Woche 04: Programmierung – Statistik mit SciPy

Skript

Erarbeitet von
Ludmila Himmelspach

Lernziele 1
Inhalt 1
Einstieg 1
Modul SciPy 2
Statistische Kennzahlen 2
Take-Home Message 6
Quellen 6
Weiterführendes Material 7
Disclaimer 7

Lernziele

- Statistische Kennzahlen wie Minimal-, Maximal- und Mittelwert, Modus und Median für *ndarrays* mit den Funktionen der Module SciPy und NumPy berechnen können

Inhalt

Einstieg

Statistik dient dazu, die reale Welt zu beschreiben, indem Daten erhoben, analysiert und interpretiert werden. Die statistischen Berechnungen helfen dabei, neues Wissen aus Daten zu extrahieren, um bessere Entscheidungen für Problemstellungen zu treffen, die weniger Einzelfälle, sondern vielmehr eine Gesamtheit oder Massenerscheinungen betreffen. Die Programmiersprache Python und seine Module können für uns Menschen noch keine

Projekt „KI für alle"
HeiCAD – Heine Center for Artificial Intelligence and Data Science

CC BY

Entscheidungen übernehmen, dennoch bieten sie eine Vielzahl an Funktionen an, um statistischen Kennzahlen zu berechnen.

Modul SciPy

Ein Modul für wissenschaftliche Berechnungen, Simulationen und Visualisierungen heißt *SciPy*.

Quelle [1]

SciPy steht für *Scientific Python*. Es ist in verschiedene Untermodule unterteilt, die unterschiedliche Bereiche des wissenschaftlichen Rechnens abdecken. Eins davon ist *Statistical functions (scipy.stats)*.

Quelle [2]

Dieses Untermodul enthält viele Funktionen für statistische Berechnungen. In diesem Kapitel konzentrieren wir uns auf die grundlegenden statistischen Kennzahlen.

Statistische Kennzahlen

Damit du die Funktionen des Moduls *SciPy* nutzen kannst, musst du dieses zuerst in dein Programm importieren. Wenn du nur ein bestimmtes Untermodul wie z. B. *scipy.stats* in deinem Programm benötigst, hast du die Möglichkeit nur dieses zu importieren. Im Folgenden werden wir mit dieser Patiententabelle arbeiten.

	Alter	**Größe**	**Gewicht**
Patient 1	45	1.82	78.5
Patient 2	35	1.73	56.3
Patient 3	67	1.87	82
Patient 4	23	1.95	90.4
Patient 5	35	1.82	85

Die Werte dieser Datentabelle speichern wir in einem *ndarray*. Dafür muss das Modul *NumPy* auch importiert werden.

```
from scipy import stats
import numpy as np

# Wir legen die Werte der Datentabelle in einem ndarray ab.
```


CC BY

```
patienten_daten = np.array([[45, 1.82, 78.5], [35, 1.73, 56.3],
                            [67, 1.87, 82], [23, 1.95, 90.4],
                            [35, 1.82, 85]])

print(patienten_daten)
```

In der Ausgabe der *print()*-Anweisung kannst du prüfen, ob du die Daten korrekt eingegeben hast.

Um den *Minimalwert* für jedes Merkmal zu berechnen, nutzen wir die Funktion *tmin()* des *SciPy*-Moduls. *tmin* steht für *trimmed minimum* und berechnet den Minimalwert entlang einer Dimension. Dabei gibt es die Möglichkeit, Werte zu ignorieren, die kleiner als eine angegebene Untergrenze sind. Das ist dann sinnvoll, wenn man z. B. an einem Minimalwert interessiert ist, der größer als null ist, es aber sehr kleine Merkmalswerte nahe null in der Datentabelle gibt, die man nicht beachten möchte, weil man darunter Rundungsfehler vermutet.

Quelle [3]

Zuerst berechnen wir den Minimalwert für jedes Merkmal über alle Merkmalsausprägungen, also geben wir keine Untergrenze an. Für den Parameter *axis* müssen wir die Dimension angeben, über die wir den Minimalwert bilden wollen. Unser *ndarray patienten_daten* ist zweidimensional. Wenn wir entlang der ersten Dimension gehen, laufen wir über unsere Beobachtungen bzw. Zeilen der Datentabelle. Wenn wir entlang der zweiten Dimension gehen, laufen wir über die Merkmale bzw. die Spalten der Tabelle. Da wir den Minimalwert über die Beobachtungen, also über die Zeilen der Datentabelle hinweg, also in diesem Beispiel den Minimalwert für jede Spalte und somit für jedes Merkmal, berechnen wollen, müssen wir für den Parameter *axis* die erste Dimension, also eine Null, angeben.

```
# Hier wird keine Untergrenze angegeben.
min_array = stats.tmin(patienten_daten, axis = 0)

print(min_array)
```

Jetzt berechnen wir den Minimalwert für jedes Merkmal mit der Angabe der unteren Grenze. Diese legen wir mit Hilfe des Parameters *lowerlimit* fest. Wenn wir für *lowerlimit* nur einen Wert angeben, wird dieser für alle Merkmale angewendet. Das macht in unserem Fall keinen Sinn, da Werte für jedes Merkmal in unterschiedlichen Bereichen liegen. Also geben wir in runden Klammern für jedes Merkmal eine eigene untere Grenze an. Für das Alter ist es 23, für die Größe 1.73 und für das Gewicht 60. Mit dem Parameter *inclusive* können wir zusätzlich festlegen, ob die Werte, die genau der unteren Grenzen entsprechen, bei der Berechnung des Minimalwerts miteinbezogen werden. Wir ignorieren diese Werte, also setzen wir den Parameter *inclusive* auf *False.*

```
# Hier wird die Untergrenze für das Alter mit 23, für die Größe
# mit 1.73 und für das Gewicht mit 60 angegeben.
min_array_ll = stats.tmin(patienten_daten, axis = 0,
                          lowerlimit = (23, 1.73, 60),
                          inclusive = False)
```


CC BY

```
print(min_array_ll)
```

Durch die Angabe der Untergrenze unterscheiden sich natürlich die Minimalwerte für die Merkmale von den zuvor berechneten Minimalwerten.

Der *Maximalwert* eines Arrays entlang der angegebenen Dimension wird mit der Funktion *tmax() (trimmed maximum)* berechnet. Bei dieser Funktion hat man wieder die Möglichkeit, mit dem Parameter *upperlimit* nur die Werte zu berücksichtigen, die kleiner als eine angegebene Obergrenze sind.

Quelle [4]

Wir berechnen den Maximalwert für jedes Merkmal über alle Merkmalsausprägungen, also geben wir keine Obergrenze an. Wie für die Funktion *tmin()* müssen wir auch hier die Dimension angeben, entlang welcher der Maximalwert bzw. die Maximalwerte berechnet werden sollen.

```
# Hier wird keine Obergrenze angegeben. Also werden alle
# Merkmalsausprägungen betrachtet.
max_array = stats.tmax(patienten_daten, axis = 0)

print(max_array)
```

In der Ausgabe sehen wir die Maximalwerte für jedes Merkmal der Datentabelle.

Der *Modus* eines Arrays entlang der angegebenen Dimension wird mit der Funktion *mode()* berechnet. Wenn man die Dimensionalität des Eingabearrays in der Ausgabe beibehalten möchte, setzt man den Parameter *keepdims* auf *True*. Möchte man, dass die Modi der Merkmale in einem eindimensionalen Array ausgegeben werden, soll der Parameter *keepdims* auf *False* gesetzt werden.

Quelle [5]

Wir lassen die Modi der Merkmale und ihre Vorkommenshäufigkeiten in jeweils einem eindimensionalen Array ausgeben, also setzen wir den Parameter *keepdims* auf *False.*

```
mode_array = stats.mode(patienten_daten, axis = 0,
                        keepdims=False)

print(mode_array)
```

In der Ausgabe sehen wir, dass in der Datentabelle zwei 35-jährige Personen vorkommen. Außerdem gibt es zwei Patienten, die 1,82 m groß sind. Für das Merkmal „Gewicht“ gibt es keine Werte, die mehrmals vorkommen, also gibt die Funktion den Minimalwert für dieses Merkmal aus. Übrigens, wenn es für ein Merkmal mehrere Modi gibt, gibt die Funktion nur einen von ihnen aus.

CC BY

Für die Berechnung des *Mittelwerts* eines Arrays entlang der angegebenen Dimension stehen mehrere Funktionen im Modul *SciPy* zur Verfügung. Am einfachsten kann man es mit der Funktion *tmean()* berechnen. *tmean* steht für *trimmed mean*, d. h. wie bei der Berechnung des Minimal- und des Maximalwerts kann man mit der Angabe einer Unter- und einer Obergrenze für jedes Feature die Werte bei der Berechnung des Mittelwerts ignorieren, die außerhalb dieser Grenzen liegen.

Quelle [6]

Wir verzichten auf die Angabe der Unter- und der Obergrenze. Also werden bei der Berechnung der Mittelwerte für die einzelnen Merkmale alle ihre Werte miteinbezogen.

```
mean_array = stats.tmean(patienten_daten, axis=0)

print(mean_array)
```

Wenn du der Funktion *tmean()* nicht traust, kannst du mit dem Taschenrechner die Mittelwerte in der Ausgabe überprüfen.

Für die Berechnung des *Medians* gab es eine Funktion namens *nanmedian()* in den älteren Versionen von *SciPy*.

Quelle [7]

Diese Funktion steht in den neueren Versionen des Moduls nicht mehr zur Verfügung. Wir können aber den *Median* eines Arrays entlang der gewählten Dimension mit der Funktion *median()* des Moduls *NumPy* berechnen.

Quelle [8]

In der Handhabung ist diese Funktion sehr ähnlich zu den statistischen Funktionen aus dem SciPy-Modul.

```
median_array = np.median(patienten_daten, axis = 0)

print(median_array)
```

Eigentlich alle oben genannten statistischen Kennzahlen bis auf den *Modus* lassen sich auch mit den entsprechenden Funktionen des *NumPy*-Moduls berechnen. Diese Funktionen bieten allerdings teilweise andere Auswahloptionen als die Funktionen im *SciPy*-Modul an.

Quelle [9]

Take-Home Message

In diesem Video hast du das Modul SciPy kennengelernt. Du hast gelernt, wie man mit seinen Funktionen einige grundlegende statistische Kennzahlen wie Minimal-, Maximal-, Mittelwert und Modus berechnen kann. Außerdem hast du gesehen, dass die Modulbestandteile sich im Laufe der Zeit ändern können und einige Funktionen in neueren Versionen nicht mehr zur Verfügung stehen können, so dass man auf andere Module ausweichen muss.

Quellen

Quelle [1] Virtanen, P., Gommers, R., Oliphant, T., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S., Brett, M., Wilson, J., Millman, K., Mayorov, N., Nelson, A., Jones, E., Kern, R., Larson, E., Carey, C., Polat, ., Feng, Y., Moore, E., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E., Harris, C., Archibald, A., Ribeiro, A., Pedregosa, F., van Mulbregt, P., & SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods, 17,* 261–272. https://doi.org/10.1038/s41592-019-0686-2

Quelle [2] SciPy (2022). Statistical functions (scipy.stats). In *SciPy Reference, Release 1.10.0* https://docs.scipy.org/doc/scipy/reference/stats.html

Quelle [3] SciPy (2022). Statistical functions (scipy.stats). In *SciPy Reference, Release 1.10.0* https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.tmin.html

Quelle [4] SciPy (2022). Statistical functions (scipy.stats). In *SciPy Reference, Release 1.10.0* https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.tmax.html

Quelle [5] SciPy (2022). Statistical functions (scipy.stats). In *SciPy Reference, Release 1.10.0* https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mode.html

Quelle [6] SciPy (2022). Statistical functions (scipy.stats). In *SciPy Reference, Release 1.10.0* https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.tmean.html

Quelle [7] SciPy (2014). Statistical functions (scipy.stats). In *SciPy Reference, Release 0.14.0* https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.nanmedian.html

Quelle [8] NumPy (2022). Statistics. In *NumPy Reference, Release 1.23*.

CC BY

https://numpy.org/doc/stable/reference/generated/numpy.median.html

Quelle [9] NumPy (2022). Mathematical functions. In *NumPy Reference, Release 1.23*. https://numpy.org/doc/stable/reference/routines.math.html

Weiterführendes Material

Schmitt, S. (2021). *Python Kompendium: Professionell Python Programmieren lernen.* BMU Media Verlag

Disclaimer

Transkript zu dem Video „Woche 04: Programmierung – Statistik mit SciPy“, Ludmila Himmelspach.
Dieses Transkript wurde im Rahmen des Projekts ai4all des Heine Center for Artificial Intelligence and Data Science (HeiCAD) an der Heinrich-Heine-Universität Düsseldorf unter der Creative Commons Lizenz CC-BY 4.0 veröffentlicht. Ausgenommen von der Lizenz sind die verwendeten Logos, alle in den Quellen ausgewiesenen Fremdmaterialien sowie alle als Quellen gekennzeichneten Elemente.