Hausarbeit

DLMDWPMP01 - Programmieren mit Python

im Master Studiengang Data Science

Modulabschlussaufgabe: Pythonprogramm „functionfinder“

Verfasser: Georg Grunsky

Matrikelnummer: IU14072015

Tutorin: Goram, Mandy

Datum: 13.01.2023

Inhaltsverzeichnis

I Abbildungsverzeichnis

II Tabellenverzeichnis

1 Einleitung

1.1 Die Aufgabenstellung

Im Rahmen der Aufgabenstellung zum Kurs DLMDWPMP01 – Programmieren mit Python soll ein Python-Programm umgesetzt werden, das verwaschene Trainingsdaten korrekt auf vier idealtypische Funktionen abbildet. Hierfür stehen 50 idealtypische Funktionen zur Auswahl. Maßgeblich für die Auswahl ist hierbei, gemäß Vorgabe, die kleinste Summe der quadratischen Abweichungen (Least Square). Anschließend soll das gleiche Programm die Punkte eines Testdatensatzes, anhand eines definierten minimalen, statistischen Abstandswertes zu den gewählten idealtypischen Funktionen, klassifizieren und jeweils der entsprechenden Funktion zuordnen.

Fokus der vorliegenden Aufgabenstellung, sowie dieser Arbeit ist primär die Erlernung der Programmiersprache Python und nicht die ausschließliche Erfüllung der beschriebenen mathematischen Aspekte. Diese, isoliert betrachtet, könnten ebenso in Form eines Jupyter-Notebooks erarbeitet werden. Die, in dieser Arbeit getroffenen, Entscheidungen zur Lösung der Aufgabe zielen daher darauf ab, möglichst viele Aspekte einer Entwicklung, sowie die Vorgaben zur Bearbeitung, abzubilden und dienen in erster Linie der Aneignung von Fähigkeiten im Umgang mit Python.

1.2 Struktur der Hausarbeit

Im Hauptteil der vorliegenden Hausarbeit werden zuerst, vom Autor getroffene Entscheidungen, in Bezug auf den strukturellen Aufbau, die Installation und Konfiguration, sowie die Dokumentation des Programmes dargelegt. In weiterer Folge wird auf die bereitgestellten Datensätze eingegangen und wie diese, im Rahmen der Aufgabenstellung, verarbeitet wurden. Anschließend wird in einem weiteren Abschnitt die Auswahl idealtypischer Funktionen anhand der vorliegenden Trainingsdaten beschrieben und dargestellt. Der letzte Abschnitt diskutiert die Klassifikation der Testdaten mithilfe der ausgewählten idealtypischen Funktionen. Hierbei wird auch auf die automatisierte Ausgabe zur Evaluierung nicht erfolgreich klassifizierter Testdaten eingegangen. In der Zusammenfassung werden, anhand der dargestellten Ergebnisse, Ansätze für mögliche Anpassungen des Programms erneut aufgegriffen und als potentielles Thema für eine weitere Hausarbeit zur Diskussion gestellt.

2 Das Python-Package functionfinder

2.1 Struktur, Installation, Konfiguration und Dokumentation

2.1.1 Struktur

Das Programm wurde vom Autor mit dem Namen *„functionfinder“* betitelt und steht auf GitHub unter dem Link

`https://github.com/GGProjects/DLMDWPMP01`

zum Download zur Verfügung.

Hinsichtlich der Struktur des vorliegenden Programmes, entschied sich der Autor für eine Aufteilung des Quellcodes in mehrere Module die im Package *functionfinder* zusammengeführt sind (Kofler 2022, S. 242). Die weitere Ordnerstruktur orientiert sich an einem Blog-Post von Henk Griffioen (2017), der, nach Meinung des Autors, ein leicht verständliches Framework für Pythonprojekte bietet.

In Anlehnung an Croitoru (2022), Kofler (2022) und Henk Griffioen (2017) werden in dieser Arbeit unterschiedliche Begriffe für die Abgrenzung der beschriebenen Elemente verwendet.

- Die Bezeichnungen *Programm* bzw. *Projekt* oder auch *Software* beziehen sich auf die Gesamtheit aller bereitgestellten Module, Packages sowie aller zusätzlichen Ordner und Dateien.
- Mit *Package* werden alle jenen Ordner innerhalb des Projektes angesprochen, die Module beinhalten und des weiteren über eine Datei __init__.py verfügen.
- *Pakete* sind im Gegensatz dazu, öffentlich verfügbare Package-Sammlungen wie zB *pandas*.
- Als *Module* werden die einzelnen, verfügbaren Pythonskripte bezeichnet.
- In Modulen abgelegte *Funktionen* bezeichnen, in sich geschlossenen, Code, der im Programmablauf von den Modulen aufgerufen wird. Im Rahmen dieser Arbeit wird jedoch der Begriff Funktion auch in Bezug auf den mathematischen Zusammenhang der Datenpunkte, innerhalb der bereitgestellten Datensätze verwendet.
- Unter dem Begriff *Methoden* werden letztlich jene Funktionen zusammengefasst, die einer bestimmten Objektklasse zugeordnet sind.

In der nachfolgenden Abbildung werden die wesentlichen Ordner und Dateien der bereitgestellten Programmstruktur dargestellt. Hervorzuheben ist hierbei das Package *functionfinder*, das sämtliche Pythonmodule beinhaltet, die für die Programmfunktionalität verantwortlich sind, sowie das Modul *ffrunner*, über das die eigentliche Ausführung des Programmes gestartet wird. Die übrigen Verzeichnisse des Projekts stehen für die bereitgestellten Daten (*data*), die Dokumentation (*docs*), die vom Programm erzeugten Ausgaben (*output*), sowie für die vorgesehenen UnitTests (*tests*) zur Verfügung.

```
DMLDWPMP01.
|   requirements.txt            # Paketerfordernisse
|   setup.py                    # Modul zur Installation des Packages
|
+---data                        # Ablage der Quelldaten
|   |   ideal.csv
|   |   test.csv
|   \   train.csv
|
+---docs                        # Sphinx-Dokumentation des Packages
|
+---functionfinder              # Hauptpackage der Software
|   |   __init__.py
|   |   classes.py              # Definition von verwendeten Klassen
|   |   config.py               # Benutzer-Konfigurierbare Parameter
|   |   datafunctions.py        # Funktionen zur Datenverarbeitung
|   |   exceptions.py           # Definition von Exceptions
|   |   ffrunner.py             # Hauptmodul, Aufruf durch CLI-Command "ff"
|   |   log.py                  # Parametrisierung der Logausgaben
|   \   setuplog.py             # Parametrisierung des Installationslogs
|
+---output                      # Verzeichnis aller generierten Ergebnisse
|   +---logs                    # Logausgaben
|   +---figures                 # Grafiken
|   +---data                    # SQLite Datenbank
|
\---tests                       # Package der UnitTests
    |   __init__.py
    \   test_unit.py           # Beinhaltet UnitTest-Klassen
```

Struktur des Programmes

2.1.2 Installation

Für die Installation bietet es sich an, lokal eine virtuelle Pythonumgebung anzulegen. Anschließend kann nach Wechsel in das Projektverzeichnis, mit dem Befehl

pip install -e .

das Modul *setup.py* ausgeführt und somit das Programm installiert werden. Mit der Installation werden die Verfügbarkeit der benötigten Dateien überprüft und UnitTests zur Sicherstellung der Funktionalität durchgeführt. Für Letzteres lieferte der Foren-Beitrag von **?** eine gute Vorlage. Benötigte Pakete werden gegebenenfalls in der virtuellen Umgebung nachinstalliert. Im Rahmen der Installation wird außerdem ein Logfile mit der Bezeichnung *setuplog_[DATUM]_[UHRZEIT]* im Verzeichnis *output/logs* angelegt. Nach erfolgreicher Installation, kann das Programm aus der Konsole mit dem einfachen Aufruf *„ff“* ausgeführt werden.

Der Konsolenbefehl *python -m unittest -v tests/test_unit.py* stellt eine detaillierte Ausgabe der Ergeb-

nisse der UnitTests zur Verfügung.

Im Zuge der Entwicklung wurde das vorliegende Programm mit der Pythonversion 3.11 unter Windows10 getestet. Die Lauffähigkeit auf Linux oder IOS/MacOS Plattformen wurde im gegenständlichen Programmdesign nicht berücksichtigt und bietet sich, aus Sicht des Autors, als Thema für eine Weiterentwicklung der Software an.

2.1.3 Konfiguration

Über Änderungen im Modul *config.py* können, falls benötigt, Verzeichnispfade, Log-Einstellungen, Bewertungsfunktionen, sowie auch die Pfade zu den vorliegenden Datendateien angepasst werden.

Letzteres ermöglicht eine einfache Änderung der verwendeten Trainings-, Funktions- und Testdaten, um das Programm auch mit anderen als den, im Rahmen der Aufgabenstellung, bereitgestellten Daten auszuführen.

Eine Anpassung der Bewertungsfunktionen bietet des weiteren die Möglichkeit, die zur Bewertung der Trainingsdaten herangezogene Funktion (der Standardwert ist hierfür die kleinste Summe der quadratischen Abweichungen) sowie den Faktor für den errechneten Fehlerwert, zur Klassifikation der Testdaten (der Standardwert ist, gemäß Aufgabenstellung, die Wurzel aus Zwei) zu verändern.

2.1.4 Dokumentation

Bei der Dokumentation der Software entschied sich der Autor für eine Formatierung gemäß NumPyDoc Style Guide (numpydoc v1.6.0). Dieser bietet den Vorteil, durch die Software *Sphinx* automatisiert zu einer übersichtlichen Dokumentation zusammengefasst werden zu können. Die *autosummary*-Funktion von Sphinx ermöglicht hierbei eine rekursive Verarbeitung der in den Modulen, Funktionen und Methoden hinterlegten Dokumentation (Leedham 2020).

Verfügbar ist diese, automatisch generierte, Dokumentation

- als HTML-Website, durch laden der Datei *docs/build/html/index.html*
- oder als LATEX-Pdf (*docs/build/latex/dlmdwpmp01.pdf*).

2.2 Bereitgestellte Datensätze

2.2.1 Beschreibung der Datensätze

Im Zuge der Aufgabenstellung wurden dem Autor drei Datensätze in Form unten aufgelisteter CSV-Dateien bereitgestellt.

- *train.csv*
- *ideal.csv*
- *test.csv*

Erstere beinhaltet eine Tabelle von vier „verwaschenen“mathematischen Funktionen in den Spalten *y1* bis *y4* mit jeweils 400 Datenpunkten. Die folgende Tabelle zeigt die exemplarische Struktur des Trainingsdatensatzes.

x	y1	y2	y3	y4
-20.0	100.216064	-19.757296	0.3461139	19.776287
-19.9	99.894684	-19.70282	0.61786354	19.789793
-19.8	99.397385	-19.564255	0.1743704	19.441765
-19.7	98.24446	-19.858267	0.7310719	19.869267
-19.6	97.926956	-19.825966	0.27302822	19.864285

Tabelle 1: Exemplarischer Auszug der Datei train.csv

Eine Visualisierung der Trainingsdaten lässt bereits optische Rückschlüsse auf mögliche, passende idealtypische Funktionen zu.

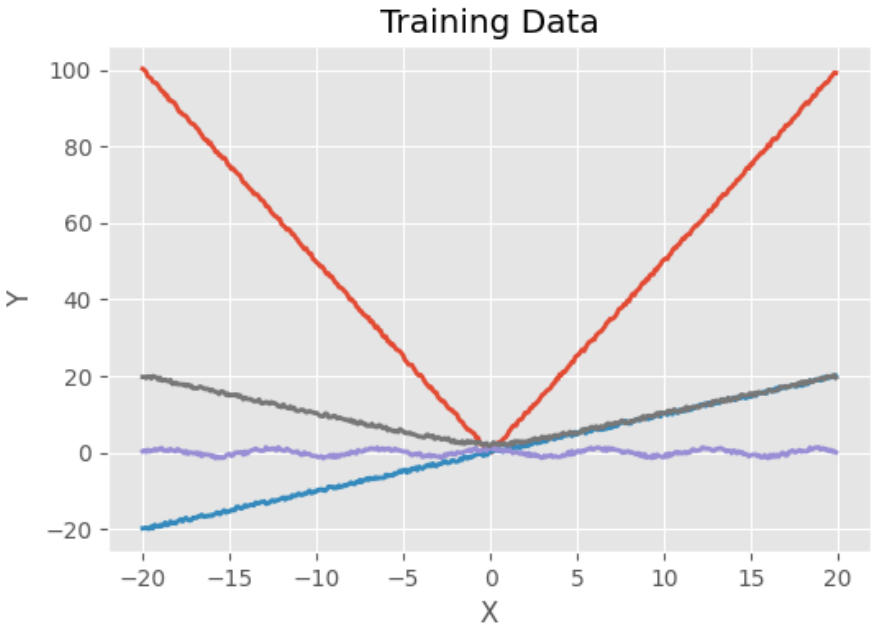

Abbildung 1: Darstellung der bereitgestellten Trainingsdatensätze

Die Datei *ideal.csv* beinhaltet 50, nicht „verwaschene“, idealtypische Funktionen in den Spalten *y1* bis *y50*. Wie im Detail im folgenden Abschnitt beschrieben, sollen diese Funktionen durch das Programm mit dem Trainingsdatensatz verglichen werden und daraus die passendsten vier Funktionen gewählt werden. Die Datei weist, bis auf die Anzahl der Spalten, eine ähnliche Struktur wie der Trainingsdatensatz auf und besteht ebenfalls je Spalte aus 400 Datenpunkten.

x	y1	y2...	y48	y49	y50
-20.0	-0.9129453	0.40808207	-0.18627828	0.9129453	0.3968496
-19.9	-0.8676441	0.4971858	-0.21569017	0.8676441	0.47695395
-19.8	-0.81367373	0.58132184	-0.23650314	0.81367373	0.5491291
-19.7	-0.75157344	0.65964943	-0.24788749	0.75157344	0.6128399
-19.6	-0.6819636	0.7313861	-0.24938935	0.6819636	0.6679019

Tabelle 2: Exemplarischer Auszug der Datei ideal.csv

Im Gegenzug zu den oben dargestellten beiden Datensätzen besteht der Testdatensatz nur aus 100

einzelnen Datenpunkten. Diese werden in einer Spalte *y* der Datei *test.csv* zusammengefasst und sind nicht als zusammengehörige Funktion zu verstehen.

x	y
4.9	4.4963365
-4.7	34.25082
7.2	6.6985793
17.9	17.754583
-6.5	81.21402

Tabelle 3: Exemplarischer Auszug der Datei test.csv

In der weiteren Bearbeitung der Aufgabenstellung sollen diese Datenpunkte mit den ausgewählten idealtypischen Funktionen verglichen und jeweils jener, mit der geringsten Abweichung, zugewiesen werden. Dieser Vorgang wird im Zuge dieser Arbeit als „Klassifikation der Testdaten“bezeichnet und im gleichnamigen Abschnitt im Detail behandelt.

Für die interne Verarbeitung der Daten wurde, im Rahmen des Programms, eine eigene Objektklasse geschaffen, die spezifische Methoden sowie die Parameter der Visualisierung bereitstellt. Diese Klasse und deren Unterklassen für die verschiedenen Datensätze sind im Modul *classes.py* definiert.

2.2.2 Ablage der Datensätze in einer SQLite Datenbank

Gemäß der Vorgabe der Aufgabenstellung sollen sowohl die Trainings- (*train.csv*) als auch die Funktionsdaten (*ideal.csv*) in eine SQLite Datenbank eingelesen werden. Diese wird im Verzeichnis *output/data* bereitgestellt. Der Autor entschied sich hierbei für das Python-Paket *sqlite3* anstelle des, in der Vorgabe vorgeschlagenen, Paketes *sqlalchemy*. Dieses bietet für die gegebene Anwendung eine vereinfachte Möglichkeit Daten mit einer SQLite Datenbank zu verarbeiten (Gosset / Wright 2017).

Über die UnitTests, die im Rahmen der, in Abschnitt 2.1.2 beschriebenen, Installation durchgeführt werden, wird unter anderem, das lokale System auf die Möglichkeit eines direkten SQL-Imports hin überprüft. Dies bedeutet, dass versucht wird, die Daten in die SQLite Datenbank zu schreiben, ohne diese zuerst über das Python-Programm einzulesen und erst in einem weiteren Verarbeitungsschritt an die Datenbank zu übergeben. Die Umsetzung hierfür erfolgt nach einem Foren-Beitrag von Stevens-Haas (2021) mit dem Paket *subprocess*. Das Ergebnis der Überprüfung wird bei Erfolg im Logfile des Setups angeführt. Auch in der Ausführung des Programms wird vorerst noch einmal der direkte Import versucht und erst im Falle eines Fehlschlages der Umweg über die Methoden des Python-Pakets *pandas* gewählt. Obwohl, aufgrund der geringen Größe, der in dieser Arbeit vorliegenden Datensätze, die Möglichkeit des direkten Imports nur eine untergeordnete Rolle spielt, schont ein solcher, im Falle größerer Datenmengen, die Systemressourcen und spart Zeit in der Programmausführung.

Die Testdaten (*test.csv*) werden, gemäß Vorgabe, in der Programmausführung zeilenweise eingelesen, mit den Daten der gewählten Funktionen verglichen und gemeinsam mit der getroffenen Klassifikation in die SQLite Datenbank geschrieben.

2.3 Auswahl idealtypischer Funktionen

Wie bereits im vorangegangenen Abschnitt angeführt, steht zur Bearbeitung der Aufgabenstellung eine Auswahl an 50 idealtypischen Funktionen in einer CSV-Datei zur Verfügung. Aus diesen sollen, für jede der vier Datenspalten der Trainingsdaten, jeweils die passendste Funktion durch das Programm gewählt werden. Das Kriterium zur Selektion ist, gemäß Aufgabenstellung, die Minimierung der Summe aller quadratischen y-Abweichungen (Least Square).

Mathematisch wird daher der kleinste Wert der Formel

$$\sum_{i=1}^{n}(TrainingData_i - IdealFunction_i)^2 \tag{2.1}$$

gesucht. Wobei die Variable $TrainingData$ durch alle vier vorliegenden Trainingsdatenspalten iteriert und die Variable $IdealFunction$ durch sämtliche verfügbaren Datenspalten der idealtypischen Funktionen. n steht hierbei für die Anzahl an Datenpunkten je Datenspalte, die, im vorliegenden Programmdesign, in beiden Datensätzen gleichgroß sein muss. Diese Anforderung ist eine wesentliche Voraussetzung für die Funktionalität der Software *functionfinder* und wird bereits im Rahmen der Installation durch einen UnitTest überprüft. Werden die Quelldatensätze nach erfolgter Installation nochmals geändert, bietet sich eine manuelle Ausführung der UnitTests an, um sich in diesem Bereich abzusichern. Die Schaffung einer Toleranz des Programmes gegenüber unterschiedlichen Datensatzgrößen (zB durch statistische Methoden zur Behandlung von fehlenden Werten) wäre, aus Sicht des Autors, eine durchaus interessante Weiterentwicklung des Projekts.

In der Ausführung des programmierten Packages wurde für den mathematischen Teil dieser Aufgabe (gemäß Gleichung 2.1) von der Eigenschaft des *pandas*-Objekts *Series* Gebrauch gemacht, die, bei der Subtraktion von zwei gleichlangen *Series* der Länge n (in diesem Fall mit $n = 400$) erneut eine *Serie* derselben Länge mit den jeweils subtrahierten Einzelwerten zurück gibt. Auch die mathematische Funktion des Quadrats wird auf jeden Einzelwert der *Serie* angewandt. Die Summenbildung der *Serie* fasst diese nun, dem vorgegebenen Kriterium entsprechend, zu einem Wert zusammen. Diese Eigenschaft ermöglichte es daher, jede Datenspalte des Trainingsdatensatzes durch die Datenspalten der verfügbaren idealtypischen Funktionen zu iterieren und die zurückgelieferten Ergebnisse direkt zu vergleichen. Das Ergebnis mit der kleinsten Summe wurde jeweils als das Passendste abgespeichert.

Im Programmdesign bot es sich daher an, hierfür zwei Funktionen getrennt zu definieren:

1. Die Funktion der Iteration und des Vergleiches zur Ermittlung des minimalen Fehlers wurde im Modul *datafunctions.py* definiert.
2. Die Funktion zur Berechnung des Fehlers (gemäß Vorgabe, entsprechend der Gleichung 2.1) wurde in der Datei *config.py* definiert.

Dies erlaubt es, auf einfache Weise in der Konfigurationsdatei *config.py*, die Funktion zur Fehlerberechnung zu verändern und gegebenenfalls andere Kriterien, wie zB die durchschnittliche mittlere Abweichung (Mean Squared Error - MSE) zur Bewertung heranzuziehen.

Als Rückgabewert der ersten genannten Funktion, wurde vom Autor der Python-Datentyp *dictionary* gewählt. In diesem wurde zu jedem *Key* (die Datenspalten des Trainingsdatensatzes) der *Value* in Form eines *Tupels* abgelegt, welcher den Namen der gewählten Datenspalte des Datensatzes

idealtypischer Funktionen, sowie die berechnete Abweichung beinhaltet.

```
{'y1': ('y36', 33.71178854422821),
 'y2': ('y11', 32.62893138832121),
 'y3': ('y2', 33.11847188090256),
 'y4': ('y33', 31.75243110139478)}
```

Darstellung des Rückgabewertes der berechneten Übereinstimmungen

Die oben angeführte Darstellung des Rückgabewertes macht ersichtlich, welche idealtypischen Funktionen für die jeweiligen Trainingsdatenspalten gewählt wurden. Die grafische Repräsentation lässt auch eine optische Bestätigung der getroffenen Auswahl zu.

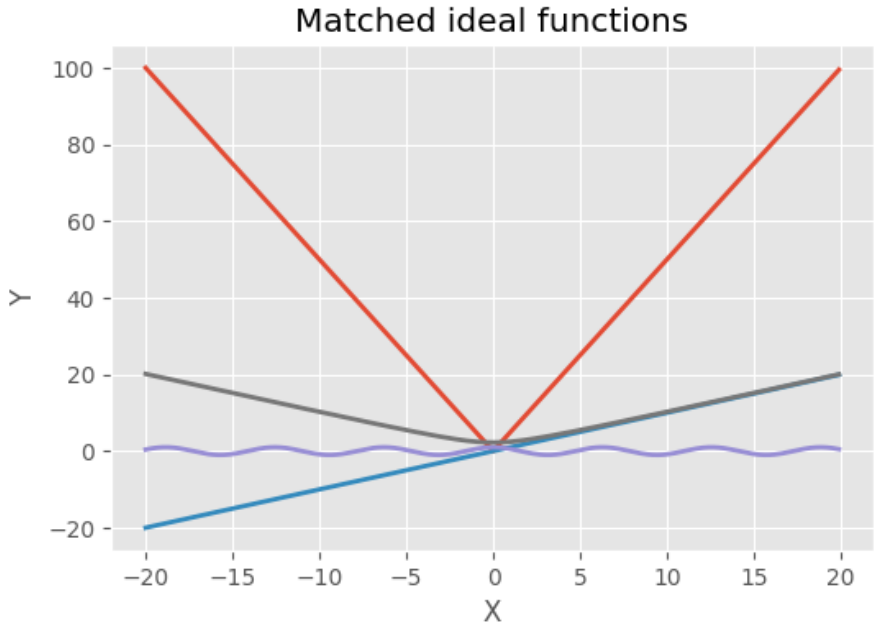

Abbildung 2: Darstellung der ausgewählten idealtypischen Funktionen

2.4 Klassifikation der Testdaten

In diesem Abschnitt wird die Klassifikation der Testdaten mithilfe der ausgewählten idealtypischen Funktionen sowie den jeweiligen berechneten Abweichungen (siehe vorangegangener Abschnitt) diskutiert.

Entsprechend der Vorgabe zur vorliegenden Aufgabenstellung werden die, in der Datei *test.csv*, abgebildeten Datenpunkte, Zeile für Zeile eingelesen und mit dem maximalen Abstand von $a * \sqrt{2}$ einer der vier gewählten Funktionen zuordnet. Wobei a den berechneten Fehlerwert des Trainingsdatensatzes der jeweiligen Funktion repräsentiert. Hierbei unterstützt das Paket *csv* den Vorgang der zeilenweisen Iteration. (Python Software Foundation)

In der Umsetzung wurde die oben angeführte Berechnungsformel wiederum in zwei Teile zerlegt. Der Wert a stammt aus einer vorangegangenen Berechnung und ist an dieser Stelle bereits bestimmt. Der zweite Teil betrifft den Multiplikationsfaktor $\sqrt{2}$, der durch die Vorgabe zur Aufgabenstellung festgelegt wurde. Um diesen gegebenenfalls anpassen zu können wurde dazu die Variable *factor* in

der Konfigruationsdatei *config.py* definiert. Dadurch wird wiederum eine flexible Herangehensweise an die Bewertung der Testdaten gewährleistet.

Für die Bewertung selbst wurde im Modul *datafunctions.py* die Funktion *calculate_best_ideal* definiert. Diese nimmt im Wesentlichen als Parameter die aktuell eingelesene Zeile der Datei *test.csv*, die Auswahl der idealtypischen Funktionen inklusive der Fehlerberechnungen, den Datensatz der idealtypischen Funktionen, sowie einen Funktionsparameter für die Filterung der Ergebnisse entgegen.

Letzterer ist für die Berechnung der passenden idealtypischen Funktion auf *min* gesetzt. Das bedeutet, dass aus den Abweichungen zu allen vier gewählten Funktionen, diejenige mit der kleinsten Abweichung gewählt wird.

Das zeilenweise Einlesen der Testdaten könnte, im Fall der Bearbeitung größerer Datenmengen die Prozessdauer des Programmes negativ beeinflussen. Das *pandas* Modul bietet die Möglichkeit die oben angeführte Funktion auf jede Observation eines Objektes vom Typ DataFrame anzuwenden. Hierdurch würde, in diesem Fall, eine vermeintliche Steigerung der Performance erzielt, wenn der Testdatensatz in Gänze als *DataFrame* bearbeitet würde. Die Struktur des Ergebnisses entspräche durch die funktionale Erweiterung des *DataFrames* auch in der Struktur dem geforderten Ergebnis. Ein solches Vorgehen bedingt allerdings eine Abweichung der Vorgabe der zeilenweisen Bearbeitung und wäre, aus Sicht des Autors in einer weiterführenden Arbeit zu evaluieren.

Die Aufgabenstellung sieht vor, dass die Testdaten um die Werte der Abweichung (DeltaY), sowie die Nummer der gewählten Funktion (Idealfunktion) erweitert werden und in eine eigene Tabelle der SQLite Datenbank geschrieben werden. Der Autor entschied sich an dieser Stelle, noch einen Wert (Off_limit) hinzuzufügen. Dieser Wert, vom Datentyp *Boolean*, zeigt an, wenn dieser Datenpunkt den gesetzten Maximalabstand ($a * \sqrt{2}$) zu keiner der gewählten Funktionen einhalten konnte und wird in diesem Fall auf *True* gesetzt. Dem Datenpunkt wird aber dennoch die Funktion mit dem geringsten Abstand zugewiesen.Der nachfolgende exemplarische Auszug der Ergebnisse aus der Tabelle *test* der SQLite Datenbank zeigt die angesprochene Struktur der ausgegebenen Daten. In der letzten Datenreihe wird auch ersichtlich, dass dieser Datenpunkt den angeführten Maximalabstand bei keiner, der gewählten idealtypischen Funktionen, einhalten konnte. Die Zuweisung ist daher so zu verstehen, dass in diesem Fall die Funktion *y36* den geringsten Abstand zu diesem Datenpunkt aufwies.

x	y	DeltaY	Idealfunktion	Off_limit
4.9	4.4963365	0.4036635	y11	0
-4.7	34.25082	10.75082	y36	0
7.2	6.6985793	0.501420700000001	y11	0
17.9	17.754583	0.145416999999998	y11	0
-6.5	81.21402	48.71402	y36	1

Tabelle 4: Exemplarischer Auszug der Ergebnisse aus der SQLite Datenbank

Für eine Evaluierung der Ergebnisse wurde in der Objektklasse *testdata* die Methode *off_limit* definiert, die aus der SQLite Datenbank jene Werte abfragt, bei denen dieser Wert auf *True* gesetzt ist. Diese Datenpunkte werden nochmals an die Methode *calculate_best_ideal* übergeben, wobei diesmal der Funktionsparameter für die Filterung auf *raw* gesetzt wird. Dadurch werden die Ergebnisse des Vergleichs zu jeder der vier gewählten idealtypischen Funktionen ausgegeben und können über das Logfile manuell evaluiert werden.

Bei der Bearbeitung der zur Aufgabe übergebenen Datensätze waren hierbei zwei Datenpunkte des

Testdatensatzes betroffen. Der aufbereitete Auszug aus dem entsprechenden Logfile zeigt die unten angeführten Ergebnisse:

```
/  2 entries had a deviation above the set limit:

|   x          y      DeltaY  Idealfunktion   Off_limit
/ -6.5   81.21402   48.714020            y36        True
| -6.5   81.21402   87.714020            y11        True
| -6.5   81.21402   80.237432             y2        True
| -6.5   81.21402   74.340156            y33        True

|   x          y      DeltaY  Idealfunktion   Off_limit
| 0.3   59.68033   58.180330            y36        True
| 0.3   59.68033   59.380330            y11        True
| 0.3   59.68033   58.724993             y2        True
/ 0.3   59.68033   57.424227            y33        True
```

Listing 2.1: Datenpunkte mit überschrittenem Maximalabstand

In den oben angeführten Tabellen wird die Distanz der beiden betroffenen Datenpunkte zu den vier gewählten Funktionen ersichtlich. In der Zuweisung wurden dennoch jene Funktionen mit dem geringsten Abstand gewählt (grafisch hervorgehoben).

In der grafischen Repräsentation der Ergebnisse ist die Annäherung der einzelnen Datenpunkte des Testdatensatzes an die gewählten idealtypischen Funktionen gut erkennbar. Jene beiden Datenpunkte, die im Vergleich den definierten Maximalabstand überschritten haben werden durch das Symbol x dargestellt und liegen etwa bei den Koordinaten $(-)7$
81 sowie $(0)//60$. Wobei hier die optische Beurteilung des zweiten Punktes eher eine Tendenz zur Funktion *y36* vermuten ließe, als zur Funktion *y33*. In der oben angeführten Darstellung zu diesem Datenpunkt wird aber auch deutlich, dass die Berechneten Abstände tatsächlich sehr nahe beieinander liegen.

Würde die Distanz allerdings unter dem Aspekt der Relation zur berechneten Abweichung betrachtet werden, entspräche die Klassifikation der oben angesprochenen, optischen Beurteilung.

$$\text{y33: } \frac{57.424227\ (Punktabweichung)}{31.752431\ (Fehlerwert\ Funktion\ y33)} = 1.808498$$

$$\text{y36: } \frac{58.180330\ (Punktabweichung)}{33.711788\ (Fehlerwert\ Funktion\ y36)} = 1.725815$$

Eine Klassifikation unter Berücksichtigung der berechneten Abweichung a für jeder der ausgewählten idealtypischen Funktionen, wäre aus Sicht des Autors, eine mögliche Weiterentwicklung des Programmes.

Zusätzlich kann in der Visualisierung beobachtet werden, dass auch an den Schnittpunkten der Funktionen die Zuordnung der Datenpunkte ausreichend gut zu funktionieren scheint und der Verlauf der idealtypischen Funktionen optisch erkennbar nachverfolgt wird.

Abbildung 3: Klassifikation der Testdatensätze **?**

3 Schlussteil

Zusammenfassend wird an dieser Stelle, wie Eingangs erwähnt, noch einmal betont, dass bei der Bearbeitung der gegenständlichen Aufgabenstellung in erster Linie die Erlernung des Umgangs mit der Programmiersprache Python im Vordergrund stand. Für den Autor war hierbei der exemplarische Aufbau und die Struktur eines selbst erstellten Pythonmoduls, sowie das Zusammenspiel und die Anpassungsfähigkeit der einzelnen Skripte von besonderer Bedeutung.

Es wurde hierbei versucht, die Vorgaben der Aufgabenstellung weitestgehend zu erfüllen. Wie jedoch bereits in den vorangegangenen Abschnitten erläutert, waren aus Sicht des Autors, Erweiterungen in Bezug auf einen automatischen Versuch des direkten Imports der Daten in die SQLite Datenbank, sowie hinsichtlich der verfügbaren Datenspalten des, an die SQLite Datenbank, übergebenen Ergebnisses der klassifizierten Testdaten sinnvoll. Letztere Anpassung begünstigt hierbei in erster Linie die Möglichkeit der Ausgabe jener Ergebnisse, die den vorgegebenen Maximalabstand zu keiner der ausgewählten idealtypischen Funktionen einhalten konnten.

Des Weiteren wurden in den vorangegangenen Abschnitten zwei Themenbereiche angesprochen, die einen Ansatz für eine Weiterentwicklung des Programmes darstellen:

1. Zeitersparnis: Tabellenweise Verarbeitung der Testdaten
2. Zuweisung nicht aufgrund des kleinsten Abstandes sondern aufgrund des kleinsten Abstandes in Bezug auf a (den berechneten Fehlerwert der Trainingsdaten)

Durch diese beiden Anpassungen könnte voraussichtlich die Effizienz des Programmes in der Verarbeitung größerer Datensätze, sowie die Genauigkeit der Klassifikation der Testdaten gesteigert werden. Die Validierung dieser Vermutung wäre, aus Sicht des Autors, eine mögliche Fragestellung für eine weiterführende Arbeit.

Kompatibilität **?**auf anderen Plattformen wie IOS IPad, Linux Toleranz gegenüber unterschiedlichen Datensatzgrößen (train + ideal) durch statistische Methoden der Behandlung fehlender Werte.

Literaturverzeichnis

Croitoru, P. D. C.: *Programmieren mit Python* (**2022**), iU Internationale Hochschule GmbH.

Gosset, J./ **Wright, A.** (**2017**): *Accessing SQLite Databases Using Python and Pandas – Data Analysis and Visualization in Python for Ecologists*.

Henk Griffioen (**2017**): *How to Start a Data Science Project in Python*.

Kofler, M. (**2022**): *Python: der Grundkurs*. Rheinwerk Computing, Bonn, 2., aktualisierte auflage edition.

Leedham, J. (**2020**): *Answer to "Sphinx autodoc is not automatic enough"*.

numpydoc v1.6.0 (): *Style guide — numpydoc v1.6.0rc1.dev0 Manual*.

Python Software Foundation (): *csv — CSV File Reading and Writing*.

Stevens-Haas, J. (**2021**): *Importing a CSV file into a sqlite3 database table using Python - Stack Overflow*.

A Annexes (optional)

(with a list of them)