

```
1 # Hausarbeit IU Akademie Programmieren mit Python
2 # Andreas Müller
3
4 # Benötigte Importe
5 from Lib.Data_import import DataToImport
6 import sqlalchemy as sql
7 from Lib.DataWriteToDB import DataToDatabase
8 import os
9 from Lib.Search_Function import lookupthings
10 from Lib.Visualize import ShowInPlot
11
12
13 def extract_tablename(path):
14     result = os.path.basename(path)
15     result = result[:-4]
16     return result
17
18
19 DataBaseName = "IU_Hausarbeit.db"
20
21 engine = sql.create_engine("sqlite+pysqlite:///+" +
    DataBaseName, echo=True)
22 connection = engine.connect()
23
24 train_data = DataToImport()
25 ideal_data = DataToDatabase()
26
27 ''' ideal.csv importieren und in DB bringen'''
28 ideal_data.importieren("ideal.csv")
29 Tablename = extract_tablename("ideal.csv")
30 ideal_data.create_table(Tablename, ideal_data,
    connection)
31 ideal_data.data_to_table(Tablename, ideal_data,
    connection)
32
33 ''' Trainingsdaten importieren und in DB bringen'''
34 train_data.importieren("train.csv")
35 Tablename = extract_tablename("train.csv")
36 train_data_to_DB = DataToDatabase()
37 train_data_to_DB.create_table(Tablename, train_data,
    connection)
```

```
38 train_data_to_DB.data_to_table(Tablename, train_data
, connection)
39
40 ''' Trainingsdaten in ideal.csv suchen '''
41 ideal_4_data = lookupthings()
42 ideal_4_data.lookup_train_in_ideal(train_data,
ideal_data)
43 ideal_4_data_to_DB = DataToDatabase()
44 ideal_4_data_to_DB.create_table('four_of_ideal',
ideal_4_data, connection) # eigtl nicht nötig
45 ideal_4_data_to_DB.data_to_table('four_of_ideal',
ideal_4_data, connection) # eigtl nicht nötig
46
47
48 ''' Testdaten importieren um mit idealen 4 zu
vergleichen'''
49 test_data_to_DB = DataToDatabase()
50 test_data_compare = lookupthings()
51 ''' import und Weiterverarbeitung mit Kind-Klasse '''
52 test_data_to_DB.importieren("test.csv")
53 Tablename = extract_tablename("test.csv")
54
55 test_data_compare.lookup_test_in_ideal4(
test_data_to_DB, ideal_4_data)
56 test_data_to_DB.create_table(Tablename,
test_data_compare, connection)
57 test_data_to_DB.data_to_table(Tablename,
test_data_compare, connection)
58
59 ShowData = ShowInPlot()
60 test_data_to_DB.y[0] = test_data_to_DB.x
61 train_data.y[0] = train_data.x
62 ShowData.show_id4_plus_test(ideal_4_data,
test_data_to_DB, train_data, 'x', 'y', 'ideal 4 vs
test Data')
63
64 print("Saved plotts to folder")
65 print("DONE")
66
```

```

1 # class for importing data from csv
2 import os
3
4 class DataToImport:
5     def __init__(self):
6         self.data_path = None
7         self.file_content = [None]
8         self.x = []
9         self.y = [] # []
10        self.Anzahl_Spalten = 0 # Zaehler fuer die
 gesamte Anzahl der Spalten
11
12    def extract_tablename(self, path):
13        """
14            Funktion extrahiert den Dateinamen aus
 demDateipfad
15            :param path: Dateipfad
16            :return: result: Dateiname
17        """
18        result = os.path.basename(path)
19        result = result[:-4]
20        return result
21
22    def separate_lines(self, f_content, spalten):
23        """
24            function seperates the file content in two
 variables.
25            :param f_content: is in two columns, e.g. ['4
 .234', '3.12']
26            :spalten: Anzahl der gesamten Spalten
27            :return: two seperate variables
28        """
29
30            sp = 1 # Spaltenindex, kann bei 1 beginnen,
 da Spaltenindex die x-Werte enthaelt
31            b_x_valid = False # konnte x gelesen werden
 ?
32            x = []
33
34            # Liste fuer y-Werte erzeugen
35            y = []

```

```

36         for i in range(spalten):
37             y.append([])
38
39             # print("Laenge file_content: ", len(
40                 f_content))
40             i = 0      # Laufindex muss zurueck gesetzt
41                 werden, von for Schleife
41             # solange das Ende des Inhalts nicht erreicht
41                 ist lese ein
42             while i < len(f_content):
43                 # x einlesen, erster Wert muss 'x' sein
44                 try:
45                     if i > 0:
46                         x.append(float(f_content[i][0]))
47                         # erste Stelle ist ein string, kein float
48                     else:
49                         x.append(f_content[i][0])
50                         b_x_valid = True
50                     except:
51                         print(f"Punkt {i} kann nicht
51 verarbeitet werden")
52
53                         # y einlesen, wenn der x-Wert eingelesen
53                         werden konnte
54
55                         if b_x_valid:
56                             for sp in range(spalten-1):
57                                 try:
58
59                                     if i > 0:
60                                         y[sp+1].append(float(
61                                         f_content[i][sp+1]))
61                                     else:
62                                         y[sp+1].append(f_content[
62                                         i][sp+1])
63                                     except:
64                                         print(f"Punkt {i} kann nicht
64 verarbeitet werden. Zugehoeriges x wird gelöscht")
65                                         del x[i]
66
67                         for j in range(1, spalten):

```

```

68                                try:
69                                    del y[j][i]
70                                except:
71                                    print(f"Index {j} |"
72                                         f"\n{i} nicht zu loeschen!")
72                                sp = sp+1          #
73                                # Index ans Ende der Spalten setzen, damit diese nicht
73                                # importiert werden
73                                break    # notwendig um die
74                                # restlichen y-Werte nicht zu lesen
74                                sp += 1
75                                sp = 1          # Ruecksetzen fuer
75                                # naechsten Durchlauf fuer x
76                                b_x_valid = False    # Gueltigkeitswert
76                                ruecksetzen
77                                i += 1
78
79                                return x, y
80
81        def importieren(self, data_path):
82            """
83                function to import a file content
84                :param data_path: file to import
85                :return: nothing, x and y are self
86            """
87            try:
88                with open(data_path, "r") as
88                csv_import_file:
89                    for line in csv_import_file.read().
89                    split("\n"):
90                        self.file_content.append(line.
90                        split(","))  # = csv_import_file.read()
91                        self.file_content.__delitem__(0)
91                        # None in erster Zeile löschen
92
93                        # Anzahl Spalten der Daten zaehlen
94                        for self.Anzahl_Spalten in range(len(
94                        self.file_content[0])):
95                            self.Anzahl_Spalten += 1
96            except:
97                print("EXCEPTION importieren, Datei kann")

```

```
97 nicht geöffnet werden")
98
99     # test Daten nach x sortieren
100    Tablename = self.extract_tablename(data_path
101)
101    # Tabelle test.csv muss gesondert behandelt
102    # werden. Daten werden nach x sortiert.
102    if Tablename == 'test':
103        i = 0
104        while i < len(self.file_content)-1:
105            try:
106                self.file_content[i][0] = float(
106                self.file_content[i][0])
107                self.file_content[i][1] = float(
107                self.file_content[i][1])
108            except:
109                print(self.file_content[i])
110                del self.file_content[i]
111                print("konnte string nicht in
111 float konvertieren -> wurde gelöscht")
112                i -= 1
113
114                i += 1
115                del self.file_content[i]
116                # Daten werden nach x sortiert.
117                self.file_content.sort()
118
119                if self.Anzahl_Spalten > 0:
120                    self.x, self.y = self.separate_lines(
120                    self.file_content, self.Anzahl_Spalten)
121                else:
122                    self.x = []
123                    self.y = []
124
125
126                print("importieren DONE")
127
```

```

1 import sqlalchemy as sql
2 from sqlalchemy.exc import OperationalError as
sqlOpErr
3 from Lib.Data_import import DataToImport
4
5
6 class DataToDatabase(DataToImport):      # erbt
    Dataimport Klasse
7
8     def __init__(self):
9
10         self.rows = 0
11
12         super().__init__()
13
14     def create_table(self, tablename, di, connection
):
15         """
16             Die Funktion erzeugt eine Tabelle in der
Datenbank
17             :param tablename: übergebener Tabellenname
18             :param di: Inhalt der Tabelle
19             :param connection: Verbindung zur Datenbank
20             kann die Tabelle nicht erzeugt werden wird
eine Exception ausgelöst
21         """
22         # text fuer Create bauen (x float, y0 float,
y1 float, ... )
23         if len(di.x) > 0:
24             text_create = "(" + di.x[0] + " float, "
25             for i in range(di.Anzahl_Spalten - 1):
26                 text_create += di.y[i + 1][0] + "
float, "
27                 if tablename == 'test':
28                     text_create = text_create[:-7]
29                     text_create += "string, "
30                     text_create = text_create[:-2]
31                     text_create += ")"
32             else:
33                 print("Array ist leer. kann nicht
ausgeführt werden")

```

```

34         pass
35
36     rowsintable = 0      # Tabelle als leer
37     vorbelegen
38
39     try:
40         connection.execute(sql.text("CREATE TABLE
41             " + tablename + " " + text_create))    # Tabelle
42             einfgen
43     except sqlOpErr:
44         print("Tabelle existiert schon")
45         result = connection.execute(sql.text(
46             "SELECT * FROM " + tablename))
47         for row in result:
48             rowsintable += 1  # i > 0 → Tabelle
49             ist schon befüllt  # prüfen, ob Tabelle befüllt ist
50             , wenn sie existiert
51         self.rows = rowsintable
52
53     def create_table_id4(self, tablename, di,
54     connection):
55         """
56             Die Funktion erzeugt die Tabelle für die 4
57             idealen Funktionen,
58             nötig weil das Format anders ist als oben
59             :param tablename: übergebener Tabellenname
60             :param di: Inhalt für die Tabelle
61             :param connection: Verbindung zur Datenbank
62             """
63
64         # text fuer Create bauen (x float, y0 float,
65         y1 float, ... )
66         text_create = "(" + di.y[1][0] + " float
67         ,
68             for i in range(1, di.Anzahl_Spalten):
69                 text_create += di.y[i][3][2] + " float, "
70                 text_create = text_create[:-2]
71                 text_create += ")"
72
73         rowsintable = 0 # Tabelle als leer vorbelegen
74
75         try:

```

```

65         connection.execute(sql.text("CREATE
66             TABLE " + tablename + " " + text_create))
67     except sqlOpErr:
68         print("Tabelle existiert schon")
69         result = connection.execute(sql.text(""
70             "SELECT * FROM " + tablename))
71         for row in result:
72             rowsintable += 1 # i > 0 → Tabelle
73             ist schon befüllt
74         self.rows = rowsintable
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98

```

ist schon befüllt

Funktion befüllt die Tabelle

:param tablename: Übergebener Tabellenname

:param di: Inhalt für die Tabelle

:param connection: Verbindung zur Datenbank

"""

rowsintable = self.rows

if rowsintable == 0:

TEXT_INSERT bauen (x, y1, y2, ...)

text_insert = "(" + di.x[0] + ", "

for i in range(di.Anzahl_Spalten - 1):

text_insert += di.y[i + 1][0] + ", "

text_insert = text_insert[:-2]

text_insert += ")"

,

TEXT VALUES bauen VARIABEL (:x, :y1

, :y2, ...)

text_values = ":" + di.x[0] + ", :"

for i in range(di.Anzahl_Spalten - 1):

text_values += di.y[i + 1][0] +

", :"

text_values = text_values[:-3]

text_values += ")"

for j in range(1, len(di.x)): # j steht
für die Zeilen

```

99                     dict2 = {di.x[0]: di.x[j]}
100                     for i in range(1, di.Anzahl_Spalten
101 ): # i steht für die Spalten
102                         dict2.update({di.y[i][0]: di.y[i
103 ][j]}) # dictionary bauen
104                     connection.execute(sql.text("INSERT
INTO " + tablename + " " + text_insert + " VALUES "
+ text_values), [dict2])
105                     connection.commit()
106
107
108
109     def data_to_table_id4(self, tablename, di,
connection):
110         """
111             Funktion befüllt die Tabelle der 4 idealen
Funktionen
112             :param tablename: übergebener Tabellenname
113             :param di: Inhalt für die Tabelle
114             :param connection: Verbindung zur Datenbank
115         """
116         rowsintable = self.rows
117
118         if rowsintable == 0:
119             ##### TEXT_INSERT bauen (x, y1, y2
, ...)
120             text_insert = "(" + di.y[1][0][0] + ", "
# 'x'
121             # for i in range(di.Anzahl_Spalten - 2):
122                 for i in range(1, di.Anzahl_Spalten
): # 'ynr' Nummer der Funktion, z.B. y36
123                     text_insert += di.y[i][1][0] + ", "
124                     text_insert = text_insert[:-2]
125                     text_insert += ")"
126
127             ##### TEXT VALUES bauen VARIABEL (:x
, :y1, :y2, ...)
128             text_values = ":" + di.y[1][0][0] +
", :"

```

```
129          # for i in range(di.Anzahl_Spalten - 2):
130          for i in range(1, di.Anzahl_Spalten):
131              text_values += di.y[i][1][0] + ", :"
132              text_values = text_values[:-3]
133              text_values += ")"
134
135          for j in range(1, len(di.y[1][0])): # j
136              steht für die Zeilen
136          dict2 = {di.y[1][0][0]: di.y[1][0][j]
137          } # 'x: x-Wert'
137          for i in range(1, di.Anzahl_Spalten
137          ): # i steht für die Spalten; 4 Spalten für die 4
137          idealen Funktionen -
138          dict2.update({di.y[i][1][0]: di.
138          y[i][1][j]}) # dictionary zeilenweise bauen
139          connection.execute(sql.text("INSERT
139          INTO " + tablename + " " + text_insert + " VALUES "
139          + text_values), [dict2]) # an Datenbank senden
140          connection.commit()
141      else:
142          print("Tabelle bereits befüllt")
143
144
```

```
1 class NoMatchNotFoundError(Exception):
2     def __init__(self):
3         message = "Es konnte eine Übereinstimmung
4             gefunden werden"
5         self.message = message
```

```

1 import math
2 from Lib.NoMatchFoundError import NoMatchFoundError
3
4
5 class lookupthings():
6     def __init__(self):
7         self.x = [[],[],[],[], []]
8         self.y
9         = [[[[],[],[]], [[],[],[],[]], [[],[],[],[]], [[],
10            [],[],[]], [[],[],[],[]]]]
11         self.criterion = 0.25
12         self.criterion_Test_in_ideal4 = 0.7
13         self.Anzahl_Spalten = 5
14
15     def lookup_train_in_ideal(self, source,
16         destination):      # source: Instanz train_data;
17         destination: Instanz ideal_data
18         """
19             zu den trainingsdaten passende Funktionen in
20             den 50 idealen Funktionen finden
21             :param source: Quelldatei -> trainings Daten
22             :param destination: Zielfile -> ideal Daten
23             """
24
25     # Header schreiben
26     self.x[0] = 'x'
27     for i in range(1, 5):
28         self.y[i][0].append('x')
29         self.y[i][1].append('y')
30         self.y[i][2].append('Differenz')
31         self.y[i][3].append('QuellFunktion')
32
33     # alle yse in Quelle (train, y1 bis y4)
34     for pos_outer_src in range(1,len(source.y)):
35
36         # alle x in Quelle (train_data, x 1 bis
37         400)
38         for pos_inner_dest in range(1, len(source
39             .x)):
40             pos_inner_src = pos_inner_dest

```

```

35          # alle y in Ziel (50 ideale) in x
36          Richtung durchlaufen 1 - 50
37          for pos_outer_dest in range(1, len(
38              destination.y)):
39              middleValue = (destination.y[
40                  pos_outer_dest][pos_inner_dest] + source.y[
41                      pos_outer_src][pos_inner_src])/2      # Mittelwert
42              difference = math.sqrt(0.5 * ((
43                  source.y[pos_outer_src][pos_inner_src] - middleValue
44                  )**2 + (destination.y[pos_outer_dest][pos_inner_dest]
45                  ] - middleValue)**2))      # quadr. mittlere
46              Abweichung
47              if difference <= self.criterion:
48                  self.x[pos_outer_src].append(
49                      source.x[pos_inner_src])
50                  self.y[pos_outer_src][0].
51                  append(source.x[pos_inner_src])      # x-Wert
52                  schreiben in y Liste
53                  self.y[pos_outer_src][1].
54                  append(destination.y[pos_outer_dest][pos_inner_dest]
55                  ])      # y-Wert schreiben
56                  self.y[pos_outer_src][2].
57                  append(difference)      # Differenz schreiben
58                  self.y[pos_outer_src][3].
59                  append(destination.y[pos_outer_dest][0])      # Name
60                  der Quellfunktion
61                  pass
62                  # if Wert <= Sqrt(2)
63                  pass
64                  # y dest
65                  pass
66                  # x_xrc
67                  pass
68
69                  self.cleanup()  # bereinigen
70
71                  print("DONE LOOKUP")
72
73
74
75      def lookup_test_in_ideal4(self, source,
76          destination):

```

```

56      """
57          testdaten mit den 4 gefundenen idealen
58          Funktionen vergleichen
59          :param source: testdaten
60          :param destination: 4 ideale funktionen
61          """
62          self.x = []
63          self.y = [[], [], [], []]
64          # Ueberschriften anlegen
65          self.y[0].append('x')
66          self.y[1].append('y')
67          self.y[2].append('Differenz')
68          self.y[3].append('QuellFunktion')
69          found = False
70
71          # lenFileContent = len(source.file_content)
72          for i in range(len(source.file_content)):
73              xTest = source.file_content[i][0]    # x
74              Wert suchen
75
76              for j in range(1, len(destination.y[0])):
77                  xDest = destination.y[0][j]
78                  found = False
79                  if destination.y[0][j] == xTest:
80                      for z in range(1, len(destination
81 .y)): # für alle y se in y bei x
82                          yTest = source.file_content[i
83 ] [1]
84                          middleValue = (destination.y[
85 z][j] + source.file_content[i][1]) / 2.0
86                          difference = math.sqrt(0.5
87 * ((source.file_content[i][1] - middleValue) ** 2
88 + (destination.y[z][j] - middleValue) ** 2)) # quadr. mittlere Abweichung
89                          try:
90                              if difference < self.
91 criterion_Test_in_ideal4:
92                                  self.y[0].append(
93 xTest)
94                                  self.y[1].append(
95 yTest)

```

```

86                         self.y[2].append(
87             difference)
88             self.y[3].append(
89             destination.y[z][0])
90             found = True
91         elif found == False and
92             z == len(destination.y)-1:
93             raise
94             NoMatchFoundError      # benutzerdefinierte Exception
95         except NoMatchFoundError:
96             print(NoMatchFoundError
97             (.message))
98         pass
99     else:
100        # print("else Pfad")
101        if xDest > xTest:
102            break      # Schleife
103            beenden, wenn x aus Destination > als x aus
104            Testdaten, da Daten sortiert sind
105        pass
106    pass
107    self.x = self.y[0]
108    self.Anzahl_Spalten = len(self.y)
109
110    def cleanup(self):
111        """
112            Funktion bestimmt die am häufigsten
113            gefundene ideale Funktion und löscht die nicht
114            zugehörigen
115        """
116        i = 0
117        for j in range(1, len(self.y)):      #
118            durchlaufe alle y 1-5
119            # die Funktion mit den meisten Matches
120            ermitteln
121            zwischending = [[], []]
122            for z in range(1, 51): # bei 1 starten
123                wegen Namen y1 bis y50

```

```

115                     zwischending[0].append('y' + str(z
116 ))      # interimsarray mit Funktionsnamen
117             zwischending[1].append(self.y[j][3].
118 count('y' + str(z)))    # zugehörige Anzahl der
119             gefundenen Matches
120
121             max_value = max(zwischending[1])    #
122             Maximale Anzahl
123             max_value_index = zwischending[1].index(
124             max_value) # Index der maximalen Anzahl
125             search_value = zwischending[0][
126             max_value_index]    # hier steht die ermittelte am
127             häufigsten gefundene Funktion
128
129             for i in range(len(self.y[j][3]), 1, -1
130 ):    # lösche die anderen Werte der Funktionen, die
131             nicht am häufigsten enthalten sind
132             if self.y[j][3][i-1] != search_value
133 :    # zaehle von hinten her
134             for todelete in range(len(self.y
135 [j])):
136                 del self.y[j][todelete][i-1
137 ]    # von i muss 1 subtrahiert werden, da hier der
138             Index benötigt wird in i ist aber die gesamte Anzahl
139             an Stellen enthalten
140
141             # Ab hier cleanup um Funktionen createTable
142             und dataToTable nutzen zu können -> umformatieren in
143             eindimensional (x) und zweidimensional (y)
144             for i in range(1, 5):          #
145             Funktionsnamen in y Spalte Platz [0] schreiben als
146             Überschrift
147             self.y[i][1][0] = self.y[i][3][1]
148
149             self.x = []      # x löschen, indem neu
150             definiert wird
151             self.x = self.y[1][0]  # es werden die x-
152             Werte aus y[0] in neues x geschrieben
153
154             for i in range(5):
155                 del self.y[i][3]      # lösche

```

```
135 Quellfunktion
136         del self.y[i][2]          # lösche
    Differenz
137
138         self.y[0][0] = self.y[1][0]      # x Werte
    in nulltes Array schreiben
139         del self.y[0][1]
140         self.y[0] = self.y[0][0]
141         del self.y[1][0]
142         self.y[1] = self.y[1][0]
143         del self.y[2][0]
144         self.y[2] = self.y[2][0]
145         del self.y[3][0]
146         self.y[3] = self.y[3][0]
147         del self.y[4][0]
148         self.y[4] = self.y[4][0]
149
150     print("DONE cleanup")
```

```

1 import unittest
2 from Data_import import DataToImport
3 import os
4
5
6 class UnitTestImport(unittest.TestCase):
7     def test_extract_tablename(self):
8         ''' Rückgabewert des Dateinamens prüfen '''
9         print("START TEST Tabellenname aus Pfad
extrahieren")
10        testpath = "C:\\Data\\Unterordner_1\\\
Unterordner_2\\Tabelle.csv"
11        print("Testpfad: " + testpath)
12        result = DataToImport.extract_tablename(self
, testpath)
13        print("Extrahierter Name: " + result)
14        self.assertEqual(result, 'Tabelle', "Der
Dateiname ist Tabelle")
15        print("Finish Test Tabellenname")
16
17
18    def test_separate_lines(self):
19        ''' Rückgabe Soll: zwei Werte x und y '''
20        ''' Eingabe: ein zweispaltiger Wert und die
Anzahl der Spalten '''
21        print("START TEST Zeilen separieren")
22        testvalue = [[['x', 'y1', 'y2', 'y3'], [1.0, 2
.0, 3.0, 4.0], [5.0, 6.0, 7.0, 8.0], [9.0, 10.0, 11.0
, 12.0]]] # Wert zum "zerlegen"
23        print("Testmatrix: " + str(testvalue))
24        testspalten = 4
25        testresult_1 = ['x', 1.0, 5.0, 9.0]
26        testresult_2 = [[[], ['y1', 2.0, 6.0, 10.0], [
'y2', 3.0, 7.0, 11.0], ['y3', 4.0, 8.0, 12.0]]]
27        result1, result2 = DataToImport.
separate_lines(self, testvalue, testspalten)
28        print("Ergebnis 1: " + str(result1))
29        print("Ergebnis 2: " + str(result2))
30        self.assertEqual(result1, testresult_1, "
Result 1 OK")
31        self.assertEqual(result2, testresult_2, "

```

```
31 Result 2 OK")
32     print("Finish Test separieren")
33
34
35
36 if __name__ == '__main__':
37     unittest.main()
```

```

1 from matplotlib import pyplot as plt
2 from matplotlib import style
3
4
5 class ShowInPlot():
6     def __init__(self):
7         x_axis = 'X AXIS'           # Name x-Achse
8         y_axis = 'Y AXIS'           # Name y_Achse
9         title = 'TITEL'             # Diagrammtitel
10
11
12     def show_id4_plus_test(self, id4, test, train,
13                           x_name, y_name, title):
14         data_name = [] # label in Diagramm
15
16         for i in range(id4.Anzahl_Spalten):
17             if type(id4.y[i][0]) is str:
18                 data_name.append(id4.y[i][0])
19                 del id4.y[i][0]
20
21             style.use('ggplot')
22             fig, ax = plt.subplots(1, 1)    # Diagramm
23             mit einer Spalte und einer Zeile
24             ax.grid(True, color="b")      # Gitterfarbe und
25             Gitter zeigen
26             ax.set_xlabel(x_name)        # x-
27             Achsenbeschriftung
28             ax.set_ylabel(y_name)        # y-
29             Achsenbeschriftung
30             ax.set_title(title)          # Diagrammtitel
31             for i in range(1, id4.Anzahl_Spalten):
32                 # zeige y Daten
33                 ax.scatter(id4.y[0], id4.y[i], label=
34 data_name[i], marker='+',
35                         alpha=0.2) # edgecolors='r
36                 # alpha = Durchsichtigkeit label=title+'_'+str(i),
37                 ax.legend()
38
39
40
41
42     ''' test Daten darstellen '''
43

```

```

34         ax.scatter(test.y[0], test.y[1], label='test
35             data', marker='o', color='black',
36             alpha=0.6) # , marker='o',
37             color='0', edgecolors='r', alpha=0.2) # alpha =
38             Durchsichtigkeit label=title+'_'+str(i),
39             ax.legend()
40
41     ''' neues Diagramm mit ideal 4 und train data
42     ...
43
44         fig2, ax2 = plt.subplots(1, 1) # Diagramm
45         mit einer Spalte und einer Zeile
46         ax2.grid(True, color="b") # Gitterfarbe und
47         Gitter zeigen
48         ax2.set_xlabel(x_name) # x-
49         Achsenbeschriftung
50         ax2.set_ylabel(y_name) # y-
51         Achsenbeschriftung
52         ax2.set_title('ideal 4 vs train data') # 
53         Diagrammtitel
54         for i in range(1, id4.Anzahl_Spalten): # 
55         zeige y Daten
56             ax2.scatter(id4.y[0], id4.y[i], label=
57             data_name[i], marker='+', alpha=0.2)
58             ax2.legend()
59
60     ''' train data in Diagramm 2 '''
61     data_name = [] # label in Diagramm
62
63
64     for i in range(train.Anzahl_Spalten):
65         if type(train.y[i][0]) is str:
66             data_name.append(train.y[i][0])
67             del train.y[i][0]
68
69     for i in range(1, train.Anzahl_Spalten): # 
70     zeige y Daten
71         ax2.scatter(train.y[0], train.y[i], label
72         =data_name[i], marker='.', alpha=0.3)
73         ax2.legend()

```

```
62
63     fig.show()
64     plt.savefig("ideal4_vs_train.png")
65
66     print("DONE Plotting")
67
```

1 https://github.com/AndiMue/IU_PwP_Hausarbeit.git