Kurs: Programmieren mit Python, DLMDWPMP01

Hausarbeit

Tutor: Dr. Thomas Kopsch

Verfasser: Andreas Müller

11.01.2024

Inhaltsverzeichnis

1. Aufgabenstellung

Gegeben drei csv Dateien:

train.csv: Diese enthält 4 Funktionen (x, y1, y2, y3, y4)

test.csv: Diese enthält einen Datensatz (x, y)

ideal.csv: Diese enthält 50 ideale Funktionen (x, y1, y2, y3, …, y50)

Es sollen die Trainingsdaten (train.csv) aus der csv Datei gelesen und in eine SQLite Datenbank auf ein Tabellenblatt geschrieben werden, ebenso soll mit den Daten aus ideal.csv verfahren werden. Verwendet wird die selbe Datenbank, aber ein neues Tabellenblatt.Es gilt weiterhin zu den in train.csv enthaltenen Daten die idealen zugehörigen Funktionen in ideal.csv zu finden und diese ebenfalls in ein Tabellenblatt der Datenbank geschrieben werden.

Die 4 gefundenen idealen Funktionen werden bildlich über die Trainingsdaten gelegt und so visualisiert. Mit den Testdaten wird ebenso verfahren. Diese werden bildlich über die 4 gefundenen idealen Funktionen gelegt.

2. Programm

2.1 Main

Es wird nur auf die wesentlichsten Merkmale eingegangen.

```
19  DataBaseName = "IU_Hausarbeit.db"
20
21  engine = sql.create_engine("sqlite+pysqlite:///" + DataBaseName, echo=True)
22  connection = engine.connect()
23
```

Quelle: [5]

Es wird er Dateiname der Datenbank festgelegt und die Datenbank angelegt und die Verbindung hergestellt.

```
23
24  train_data = DataToImport()
25  ideal_data = DataToDatabase()
26
```

Quelle: [5]

Es werden Instanzen angelegt für die Trainingsdaten (train_data) und die idealen Daten (ideal_data). Die Klasse DataToDatabase ist eine Kindklasse der DataToImport Klasse. Darauf wird später noch kurz eingegangen. Vererbung war eine Forderung der Aufgabenstellung.

```
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
```

Quelle: [5]

Dieser Teil importiert die Daten aus ideal.csv, legt eine Tabelle in der Datenbank an und schreibt diese Daten in das eben erstellte Tabellenblatt. Die Extraktion des Dateinamens mag umständlich und überflüssig erscheinen. Dem kann der Autor an dieser Stelle nur recht geben. Zur Zeit des Entwurfs wurde mit absoluten Pfaden gearbeitet und so aus dem kompletten Pfad, der in einer eigenen Variablen stand, der Dateiname extrahiert und als Name für das Tabellenblatt weiter verwendet. Es wurde hier die ursprüngliche Struktur bei behalten.

```
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
```

Quelle: [5]

Ebenso wird mit den Trainingsdaten aus train.csv verfahren. Der aufmerksame Leser bemerkt sofort den Unterschied, dass hier eine weitere Instanz angelegt werden muss (Zeile 36), damit die Daten in die Datenbank geschrieben werden können. Dies ist hier nötig, weil ohne Vererbung gearbeitet wird.

```
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( tablename: 'four_of_ideal', ideal_4_data, connection)
45  ideal_4_data_to_DB.data_to_table( tablename: 'four_of_ideal', ideal_4_data, connection)
46
```

Quelle: [5]

Mit der Instanz von lookupthings() werden die 4 Funktionen in ideal gesucht, die am besten zu den

Trainingsdaten passen. Diese werden dann ebenfalls in die Datenbank auf ein Tabellenblatt, four_of_ideal, geschrieben.
Mit den Testdaten wird ebenso verfahren wie mit den idealen Daten. Es wird hier ebenfalls eine Instanz von lookupthings() angelegt. Alternativ könnte man auch sicherheitshalber die Daten der vorherigen Instanz löschen und diese weiter verwenden.
Es werden die gefundenen idealen 4 Funktionen durchsucht nach passenden Ergebnissen zu den Testdaten.

Zum Schluss werden die 4 gefundenen Funktionen über die passenden idealen Daten visualisiert und die Trainingsdaten über die gefundenen vier Funktionen. So ist sehr leicht ersichtlich, dass das gefundene Ergebnis richtig ist.

2.2 Klassen

2.2.1 Beschreibung Data_import

Die Klasse DataToImport besteht im Wesentlichen aus drei Methoden.
extract_tablename liest aus einem gegeben Pfad den Dateinamen, dieser wird später als Name für das Tabellenblatt weiter verwendet.
import liest die Daten aus der jeweiligen csv Datei aus, konvertiert diese in float und schreibt diese in eine Liste file_content. Sollte die Datei nicht vorhanden sein, wird eine Ausnahmebehandlung ausgelöst. Da die csv Datei test.csv unsortierte Daten enthält, werden diese nach den x-Werten sortiert. Wenn die Daten nicht nach float konvertiert werden können, werden diese gelöscht. Dazu wird dann auch eine Meldung ausgegeben.
Die Methode separate_lines trennt die importierte Liste f_content in x- und y- Werte auf. Dies wurde vom Autor als sinnvoll erachtet. Dazu wird der erste Werte als x angesehen und stellt einen string dar: 'x' und wird nicht nach float konvertiert, geht auch nicht, wenn dann der Index größer als null ist, werden die nachfolgenden Werte in float konvertiert und ein Gültigkeitsstatus auf true gesetzt. Sollte dies nicht möglich sein, wird eine Meldung ausgegeben.Der Status bleibt dann false.
Wenn der x-Wert erfolgreich verarbeitet werden konnte, werden die y-Werte versucht zu konvertieren und an die y Liste anzuhängen. Kann ein y-Wert nicht erfolgreich verarbeitet werden, wird eine Meldung ausgegeben und der zugehörige x-Wert aus der x-Liste gelöscht (Zeile 65), ebenso die übrigen zugehörigen y-Werte.
Am Ende wird der Gültigkeitsstatus für den erfolgreichen Import von x wieder false gesetzt für den nächsten Durchlauf, solange ein Index kleiner als die Länge von f_content.
Die Methoden import und separate_lines wurden inspiriert von [2] und entsprechend und

umfangreich angepasst und abgeändert.

2.2.2 Beschreibung DataWriteToDB

Die Klasse DataToDatabase() besitzt vier Methoden, je zwei um ein Tabellenblatt zu erzeugen und zwei um das Tabellenblatt zu befüllen. Dies ist hier notwendig, weil die Dateninhalte „di“ sich unterscheiden zwischen den idealen vier Funktionen und importierten Daten. Die Funktionsweise ist aber identisch, daher werden nur die Methoden create_Table(self, tablename, di, connection) und data_to_table(self, tablename, di, connection) beschrieben.
create_Table(self, tablename, di, connection) erzeugt in der datenbank ein Tabellenblatt. Die Übergabaparameter sind der Name des Tabellenblattes (tablename), die importierten Daten (di) und die Verbindung zur Datenbank.
Es wird zunächst der Testbaustein zum erzeugen des Tabellenblattes zusammengebaut:

```
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")
```

Quelle: [5]
Es wird noch unterschieden, ob das Tabellenblatt „test“ erzeugt werden soll, hier muss zu den idealen Daten beispielsweise etwas unterschieden werden.
Ergebnis beispielsweise für die idealen Daten:
(x float, y1 float, y2 float, y3 float, y4 float, y5 float, y6 float, .., y50 float)
Damit wird im nächsten Abschnitt das Tabellenblatt angelegt:

```
38          try:
39              connection.execute(sql.text("CREATE TABLE " + tablename + " " + text_create))   # Tabelle
40          except sqlOpErr:
41              print("Tabelle existiert schon")
42              result = connection.execute(sql.text("SELECT * FROM " + tablename))
43              for row in result:
44                  rowsintable += 1  # i > 0 → Tabelle ist schon befüllt   # prüfen, ob Tabelle befüllt i
45          self.rows = rowsintable
```

Quelle: [5]
Wenn das Tabellenblatt in der Datenbank bereits existiert, wird eine Meldung ausgegeben: „Tabelle

existiert schon“ und keine weitere Tabelle angelegt.
Die Anzahl der Spalten muss zwischengespeichert werden, damit im nächsten Schritt die Tabelle auch richtig befüllt werden kann, das passier nur, wenn die Spaltenanzahl größer als null ist, ansonsten macht das Befüllen auch keinen Sinn.
Es wird ebenso zuerst der Textbaustein zum Befüllen erzeugt und danach versucht die Daten in die Tabelle zu schreiben.

```
83          if rowsintable == 0:
84              # TEXT_INSERT bauen (x, y1, y2, ...)
85              text_insert = "(" + di.x[0] + ", "
86              for i in range(di.Anzahl_Spalten - 1):
87                  text_insert += di.y[i + 1][0] + ", "
88              text_insert = text_insert[:-2]
89              text_insert += ")"
90
91              # TEXT VALUES bauen VARIABEL  (:x, :y1, :y2, ...)
92              text_values = "(:" + di.x[0] + ", :"
93              for i in range(di.Anzahl_Spalten - 1):
94                  text_values += di.y[i + 1][0] + ", :"
95              text_values = text_values[:-3]
96              text_values += ")"
97
98              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):  # i steht für die Spalten
101                     dict2.update({di.y[i][0]: di.y[i][j]})  # dictionary bauen
102                 connection.execute(sql.text("INSERT INTO " + tablename + " " + text_insert + " VALUES " + text_values), [dict2])
103             connection.commit()
104         else:
105             print("Tabelle bereits befüllt")
```

Quelle: [5]

2.2.3 Beschreibung NoMatchFoundError

Diese Klasse bildet eine Ausnahmebehandlung ab. Sie wird aufgerufen aus Search_Function.py (Zeile 90) aufgerufen, wenn keine Übereinstimmung gefunden werden kann, bei der die ermittelte Differenz kleiner als das Kriterium ist, heißt, der Punkt außerhalb einer der idealen vier Funktionen. Ausgabe ist eine Fehlermeldung

2.2.4 Beschreibung Search_Function

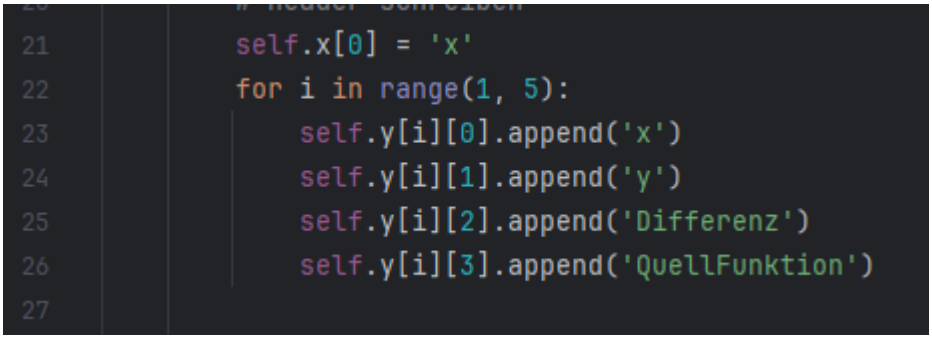
Die Klasse lookupthings beinhaltet drei Methoden:

lookup_train_in_ideal,
lookup_test_in_ideal,
cleanup.

Die beiden Methoden lookup_train_in_ideal und lookup_test_in_ideal arbeiten analog, daher wird nur eine Methode beschreiben. Der Unterschied besteht in einem formalen Unterschied der Eingangsdaten und dem Kriterium.
lookup_train_in_ideal sucht die passenden Funktionen für die Trainingsdaten in den idealen

Funktionen. Daher wird dieser Methode eine Quelle für das Suchen und ein Ziel, in dem gesucht wird übergeben, hier: source: train_data und destination: ideal_data

Die gefundenen Funktionen sollen in eine Tabelle der Datenbank geschrieben werden, daher wird zuerst der Header, die erste Zeile gebildet:



```
21          self.x[0] = 'x'
22          for i in range(1, 5):
23              self.y[i][0].append('x')
24              self.y[i][1].append('y')
25              self.y[i][2].append('Differenz')
26              self.y[i][3].append('QuellFunktion')
27
```

Quelle: [5]

Danach wird das Ziel zeilenweise durchsucht. Es wird ein Mittelwert berechnet aus Quelle und Ziel. Aus Quelle und Ziel dann eine quadratische Differenz. Diese Differenz muss kleiner dem Kriterium sein, um die Abweichung als gültig anzuerkennen.

Ist dies der Fall, werden folgende Werte als x- und y-Wert abgespeichert:

x-Wert

x-Wert aus Quelldaten

Differenz

Name der als gültig erkannten Funktion (z.B. y36).

Anschließend wird noch ein cleanup durchgeführt

Es wird der Fall sein, dass eine oder mehrere Funktionen als gültig erkannt werden. Daher wird das Ergebnis gesäubert. Es wird die Funktion ermittelt, die am häufigsten ein gültiges Ergebnis geliefert hat, alle anderen Werte werden entfernt.

Damit ist die Berechnung fertig.

2.2.5 Beschreibung Test_Data_import

Die Unittestklasse beinhaltet eigentlich zwei Tests. Der erste Test ist die Prüfung, ob ein Dateiname richtig aus einer Pfadangabe extrahiert wird. Gegeben ist testpath und daraus soll der Dateiname gefunden werden. Der Test ist für „Tabelle“ positiv.

Der zweite Test separiert eine eingelesene Matrix in x und y Werte. Die Testmatrix testvalue besteht aus vier Spalten und vier Zeilen. In dieser Form werden die Daten auch aus den csv Dateien ausgelesen. Diese Daten werden nun separiert in die x- und zugehörigen y-Werte. Der Test fällt positiv aus, wenn aus der Testmatrix (Zeile 22) die Werte richtig extrahiert werden (Zeilen 25, 26).

Es wurde beispielhaft ein Unittest geschrieben. Weitere Unittests funktionieren analog.

2.2.6 Beschreibung Visualize

Mittels der Klasse „ShowInPlot“ werden die Ergebnisse dargestellt. Diese besteht im Wesentlichen aus einer Methode, show_id4_plus_test. Dieser Name ist historisch gewachsen und könnte nun auch umbenannt werden, da es nur diese eine Methode und nicht mehrere für die Darstellung der Ergebnisse gibt. Es werden Daten übergeben:

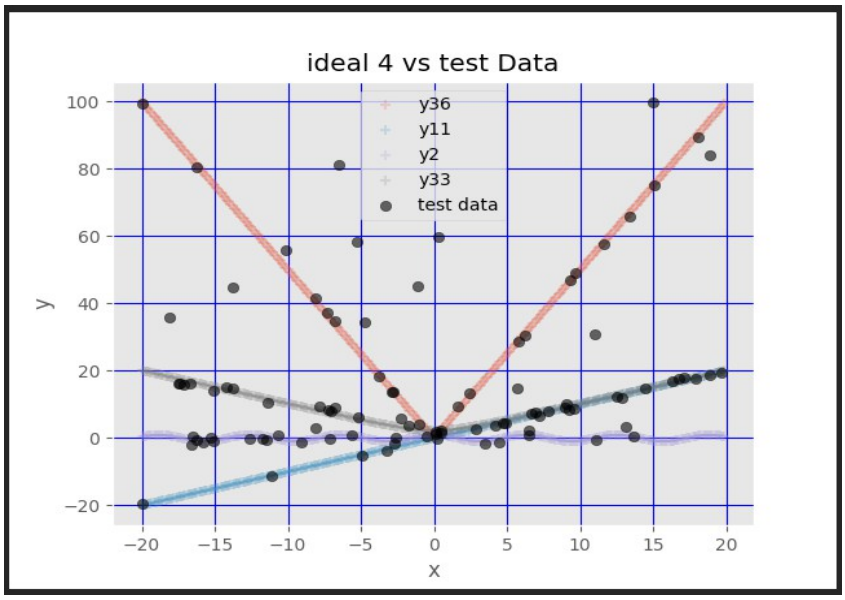
die idealen vier Funktionen, id4,
die Testdaten, test,
die Trainingsdaten, train,
x-Achsenbeschriftung, x_name,
y-Achsenbeschriftung, y_name,
Titel, title.

In Zeile 15 bis 18 werden die Kopfdaten aus id4 gelesen, wenn diese ein String sind, an data_name angehangen und aus id4 gelöscht, [5]:

```
15          for i in range(id4.Anzahl_Spalten):
16              if type(id4.y[i][0]) is str:
17                  data_name.append(id4.y[i][0])
18                  del id4.y[i][0]
```

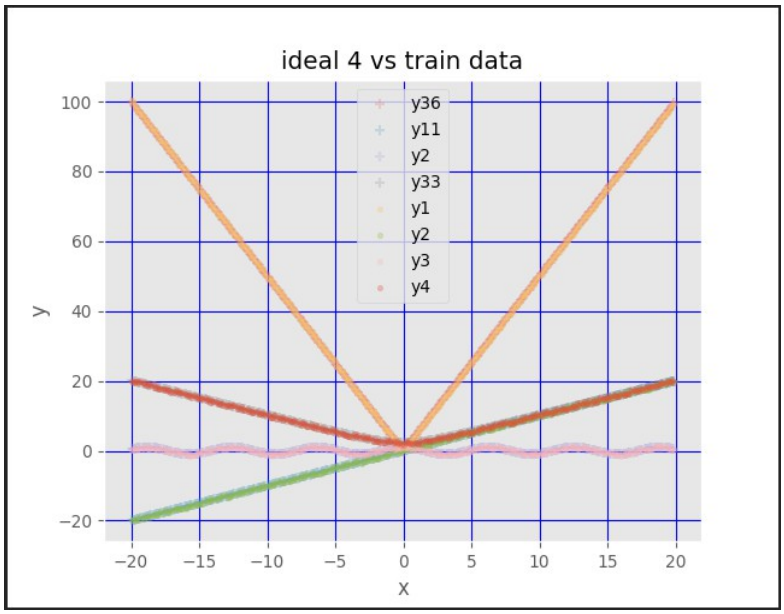
Quelle: [5]
Dies dient später der Beschriftung.
test und train haben ein etwas anderes Datenformat, hier ist dies nicht notwendig.

Ab Zeile 20 wird das erste Diagramm erzeugt und abgespeichert, hier das Ergebnis:



Quelle: [6]

Darstellung der Test-Daten (schwarze Punkte) und der gefundenen idealen vier Funktionen (rot, grau, lila und blau). Es ist leicht zu erkennen, dass bis auf einzelne Punkte die Daten gut an diese Funktionen passen.
Auf der folgenden Seite werden die idealen vier Funktionen (y36, y11, y2, y33) mit den Trainingsdaten verglichen (y1, y2, y3, y4).
Es ist gut zu erkennen, dass y33 gleich mit y1 ist, y11 mit y2, y2 mit y3 und y33 mit y4.
Beide Diagramme werden im Root Ordner abgespeichert.



Quelle: [6]

3. Zusammenfassung

Abschließend kann gesagt werden, dass die Berechnung der gültigen Daten etwas anders erfolgt als in der Aufgabenstellung vorgeschlagen. Dies rührt daher, dass die Aufgabenstellung dem Autor sehr uneindeutig formuliert ist. Das ist auch an manchen Fragen im Course Feed zu sehen, dass der Autor an dieser Stelle nicht alleine ist. Die Formulierung sollte besser sein. Das Ergebnis ist aber das Selbe und daher der Berechnungsweg nicht „kriegsentscheidend“.
Ebenso wurden keine Pandas verwendet, da dem Autor nicht zu 100% der Vorteil gegenüber der verwendeten Berechnung klar war. Die Formulierung „...solltest Du Pandas benutzen...“ (Quelle: [4]) legt auch nicht nahe, dass Pandas benutzt werden müssen, denn dann muss „solltest“ durch „musst“ ersetzt werden.

4. Anhang

4.1 Aufgabenstellung

Wesentlicher Auszug der Aufgabenstellung [4]:

„1. AUFGABENSTELLUNG

Für die Hausarbeit steht folgende Aufgabenstellung zur Verfügung.

Als Ausgangsbasis für die Hausarbeit dient zunächst das Studienskript, dessen Inhalte als Basiswissen die Voraussetzung für die vertiefende Betrachtung der nachfolgenden Fragestellung darstellen. Es wird erwartet, dass in der Hausarbeit weitere Literaturquellen zu dieser Fragestellung recherchiert und verarbeitet werden.

1.1. Die Aufgabe

Du erhältst:

A) 4 Training-Datensätze

B) einen Test-Datensatz

C) einen Datensatz, der 50 ideale Funktionen beschreibt

Alle Daten bestehen aus x-y-Paaren. Die Struktur in den CSV-Files ist wie folgt:

x	y
X1	Y1
…	…
Xn	Yn

Deine Aufgabe ist, ein Python Programm zu schreiben, welches mittels der vier Trainingsdatensätze (A) die vier besten Passungen / Fits aus dem Datensatz von 50 idealen Funktionen (C) findet. Die folgenden Kriterien sollen beachtet werden:

1. Das Kriterium zur Selektion idealer Funktionen für den Training-Datensatz ist die Minimierung der

Summe aller quadratischen y-Abweichungen (Least-Square).

2. Dein Programm muss den Test-Datensatz B zur Validierung der Selektion benutzen. Hierbei soll für jedes

x-y-Paar im Test-Datensatz überprüft werden, ob die Werte zu den vier idealen Funktionen passen.

a. Benutze ein Kriterium, welches sicherstellt, dass die maximale Abweichung zwischen der vorher ermittelten idealen Funktion und den Testwerten nicht die maximale Abweichung zwischen den Trainingsdaten (A) und den vier idealen Funktionen aus (C) um mehr als den Faktor Wurzel aus zwei (sqrt(2)) übersteigt.

b. Sollten die Testdaten an die von Dir gefundenen vier Funktionen anpassbar sein, speichere für jeden Testdatensatz die entsprechenden Abweichungen ab.

3. Alle Daten sollten logisch visualisiert werden.

4. Schreibe Unit-Tests, wo immer möglich.

Um Deine im Kurs erlernten Fähigkeiten unter Beweis zu stellen, musst Du die im folgenden Kapitel (Details) dargestellten Kriterien erfüllen.

1.2 Details

Datenbanken und Tabellen

Du erhältst vier Trainingsdatensätze in Form von CSV-Dateien. Dein Python-Programm muss in der Lage sein, eine SQLite-Datenbank (Datei) idealerweise über sqlalchemy unabhängig zu kompilieren und die Trainingsdaten in eine einzelne, fünfspaltige Tabelle zu laden. Die erste Spalte zeigt die x-Werte aller Funktionen. Tabelle 1 am Ende dieses Unterabschnitts zeigt Dir, welche Struktur Deine Tabelle
voraussichtlich haben wird.

• Die fünfzig idealen Funktionen, die auch über eine CSV-Datei bereitgestellt werden, müssen in eine
andere Tabelle geladen werden. Ebenso zeigt die erste Spalte die x-Werte, was bedeutet, dass insgesamt 51 Spalten vorhanden sind. Tabelle 2 am Ende dieses Unterabschnitts beschreibt schematisch, welche Struktur erwartet wird.

• Nachdem die Trainingsdaten und die idealen Funktionen in die Datenbank geladen wurden, müssen die
Testdaten (B) Zeile für Zeile aus einer anderen CSV-Datei geladen und - wenn sie das Kriterium im Unterabschnitt 2 erfüllt - mit einer der vier abgeglichen Funktionen abgespeichert werden.

• Anschließend müssen die Ergebnisse in einer anderen vierspaltigen Tabelle in der SQLite-Datenbank
gespeichert werden. Gemäß Tabelle 3 am Ende dieses Unterabschnitts enthält diese Tabelle vier Spalten mit x- und y-Werten sowie die entsprechend gewählte ideale Funktion und die damit verbundene Abweichung.

• Schließlich werden die Trainingsdaten, die Testdaten, die gewählten Idealfunktionen sowie die entsprechenden / zugewiesenen Datensätze unter einer entsprechend gewählten Darstellung der Abweichung visualisiert.

Struktur des Python Programms

• Das Programm soll soweit wie möglich Objekt-orientiert sein.

• Es soll mindestens eine Vererbungshierarchie (inheritance) haben.

• Benutze sowohl Standard als auch user-definiertes Exception Handling.

• Für die Programmlogik solltest Du Pandas benutzen, aber auch Visualisierung mittels Bokeh, matlibplot
etc.

• Schreibe Unit-Tests, wo immer es sich anbietet.

• Dokumentiere Dein Programm vollständig und mache von docstrings Gebrauch.

Verwendung von Git

• Bitte verwende Git zur Versionskontrolle Deines Codes.

Tabelle 1: Training Daten Datenbank Tabelle

X	Y1 (Training Funktion)	Y2 (Training Funktion)	Y3 (Training Funktion)	Y4 (Training Funktion)
x1	y11	y21	y31	y41
…	…	…	…	…
xn	y1n	y2n	y3n	y4n

Tabelle 2: Tabelle der idealen Funktionen

X	Y1 (Ideale Funktion)	Y2 (Ideale Funktion)	…	Y50 (Ideale Funktion)
x1	y11	y21	…	y41
…	…	…	…	…
xn	y1n	y2n	…	y4n

Tabelle 3: Test-Daten Tabelle

X (Test Funktion)	Y1 (Test Funktion)	Delta Y (Abweichung)	Nummer der Idealen Funktion (z.B. Funk37)
x1	y11	y21	y31
…	…	…	…
xn	y1n	y2n	y3n

1.3. Anmerkungen

Der Datensatz für diese Aufgabe wird auf Anfrage für jeden einzelnen Studenten zur Verfügung gestellt. Daher sollte ein Ticket für den Tutor geöffnet werden, woraufhin der Zugriff auf die Daten gewährt wird. Eine Kopie wird an die verantwortlichen Personen gesendet - so wird eine spätere Manipulation durch die Studierenden verhindert.

Es wird erwartet, dass Dein gesamter Quellcode im Anhang Deiner schriftlichen Aufgabe enthalten ist, damit Dein gesamtes Programm einschließlich der Ausgaben getestet werden kann. Deine Eingabedaten sind nicht erforderlich.

Ziel ist es, Deine Arbeit, Deine Entscheidungen und Deine Einschätzung des Aufgabenergebnisses durch Deine Abgabe vollständig zu rekonstruieren.

2. ZUSATZINFORMATIONEN ZUR BEWERTUNG DER HAUSARBEIT

Bei der Konzeption und Erstellung der Hausarbeit sollten die im Prüfungsleitfaden aufgeführten Bewertungskriterien und Erläuterungen berücksichtigt werden.

Bezüglich Einführung und thematischer Abgrenzung sollte darauf geachtet werden, dass diese im gewählten Lösungsansatz der Aufgabe demonstriert werden.

Die Bewertung der Struktur bezieht sich auf das Design des Programms, Klassenstruktur, Wahl von Verallgemeinerungen im Programm und die Komposition des Programms.

In der Argumentation werden die finale Funktionalität und die Richtigkeit der Ausführung des

Programms bewertet.

Der Abschluss soll einen wissenschaftlich adäquaten Text und eine Diskussion der Vor- & Nachteile des gewählten Lösungsansatzes, im Speziellen eine Diskussion der Abgrenzung zu anderen möglichen Lösungen, der Programmstruktur und der Module und Frameworks, die für die Lösung verwendet wurden, darstellen. “

4.2 Main

```
# Hausarbeit IU Akademie Programmieren mit Python
# Andreas Müller

# Benötigte Importe
from Lib.Data_import import DataToImport
import sqlalchemy as sql
from Lib.DataWriteToDB import DataToDatabase
import os
from Lib.Search_Function import lookupthings
from Lib.Visualize import ShowInPlot


def extract_tablename(path):
    result = os.path.basename(path)
    result = result[:-4]
    return result


DataBaseName = "IU_Hausarbeit.db"

engine = sql.create_engine("sqlite+pysqlite:///" + DataBaseName, echo=True)
connection = engine.connect()

train_data = DataToImport()
ideal_data = DataToDatabase()

''' ideal.csv importieren und in DB bringen'''
ideal_data.importieren("ideal.csv")
Tablename = extract_tablename("ideal.csv")
ideal_data.create_table(Tablename, ideal_data, connection)
ideal_data.data_to_table(Tablename, ideal_data, connection)

''' Trainingsdaten importieren und in DB bringen'''
train_data.importieren("train.csv")
Tablename = extract_tablename("train.csv")
train_data_to_DB = DataToDatabase()
train_data_to_DB.create_table(Tablename, train_data, connection)
train_data_to_DB.data_to_table(Tablename, train_data, connection)

''' Trainingsdaten in ideal.csv suchen '''
ideal_4_data = lookupthings()
ideal_4_data.lookup_train_in_ideal(train_data, ideal_data)
ideal_4_data_to_DB = DataToDatabase()
ideal_4_data_to_DB.create_table('four_of_ideal', ideal_4_data, connection)  # eigtl nicht nötig
ideal_4_data_to_DB.data_to_table('four_of_ideal', ideal_4_data, connection)     # eigtl nicht nötig


''' Testdaten importieren um mit idealen 4 zu vergleichen'''
test_data_to_DB = DataToDatabase()
```

```
test_data_compare = lookupthings()
''' import und Weiterverarbeitung mit Kind-Klasse '''
test_data_to_DB.importieren("test.csv")
Tablename = extract_tablename("test.csv")

test_data_compare.lookup_test_in_ideal4(test_data_to_DB, ideal_4_data)
test_data_to_DB.create_table(Tablename, test_data_compare, connection)
test_data_to_DB.data_to_table(Tablename, test_data_compare, connection)

ShowData = ShowInPlot()
test_data_to_DB.y[0] = test_data_to_DB.x
train_data.y[0] = train_data.x
ShowData.show_id4_plus_test(ideal_4_data, test_data_to_DB, train_data, 'x', 'y', 'ideal 4 vs test Data')

print("Saved plotts to folder")
print("DONE")
```

4.3 Data_import

```
# class for importing data from csv
import os

class DataToImport:
    def __init__(self):
        self.data_path = None
        self.file_content = [None]
        self.x = []
        self.y = []  # []
        self.Anzahl_Spalten = 0  # Zaehler fuer die gesamte Anzahl der Spalten

    def extract_tablename(self, path):
        """
        Funktion extrahiert den Dateinamen aus demDateipfad
        :param path: Dateipfad
        :return: result: Dateiname
        """
        result = os.path.basename(path)
        result = result[:-4]
        return result

    def separate_lines(self, f_content, spalten):
        """
        function seperates the file content in two variables.
        :param f_content: is in two columns, e.g. ['4.234','3.12']
        :spalten: Anzahl der gesamten Spalten
        :return: two seperate variables
        """

        sp = 1  # Spaltenindex, kann bei 1 beginnen, da Spaltenindex die x-Werte enthaelt
        b_x_valid = False   # konnte x gelesen werden?
        x = []

        # Liste fuer y-Werte erzeugen
        y = []
        for i in range(spalten):
            y.append([])

        # print("Laenge file_content: ", len(f_content))
        i = 0     # Laufindex muss zurueck gesetzt werden, von for Schleife
        # solange das Ende des Inhalts nicht erreicht ist lese ein
```

```
    while i < len(f_content):
        # x einlesen, erster Wert muss 'x' sein
        try:
            if i > 0:
                x.append(float(f_content[i][0])) # erste Stelle ist ein string, kein float
            else:
                x.append(f_content[i][0])
            b_x_valid = True
        except:
            print(f"Punkt {i}  kann nicht verarbeitet werden")

        # y einlesen, wenn der x-Wert eingelesen werden konnte

        if b_x_valid:
            for sp in range(spalten-1):
                try:

                    if i > 0:
                        y[sp+1].append(float(f_content[i][sp+1]))
                    else:
                        y[sp+1].append(f_content[i][sp+1])
                except:
                    print(f"Punkt {i} kann nicht verarbeitet werden. Zugehoeriges x wird gelÃ¶scht")
                    del x[i]

                    for j in range(1, spalten):
                        try:
                            del y[j][i]
                        except:
                            print(f"Index {j} | {i} nicht zu loeschen!")
                            sp = sp+1       # Index ans Ende der Spalten setzen, damit diese nicht importiert werden
                    break   # notwendig um die restlichen y-Werte nicht zu lesen
            sp += 1
        sp = 1              # Ruecksetzen fuer naechsten Durchlauf fuer x
        b_x_valid = False   # Gueltigkeitswert ruecksetzen
        i += 1

    return x, y

def importieren(self, data_path):
    """
    function to import a file content
    :param data_path: file to import
    :return: nothing, x and y are self
    """
    try:
        with open(data_path, "r") as csv_import_file:
            for line in csv_import_file.read().split("\n"):
                self.file_content.append(line.split(","))  # = csv_import_file.read()
            self.file_content.__delitem__(0)    # None in erster Zeile lÃ¶schen

        # Anzahl Spalten der Daten zaehlen
        for self.Anzahl_Spalten in range(len(self.file_content[0])):
            self.Anzahl_Spalten += 1
    except:
        print("EXCEPTION importieren, Datei kann nicht geÃ¶ffnet werden")

    # test Daten nach x sortieren
    Tablename = self.extract_tablename(data_path)
    # Tabelle test.csv muss gesondert behandelt werden. Daten werden nach x sortiert.
    if Tablename == 'test':
```

```
            i = 0
            while i < len(self.file_content)-1:
                try:
                    self.file_content[i][0] = float(self.file_content[i][0])
                    self.file_content[i][1] = float(self.file_content[i][1])
                except:
                    print(self.file_content[i])
                    del self.file_content[i]
                    print("konnte string nicht in float konvertieren -> wurde gelÃ¶scht")
                    i -= 1

                i += 1
            del self.file_content[i]
            # Daten werden nach x sortiert.
            self.file_content.sort()

        if self.Anzahl_Spalten > 0:
            self.x, self.y = self.separate_lines(self.file_content, self.Anzahl_Spalten)
        else:
            self.x = []
            self.y = []


        print("importieren DONE")
```

4.4 DataWriteToDB

```
import sqlalchemy as sql
from sqlalchemy.exc import OperationalError as sqlOpErr
from Lib.Data_import import DataToImport


class DataToDatabase(DataToImport):     # erbt Dataimport Klasse

    def __init__(self):

        self.rows = 0

        super().__init__()

    def create_table(self, tablename, di, connection):
        """
        Die Funktion erzeugt eine Tabelle in der Datenank
        :param tablename: Ã¼bergebener Tabellenname
        :param di: Inhalt der Tabelle
        :param connection: Verbindung zur Datenbank
        kann die Tabelle nicht erzeugt werden wird eine Exception ausgelÃ¶st
        """
        # text fuer Create bauen (x float, y0 float, y1 float, ... )
        if len(di.x) > 0:
            text_create = "(" + di.x[0] + " float, "
            for i in range(di.Anzahl_Spalten - 1):
                text_create += di.y[i + 1][0] + " float, "
            if tablename == 'test':
                text_create = text_create[:-7]
                text_create += "string, "
            text_create = text_create[:-2]
            text_create += ")"
        else:
            print("Array ist leer. kann nicht ausgefÃ¼hrt werden")
            pass
```

```
        rowsintable = 0     # Tabelle als leer vorbelegen

        try:
            connection.execute(sql.text("CREATE TABLE " + tablename + " " + text_create))   # Tabelle
einfÃ¼gen
        except sqlOpErr:
            print("Tabelle existiert schon")
            result = connection.execute(sql.text("SELECT * FROM " + tablename))
            for row in result:
                rowsintable += 1  # i > 0 â†’ Tabelle ist schon befÃ¼llt   # prÃ¼fen, ob Tabelle befÃ¼llt ist, wenn
sie existiert
        self.rows = rowsintable

    def create_table_id4(self, tablename, di, connection):
        """
        Die Funktin erzeugt die Tabelle fÃ¼r die 4 idealen Funktionen,
        nÃ¶tig weil das Format anders ist als oben
        :param tablename: Ã¼bergebener Tabellenname
        :param di: Inhalt fÃ¼r die Tabelle
        :param connection: Verbindung zur Datenbank
        """
        # text fuer Create bauen (x float, y0 float, y1 float, ... )
        text_create = "(" + di.y[1][0][0] + " float, "
        for i in range(1, di.Anzahl_Spalten):
            text_create += di.y[i][3][2] + " float, "
        text_create = text_create[:-2]
        text_create += ")"

        rowsintable = 0 # Tabelle als leer vorbelegen

        try:
            connection.execute(sql.text("CREATE TABLE " + tablename + " " + text_create))
        except sqlOpErr:
            print("Tabelle existiert schon")
            result = connection.execute(sql.text("SELECT * FROM " + tablename))
            for row in result:
                rowsintable += 1  # i > 0 â†’ Tabelle ist schon befÃ¼llt
        self.rows = rowsintable


    def data_to_table(self, tablename, di, connection):
        """
        Funktion befÃ¼llt die Tabelle
        :param tablename: Ã¼bergebener Tabellenname
        :param di: Inhalt fÃ¼r 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 fÃ¼r die Zeilen
            dict2 = {di.x[0]: di.x[j]}
            for i in range(1, di.Anzahl_Spalten):  # i steht fÃ¼r die Spalten
                dict2.update({di.y[i][0]: di.y[i][j]})  # dictionary bauen
            connection.execute(sql.text("INSERT INTO " + tablename + " " + text_insert + " VALUES " +
text_values), [dict2])
        connection.commit()
    else:
        print("Tabelle bereits befÃ¼llt")



  def data_to_table_id4(self, tablename, di, connection):
    """
    Funktion befÃ¼llt die Tabelle der 4 idealen Funktionen
    :param tablename: Ã¼bergebener Tabellenname
    :param di: Inhalt fÃ¼r die Tabelle
    :param connection: Verbindung zur Datenbank
    """
    rowsintable = self.rows

    if rowsintable == 0:
        ######## TEXT_INSERT bauen (x, y1, y2, ...)
        text_insert = "(" + di.y[1][0][0] + ", "    # 'x'
        # for i in range(di.Anzahl_Spalten - 2):
        for i in range(1, di.Anzahl_Spalten):      # 'ynr' Nummer der Funktion, z.B. y36
            text_insert += di.y[i][1][0] + ", "
        text_insert = text_insert[:-2]
        text_insert += ")"

        ######## TEXT VALUES bauen VARIABEL  (:x, :y1, :y2, ...)
        text_values = "(:" + di.y[1][0][0] + ", :"
        # for i in range(di.Anzahl_Spalten - 2):
        for i in range(1, di.Anzahl_Spalten):
            text_values += di.y[i][1][0] + ", :"
        text_values = text_values[:-3]
        text_values += ")"

        for j in range(1, len(di.y[1][0])):  # j steht fÃ¼r die Zeilen
            dict2 = {di.y[1][0][0]: di.y[1][0][j]}      # 'x: x-Wert'
            for i in range(1, di.Anzahl_Spalten):  # i steht fÃ¼r die Spalten; 4 Spalten fÃ¼r die 4 idealen
Funktionen -
                dict2.update({di.y[i][1][0]: di.y[i][1][j]})  # dictionary zeilenweise bauen
            connection.execute(sql.text("INSERT INTO " + tablename + " " + text_insert + " VALUES " +
text_values), [dict2])  # an Datenbank senden
        connection.commit()
    else:
        print("Tabelle bereits befÃ¼llt")
```

4.5 NoMatchFoundError

```
class NoMatchFoundError(Exception):
  def __init__(self):
    message = "Es konnte eine Ãœbereinstimmung gefunden werden"
    self.message = message
```

4.6 Search_Funktion

```
import math
from Lib.NoMatchFoundError import NoMatchFoundError


class lookupthings():
    def __init__(self):
        self.x = [[],[],[],[], []]
        self.y = [[[],[],[],[]], [[],[],[],[]], [[],[],[],[]], [[],[],[],[]], [[],[],[],[]]]
        self.criterion = 0.25
        self.criterion_Test_in_ideal4 = 0.7
        self.Anzahl_Spalten = 5

    def lookup_train_in_ideal(self, source, destination):       # source: Instanz train_data; destination: Instanz ideal_data
        """
        zu den trainingsdaten passende Funktionen in den 50 idealen Funktionen finden
        :param source: Quelldatei -> trainings Daten
        :param destination: Zieldatei -> ideal Daten
        """

        # Header schreiben
        self.x[0] = 'x'
        for i in range(1, 5):
            self.y[i][0].append('x')
            self.y[i][1].append('y')
            self.y[i][2].append('Differenz')
            self.y[i][3].append('QuellFunktion')

        # alle yse in Quelle (train, y1 bis y4)
        for pos_outer_src in range(1,len(source.y)):

            # alle x in Quelle (train_data, x 1 bis 400)
            for pos_inner_dest in range(1, len(source.x)):
                pos_inner_src = pos_inner_dest

                # alle y in Ziel (50 ideale) in x Richtung durchlaufen 1 - 50
                for pos_outer_dest in range(1, len(destination.y)):
                    middleValue = (destination.y[pos_outer_dest][pos_inner_dest] + source.y[pos_outer_src][pos_inner_src])/2       # Mittelwert
                    difference = math.sqrt(0.5 * ((source.y[pos_outer_src][pos_inner_src] - middleValue)**2 + (destination.y[pos_outer_dest][pos_inner_dest] - middleValue)**2))       # quadr. mittlere Abweichung
                    if difference <= self.criterion:
                        self.x[pos_outer_src].append(source.x[pos_inner_src])
                        self.y[pos_outer_src][0].append(source.x[pos_inner_src])       # x-Wert schreiben in y Liste
                        self.y[pos_outer_src][1].append(destination.y[pos_outer_dest][pos_inner_dest])     # y-Wert schreiben
                        self.y[pos_outer_src][2].append(difference)     # Differenz schreiben
                        self.y[pos_outer_src][3].append(destination.y[pos_outer_dest][0])       # Name der Quellfunktion
                        pass                    # if Wert <= Sqrt(2)
                    pass                        # y dest
                pass                            # x_xrc
            pass

        self.cleanup()  # bereinigen

        print("DONE LOOKUP")
```

```
    def lookup_test_in_ideal4(self, source, destination):
        """
        testdaten mit den 4 gefundenen idealen Funktionen vergleichen
        :param source: testdaten
        :param destination: 4 ideale funktionen
        """
        self.x = []
        self.y = [[], [], [], []]
        # Ueberschriften anlegen
        self.y[0].append('x')
        self.y[1].append('y')
        self.y[2].append('Differenz')
        self.y[3].append('QuellFunktion')
        found = False

        # lenFileContent = len(source.file_content)
        for i in range(len(source.file_content)):
            xTest = source.file_content[i][0]   # x Wert suchen

            for j in range(1, len(destination.y[0])):
                xDest = destination.y[0][j]
                found = False
                if destination.y[0][j] == xTest:
                    for z in range(1, len(destination.y)):  # fÃ¼r alle y se in y bei x
                        yTest = source.file_content[i][1]
                        middleValue = (destination.y[z][j] + source.file_content[i][1]) / 2.0
                        difference = math.sqrt(0.5 * ((source.file_content[i][1] - middleValue) ** 2 + (destination.y[z][j] -
middleValue) ** 2))  # quadr. mittlere Abweichung
                        try:
                            if difference < self.criterion_Test_in_ideal4:
                                self.y[0].append(xTest)
                                self.y[1].append(yTest)
                                self.y[2].append(difference)
                                self.y[3].append(destination.y[z][0])
                                found = True
                            elif found == False and z == len(destination.y)-1:
                                raise NoMatchFoundError     # benutzerdefinierte Excpetion

                        except NoMatchFoundError:
                            print(NoMatchFoundError().message)

                        pass
                else:
                    # print("else Pfad")
                    if xDest > xTest:
                        break       # Schleife beenden, wenn x aus Destination > als x aus Testdaten, da Daten sortiert
sind
                    pass
                pass
        self.x = self.y[0]
        self.Anzahl_Spalten = len(self.y)


    def cleanup(self):
        """
        Funktion bestimmt die am hÃ¤ufigsten gefundene ideale Funktion und lÃ¶scht die nicht zugehÃ¶rigen
        """
        i = 0
        for j in range(1, len(self.y)):     # durchlaufe alle y 1-5
            # die Funktion mit den meisten Matches ermitteln
            zwischending = [[], []]
```

```
            for z in range(1, 51):  # bei 1 starten wegen Namen y1 bis y50
                zwischending[0].append('y' + str(z))    # interimsarray mit Funktionsnamen
                zwischending[1].append(self.y[j][3].count('y' + str(z)))    # zugehÃ¶rige Anzahl der gefundenen
Matches

            max_value = max(zwischending[1])    # Maximale Anzahl
            max_value_index = zwischending[1].index(max_value)  # Index der maximalen Anzahl
            search_value = zwischending[0][max_value_index]     # hier steht die ermittelte am hÃ¤ufigsten
gefundene Funktion

            for i in range(len(self.y[j][3]), 1, -1):   # lÃ¶sche die anderen Werte der Funktionen, die nicht am
hÃ¤ufigsten enthalten sind
                if self.y[j][3][i-1] != search_value:   # zaehle von hinten her
                    for todelete in range(len(self.y[j])):
                        del self.y[j][todelete][i-1]    # von i muss 1 subtrahiert werden, da hier der Index benÃ¶tigt wird
in i ist aber die gesamte Anzahl an Stellen enthalten

        # Ab hier cleanup um Funktionen createTable und dataToTable nutzen zu kÃ¶nnen -> umformatieren in
eindimensional (x) und zweidimensional (y)
        for i in range(1, 5):           # Funktionsnamen in y Spalte Platz [0] schreiben als Ãœberschrift
            self.y[i][1][0] = self.y[i][3][1]

        self.x = []       # x lÃ¶schen, indem neu definiert wird
        self.x = self.y[1][0]   # es werden die x-Werte aus y[0] in neues x geschrieben

        for i in range(5):
            del self.y[i][3]        # lÃ¶sche Quellfunktion
            del self.y[i][2]        # lÃ¶sche Differenz

        self.y[0][0] = self.y[1][0]       # x Werte in nulltes Array schreiben
        del self.y[0][1]
        self.y[0] = self.y[0][0]
        del self.y[1][0]
        self.y[1] = self.y[1][0]
        del self.y[2][0]
        self.y[2] = self.y[2][0]
        del self.y[3][0]
        self.y[3] = self.y[3][0]
        del self.y[4][0]
        self.y[4] = self.y[4][0]

        print("DONE cleanup")
```

4.7 Test_Data_import

```
import unittest
from Data_import import DataToImport
import os


class UnitTestImport(unittest.TestCase):
    def test_extract_tablename(self):
        ''' RÃ¼ckgabewert des Dateinamens prÃ¼fen '''
        print("START TEST Tabellenname aus Pfad extrahieren")
        testpath = "C:\\Data\\Unterordner_1\\Unterordner_2\\Tabelle.csv"
        print("Testpfad: "+ testpath)
        result = DataToImport.extract_tablename(self, testpath)
        print("Extrahierter Name: " + result)
        self.assertEqual(result, 'Tabelle', "Der Dateiname ist Tabelle")
        print("Finish Test Tabellenname")
```

```
    def test_separate_lines(self):
        ''' RÃ¼ckgabe Soll: zwei Werte x und y '''
        ''' Eingabe: ein zweispaltiger Wert und die Anzahl der Spalten '''
        print("START TEST Zeilen separieren")
        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"
        print("Testmatrix: " + str(testvalue))
        testspalten = 4
        testresult_1 = ['x', 1.0, 5.0, 9.0]
        testresult_2 = [[], ['y1', 2.0, 6.0, 10.0], ['y2', 3.0, 7.0, 11.0], ['y3', 4.0, 8.0, 12.0]]
        result1, result2 = DataToImport.separate_lines(self, testvalue, testspalten)
        print("Ergebnis 1: " + str(result1))
        print("Ergebnis 2: " + str(result2))
        self.assertEqual(result1, testresult_1, "Result 1 OK")
        self.assertEqual(result2, testresult_2, "Result 2 OK")
        print("Finish Test separieren")



if __name__ == '__main__':
    unittest.main()
```

4.8 Visualize

```
from matplotlib import pyplot as plt
from matplotlib import style


class ShowInPlot():
    def __init__(self):
        x_axis = 'X AXIS'       # Name x-Achse
        y_axis = 'Y AXIS'       # Name y_Achse
        title = 'TITEL'         # Diagrammtitel


    def show_id4_plus_test(self, id4, test, train, x_name, y_name, title):
        data_name = []  # label in Diagramm

        for i in range(id4.Anzahl_Spalten):
            if type(id4.y[i][0]) is str:
                data_name.append(id4.y[i][0])
                del id4.y[i][0]

        style.use('ggplot')
        fig, ax = plt.subplots(1, 1)    # Diagramm mit einer Spalte und einer Zeile
        ax.grid(True, color="b")    # Gitterfarbe und Gitter zeigen
        ax.set_xlabel(x_name)       # x-Achsenbeschriftung
        ax.set_ylabel(y_name)       # y- Achsenbeschriftung
        ax.set_title(title)         # Diagrammtitel
        for i in range(1, id4.Anzahl_Spalten):      # zeige y Daten
            ax.scatter(id4.y[0], id4.y[i], label=data_name[i], marker='+',
                       alpha=0.2)  # edgecolors='r'  # alpha = Durchsichtigkeit label=title+'_'+str(i),
            ax.legend()


        ''' test Daten darstellen '''

        ax.scatter(test.y[0], test.y[1], label='test data', marker='o', color='black',
                   alpha=0.6)  # , marker='o', color='0', edgecolors='r', alpha=0.2)  # alpha = Durchsichtigkeit
label=title+'_'+str(i),
```

```
ax.legend()

fig.show()
plt.savefig("ideal4_vs_test.png")

''' neues Diagramm mit ideal 4 und train data'''
fig2, ax2 = plt.subplots(1, 1)  # Diagramm mit einer Spalte und einer Zeile
ax2.grid(True, color="b")  # Gitterfarbe und Gitter zeigen
ax2.set_xlabel(x_name)  # x-Achsenbeschriftung
ax2.set_ylabel(y_name)  # y- Achsenbeschriftung
ax2.set_title('ideal 4 vs train data')  # Diagrammtitel
for i in range(1, id4.Anzahl_Spalten):  # zeige y Daten
    ax2.scatter(id4.y[0], id4.y[i], label=data_name[i], marker='+', alpha=0.2)
    ax2.legend()

''' train data in Diagramm 2 '''
data_name = []  # label in Diagramm

for i in range(train.Anzahl_Spalten):
    if type(train.y[i][0]) is str:
        data_name.append(train.y[i][0])
        del train.y[i][0]

for i in range(1, train.Anzahl_Spalten):  # zeige y Daten
    ax2.scatter(train.y[0], train.y[i], label=data_name[i], marker='.', alpha=0.3)
    ax2.legend()

fig.show()
plt.savefig("ideal4_vs_train.png")

print("DONE Plotting")
```

4.9 Quellen

[1]: Python 3 Das umfassende Handbuch, Johannes Ernesti, Peter Kaiser, 3. Auflage, korrigierter Nachdruck 2014

[2]: Vorlesung Industrie 4.0, Virtuelle Hochschule Bayern, Prof.Dr-.Ing. Thomas Kirchmeier, (Horizontale Integration)

[3]: matplotlib.org

[4]: IU Akademie, Aufgabenstellung_Hausarbeit_DLMDWPMP01.pdf

[5]: Abgegebener Code der Hausarbeit, Andreas Müller

[6]: Abgegebene Ergebnisse der Hausarbeit, Andreas Müller