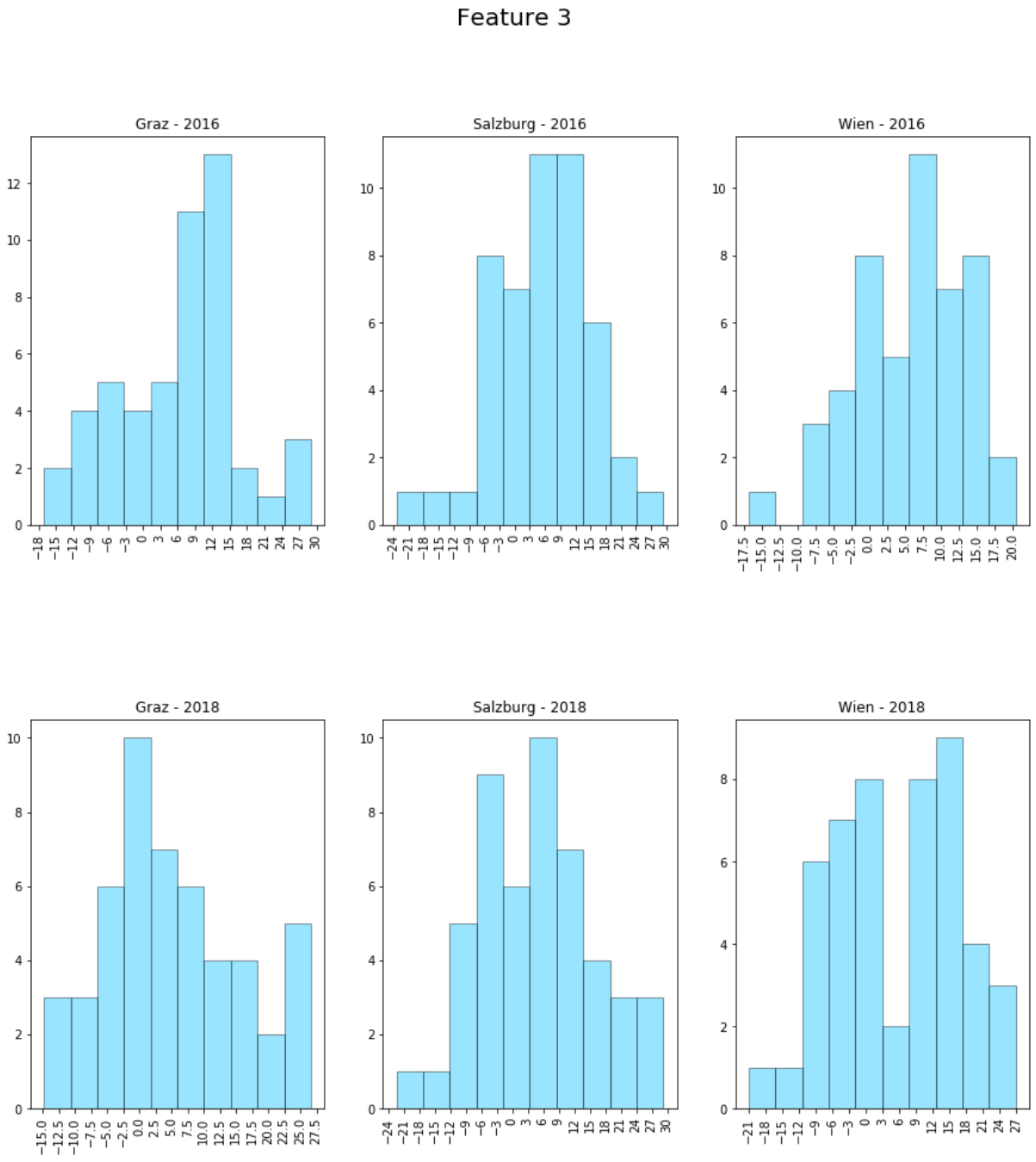
Zur Visualisierung der Histogramme pro Gruppe der jeweiligen Features dient die Funktion *plotGroupsPerFeature*, die „*daten_feature*“ und „featurenumber“ als Parameter nimmt. Die Funktion bildet die 6 Gruppe innerhalb des Features und erstellt mittels „*plt.figure*“ und „*subplot*“ die 6 Plots, eingeteilt in Stadtinformationen horizontal und Datumsinformationen vertikal.

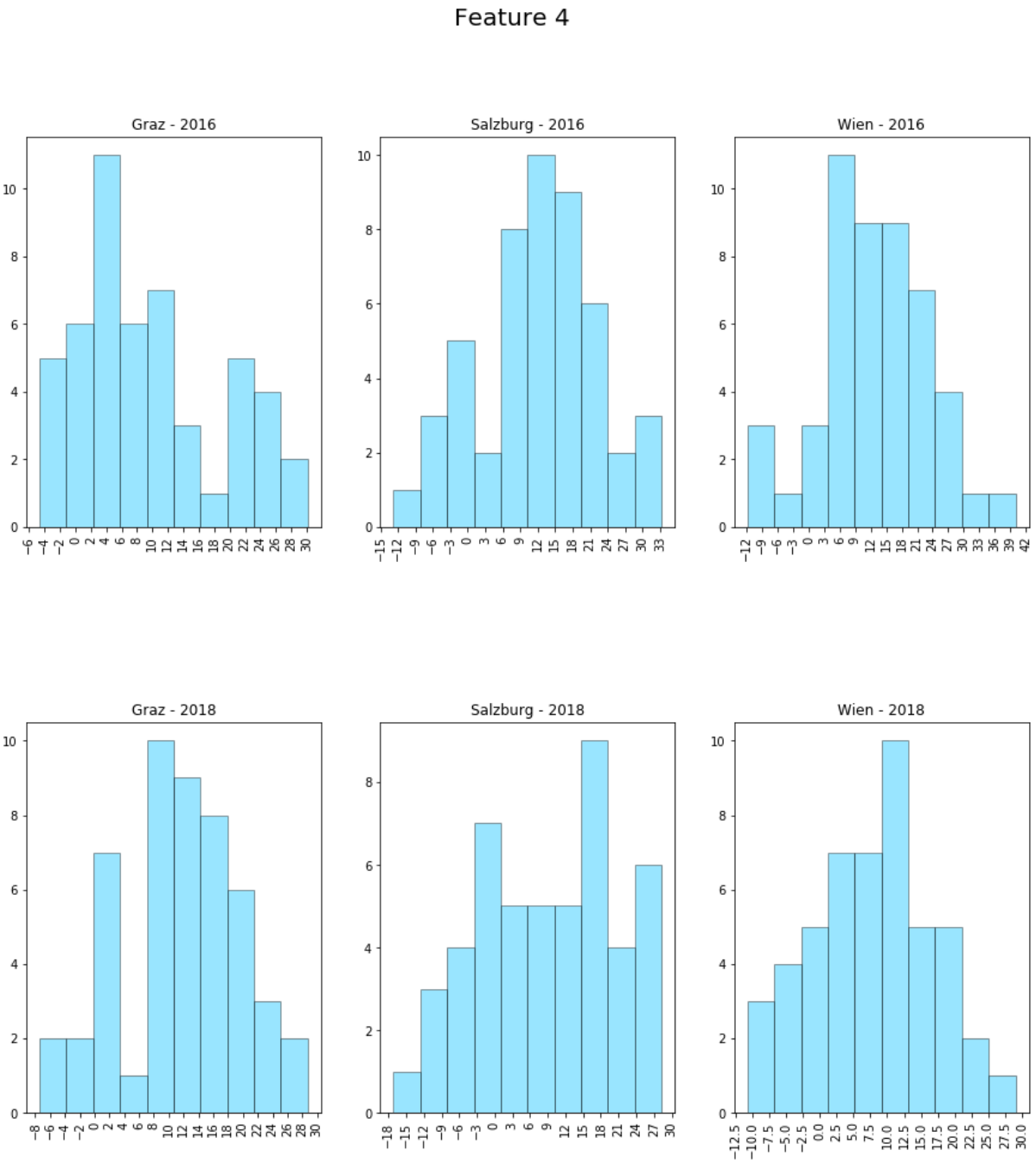
```
# FUNKTION plotGroupsPerFeature fuer alle features aufrufen
for i in range(0,len(liste_featuresdaten)):
    plotGroupsPerFeature(liste_featuresdaten[i], i)
```

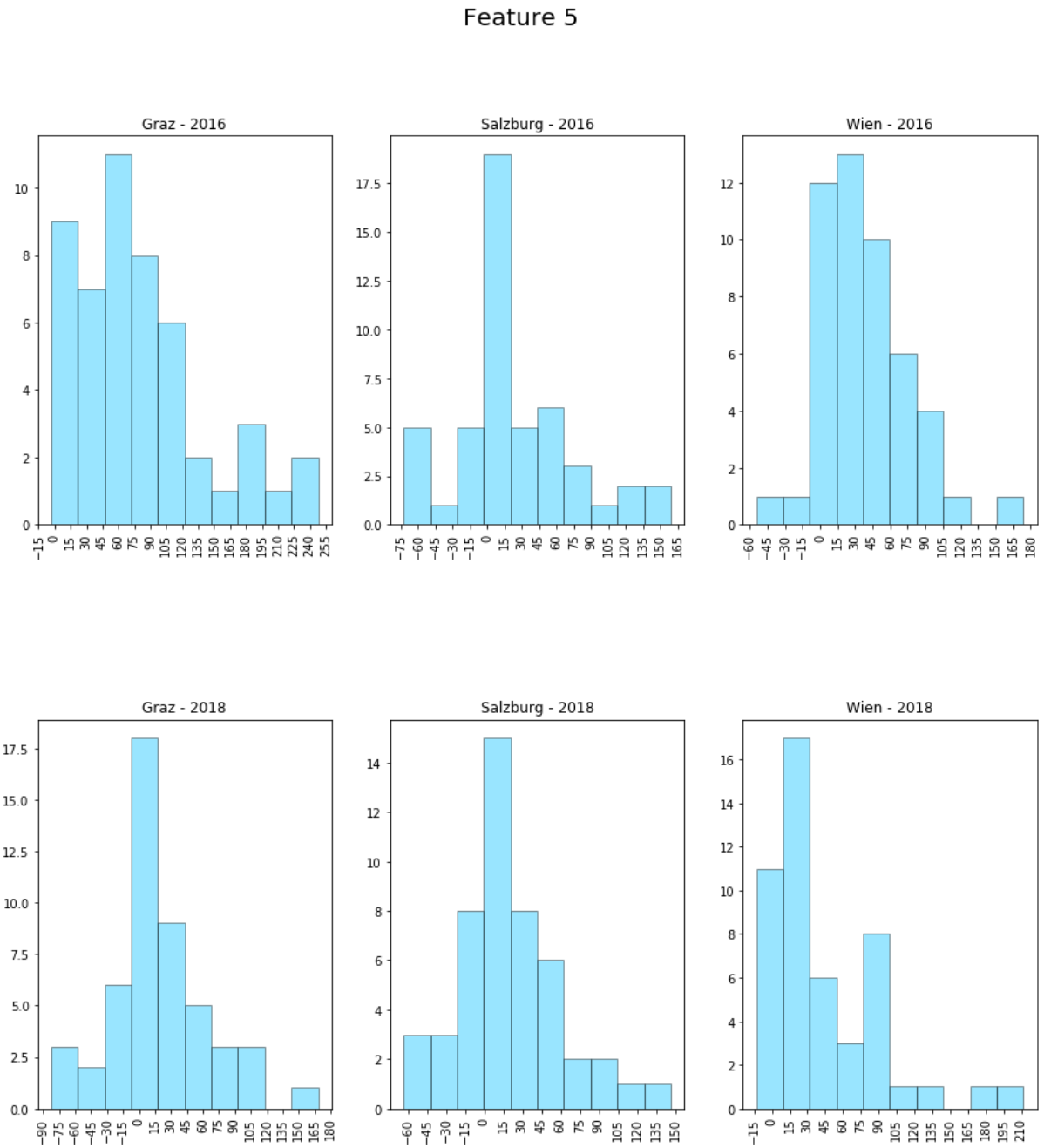
Die Funktion wird für die gesamte Liste an Featuresdaten aufgerufen, was folgende 6 Histogramm-Plots in der laut Angabe gewünschten Anordnung pro Feature generiert:

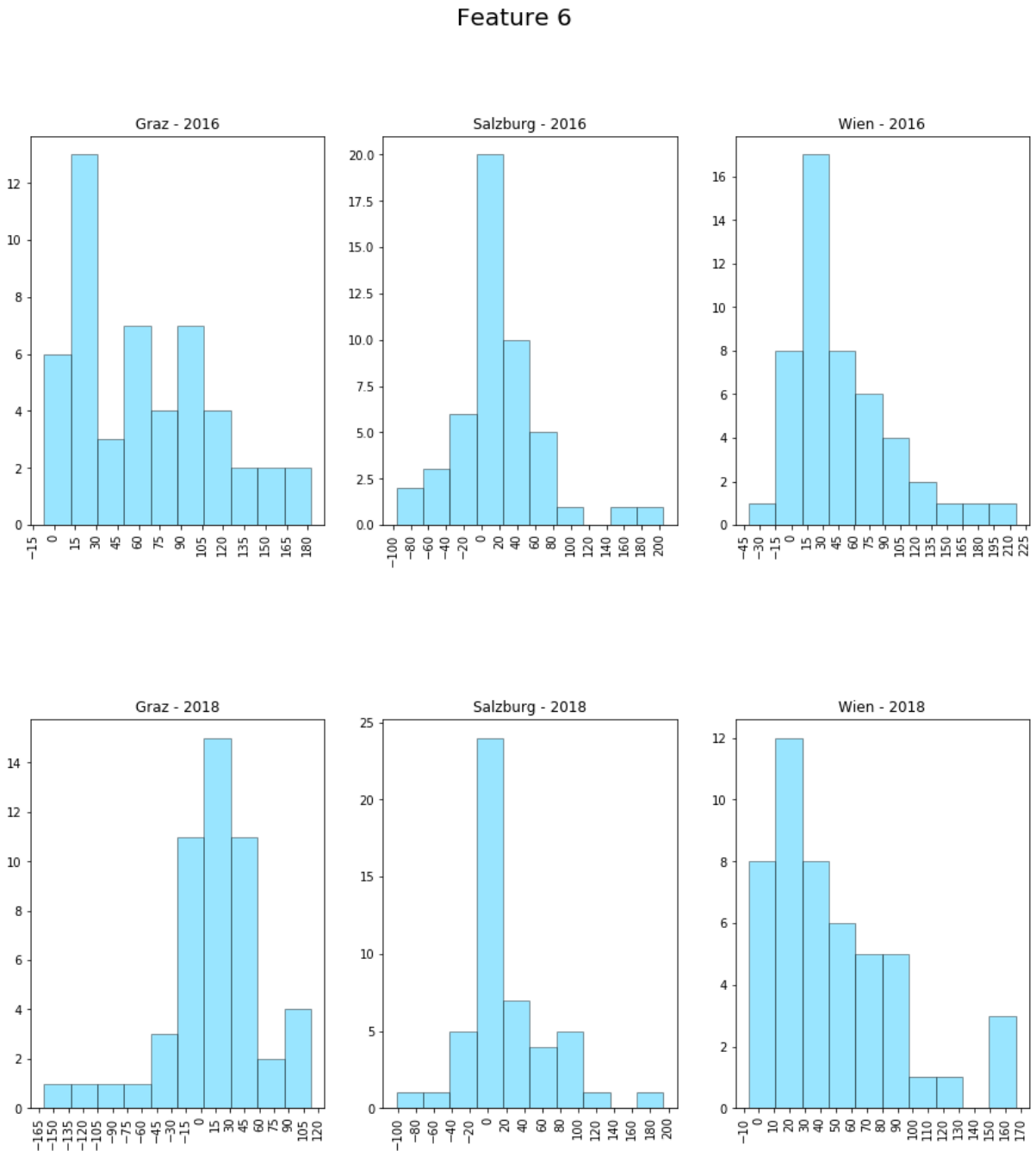

Feature 1
Graz - 2016
Salzburg - 2016
Wien - 2016
Graz - 2018
Salzburg - 2018
Wien - 2018


Feature 2
Graz - 2016
Salzburg - 2016
Wien - 2016
Graz - 2018
Salzburg - 2018
Wien - 2018


Feature 3
Graz - 2016
Salzburg - 2016
Wien - 2016
Graz - 2018
Salzburg - 2018
Wien - 2018


Feature 4
Graz - 2016
Salzburg - 2016
Wien - 2016
Graz - 2018
Salzburg - 2018
Wien - 2018


Feature 5
Graz - 2016
Salzburg - 2016
Wien - 2016
Graz - 2018
Salzburg - 2018
Wien - 2018


Feature 6
Graz - 2016
Salzburg - 2016
Wien - 2016
Graz - 2018
Salzburg - 2018
Wien - 2018

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

### GESAMT-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)
liste_featuresdaten

feature_values  = {'f1': list(liste_featuresdaten[0]["value"]),
        'f2': list(liste_featuresdaten[1]["value"]),
        'f3': list(liste_featuresdaten[2]["value"]),
        'f4': list(liste_featuresdaten[3]["value"]),
        'f5': list(liste_featuresdaten[4]["value"]),
        'f6': list(liste_featuresdaten[5]["value"]),
        }

labels = list(liste_featuresdaten[1]["UID"])

df_values = pd.DataFrame(feature_values, index=labels)

# KORRELATION MATRIX BERECHNEN
correlation_matrix = df_values.corr()
print(correlation_matrix)

# KORRELATION MATRIX VISUALISIEREN
# Quelle: https://machinelearningmastery.com/visualize-machine-learning-data-python-pandas/
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111)
cax = ax.matshow(correlation_matrix, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,6,1)
names = ["f1", "f2", "f3", "f4", "f5", "f6"]
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
plt.show()
```

Für die Visualisierung der Korrelationsmatrizen wird ein Dataframe aus den „values“ der Features gebildet. Darauf wird die Funktion corr() aufgerufen, um die Korrelationsmatrix zu berechnen. Diese wird mittels „plt.figure“ und „ax.matshow“ geplottet.

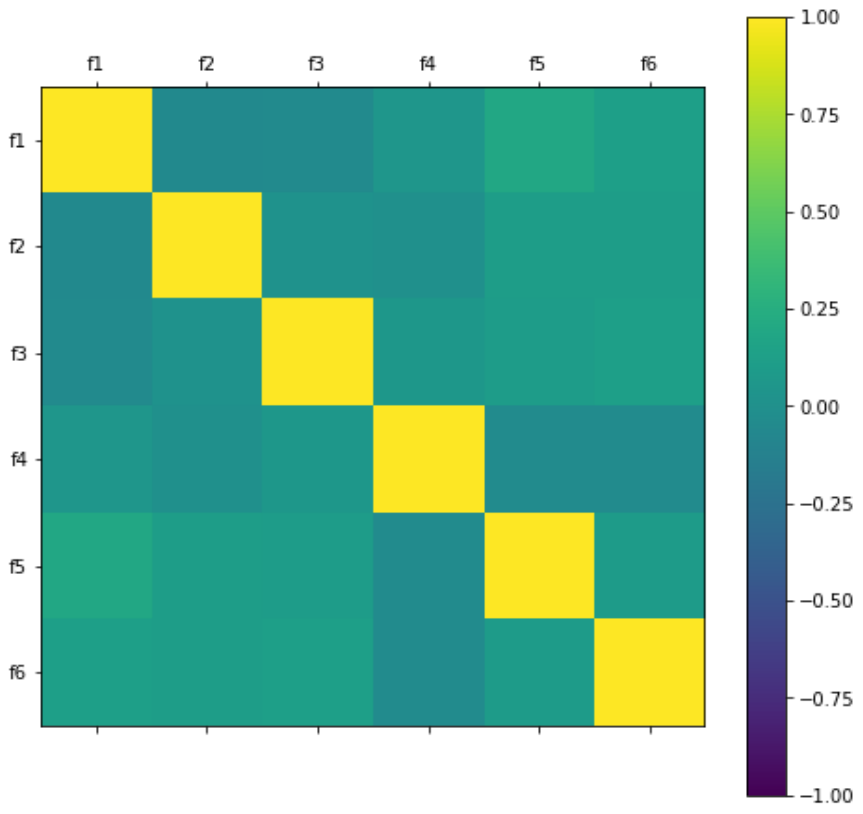
Das Dataframe, das die für die Berechnung notwendigen Feature-Values enthält, sieht folgendermaßen aus:

```
In [456]: df_values
Out[456]:
          f1         f2         f3         f4          f5          f6
10 -19.682142   3.554936   7.846988   6.061614   44.804247   41.769487
10   7.454699 -10.998649  -2.878346  -3.580512  -54.428479   -5.413186
42   9.259593  27.649605  15.188278  15.053493  211.745919   74.401034
42   1.158177   4.183757  -2.997264  23.808195   41.003214    1.850606
20   2.066110   6.896700  -3.958214  30.403989  118.380326   33.662859
20   5.490661  14.043380  -0.975849  19.620544  141.780105   29.631927
99  -3.676499  14.555763  10.382064  13.074585    5.053960    2.876545
99  -3.051798  -8.435430  10.988919  24.805101   57.468664   21.184415
73   5.200505   4.431096   3.699571  20.102395   51.943200  105.277252
73  -0.278199 -12.185994   6.809528  21.220517   83.871073   76.942002
74   7.948902   3.016128  12.302716   7.162328   68.535104   78.406128
74  19.516241   1.730303  -2.450919   7.536040   52.670083   78.734726
```

Die Korrelationsmatrix:

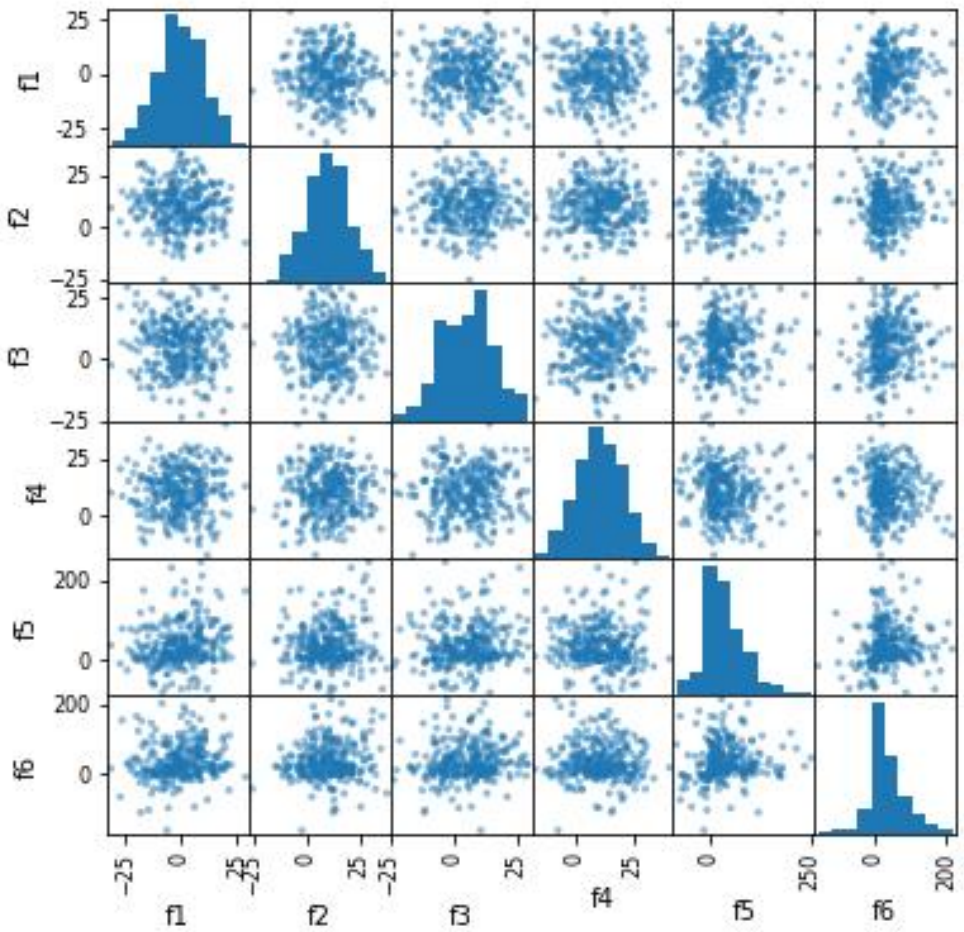
```
In [457]: correlation_matrix
Out[457]:
          f1        f2        f3        f4        f5        f6
f1  1.000000 -0.050644 -0.040476  0.054269  0.199429  0.129908
f2 -0.050644  1.000000  0.016816  0.005453  0.112926  0.112189
f3 -0.040476  0.016816  1.000000  0.060480  0.105827  0.127470
f4  0.054269  0.005453  0.060480  1.000000 -0.034859 -0.033460
f5  0.199429  0.112926  0.105827 -0.034859  1.000000  0.099248
f6  0.129908  0.112189  0.127470 -0.033460  0.099248  1.000000
```

Die Visualisierung der Korrelationsmatrix:



Die folgende Abbildung zeigt das Ergebnis der Darstellung von den Features f1 bis f6 mittels pandas *scatter_matrix*.

```
# ÜBERBLICK ÜBER df_values MITTELS SCATTERMATRIX
pd.scatter_matrix(df_values, figsize=(6, 6))
plt.show()
```



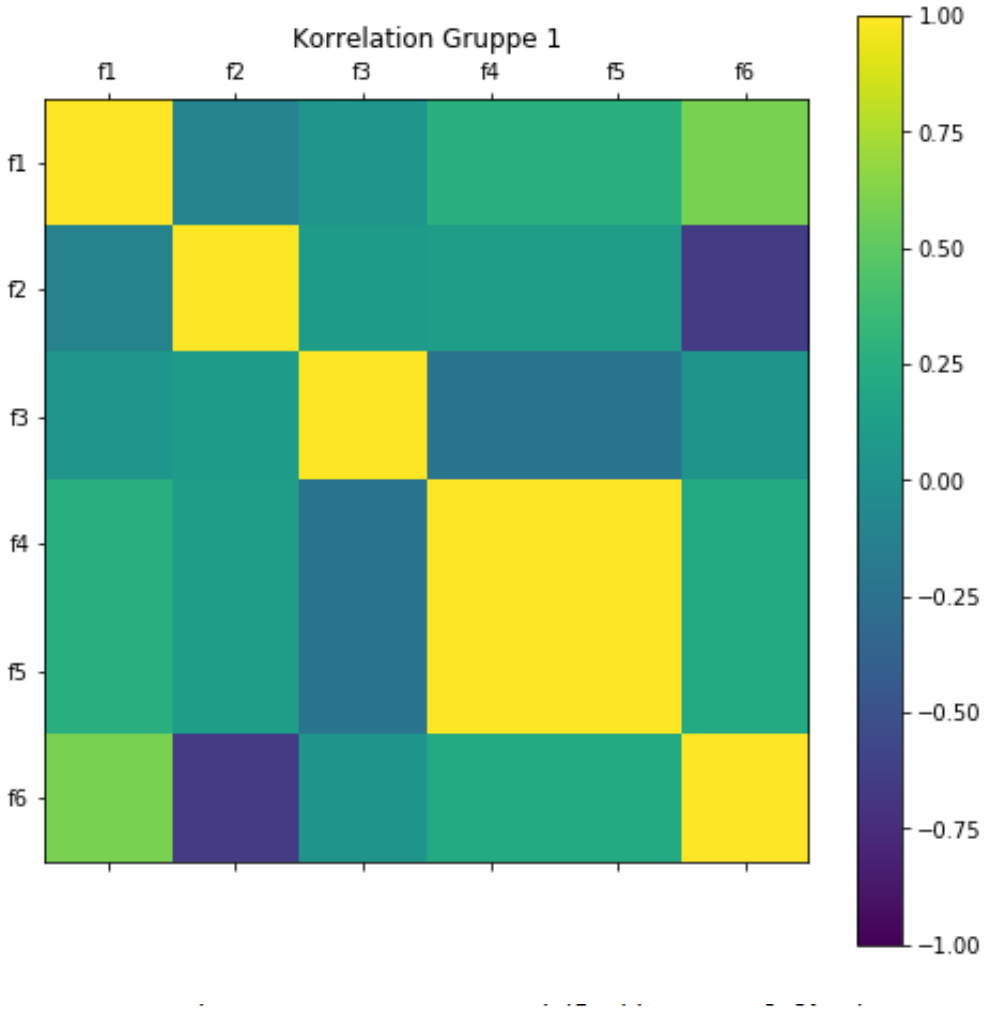
Korrelation pro Gruppe (für jeweils alle Features):

```
## ------------------------------------------------------
## KORRELATION PRO GRUPPE (alle features betrachtet)
## ------------------------------------------------------
# basisdaten
joined_all

# gruppen bilden
g1 = joined_all[ (joined_all["Stadt"]=="Graz") & (joined_all["Datum"]=="20180416T00:00Z") ]
g2 = joined_all[ (joined_all["Stadt"]=="Graz") & (joined_all["Datum"]=="20160813T00:00Z") ]
g3 = joined_all[ (joined_all["Stadt"]=="Salzburg") & (joined_all["Datum"]=="20180416T00:00Z") ]
g4 = joined_all[ (joined_all["Stadt"]=="Salzburg") & (joined_all["Datum"]=="20160813T00:00Z") ]
g5 = joined_all[ (joined_all["Stadt"]=="Wien") & (joined_all["Datum"]=="20180416T00:00Z") ]
g6 = joined_all[ (joined_all["Stadt"]=="Wien") & (joined_all["Datum"]=="20160813T00:00Z") ]

# gruppenliste
gruppenliste = [g1,g2,g3,g4,g5,g6]
```

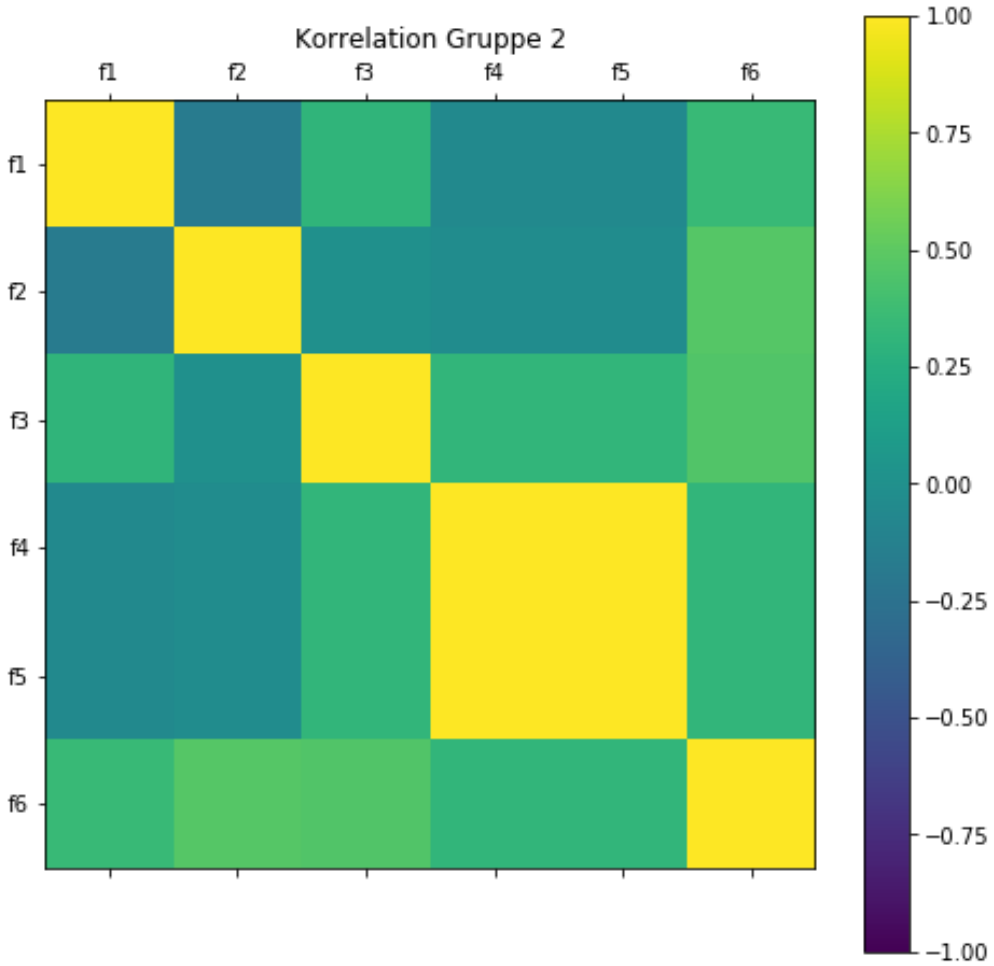
```
# FUNKTION: plot fuer jeweilige gruppe korrelation zwischen den features 1-6
def plotCorrelationPerGroup(groupData, groupNr):
    """
    funktion visualisiert fuer das uebergebene groupdata die korrelation
    param: groupData
    param: groupNr
    """

    f1 = groupData[ groupData["variable"]=="Feature 1" ]["value"]
    f2 = groupData[ groupData["variable"]=="Feature 2" ]["value"]
    f3 = groupData[ groupData["variable"]=="Feature 3" ]["value"]
    f4 = groupData[ groupData["variable"]=="Feature 4" ]["value"]
    f5 = groupData[ groupData["variable"]=="Feature 5" ]["value"]
    f6 = groupData[ groupData["variable"]=="Feature 6" ]["value"]

    data = {'f1': list(f1),
            'f2': list(f2),
            'f3': list(f3),
            'f4': list(f4),
            'f5': list(f5),
            'f6': list(f6)
            }

    df_values = pd.DataFrame(data, columns=['f1','f2','f3','f4','f4','f5'])

    # KORRELATION MATRIX BERECHNEN
    correlation_matrix = df_values.corr()
    print(correlation_matrix)

    # KORRELATION MATRIX VISUALISIEREN
    # Quelle: https://machinelearningmastery.com/visualize-machine-learning-data-python-
    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(111)
    plt.subplot().set_title("Korrelation Gruppe " + str(groupNr+1))
    cax = ax.matshow(correlation_matrix, vmin=-1, vmax=1)
    fig.colorbar(cax)
    ticks = np.arange(0,6,1)
    names = ["f1", "f2", "f3", "f4", "f5", "f6"]
    ax.set_xticks(ticks)
    ax.set_yticks(ticks)
    ax.set_xticklabels(names)
    ax.set_yticklabels(names)
    plt.show()
```

```
# FUNKTION plotCorrelationPerGroup fuer alle gruppen aufrufen
for i in range(0,len(gruppenliste)):
    plotCorrelationPerGroup(gruppenliste[i], i)
```

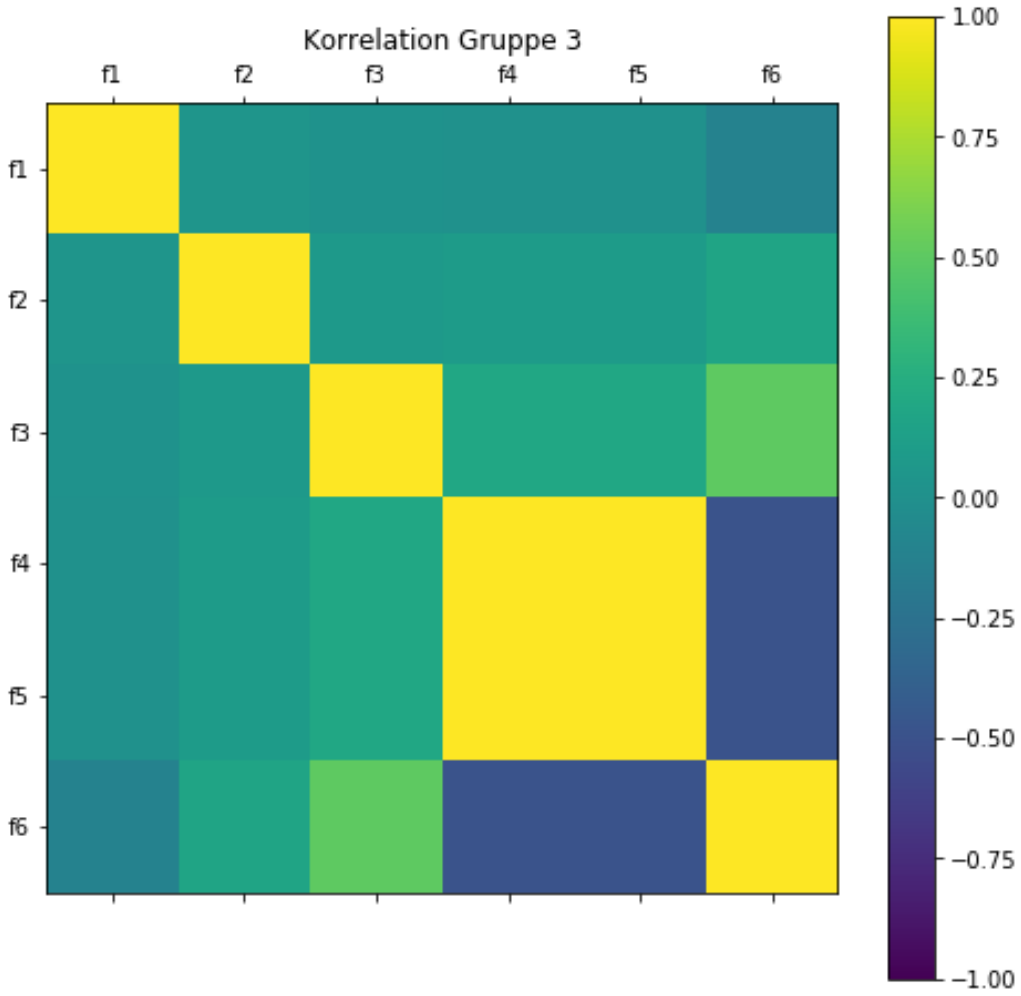
Visualisierung:



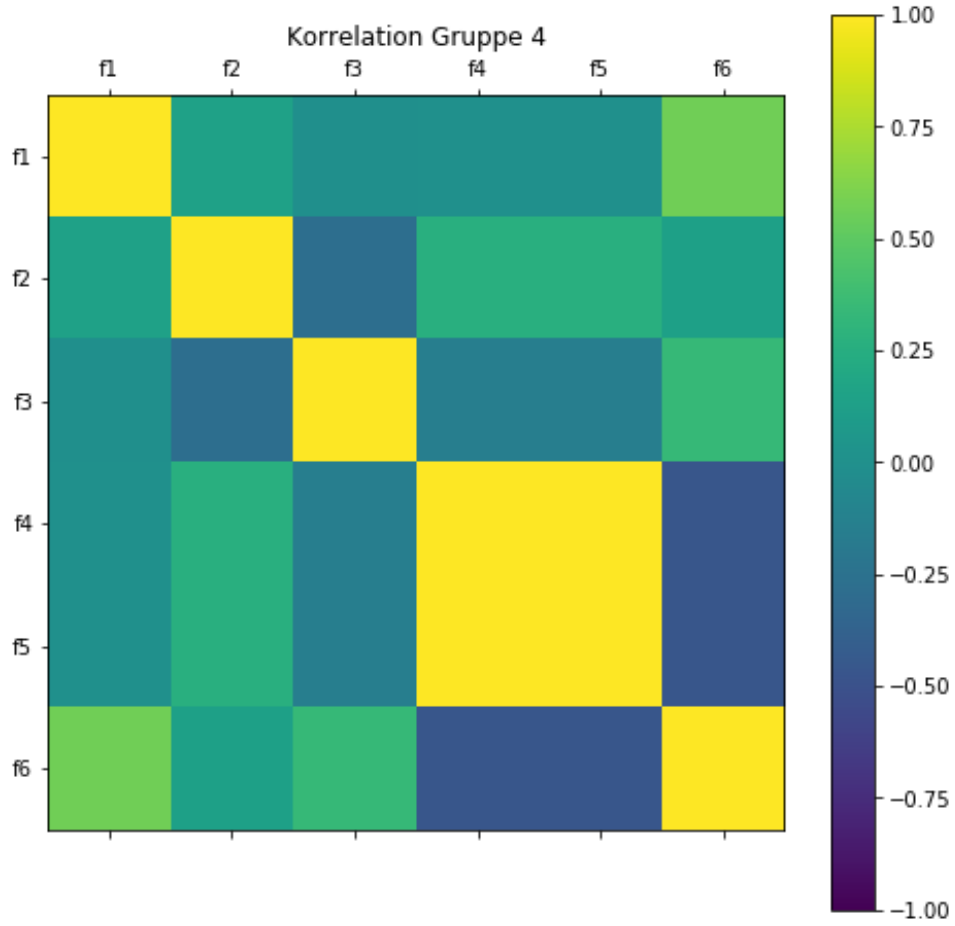
	f1	f2	f3	f4	f4	f5
f1	1.000000	-0.105789	0.046934	0.267477	0.267477	0.601203
f2	-0.105789	1.000000	0.108814	0.115114	0.115114	-0.640812
f3	0.046934	0.108814	1.000000	-0.226621	-0.226621	0.033232
f4	0.267477	0.115114	-0.226621	1.000000	1.000000	0.222937
f4	0.267477	0.115114	-0.226621	1.000000	1.000000	0.222937
f5	0.601203	-0.640812	0.033232	0.222937	0.222937	1.000000



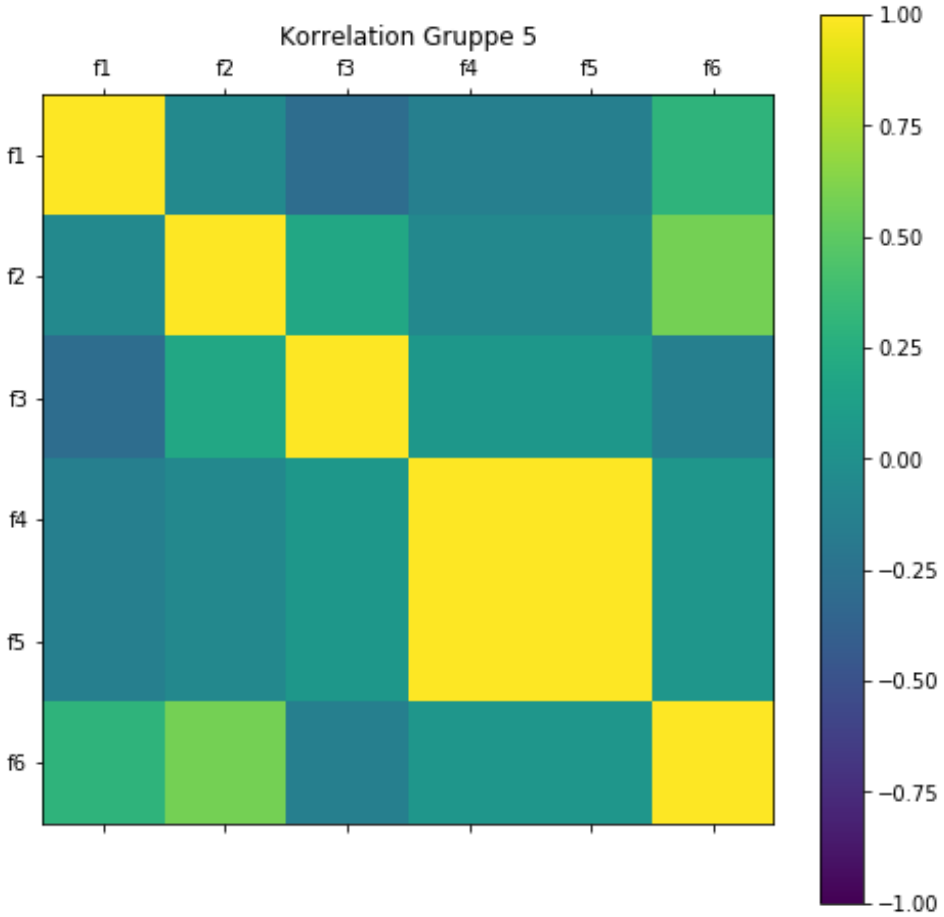
	f1	f2	f3	f4	f4	f5
f1	1.000000	-0.171523	0.306136	-0.050858	-0.050858	0.345835
f2	-0.171523	1.000000	0.003869	-0.029364	-0.029364	0.475011
f3	0.306136	0.003869	1.000000	0.317731	0.317731	0.453553
f4	-0.050858	-0.029364	0.317731	1.000000	1.000000	0.312737
f4	-0.050858	-0.029364	0.317731	1.000000	1.000000	0.312737
f5	0.345835	0.475011	0.453553	0.312737	0.312737	1.000000



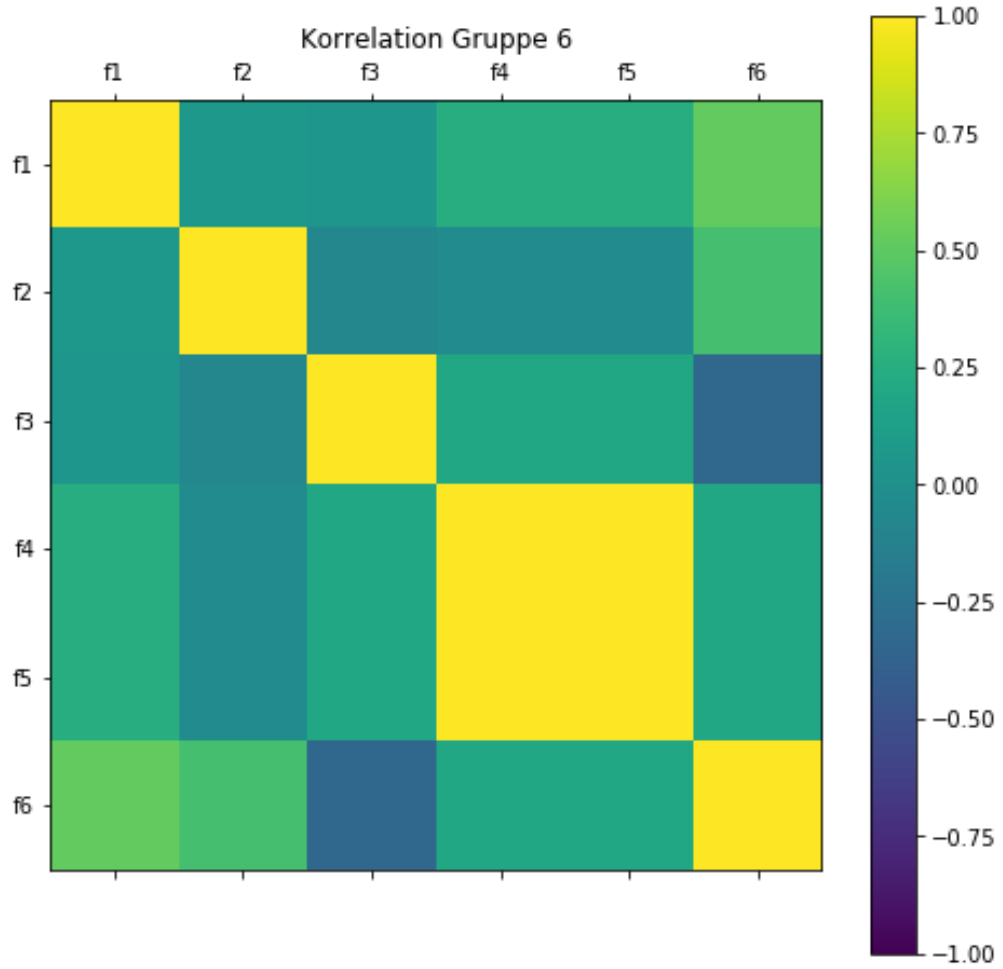
	f1	f2	f3	f4	f4	f5
f1	1.000000	0.046192	0.022138	0.010878	0.010878	-0.115899
f2	0.046192	1.000000	0.083466	0.096638	0.096638	0.170886
f3	0.022138	0.083466	1.000000	0.194845	0.194845	0.503635
f4	0.010878	0.096638	0.194845	1.000000	1.000000	-0.486912
f4	0.010878	0.096638	0.194845	1.000000	1.000000	-0.486912
f5	-0.115899	0.170886	0.503635	-0.486912	-0.486912	1.000000



	f1	f2	f3	f4	f4	f5
f1	1.000000	0.143353	-0.003789	0.005111	0.005111	0.569953
f2	0.143353	1.000000	-0.274157	0.259904	0.259904	0.133285
f3	-0.003789	-0.274157	1.000000	-0.142696	-0.142696	0.342952
f4	0.005111	0.259904	-0.142696	1.000000	1.000000	-0.468655
f4	0.005111	0.259904	-0.142696	1.000000	1.000000	-0.468655
f5	0.569953	0.133285	0.342952	-0.468655	-0.468655	1.000000



	f1	f2	f3	f4	f4	f5
f1	1.000000	-0.054764	-0.284818	-0.134432	-0.134432	0.300984
f2	-0.054764	1.000000	0.198689	-0.068728	-0.068728	0.583468
f3	-0.284818	0.198689	1.000000	0.056996	0.056996	-0.138401
f4	-0.134432	-0.068728	0.056996	1.000000	1.000000	0.048880
f4	-0.134432	-0.068728	0.056996	1.000000	1.000000	0.048880
f5	0.300984	0.583468	-0.138401	0.048880	0.048880	1.000000



	f1	f2	f3	f4	f4	f5
f1	1.000000	0.069899	0.049652	0.248402	0.248402	0.516285
f2	0.069899	1.000000	-0.070477	-0.033397	-0.033397	0.412686
f3	0.049652	-0.070477	1.000000	0.187762	0.187762	-0.321560
f4	0.248402	-0.033397	0.187762	1.000000	1.000000	0.201112
f4	0.248402	-0.033397	0.187762	1.000000	1.000000	0.201112
f5	0.516285	0.412686	-0.321560	0.201112	0.201112	1.000000

Anmerkung:

Die Korrelation zwischen den Features 1-6 ist je nach Gruppe unterschiedlich.

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

# neue abgeleitete variable aus summe von f5 und f6 bilden
variable_sum56 = list()
for i in range(0, len(list(liste_featuresdaten[5]["value"]))):
    variable_sum56.append( list(liste_featuresdaten[4]["value"])[i] + list(liste_featuresdaten[5]["value"])[i] )
len(variable_sum56)

# neues dataframe feature_values_2 erstellen fuer die analyse
feature_values_2  = {'f1': list(liste_featuresdaten[0]["value"]),
        'f2': list(liste_featuresdaten[1]["value"]),
        'f3': list(liste_featuresdaten[2]["value"]),
        'f4': list(liste_featuresdaten[3]["value"]),
        'fNEU': variable_sum56,
        'stadt': list(liste_featuresdaten[5]["Stadt"]),
        'datum': list(liste_featuresdaten[5]["Datum"])
        }

labels = list(liste_featuresdaten[1]["UID"])

df_variables_2 = pd.DataFrame(feature_values_2, index=labels)
```

Aus der Summe von Feature 5 und Feature 6 wurde eine neue Variable namens „*variable_sum56*“ erstellt. Dann wurde ein neues Dataframe, das die Spalten „*f1*“, „*f2*“, „*f3*“, „*f4*“, „*fNEU*“, „*stadt*“ und „*datum*“ enthält, erstellt. Die neue Variable wurde „*fNEU*“ genannt.

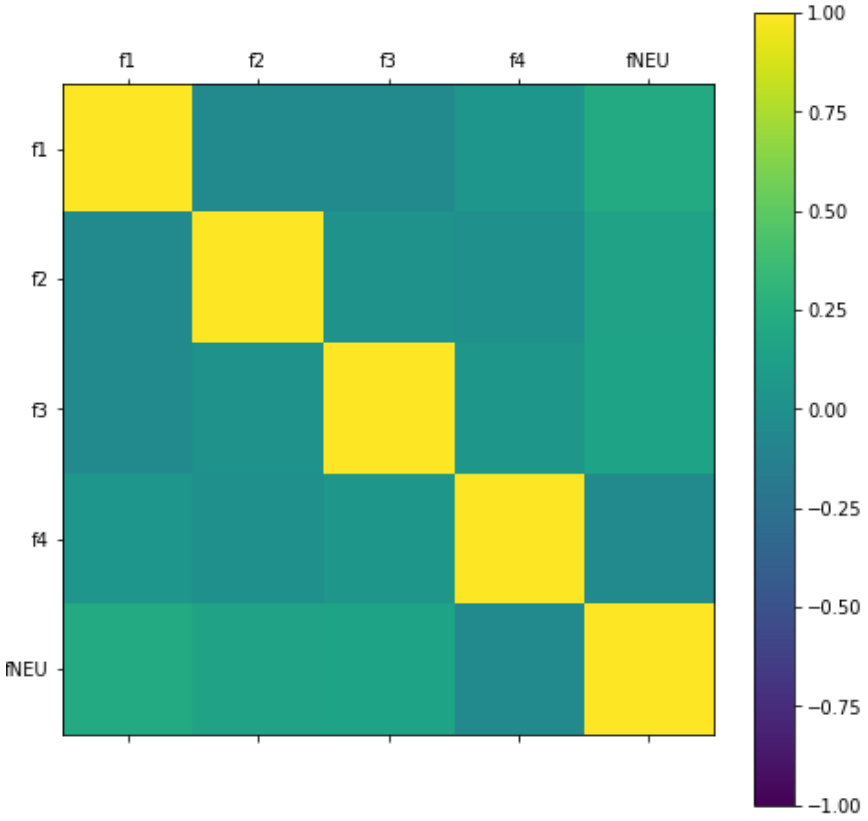
Das neue Dataframe sieht folgendermaßen aus:

```
In [460]: df_variables_2
Out[460]:
              datum         f1         f2         f3         f4         fNEU     stadt
10  20180416T00:00Z -19.682142   3.554936   7.846988   6.061614   86.573734  10  Wien
10  20160813T00:00Z   7.454699 -10.998649  -2.878346  -3.580512  -59.841664  10  Wien
42  20160813T00:00Z   9.259593  27.649605  15.188278  15.053493  286.146953  42  Wien
42  20180416T00:00Z   1.158177   4.183757  -2.997264  23.808195   42.853820  42  Wien
20  20160813T00:00Z   2.066110   6.896700  -3.958214  30.403989  152.043185  20  Wien
20  20180416T00:00Z   5.490661  14.043380  -0.975849  19.620544  171.412032  20  Wien
99  20180416T00:00Z  -3.676499  14.555763  10.382064  13.074585    7.930505  99  Graz
99  20160813T00:00Z  -3.051798  -8.435430  10.988919  24.805101   78.653079  99  Graz
73  20180416T00:00Z   5.200505   4.431096   3.699571  20.102395  157.220452  73  Graz
73  20160813T00:00Z  -0.278199 -12.185994   6.809528  21.220517  160.813075  73  Graz
74  20180416T00:00Z   7.948902   3.016128  12.302716   7.162328  146.941232  74  Graz
74  20160813T00:00Z  19.516241   1.730303  -2.450919   7.536040  131.404808  74  Graz
25  20160813T00:00Z  -5.301680  16.487270 -16.769324  13.386031  115.868799  25  Wien
25  20180416T00:00Z  -6.185252   0.451852   2.217058  17.144076   64.047546  25  Wien
1   20160813T00:00Z   0.533297  16.003032  -6.477621  15.693470   48.684902  1   Wien
1   20180416T00:00Z -15.418203  11.824135   8.742878  -1.314364  126.788423  1   Wien
16  20160813T00:00Z -19.232991  12.336079  -7.911563   4.945652    3.190489  16  Wien
16  20180416T00:00Z -21.740285   8.949513   4.354523   6.499847   63.710698  16  Wien
```

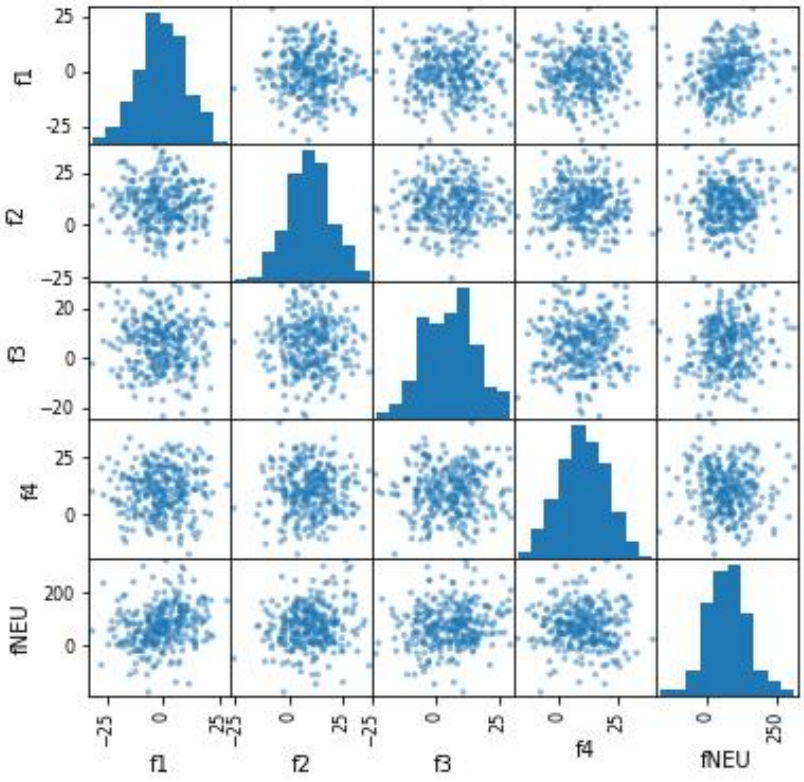
```
# 2.b. Gibt es Korrelationen zwischen den verbleibenden Variablen und der neuen abgeleiteten

df_fuerKorr = df_variables_2.loc[:, ['f1', 'f2', 'f3', 'f4', 'fNEU',]]

# KORRELATION MATRIX BERECHNEN
correlation_matrix_neu = df_variables_2.corr()
print(correlation_matrix_neu)

# KORRELATION MATRIX VISUALISIEREN
# Quelle: https://machinelearningmastery.com/visualize-machine-learning-data-python-pandas/
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111)
cax = ax.matshow(correlation_matrix_neu, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,5,1)
names = ["f1", "f2", "f3", "f4", "fNEU"]
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
plt.show()

# ÜBERBLICK ÜBER df_values MITTELS SCATTERMATRIX
pd.scatter_matrix(df_variables_2, figsize=(6, 6))
plt.show()
```

Die Korrelation zwischen den verbleibenden Variablen wird ebenfalls mit einer Korrelationsmatrix untersucht. Dafür wird die Funktion *corr()* verwendet, die die Korrelationsmatrix berechnet. Diese wird mittels „*matshow*“ geplottet.

Die Korrelationsmatrix inkl. der neuen abgeleiteten Variable („fNEU“):



Scatterplot inclusive der neuen abgeleiteten Variable („fNEU“):



```
# 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 dokume
# 2.f. Welche Modellierungsmethode verwenden Sie und warum haben Sie sich für dieses Modell entschieden?

# LINEARES MODELL MIT MODELLIERUNG ALS ADDITIVEM ZUSAMMENHANG
# zielvariable ist fNEU
# predictorvariablen sind f1, f2, f3 und f4
results = smf.ols('fNEU ~ f1+f2+f3+f4', data=df_variables_2).fit()
print(results.summary())
   # ANMERKUNG:
   # f1, f2, f3 signifikant
   # f4 nicht signifikant
   # (adj.) R squared ca. 10%
```

Als erster Versuch wird ein **lineares Modell (OLS) mit additivem Zusammenhang zwischen f1, f2, f3 und f4** zur Erklärung der Variable fNEU berechnet. Die erklärte Varianz (R-Squared bzw. adj. R-Squared) liegt bei 10% bzw. 11%. Die Variablen f1, f2 und f3 sind signifikant, der Variable f4 ist nicht signifikant beitragend zur Erklärung der Variable fNEU. Die Zusammenfassung der Resultate ergibt Folgendes:

```
In [463]: print(results.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                   fNEU   R-squared:                       0.108
Model:                           OLS   Adj. R-squared:                  0.096
Method:                Least Squares   F-statistic:                     8.801
Date:               Sat, 02 Jun 2018   Prob (F-statistic):           1.01e-06
Time:                       00:13:31   Log-Likelihood:                -1690.2
No. Observations:                296   AIC:                             3390.
Df Residuals:                    291   BIC:                             3409.
Df Model:                          4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     60.0064      7.730      7.762      0.000      44.792      75.221
f1             1.7690      0.408      4.341      0.000       0.967       2.571
f2             1.2173      0.418      2.914      0.004       0.395       2.039
f3             1.2736      0.420      3.035      0.003       0.448       2.099
f4            -0.5462      0.432     -1.264      0.207      -1.396       0.304
==============================================================================
Omnibus:                        6.587   Durbin-Watson:                   2.233
Prob(Omnibus):                  0.037   Jarque-Bera (JB):                9.827
Skew:                           0.083   Prob(JB):                      0.00735
Kurtosis:                       3.877   Cond. No.                         33.7
==============================================================================
```

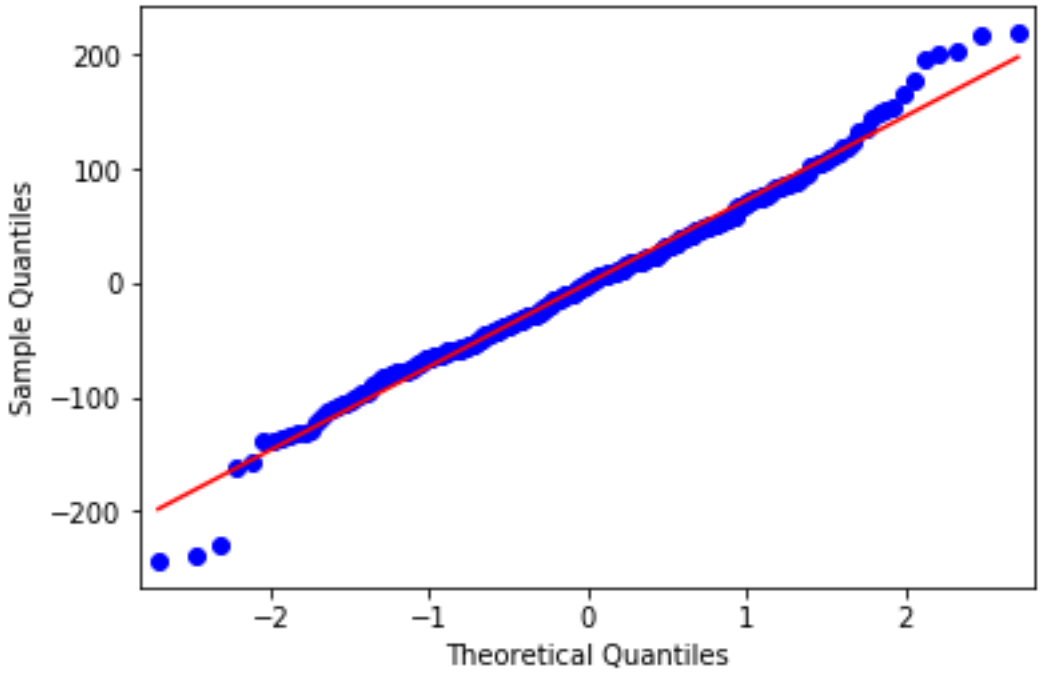
```
# LINEARES MODELL MIT BERUECKSICHTIGUNG DER INTERAKTIONEN
# zielvariable ist fNEU
# predictorvariablen sind f1, f2, f3 und f4
results2 = smf.ols('fNEU ~ f1*f2*f3*f4', data=df_variables_2).fit()
print(results2.summary())
   # ANMERKUNG:
   # interaktionen nicht signifikant
   # (adj.) R squared ca. 10% bzw. R-squared 15%
```

Der zweite Versuch ist **ein lineares Modell (OLS) u.a. unter Berücksichtigung der Interaktionen zwischen den Variablen** f1, f2, f3, f4. Die Interaktionen sind nicht signifikant. Die erklärte Varianz (R-Squared bzw. adj. R-Squared) liegt bei 15% bzw. 10%, d.h. sie konnte im Vergleich zum ersten linearen Modell erhöht werden.

```
In [465]: print(results2.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                   fNEU   R-squared:                       0.153
Model:                           OLS   Adj. R-squared:                  0.108
Method:                Least Squares   F-statistic:                     3.383
Date:               Sat, 02 Jun 2018   Prob (F-statistic):           2.84e-05
Time:                       00:14:36   Log-Likelihood:                -1682.5
No. Observations:                296   AIC:                             3397.
Df Residuals:                    280   BIC:                             3456.
Df Model:                         15
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      59.3128     10.044      5.905      0.000      39.541      79.084
f1              2.6179      1.125      2.326      0.021       0.403       4.833
f2              1.3788      0.743      1.855      0.065      -0.085       2.842
f1:f2          -0.1086      0.088     -1.229      0.220      -0.283       0.065
f3              0.7536      0.902      0.836      0.404      -1.022       2.529
f1:f3           0.0661      0.093      0.713      0.477      -0.117       0.249
f2:f3           0.0173      0.077      0.224      0.823      -0.135       0.169
f1:f2:f3       -0.0086      0.008     -1.064      0.288      -0.025       0.007
f4              0.0909      0.737      0.123      0.902      -1.359       1.541
f1:f4          -0.0670      0.079     -0.849      0.397      -0.222       0.088
f2:f4          -0.0694      0.054     -1.279      0.202      -0.176       0.037
f1:f2:f4        0.0084      0.006      1.457      0.146      -0.003       0.020
f3:f4          -0.0238      0.059     -0.401      0.689      -0.141       0.093
f1:f3:f4        0.0010      0.005      0.182      0.856      -0.010       0.012
f2:f3:f4        0.0046      0.005      0.904      0.367      -0.005       0.014
f1:f2:f3:f4 -1.228e-05      0.000     -0.026      0.979      -0.001       0.001
==============================================================================
Omnibus:                        6.593   Durbin-Watson:                   2.223
Prob(Omnibus):                  0.037   Jarque-Bera (JB):                9.514
Skew:                           0.109   Prob(JB):                      0.00859
Kurtosis:                       3.851   Cond. No.                     6.17e+04
==============================================================================
```

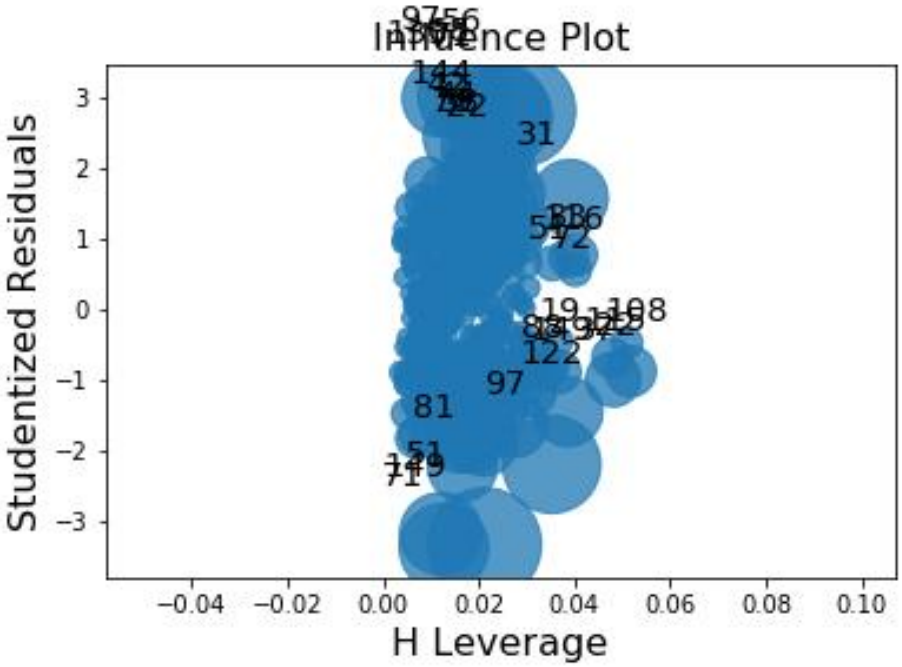
```
# REGRESSION-DIAGNOSE

# QQ plot
resid = results.resid
fig  = sm.qqplot(resid, line="s")
plt.show()
# interpretation:
# im quantile-quantile-plot (i.e. theoretische
# quantile vs. empirische quantile) sollten die punkte
# im falle der erfuellten normalverteilungsannahme
# bzgl. der fehlerterme auf der gerade liegen.
# im vorliegenden fall is dies im intervall von ca. [-2,2]
# so, aber an den enden nicht
```

Zur **Regressionsdiagnose** des ersten linearen Modells wird ein Q-Q-Plot berechnet, der die theoretischen Quantile mit den empirischen Quantilen vergleicht. Wenn die Punkte direkt auf der Gerade liegen würden, wäre die Normalverteilungsannahme der Fehlerterme voll erfüllt. In diesem Fall liegen die Punkte im Intervall von ca. -2 bis 2 auf der Gerade, aber fransen außerhalb dieses Intervalls aus und weichen stärker von der Gerade ab. Die Normalverteilungsannahme der Fehlerterme ist also nicht voll erfüllt.



```
# influence plot
sm.graphics.influence_plot(results, criterion="Cooks")
plt.show()
# interpretation:
# punkte, die weit rechts oben (od. rechts unten) liegen, sind
# einflussreiche punkte auf die regression
# hier: die punkte liegen alle bzgl. leverage mittig, aber teilweise
# bzgl. studentized residuals (outlyingness) über die ganze bandbreite
# reichend
# influence wird durch punktgroesse dargestellt
# hier: eine reihe von punkten wird vergleichsweise groß dargestellt
# (e.g. 108, 31, 122 etc.)
```

Für die Ergebnisse des ersten linearen Modells wurde ebenfalls ein Influence Plot erstellt. Dieser plottet Leverage und Residuals. „Einflussreiche“ Punkte, die Auswirkungen auf die Regressionsergebnisse haben, liegen rechts oben bzw. insgesamt rechts. In diesem Fall liegen die Punkte alle mittig bzgl. Leverage, aber teilweise bzgl. Studentized Residuals („outlyingness“) über die ganze Bandbreite reichend. Der Einfluss wird durch die Punktgröße dargestellt. Vergleichsweise groß werden hier z.B. die Punkte 97, 108, 19, 122 oder auch 72 dargestellt.



```
# VERSUCH DER MODELLIERUNG MIT ROBUSTER METHODE
# (unter verwendung des huber estimator)
# robuste methoden, versuchen mit outliern umzugehen auf eine art und weise,
# dass diese nicht so stark ins gewicht fallen in den regressionsresultaten
results3 = smf.RLM.from_formula(formula='fNEU ~ f1+f2+f3+f4', data=df_variables_2,
                                M=sm.robust.norms.HuberT()).fit()
print(results3.summary())
# interpretation:
# f1, f2, f3 sind signifikant
# ergebnis hat sich im vgl. zu den 2 vorigen modellen
# nicht sehr stark verändert
```

Der dritte Versuch berechnet ein **robustes lineares Modell nach dem Ansatz von Huber unter Betrachtung eines linearen Zusammenhangs zwischen den Variablen f1, f2, f3 und f4**. Robuste Methode versuchen mit Outliern umzugehen, indem sie diese nicht so stark ins Gewicht fallen lassen im Zuge der Regression. In der untenstehenden Zusammenfassung der Resultate sieht man, dass f1, f2 und f3 signifikant sind.

```
In [468]: print(results3.summary())
                    Robust linear Model Regression Results
==============================================================================
Dep. Variable:                   fNEU   No. Observations:                  296
Model:                           RLM   Df Residuals:                      291
Method:                         IRLS   Df Model:                            4
Norm:                          HuberT
Scale Est.:                       mad
Cov Type:                          H1
Date:                Sat, 02 Jun 2018
Time:                        00:16:37
No. Iterations:                    13
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     62.7509      7.308      8.587      0.000      48.428      77.074
f1             1.8528      0.385      4.809      0.000       1.098       2.608
f2             1.0183      0.395      2.579      0.010       0.244       1.792
f3             1.1375      0.397      2.868      0.004       0.360       1.915
f4            -0.6396      0.408     -1.566      0.117      -1.440       0.161
==============================================================================

----------------------------------------------------------------------------------------------------------------
```

```
# UMGANG MIT KATEGORIELLEN VARIABLEN:
# LINEARES MODELL MIT MODELLIERUNG ALS ADDITIVEM ZUSAMMENHANG UND INTERAKTIONEN
# zielvariable ist fNEU
# predictorvariablen sind f1, f2, f3, stadt, datum
# dummy-codierung (bzw. treatment-codierung) der kategoriellen variablen)
results4 = smf.ols('fNEU ~ (f1+f2+f3+f4)*C(stadt)*C(datum)', data=df_variables_2).fit()
print(results4.summary())
   # ANMERKUNG:
   # f2:C(datum)[T.20180416T00:00Z] signifikant
   # f3 signifikant
   # f4:C(stadt)[T.Salzburg] signifikant
   # erklärte varianz wurde auf R-square=36% und adj. R-squared=29% erhöht
   # dieses modell behandelt bisher am besten die zusammenhaenge in den daten
```

Die **kategoriellen Variablen** (d.h. Faktoren, die keine numerischen Werte habe, wie hier z.B. „stadt“ und „datum“) können **mittels Dummy-Codierung berücksichtigt** werden, wie es im vierten Modellversuch gezeigt wird. Es ergibt sich, dass durch diese Konstellation die erklärte Varianz (R-Squared bzw. adj. R-Squared) auf 29% bzw. 36% erhöht werden kann (siehe untenstehenden Output). Bezüglich der erklärten Varianz ist dieses bisher das beste Modell.

```
In [471]: print(results4.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                   fNEU   R-squared:                       0.360
Model:                           OLS   Adj. R-squared:                  0.290
Method:                Least Squares   F-statistic:                     5.150
Date:               Sat, 02 Jun 2018   Prob (F-statistic):           9.24e-14
Time:                       00:18:09   Log-Likelihood:                -1641.2
No. Observations:                296   AIC:                             3342.
Df Residuals:                    266   BIC:                             3453.
Df Model:                         29
Covariance Type:            nonrobust
================================================================================================================
                                                   coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------------------------
Intercept                                       67.6490     19.487      3.472      0.001      29.281     106.017
C(stadt)[T.Salzburg]                           -32.3042     26.384     -1.224      0.222     -84.252      19.644
C(stadt)[T.Wien]                               -14.7617     25.211     -0.586      0.559     -64.400      34.877
C(datum)[T.20180416T00:00Z]                    -14.2547     26.895     -0.530      0.597     -67.209      38.700
C(stadt)[T.Salzburg]:C(datum)[T.20180416T00:00Z]  36.8984     35.678      1.034      0.302     -33.349     107.146
C(stadt)[T.Wien]:C(datum)[T.20180416T00:00Z]      41.9780     35.523      1.182      0.238     -27.964     111.920
f1                                                 0.4724      0.871      0.542      0.588      -1.243       2.188
f1:C(stadt)[T.Salzburg]                            2.2109      1.354      1.633      0.104      -0.454       4.876
f1:C(stadt)[T.Wien]                                2.1580      1.365      1.581      0.115      -0.529       4.845
f1:C(datum)[T.20180416T00:00Z]                     1.2721      1.188      1.071      0.285      -1.067       3.611
f1:C(stadt)[T.Salzburg]:C(datum)[T.20180416T00:00Z]  -1.1286      1.901     -0.594      0.553      -4.872       2.615
f1:C(stadt)[T.Wien]:C(datum)[T.20180416T00:00Z]    -2.6143      1.791     -1.460      0.145      -6.140       0.912
f2                                                 1.4055      0.804      1.749      0.081      -0.177       2.988
f2:C(stadt)[T.Salzburg]                           -0.0791      1.281     -0.062      0.951      -2.601       2.443
f2:C(stadt)[T.Wien]                                1.7939      1.185      1.513      0.131      -0.540       4.128
f2:C(datum)[T.20180416T00:00Z]                    -3.2197      1.381     -2.332      0.020      -5.938      -0.501
f2:C(stadt)[T.Salzburg]:C(datum)[T.20180416T00:00Z]  2.1973      1.919      1.145      0.253      -1.581       5.975
f2:C(stadt)[T.Wien]:C(datum)[T.20180416T00:00Z]    1.7318      1.964      0.882      0.379      -2.134       5.598
f3                                                 2.0594      0.897      2.296      0.022       0.293       3.826
f3:C(stadt)[T.Salzburg]                            0.7126      1.280      0.557      0.578      -1.807       3.233
f3:C(stadt)[T.Wien]                               -1.9095      1.301     -1.468      0.143      -4.470       0.651
f3:C(datum)[T.20180416T00:00Z]                    -1.4054      1.255     -1.120      0.264      -3.877       1.066
f3:C(stadt)[T.Salzburg]:C(datum)[T.20180416T00:00Z]  0.5959      1.816      0.328      0.743      -2.980       4.172
f3:C(stadt)[T.Wien]:C(datum)[T.20180416T00:00Z]    0.3718      1.928      0.193      0.847      -3.424       4.168
f4                                                 1.8437      1.128      1.634      0.103      -0.378       4.065
f4:C(stadt)[T.Salzburg]                           -4.8207      1.492     -3.232      0.001      -7.758      -1.884
f4:C(stadt)[T.Wien]                               -1.0571      1.466     -0.721      0.471      -3.943       1.829
f4:C(datum)[T.20180416T00:00Z]                     0.6559      1.554      0.422      0.673      -2.403       3.715
f4:C(stadt)[T.Salzburg]:C(datum)[T.20180416T00:00Z]  0.2512      2.021      0.124      0.901      -3.728       4.231
f4:C(stadt)[T.Wien]:C(datum)[T.20180416T00:00Z]   -1.1929      2.060     -0.579      0.563      -5.248       2.862
==============================================================================
Omnibus:                       10.142   Durbin-Watson:                   2.489
Prob(Omnibus):                  0.006   Jarque-Bera (JB):               16.371
Skew:                           0.192   Prob(JB):                     0.000279
Kurtosis:                       4.086   Cond. No.                         365.
==============================================================================
```