

```

# -*- coding: utf-8 -*-
"""
Created on Mon Mar 25 16:58:25 2024

@author: bauma
"""

import pandas as pd
import os.path
import os
import math as math
from matplotlib import pyplot as plt
from matplotlib import style
from sqlalchemy import create_engine
from sqlalchemy import text
import sqlalchemy as db
from tkinter import *
from tkinter import filedialog

class FunktionError (Exception):
    """
    Diese Klasse ermöglicht das gezielte Abfangen von Fehlern in der Klasse
    Funktion und die Ausgabe in der Konsole.
    """
    def __init__(self,message=""):
        self.funktion_message = 'Es wurde ein falscher Datentyp zur Erstellung\
einer Funktion übergeben. Es muss ein Panda-Dataframe übergeben werden!'

class DatensatzError (Exception):
    """
    Diese Klasse ermöglicht das gezielte Abfangen von Fehlern in der Klasse
    Datensatz und die Ausgabe in der Konsole.
    """
    def __init__(self,message=""):
        self.datensatz_message = 'Es wurde dem Datensatz kein Dataframe\
zugewiesen.'
        self.datensatz_funktion_message = "Zur Bestimmung der Idealfunktion\
muss eine Instanz der Klasse Datensatz übergeben werden!"

class CSVError (Exception):
    """
    Diese Klasse ermöglicht das gezielte Abfangen von Fehlern beim Import
    eines Datensatzes im CSV-Format.
    """
    def __init__(self):
        self.csv_message = 'Es wurde keine CSV-Datei ausgewählt!'

class PunktError(Exception):
    """
    Diese Klasse ermöglicht das gezielte Abfangen von Fehlern in der Klasse
    Punkt und die Ausgabe in der Konsole.
    """
    def __init__(self,message=""):
        self.punkt_message = 'Mindestens einer der Variablen ist keine Zahl\
(Float oder Integer)'
        self.punkt_instanz_message = 'Die zugewiesene Variable ist keine\
Instanz der Klasse Punkt'

class Datensatz():

```

```

'''
Diese Klasse hat als Grundlage einen Pandas-Dataframe. Sie vereinfacht das
Arbeiten mit dem Dataframe bezogen auf die Struktur der zu importierenden
CSV-Dateien (x,y1,y2,...)
'''
def __init__(self, dataframe=""):
    '''
    Diese Methode wird beim Erstellen einer Instanz der Klasse Datensatz
    ausgeführt. Da es verschiedene Möglichkeiten gibt, der Klasse Datensatz
    einen Pandas-Datensatz zu übergeben, muss nicht zwingend bei der
    Erstellung der Klasse ein Dataframe übergeben werden.

    Input Argument:
    - dataframe = Pandas Dataframe oder leere Zeichenkette

    Output Argument:
    - keine

    Bei der Erstellung der Instanz wird überprüft, ob der übergebene Wert
    ein Dataframe oder eine leere Zeichenkette ist. Bei allen anderen Typen
    wird ein DatensatzError hervorgerufen.
    '''

    #Prüfen, ob es sich um einen Pandas Dataframe (Fall 1) handelt
    if isinstance(dataframe, pd.DataFrame):
        self._dataframe = dataframe

    #Prüfen, ob es sich um eine leere Zeichenkette (Fall 2) handelt
    elif dataframe == "":
        self._dataframe = dataframe

    #Treten Fall 1 und Fall 2 nicht ein --> DatensatzError
    else:
        print('ert')
        raise DatensatzError ('Es wurde dem Datensatz kein Dataframe \
zugewiesen.')

def import_dataframe(self, pfad):
    '''
    Diese Methode erlaubt das Hinzufügen eines Dataframes durch das
    Importieren aus einer CSV-Datei.

    Input-Argument:
    - Dateipfad

    Output-Argument
    - keine
    '''

    #Prüfen, ob die Datei aus dem Pfad existiert
    if os.path.isfile(pfad):
        #Prüfen, ob die Datei eine CSV-Datei ist
        if pfad.endswith('.csv'):
            self._dataframe = pd.read_csv(pfad)
        #Ist die Datei keine CSV-Datei --> CSVError
        else:
            raise CSVError (CSVError().csv_message)

    #Wenn die Datei nicht existiert --> Manuelles Auswählen der
    #entsprechenden Dateien

```

```

else:
    #Auswahl Trainingsfunktionen
    if pfad.endswith('train.csv'):
        titel = 'Bitte Trainingsfunktionen auswählen!'
    #Auswahl Idealfunktionen
    elif pfad.endswith('ideal.csv'):
        titel = 'Bitte Idealfunktionen auswählen!'
    #Auswahl Testpunkte
    elif pfad.endswith('test.csv'):
        titel = 'Bitte Testpunkte auswählen!'
    else:
        titel = 'Bitte Datei auswählen'

while True:

    try:

        pfad = filedialog.askopenfilename(title = titel,filetypes =\
        (('CSV Files','*.csv'),('All Files','*.*')))
        if pfad.endswith('.csv'):
            self._dataframe = pd.read_csv(pfad)
            break
        else:
            raise CSVError

    except CSVError:
        print (CSVError().csv_message)

def setze_dataframe (self, dataframe):
    """
    Diese Methode ermöglicht die Übergabe eines Pandas-Dataframe an
    die erstellte Instanz der Klasse Datensatz.

    Input-Argument:
    - dataframe = Pandas Dataframe

    Output-Argument:
    - keine
    """

    try:
        #Prüfen, ob dataframe ein Pandas Dataframe ist
        if isinstance(dataframe, pd.DataFrame):
            self._dataframe = dataframe
        #Ist die Variable Dataframe keine Instanz der Klasse Dataframe
        #--> DatensatzError
        else:
            raise DatensatzError

    except DatensatzError:
        print (DatensatzError().datensatz_message)

def hole_dataframe (self):
    """
    Diese Methode gibt den Pandas Dataframes des Datensatzes zurück.

    Input-Argument:
    -keine

    Output-Argument:

```

```

- self.__dataframe (Instanz der Klasse Pandas DataFrame)
'''

#Prüfen, ob dem Datensatz bereits ein Dataframe zugewiesen wurde
if isinstance(self.__dataframe,pd.DataFrame):
    return self.__dataframe

else:
    raise DatensatzError('Dem Datensatz wurde kein Dataframe zugewiesen!')

def hole_spaltenbezeichnung(self):
    '''
    Diese Methode gibt die Spaltenbezeichnungen des Dataframes zurück.

    Input-Argument:
    -keine

    Output-Argument:
    - self.__dataframe.columns (Spaltenbezeichnungen)
    '''

    #Prüfen, ob dem Datensatz bereits ein Dataframe zugewiesen wurde
    if isinstance(self.__dataframe,pd.DataFrame):
        return self.__dataframe.columns

    else:
        raise DatensatzError('Dem Datensatz wurde kein Dataframe zugewiesen!')

def anzahl_zeilen(self):
    '''
    Diese Methode gibt die Anzahl der Zeilen des Dataframes zurück.

    Input-Argument:
    -keine

    Output-Argument:
    - self.__dataframe.shape[0] (Anzahl der Zeilen)
    '''

    #Prüfen, ob dem Datensatz bereits ein Dataframe zugewiesen wurde
    if isinstance(self.__dataframe,pd.DataFrame):
        return self.__dataframe.shape[0]

    else:
        raise DatensatzError('Dem Datensatz wurde kein Dataframe zugewiesen!')

def hole_x_spalte (self):
    '''
    Diese Methode gibt die X-Spalte (X-Werte) der Funktionen des Dataframes
    zurück.

    Input-Argument:
    -keine

    Output-Argument:
    - self.__dataframe['x'] (X-Werte der Funktionen)
    '''

    #Prüfen, ob dem Datensatz bereits ein Dataframe zugewiesen wurde
    if isinstance(self.__dataframe,pd.DataFrame):
        return self.__dataframe['x']

    else:

```

```

        raise DatensatzError('Dem Datensatz wurde kein Dataframe zugewiesen!')

def hole_y_spalte (self, spaltenbez):
    '''
    Diese Methode gibt die Y-Spalte (Y-Werte) der Funktionen des Dataframes
    zurück.

    Input-Argument:
    -spaltenbez (Spaltenbezeichnung, die zurückgegeben werden soll)

    Output-Argument:
    - self.__dataframe[spaltenbez] (Y-Werte der Funktionen)
    '''
    #Prüfen, ob dem Datensatz bereits ein Dataframe zugewiesen wurde
    if isinstance(self._dataframe,pd.DataFrame):
        return self._dataframe[spaltenbez]

    else:
        raise DatensatzError('Dem Datensatz wurde kein Dataframe zugewiesen!')

def hole_punkte (self):
    '''
    Diese Methode gibt die Punkte der Funktionen des Dataframes zurück.

    Input-Argument:
    -keine

    Output-Argument:
    - self.__dic_punkte = {'y1':[Punkt(x,y),Punkt(x,y),...],
                           'y2':[Punkt(x,y),...],...}
      Key = Funktionsname
      Value = Liste mit Instanzen der Klasse Punkt
    '''
    #Prüfen, ob dem Datensatz bereits ein Dataframe zugewiesen wurde
    if isinstance(self._dataframe,pd.DataFrame):
        #Erstellen des Dictionary
        self.__dic_punkte = {}
        #Iteration über die Spaltenbezeichnung des Dataframes (x,y1,..)
        for c in self.hole_spaltenbezeichnung():
            #Erstellen der Liste für die Punkte pro Spaltenbezeichnung
            self.__list_punkte = []
            #Über die Spalte x soll nicht iteriert werden --> x-Werte für
            #alle Funktionen identisch
            if c == 'x':
                continue

            #Iteration über die Y-Werte der jeweiligen Funktionen
            for i in range (self.anzahl_zeilen()):
                #Erstellen einer Instanz Punkt und Hinzufügen zur Liste der Punkte
                self.__list_punkte.append(Punkt(self._dataframe.loc[i,'x'],\
                                                self._dataframe.loc[i,c]))

            #Komplette Liste eine Spaltenbezeichnung (Funktion) wird zum
            #Dictionary hinzugefügt
            self.__dic_punkte[c] = self.__list_punkte
        #Ausgabe des Dictionary
        return self.__dic_punkte

    else:
        raise DatensatzError('Dem Datensatz wurde kein Dataframe zugewiesen!')

def export_dic (self):

```

```

'''
Diese Methode gibt den Dataframe als Dictionary zurück.

Input-Argument:
-keine

Output-Argument:
- self.__dataframe als Dictionary
'''

#Prüfen, ob dem Datensatz bereits ein Dataframe zugewiesen wurde
try:
    if isinstance(self.__dataframe,pd.DataFrame):
        #raise DatensatzError('Dem Datensatz wurde kein Dataframe
        #zugewiesen!')
        return self.__dataframe.to_dict('records')

    else:
        #return self.__dataframe.to_dict('records')
        raise DatensatzError

except DatensatzError:
    print (DatensatzError().datensatz_message)

class Punkt ():

    def __init__(self, x_wert,y_wert):
        '''
        Diese Methode wird beim Erstellen einer Instanz der Klasse Punkt ausgeführt.

        Input Argumente:
        - x_wert (Intger oder Float)
        - y_wert (Integer oder Float)

        Output Argumente:
        - keine
        '''

        #Prüfen, ob die übergebenen Werte vom Typ Float oder Integer sind
        if isinstance(x_wert, (float,int)) and isinstance(y_wert,(float,int)):
            #Zuordnung zu den lokalen Variablen
            self.__x_wert = x_wert
            self.__y_wert = y_wert
        #Fehlermeldung, wenn die Werte nicht vom Typ Intger oder Float sind
        else:
            raise PunktError (PunktError().punkt_message)

    def hole_x_wert (self):
        '''
        Diese Methode gibt den X-Wert zurück.

        Input Argumente:
        -keine

        Output Argument:
        - X-Wert
        '''

        #Fehler abfangen, falls dem Punkt kein X-Wert zugeordnet wurde.
        try:
            return self.__x_wert

        except AttributeError:
            print ('Dem Punkt ist kein x-Wert zugeordnet!')

```

```

def hole_y_wert (self):
    """
    Diese Funktion wird beim gibt den Y-Wert zurück.

    Input Argumente:
    -keine

    Output Argument:
    - Y-Wert
    """
    #Fehler abfangen, falls dem Punkt kein Y-Wert zugeordnet wurde.
    try:
        return self.__y_wert

    except AttributeError:
        print ('Dem Punkt ist kein y-Wert zugeordnet!')

def berechne_y_abweichung (self, p):
    """
    Diese Methode berechnet die Y-Abweichung von zwei gegebenen Y-Werten.

    Input Argumente:
    -p (Instanz der Klasse Punkt)

    Output Argument:
    - y-Abweichung
    """
    #Prüfen, ob p eine Instanz der Klasse Punkt ist.
    if isinstance(p,Punkt):
        #Berechnen der Y-Abweichung.
        self.__y_abweichung = self.__y_wert - p.hole_y_wert()

        return self.__y_abweichung
    #Ist p keine Instanz der Klasse Punkt --> PunktError
    else:
        raise PunktError(PunktError().punkt_instanz_message)

def setze_idealfunktion (self, idealfunktion):
    """
    Diese Methode ermöglicht das Setzen einer Idealfunktion zu dem Punkt.

    Input Argumente:
    -idealfunktion (Instanz der Klasse Idealfunktion)

    """
    #Prüfen, ob idealfunktion eine Instanz der Klasse Idealfunktion ist.
    if isinstance(idealfunktion,Idealfunktion):
        self.__idealfunktion = idealfunktion

    #Wenn die Variable idealfunktion keine Instanz der Klasse Idealfunktion
    #ist --> Error
    else:
        raise PunktError('Die zugewiesene Variable ist keine Instanz der\
        Klasse Idealfunktion!')

def hole_idealfunktion (self):
    """
    Diese Methode gibt die dem Punkt zugewiesene Idealfunktion zurück.

    Input Argument:
    
```

```

- keine

Output Argument:
- self.__idealfunktion (Instanz der Klasse Idealfunktion!)
'''

#Fehler abfangen, falls self.__idealfunktion nicht definiert ist
try:
    return self.__idealfunktion

except AttributeError:
    print ('Es wurde noch keine Idealfunktion bestimmt!')

def setze_y_abweichung (self, y_abweichung):
    '''
    Diese Methode ermöglicht das zuweisen einer Y-Abweichung zu einem Punkt.

    Input Argument:
    - y_abweichung

    Output Argument:
    -keine
    '''
    self.__y_abweichung = y_abweichung

def hole_y_abweichung (self):
    '''
    Diese Methode gibt die y_abweichung des Punktes zurück.

    Input Argument:
    - keine

    Output Argument:
    - y_abweichung
    '''
    #Abfangen des Fehlers, falls das Attribut self.__y_abweichung nicht
    #definiert ist.
    try:
        return self.__y_abweichung

    except AttributeError:
        print ('Es wurde noch keine Y-Abweichung berechnet!')

class Funktion (Datensatz):

    def __init__ (self, funktion):
        '''
        Diese Methode wird beim Erstellen einer Instanz der Klasse Funktion
        ausgeführt.

        Input Argumente:
        - funktion (Instanz der Klasse Pandas Dataframe)
          - nur zwei Spalten (x,y)

        Output Argument:
        -keine
        '''
        #Abfrage, ob die Variable funktion eine Instanz der Klasse Pandas
        #DataFrame ist
        if isinstance(funktion, pd.DataFrame):
            #Zuweisung zu einer lokalen Variable

```



```

        Datensatz.__init__(self, funktion)

    else:
        raise FunktionError (FunktionError().funktion_message)

def hole_funktionsname (self):
    """
    Diese Methode gibt den Namen der Funktion zurück.

    Input Argumente:
    - keine

    Output Argument:
    - funktionsname (Bezeichnung der zweiten Spalte.)

    """
    return self._dataframe.columns[1]

def hole_y_werte (self):
    """
    Diese Methode gibt die Y-Werte (zweite Spalte) der Funktion zurück.

    Input Argumente:
    - keine

    Output Argument:
    - Y-Werte

    """
    return self._dataframe[self.hole_funktionsname()]

def hole_funktion (self):
    """
    Diese Methode gibt die Funktion (Pandas DataFrame) zurück.

    Input Argumente:
    - keine

    Output Argument:
    - funktion (Pandas Dataframe)

    """
    return self._dataframe

class Trainingsfunktion (Funktion):
    """
    Diese Klasse erbt von der Elternklasse Funktion. Sie dient dazu, um der
    Trainingsfunktion eine Idealfunktion zuweisen zu können.
    """

    def __init__(self, funktion):
        """
        Diese Methode wird beim Erstellen einer Instanz der Klasse
        Trainingsfunktion ausgeführt.
        """
        #Init-Methode der Elternklasse
        Funktion.__init__(self, funktion)

    def suche_idealfunktion (self, datensatz):
        """
        Diese Methode bestimmt durch Berechnungen eine Idealfunktion für die

```

Trainingsfunktion.

Input-Argument:

- datensatz (Instanz der Klasse Datensatz)

Output-Argumente:

-keine

```
...
#Prüfen, ob datensatz eine Instanz der Klasse Datensatz ist.
if isinstance(datensatz, Datensatz):
    #Abspeichern von datensatz als lokales Attribut
    self.__datensatz_ideal = datensatz
    #self.__minimale Abweichung zum Bestimmen der minimalen Abweichung
    #--> Startwert muss groß sein
    self.__minimale_abweichung = 10000000.0
    #Iteration über die Spalten des Attributes self.__datensatz
    for c_i in self.__datensatz_ideal.hole_spaltenbezeichnung():
        #Spaltenbezeichnung x soll ignoriert werden, da sie 'fix' ist.
        if c_i == 'x':
            #Nächster Schleifendurchlauf
            continue
        #Erstellen des lokalen Attributes self.__vergleichsfunktion
        #(Instanz der Klasse Funktion)
        self.__vergleichsfunktion = Funktion(pd.concat(\
            [self.__datensatz_ideal.hole_x_spalte(),\
             self.__datensatz_ideal.hole_y_spalte(c_i)],axis=1))
        #Bestimmen der maximalen Y-Abweichung (für Aufgabenteil 2)
        self.__maximale_y_abweichung = 0

        self.__sum_leastsquare = 0
        #Liste der Y-Abweichungen (für späteren Plot)
        self.__list_y_abw = []
        #Iteration über die Zeilen der Spalte c_i
        for i in range(self.anzahl_zeilen()):
            #Y-Wert der Trainingsfunktion - Y-Wert der Vergleichsfunktion
            self.__y_abweichung = self._dataframe.loc[i,\
                self.hole_funktionsname()]-\
                self.__vergleichsfunktion.hole_funktion().loc[i,\
                    self.__vergleichsfunktion.hole_funktionsname()]
            #Hinzufügen der Y-Abweichung zur Liste
            self.__list_y_abw.append(self.__y_abweichung)
            self.__sum_leastsquare += (self.__y_abweichung)**2
            #Wenn die Y-Abweichung größer als self.__maximale_y_abweichung
            #--> neue mximale Y-Abweichung
            if self.__y_abweichung > self.__maximale_y_abweichung:
                self.__maximale_y_abweichung = self.__y_abweichung

        #Finale Berechnung der Least-Square-Funktion
        self.__sum_leastsquare = self.__sum_leastsquare/self.anzahl_zeilen()

    #Wenn Least Square größer als minimale abweichung --> neue
    #minimale Abweichung und neue Idelfunktion
    if self.__sum_leastsquare < self.__minimale_abweichung:
        self.__minimale_abweichung = self.__sum_leastsquare
        #Neue Zuweisung für das Attribut self.__ideal_funktion
        #(Instanz der Klasse Idealfunktion)
        self.__ideal_funktion= Idealfunktion (\
            self.__vergleichsfunktion.hole_funktion(),\
            self._dataframe, self.__sum_leastsquare, \
```

```

        self.__maximale_y_abweichung, self.__list_y_abw)
    else:

        raise DatensatzError (DatensatzError().datensatz_funktion_message)

def setze_idealfunktion (self, idealfunktion):
    """
    Diese Methode ermöglicht das Zuweisen einer Idealfunktion zur
    Trainingsfunktion ohne Berechnung.

    Input Argumente:
    -idealfunktion (Instanz der Klasse Idealfunktion)

    Output Argument:
    -keine
    """
    if isinstance(idealfunktion, Idealfunktion):
        self.__ideal_funktion = idealfunktion
    else:
        raise FunktionError('Die Variable idealfunktion ist keine Instanz\
        der Klasse Idealfunktion.')

def hole_idealfunktion (self):
    """
    Diese Methode gibt die Idealfunktion der Trainingsfunktion zurück.

    Input Argumente:
    -keine

    Output Argument:
    -self.__ideal_funktion (Instanz der Klasse Idealfunktion.)
    """
    #Abfangen des Fehler, falls das Attribut self.__ideal_funktion nicht
    #definiert ist.
    try:
        return self.__ideal_funktion
    except AttributeError:
        print ('Es wurde noch keine Idealfunktion bestimmt!')

class Idealfunktion (Funktion, Datensatz):

    def __init__(self, funktion, trainingsfunktion, least_square,\
        maximale_y_abweichung, list_y_abweichungen):
        """
        Diese Methode wird beim Erstellen einer Instanz der Klasse Idealfunktion
        ausgeführt.

        Input Argumente:
        - funktion (Instanz der Klasse Pandas Dataframe)
          - nur zwei Spalten (x,y)
        - trainingsfunktion (Instanz der Klasse Trainingsfunktion)
        - least-square (Least-Square-Abweichung von Idealfunktion zur
          Trainingsfunktion)
        - maximale_y_abweichung (Maximale Y-Abweichung zwischen den Y-Werten)
        - list_y_abweichung (Liste aller Y-Abweichungen)

        Output Argument:
        -keine
        """

```

```

#Ausführen der Init-Methode der Elternklasse
Funktion.__init__(self, funktion)
#Zuweisen zu lokalen Attributen
self.__train_funktion = trainingsfunktion
self.__least_square = least_square
self.__maximale_y_abweichung = maximale_y_abweichung
self.__list_y_abweichungen = list_y_abweichungen

def zuweisung_testpunkte (self, testdatensatz, testkriterium):
    #Festlegen des Testkriterium, hier Faktor Wurzel 2

    if isinstance(testdatensatz,Datensatz):
        #Punkte der Idealfunktionen und Testdaten extrahieren
        punkte_test = testdatensatz.hole_punkte()

        #Festlegen der Struktur eines Dictionaries für Aufgabenteil 2
        #--> Testpunkte zur Idealfunktion
        p_test_data = {'X (Testfunktion)':[], 'Y (Testfunktion)':[],\
                        'Delta Y':[], 'Idealfunktion':[]}
        df_p_test = pd.DataFrame(p_test_data)

        #Iteration über die Values des Dictionaries der Punkte Test
        #--> Liste mit Instanzen der Klasse Punkt
        for v in punkte_test.values():
            #Iteration über die Instanzen der Klasse Punkt
            for p_test in v:
                #Es wird für jeden Punkt aus dem Testdatensatz überprüft,
                #ob er zu einer Idealfunktion passt.
                #Iteration über das Dictionary der Idealfunktionen
                for p_ideal in self.hole_punkte()[self.hole_funktionsname()]:
                    #Prüfen, ob beide Punkte den gleichen X-Wert habe
                    if p_test.hole_x_wert() == p_ideal.hole_x_wert():
                        #Berechnung der Y-Abweichung und Multiplikation mit
                        #dem Testkriterium
                        y_abweichung = p_test.berechne_y_abweichung(p_ideal)
                        y_abw_testkriterium = y_abweichung * testkriterium
                        '''Prüfen, ob die Abweichung inkl. Faktor des
                        Testkriteriums kleiner als die maximale Y-Abweichung
                        von der Idealfunktion zur Trainingsfunktion ist'''
                        if abs(y_abw_testkriterium) <= \
                            abs(self.hole_maximale_y_abweichung()):
                            #Füllen des Panda Dataframes für Aufgabenteil 2
                            df_temp_data = {'X (Testfunktion)':\
                                            [p_test.hole_x_wert()],\
                                            'Y (Testfunktion)':\
                                            [p_test.hole_y_wert()],\
                                            'Delta Y':[y_abweichung],\
                                            'Idealfunktion':\
                                            [self.hole_funktionsname()]}
                            df_temp = pd.DataFrame(df_temp_data)
                            df_p_test = pd.concat([df_p_test,df_temp])

                        else:
                            continue
                    else:
                        continue
            return df_p_test

    else:
        raise DatensatzError('Die Variable testdatensatz ist keine Instanz\
        der Klasse Datensatz.')

```

```

def setze_trainingsfunktion (self, trainingsfunktion):
    '''
    Diese Methode erlaubt das Zuweisen einer Trainingsfunktion

    Input Argumente:
    -trainingsfunktion (Instanz der Klasse Trainingsfunktion)

    Output Argument:
    -keine
    '''
    #Prüfen, ob trainingsfunktion eine Instanz der Klasse Trainingsfunktion ist.
    if isinstance(trainingsfunktion, Trainingsfunktion):
        self.__train_funktion = trainingsfunktion

    else:
        raise FunktionError('Die Variable trainingsfunktion ist keine\
                               Instanz der Klasse Trainingsfunktion.')

def hole_trainingsfunktion (self):
    '''
    Diese Methode gibt die Trainingsfunktion zurück.

    Input Argumente:
    -keine

    Output Argument:
    -self.__train_funktion (Instanz der Klasse Trainingsfunktion)
    '''
    return self.__train_funktion

def setze_least_square (self, least_square):
    '''
    Diese Methode erlaubt das Zuweisen eines Least Square Wertes.

    Input Argumente:
    -least_square

    Output Argument:
    -keine
    '''
    #Prüfen, ob least_square eine Zahl ist.
    if isinstance(least_square, (float, int)):
        #Zuweisen zu einer lokalen Variable
        self.__least_square = least_square

    else:
        raise FunktionError('Der zugewiesene Least-Square-Wert ist keine\
                               Zahl!')

def hole_least_square (self):
    '''
    Diese Methode gibt den Least-Square-Wert zurück.

    Input Argumente:
    -keine

    Output Argument:
    -self.__least_square
    '''
    return self.__least_square

```

```

def setze_maximale_y_abweichung (self, maximale_y_abweichung):
    """
    Diese Methode erlaubt das Zuweisen eines Maximale-Y-Abweichung Wertes.

    Input Argumente:
    -maximale_y_abweichung

    Output Argument:
    -keine
    """
    #Prüfen, ob maximale_y_abweichung eine Zahl ist.
    if isinstance(maximale_y_abweichung, (float,int)):
        #Zuweisen zu einer lokalen Variable
        self.__maximale_y_abweichung = maximale_y_abweichung
    else:
        raise FunktionError('Der zugewiesene Maximale-Y-Abweichung-Wert ist\
        keine Zahl!')

def hole_maximale_y_abweichung (self):
    """
    Diese Methode gibt den Maximale-Y-Abweichung-Wert zurück.

    Input Argumente:
    -keine

    Output Argument:
    -self.__maximale_y_abweichung
    """
    return self.__maximale_y_abweichung

def setze_y_abweichung (self,list_y_abw):
    """
    Diese Methode erlaubt das Zuweisen einer Liste mit Y-Abweichungen.

    Input Argumente:
    -list_y_abw (Liste)

    Output Argument:
    -keine
    """
    #Prüfen, ob list_y_abw eine Instanz der Klasse Liste ist.
    if isinstance(list_y_abw, list):
        #Zuweisen zu einer lokalen Variable
        self.__list_y_abweichungen = list_y_abw
    else:
        raise FunktionError('Der zugewiesene Y-Abweichungen sind keine\
        Liste!')

def hole_y_abweichungen (self):
    """
    Diese Methode gibt die Y-Abweichungen zurück.

    Input Argumente:
    -keine

    Output Argument:
    -self.__list_y_abweichungen (Liste)
    """

```

```

        return self.__list_y_abweichungen

#Trainingsdatensatz erstellen (Instanz der Klasse Datensatz)
trainingsdatensatz = Datensatz()
trainingsdatensatz.import_dataframe('Beispiel_Datensaetze/train.csv')

#Idealdatensatz erstellen (Instanz der Klasse Datensatz)
idealdatensatz = Datensatz()
idealdatensatz.import_dataframe('Beispiel_Datensaetze/ideal.csv')

#Testdatensatz erstellen (Instanz der Klasse Datensatz)
testdatensatz = Datensatz()
testdatensatz.import_dataframe('Beispiel_Datensaetze/test.csv')

#Erstellen zweier Dictionaries zur Verwaltung der Trainings- und Idealfunktionen
dic_trainingsfunktionen = {}
dic_idealfunktionen = {}

'''
    Füllen der Dictionaries und Zuweisen der Idealfunktionen zur einer
    Trainingsfunktion.
    Struktur der Dictionaries :{Funktionsname (z.B. y1):Instanz der Klasse
                                Trainings-oder Idealfunktion,y2:...}
'''
for d in trainingsdatensatz.hole_spaltenbezeichnung():
    if d == 'x':
        continue
    trainingsfunktion = Trainingsfunktion(pd.concat(\
                                          [trainingsdatensatz.hole_x_spalte(),\
                                           trainingsdatensatz.hole_y_spalte(d)],axis=1))

    trainingsfunktion.suche_idealfunktion(idealdatensatz)
    dic_trainingsfunktionen [trainingsfunktion.hole_funktionsname()] =\
        trainingsfunktion
    dic_idealfunktionen [trainingsfunktion.hole_idealfunktion().hole_funktionsname()]=\
        trainingsfunktion.hole_idealfunktion()

x_spalte = False
#Iteration über das Dictionaty mit den Idealfunktionen
#(Struktur:{Funktionsname:Funktion,Funktionsname:Funktion,...})
#Ziel: Erstellen einer Instanz der Klasse Datensatz für die vier Idealfunktionen
for key,value in dic_idealfunktionen.items():
    #key = Funktionsname, value=Instanz der Klasse Idealfunktion

    if x_spalte == False:
        #Erstellen eines Datensatzes mit dem ersten Eintrag des Dictionarys
        #(zwei Spalten, x und y)
        idealfunktionen_df = Datensatz (value.hole_funktion())
        x_spalte = True
    else:
        #Datensatz idealfunktionen_df erweitern um die Y-Spalte des nächsten
        #Eintrages des Dictionarys
        idealfunktionen_df = Datensatz (pd.concat([\
                                                    idealfunktionen_df.hole_dataframe(),\
                                                    value.hole_y_werte()], axis =1))

#Festlegen des Testkriterium, hier Faktor Wurzel 2
testkriterium = math.sqrt(2)

```

```

#Punkte der Idealfunktionen und Testdaten extrahieren
punkte_test = testdatensatz.hole_punkte()

#Festlegen der Struktur eines Dictionaries für Aufgabenteil 2 --> Testpunkte
#zur Idealfunktion
p_test_data = {'X (Testfunktion)':[], 'Y (Testfunktion)':[], \
               'Delta Y':[], 'Idealfunktion':[]}

df_p_testdaten = pd.DataFrame(p_test_data)

for ideal_f in dic_idealfunktionen.values():
    #ds_temp = ideal_f.zuweisung_testpunkte(testdatensatz, testkriterium)
    df_temp = ideal_f.zuweisung_testpunkte(testdatensatz, testkriterium)

    df_p_testdaten = pd.concat([df_p_testdaten, df_temp])

#Erstellen einer Liste mit den Indizes für den DataFrame df_pp_test
liste_index=[]
for index in range(df_p_testdaten.shape[0]):
    liste_index.append(index)
#Zuweisen der Indizes
df_p_testdaten.index=liste_index

#Erstellen einer Instanz der Klasse Datensatz
p_test_anpassbar = Datensatz(df_p_testdaten)

style.use('ggplot')

#Plotten der Ergebnisse für Aufgabenteil 1
for key, value in dic_trainingsfunktionen.items():
    '''
    ax1 = Plot der Trainingsfunktion
    ax2 = Plot der Idealfunktion
    ax3 = Plot der Y-Abweichungen
    '''
    fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, ncols=1, figsize=(20,20))

    ax1.plot(dic_trainingsfunktionen[key].hole_x_spalte(), \
            dic_trainingsfunktionen[key].hole_y_werte(), \
            label = 'Trainingsfunktion '+key)
    ax2.plot(dic_trainingsfunktionen[key].hole_idealfunktion().hole_x_spalte(), \
            dic_trainingsfunktionen[key].hole_idealfunktion().hole_y_werte(), \
            label = 'Idealfunktion '+ \
            dic_trainingsfunktionen[key].hole_idealfunktion().hole_funktionsname())
    ax3.plot(dic_trainingsfunktionen[key].hole_idealfunktion().hole_x_spalte(), \
            dic_trainingsfunktionen[key].hole_idealfunktion().hole_y_abweichungen(), \
            label = 'Abweichungen (y)')
    ax1.set_xlabel("X-Werte", fontsize=14)
    ax1.set_ylabel("Y-Werte", fontsize=14)
    ax2.set_xlabel("X-Werte", fontsize=14)
    ax2.set_ylabel("Y-Werte", fontsize=14)
    ax3.set_xlabel("X-Werte", fontsize=14)
    ax3.set_ylabel("Y-Werte", fontsize=14)
    ax1.legend(fontsize="15", loc="lower right")
    ax2.legend(fontsize="15", loc="lower right")
    ax3.legend(fontsize="15", loc="lower right")
    ax1.set_title('Trainingsfunktion'+ ' '+key, fontsize = 40)

    fig.savefig('Trainingsfunktion'+key+'.png')

#Plotten der Idealfunktionen inkl. passender Punkte

```



```

#Iteration über das Dictionary der Idealfunktionen
for key, value in dic_idealfunktionen.items():
    #key = Funktionsname, value = Instanz der Klasse Idealfunktion
    #Erzeuge von leeren Listen (X bzw Y-Liste)
    pp_liste_x = []
    pp_liste_y = []
    #Iteration über die Instanz der Klasse Datensatz
    for i in range (p_test_anpassbar.anzahl_zeilen()):
        #Abfrage ob Funktionsnamen identisch sind

        if p_test_anpassbar.hole_dataframe().loc[i,'Idealfunktion'] == key:
            pp_liste_x.append(p_test_anpassbar.hole_dataframe().iloc[i,0])
            pp_liste_y.append(p_test_anpassbar.hole_dataframe().iloc[i,1])

        else:
            continue
    ...

    ax.plot = Plot der Idealfunktion
    ax.scatter = Plot der passenden Punkte
    ...

    fig, ax = plt.subplots(figsize=(20,20))
    ax.plot(dic_idealfunktionen[key].hole_x_spalte(),\
            dic_idealfunktionen[key].hole_y_werte(), \
            label = 'Idealfunktion '+key )
    ax.scatter(pp_liste_x,pp_liste_y,label = "Testpunkte", color ="blue")
    ax.set_xlabel("X-Werte", fontsize=14)
    ax.set_ylabel("Y-Werte", fontsize=14)
    ax.legend(fontsize = "15", loc="lower right")
    plt.title ('Idealfunktion mit passenden Testpunkten', fontsize = "40")
    fig.savefig ('Idealfunktionfunktion_'+key+'_Testpunkte.png')

#Liste der Datensätze
datensaetze_list = [trainingsdatensatz, idealdatensatz, p_test_anpassbar]
#Liste der Dateinamen für die SQLite-Datenbank (gleiche Reihenfolge wie
#Liste der Datensätze)
dateinamen_list = ['Trainingsfunktion', 'Idealfunktionen', 'Testdaten']

#Für jeden Datensatz wird eine separate SQLite-Datei angelegt
for dn in range (len(dateinamen_list)):
    #Herstellen der Verbindung zur Datenbank -Speicherort im selben Ordner
    engine = create_engine('sqlite+pysqlite:///'+dateinamen_list[dn]+'_sqlite',\
                           echo=True)

    conn= engine.connect()
    #Falls die SQLite-Datei bereits existiert und Tabellen enthält --> Löschen
    conn.execute(text('DROP TABLE IF EXISTS '+ dateinamen_list[dn]))
    #Exportieren des DataFrames in die SQLite-Datei
    datensaetze_list[dn].hole_dataframe().to_sql(dateinamen_list[dn],\
                                                  conn, if_exists='replace',\
                                                  index=False)

    conn.commit()
    #Trennen der Verbindung zur SQLite-Datenbank
    conn.close()

```