

Für  
Ausbildung  
und Beruf

Metin Karatas

# Eigene KI-Anwendungen programmieren

- + Bilderkennung und -generierung, ChatGPT, Neuronale Netze u. v. m.
- + Datenanalyse ohne Code mit KNIME und Orange
- + Projekte in einfachem Python – ohne Vorkenntnisse einsteigen

2. Auflage



Jupyter Notebooks, Datenmaterial und  
Beispielprojekte zum Download

 Rheinwerk  
Computing

# Kapitel 1

## Einleitung

*Sie haben also beschlossen, sich mit dem Thema »Künstliche Intelligenz« vertraut zu machen? Eine sehr gute Idee! Verschaffen wir uns in diesem Kapitel erstmal einen Überblick.*

### **Worum geht es in diesem Kapitel?**

- ▶ Warum sind Kenntnisse im Bereich der künstlichen Intelligenz (KI) so wichtig?
- ▶ Können Programmieranfänger\*innen KI-Programme entwickeln?
- ▶ Was ist eine künstliche Intelligenz?
- ▶ Welche Unterkategorien gibt es?
- ▶ ein kurzer Überblick über die Geschichte der KI
- ▶ Vorstellung der verwendeten Werkzeuge

Künstliche Intelligenz hat schon vollends Einzug gehalten in unser tägliches Leben. Sei es in der Medizin, Qualitätssicherung, Kunst oder aber in der Werbung – um einige Bereiche exemplarisch aufzuzählen. Und alle Zeichen deuten darauf hin, dass die KI zukünftig noch wichtiger wird. Sie, liebe Leserin bzw. lieber Leser, haben das offensichtlich erkannt. Ob in der Schule oder Ausbildung, im Studium oder Beruf: Überall werden Sie der künstlichen Intelligenz begegnen.

Wenn Sie verstehen, wie diese Technologie funktioniert, können Sie neue Einsatzmöglichkeiten in Ihrem Umfeld identifizieren. Vielleicht liegen bereits viele Daten vor (z. B. Excel), und die KI kann diese für Sie auswerten, gruppieren, klassifizieren oder einen Hinweis für zukünftige Ereignisse liefern. Sie kann Ihnen Arbeit abnehmen oder Sie bei der täglichen Arbeit unterstützen.

Selbst wenn Sie nicht beruflich KIs entwickeln sollten, sind Kenntnisse in diesem Bereich sehr hilfreich. Sie sind dann in der Lage, in Projekten zu erkennen, wann und wie eine künstliche Intelligenz Einsatz finden könnte. Sie können auch Vorschläge von KIs besser einordnen, da Sie das »System« dahinter kennen. Nicht zu vergessen: Es macht einfach sehr viel Spaß, Programme zu entwickeln, die den Anschein von Intelligenz haben.

### 1.1 Was bietet dieses Buch?

Was ist die richtige Strategie, um Interessierte in dieses Themengebiet einzuführen? Sollte erst eine gründliche Einführung in Python erfolgen? Mit allen benötigten Modulen? Meine Erfahrungen aus dem Lehrbetrieb an der Technikerschule (Fachbereiche Maschinenbautechnik, Mechatroniktechnik und Wirtschaftsinformatik) haben gezeigt: Nein, das ist nicht notwendig. Auch angehende Maschinenbautechniker\*innen, die kaum oder gar keine Programmierkenntnisse hatten, konnten im Verlaufe des Unterrichtes Datensätze aus dem Internet herunterladen und KIs dazu programmieren. Wobei das Wort »programmieren« es nicht ganz trifft, man könnte hier eher von »konfigurieren« sprechen.

Wir werden uns gleich zu Beginn ein fertiges Python-Programm für eine bestimmte Problemstellung anschauen und intensiv besprechen. Im Anschluss werden wir dieses Programm derart ändern, dass wir damit andere Problemstellungen lösen können. Auf diese Art und Weise werden wir fortfahren. Einige Aufgaben werden wir zusätzlich mit der Software KNIME mittels grafischer Bausteine lösen. Sie müssen also keine Programme von Grund auf neu schreiben.

Es geht nicht darum, Sie umfänglich in die Programmiersprache Python oder die verwendeten Module einzuführen. Vielmehr sind Sie nach der Lektüre in der Lage, z. B. Datensätze aus dem Internet herunterzuladen und KIs dazu zu entwickeln bzw. Ihre vorhandenen Programme dafür anzupassen. Das ist ein sehr pragmatischer Ansatz. Die Programmiersprache ist nur Mittel zum Zweck. Wenn Sie das Buch durchgearbeitet haben und Ihre Kenntnisse in diesem Bereich weiter vertiefen wollen, empfehle ich Ihnen aber, die Programmiersprache Python zu erlernen.

Die hier besprochenen Programme können Sie herunterladen, analysieren und ausführen. Trauen Sie sich, diese auch zu verändern und die Ergebnisse kritisch zu betrachten. Die Kapitel enthalten auch Übungen, an denen Sie das Erlernte gleich anwenden können. Nehmen Sie sich Zeit für diese Übungen. Sie können zur Lösung ein vorhandenes, hier besprochenes Programm kopieren und anpassen. Wichtig ist, dass Sie sich nicht gleich die Musterlösung anschauen. Es wird viel mehr hängenbleiben, wenn Sie sich erst einmal selbst daran versuchen. Dann werden Fragen auftauchen, die Sie sich sonst nie gestellt hätten. Die Analyse der Musterlösung wird auch dann den Lernprozess unterstützen, wenn Sie das Programm nicht fehlerfrei erstellen konnten.

Natürlich ist es legitim, bei offenen Fragen im Internet zu recherchieren. Machen Sie aber nicht den Fehler, Codezeilen einfach zu kopieren, ohne ihre Funktionalität zu verstehen. Passen Sie die Codezeilen an Ihr Programm an. Die Struktur und Variablennamen Ihres Programmes sollten z. B. von Ihnen stammen. Verändern Sie die übernom-

menen Codezeilen so, dass sie zu Ihrem Programm passen. Übernehmen Sie keine Zeile, ohne zu verstehen, was sie eigentlich macht.

Durch Lesen allein werden Sie weder KI-Entwicklung noch Programmierung lernen. Sie würden auch nicht auf die Idee kommen, nur durch ein Buch die französische Sprache zu erlernen. Lesen, analysieren, verändern, hinterfragen und an neue Aufgabenstellungen anpassen – diese Vorgehensweise wird in diesem Themengebiet zum Erfolg führen.

Es ist leider die harte Wahrheit, dass Sie auch dann kein KI-Profi werden, wenn Sie das Buch gewissenhaft durcharbeiten. Sie haben aber somit die Grundlagen verstanden und können schon eigene KIs entwickeln. Der weitere Lernprozess wird mehr Spaß machen, da Sie sich nicht nur mit der trockenen Theorie auseinandersetzen müssen. Außerdem können Sie das Erlernte gleich an neuen Programmen anwenden.

Rückmeldungen von vielen Schülerinnen und Schülern haben gezeigt, dass die visuelle Programmierung mit Orange und KNIME besonders viel Spaß macht. Das KI-Programm wird mittels grafischer Bausteine wie ein Puzzle zusammengesetzt. Und das ist keine Spielerei, vielmehr ist KNIME in der professionellen Entwicklung weit verbreitet, Orange wird in vielen Bildungseinrichtungen verwendet.

Sie werden keine Veranstaltung zu KI erleben, welche ohne ChatGPT oder Gemini auskommt. Kein Wunder, insbesondere ChatGPT dominiert die Berichterstattung, wenn es um KIs geht. Wir werden uns diese Tools nicht nur anschauen, sondern auch ihre Programmierschnittstellen verwenden, sodass Sie ganz leicht eigene Anwendungen programmieren können, die im Stande sind, Beeindruckendes zu leisten.

## 1.2 Was ist eine »künstliche Intelligenz«?

Die Encyclopedia Britannica ([www.britannica.com](http://www.britannica.com)) definiert die KI bzw. AI (Artificial Intelligence) folgendermaßen:

*»Artificial intelligence (AI) is the ability of a computer or a robot controlled by a computer to do tasks that are usually done by humans because they require human intelligence and discernment. Although there are no AIs that can perform the wide variety of tasks an ordinary human can do, some AIs can match humans in specific tasks.«*

Die Webseite des Europäischen Parlaments ([www.europarl.europa.eu](http://www.europarl.europa.eu)) hat auch eine Definition dafür:

*»Künstliche Intelligenz ist die Fähigkeit einer Maschine, menschliche Fähigkeiten wie logisches Denken, Lernen, Planen und Kreativität zu imitieren.«*

Auch in der Literatur finden Sie ähnliche Formulierungen. Nach diesen Definitionen kann man somit auch das Navigationsgerät im Auto als künstliche Intelligenz interpretieren. Es ist in der Tat nicht einfach, die KI eindeutig zu definieren. Das kann schon ein gewiefter Algorithmus sein, um den schnellsten Weg von A nach B zu finden. Wir können aber festhalten, dass es darum geht, menschliche Intelligenz in irgendeiner Form zu imitieren.

Das maschinelle Lernen (*Machine Learning*, ML) ist eine Teilmenge der künstlichen Intelligenz. Darunter fallen selbstlernende Programme ohne vorgegebenen Algorithmus. Wir wollen das an einem Beispiel verdeutlichen: Sie trainieren ein Programm, die Summe zweier Zahlen zu berechnen. Dazu geben Sie viele mögliche Kombinationen nacheinander ein, z. B.  $1 + 3 = 4$ ,  $2 + 2 = 4$ ,  $2 + 6 = 8$  usw. Das Programm versucht schon während der Eingabe der Summanden, eine Verknüpfung zu finden und das von Ihnen eingegebene Ergebnis noch vor dem Eintippen vorherzusehen. Jedes Mal, wenn Sie schließlich das Ergebnis eingeben, wird kontrolliert, ob die Vorhersage richtig war (Trainingsphase). Wenn schließlich irgendwann eine korrekte Verknüpfung gefunden wird und diese sich auch für weitere Summanden als richtig erweist (Testphase), ist das ML-Programm so weit. Jetzt müssen nur die Summanden eingegeben werden, das Ergebnis wird automatisch ermittelt. Bei diesem Ansatz muss also kein Algorithmus eingegeben werden, um eine Summe zu berechnen. Die KI erlernt mithilfe der Trainings- und Testdaten selbst die richtige Verknüpfung.

Das maschinelle Lernen lässt sich je nach Fachliteratur in verschiedene Unterkategorien einordnen. Wir wollen uns hier auf drei Unterkategorien einigen.

### ► *Supervised Learning*

Das überwachte Lernen braucht Trainingsdaten, um zu lernen. Beispiel: Sie haben Bilder von Hunden und Katzen. Eine Person hat im Vorfeld alle vorhandenen Bilder richtig zugeordnet (man spricht hier von »labeln«). Die Trainingsdaten bestehen also aus Bildern und den zugehörigen Labeln. Sie trainieren mit diesen Trainingsdaten das Programm. Anschließend ist das Programm selbst in der Lage, neue Bilder von Hunden und Katzen richtig zuzuordnen. Auch das weiter oben beschriebene Beispiel mit den Summanden fällt unter diese Kategorie.

### ► *Unsupervised Learning*

Das unüberwachte Lernen wird oft verwendet, um in großen Datenmengen nach Mustern zu suchen und diese zu klassifizieren. Aus den Sensordaten einer Maschine können z. B. die Zustände »kritisch« oder »unkritisch« abgeleitet werden.

### ► *Reinforcement Learning*

Das verstärkende Lernen findet z. B. in Spielen bzw. beim Testen von Spielen Anwendung. Eine Figur lernt selbstständig richtige Spielzüge. Bei falschen Spielzügen er-

folgt eine Bestrafung (z. B. Punktabzug), bei richtigen eine Belohnung. Mit der Zeit lernt die Figur die richtigen Spielzüge und kommt immer weiter voran.

Wir werden uns in diesem Buch mit allen drei Unterkategorien des maschinellen Lernens beschäftigen. Die Begriffsdefinitionen ließen sich hier noch lange fortführen (z. B. künstliches neuronales Netz, Entscheidungsbäume, Deep Learning), darauf soll an dieser Stelle aber verzichtet werden. Vielmehr wollen wir erst darauf eingehen, wenn an entsprechender Stelle auch die Programme dazu entwickelt werden.

### 1.3 Geschichte der KI – ein kurzer Überblick

Schon seit langer Zeit befassen sich die Menschen mit der Idee von Maschinen, die menschliche Fähigkeiten besitzen. Der griechische Philosoph Aristoteles (334–322 v. Chr.) hat in seinem Werk »Politik« geschrieben, dass eines Tages Automaten die Sklaven ersetzen könnten:

*»Denn wenn es möglich wäre, dass jedes Werkzeug auf Geheiß oder vorbewusst sein Werk vollbringen könnte, wie angeblich die Statuen des Dädalos oder die Dreifüße des Hephästos, von denen der Dichter sagt, dass sie von selbst sich in die Versammlung der Götter begeben hätten und wenn so auch das Weberschiff von selbst webte und die Zither von selbst spielte, so bedürften weder die Künstler der Gehilfen noch die Herren der Sklaven.«*

Der Universalgelehrte Ismael al-Dschazari (1136–1206) hat für seine Dienstherrn einige mechanische Apparate konstruiert, wie z. B. einen von Wasser angetriebenen Pfau oder musizierende Automaten mit wechselnden Gesichtsausdrücken. Es existieren auch Skizzen von Leonardo da Vinci (1452–1519) über Roboterritter mit Gelenkverbindungen aus Flaschenzügen. Beachtlich, welche Visionen diese Vordenker\*innen (und es gab noch viele weitere) zu ihrer Zeit hatten.

Einige wichtige Meilensteine in der Moderne zeigen die rasante Entwicklung seit den 1940ern:

- ▶ 1943 wurde die Arbeit »A Logical Calculus oft the Ideas Immanent in Nervous Activity« vom Neuropsychologen Warren McCulloch und dem Logiker Walter Pitts veröffentlicht. Diese bahnbrechende Veröffentlichung kann als Grundlage für das Prinzip der künstlichen neuronalen Netze angesehen werden, welche wir im Verlauf dieses Buches noch kennenlernen werden.
- ▶ Im Jahre 1956 fand in den USA eine Konferenz mit dem Titel »The Dartmouth Summer Research Project on Artificial Intelligence« statt, wo sich Forscher zu dem The-

mengebiet austauschen konnten und der Begriff »künstliche Intelligenz« zum ersten Mal schriftlich festgehalten wurde.

- ▶ Der Ingenieur Arthur Samuel sorgte 1959 mit der Veröffentlichung seines Aufsatzes »Some Studies in Machine Learning Using the Game Checkers« für die Geburtsstunde des Begriffes »Machine Learning«. Das von ihm entwickelte Programm, welche auf ML basierte, konnte gegen Menschen und sich selbst Dame spielen.
- ▶ Das Programm BKG 9.8 von Hans Berliner (Wissenschaftler an der Carnegie Mellon University Pittsburgh) besiegte 1979 den damals amtierenden Weltmeister in Backgammon. Das war das erste Mal, dass ein Weltmeister bzw. eine Weltmeisterin von einem Computerprogramm besiegt wurde.
- ▶ 1997 schlug IBMs Computer Deep Blue den damaligen Schachweltmeister Garri Kasparow. Deep Blue konnte bis zu 200 Millionen Schachstellungen pro Sekunde und über sechs Züge im Voraus berechnen.
- ▶ Das Programm AlphaGo der Firma DeepMind schlug 2016 im Spiel Go den damaligen Weltmeister Lee Sedol. Die Anzahl möglicher Züge in diesem Spiel ist größer als die Anzahl der Atome im Universum.
- ▶ 2022 veröffentlichte die Firma OpenAI den Chatbot ChatGPT, eine KI von ganz neuer Qualität. Mit diesem Chatbot kann man Konversation wie mit einem Menschen führen. ChatGPT kann z. B. vereinfachen, zusammenfassen, komponieren, programmieren und philosophieren. Sie kann sogar Kurse an Elite-Universitäten bestehen, wie Tests gezeigt haben.

Die Entwicklung der letzten 80 Jahre ist bemerkenswert, und die KI leistet in vielen Bereichen, wie z. B. der Medizin, große Dienste. Die Nachteile darf man aber nicht aus den Augen verlieren. In sozialen Medien wird mithilfe dieser Technologie dafür gesorgt, dass User für sie gezielt ausgewählte Nachrichten bzw. Inhalte präsentiert bekommen. Kann man noch von freiem Willen sprechen, wenn man seit der frühen Jugend in dieser Nachrichtenblase lebt? Ist man in seinen Entscheidungen objektiv und kann jahrelangen, gezielten Nachrichtenfluss wirklich ausblenden? Wollen wir zulassen, dass die KI uns Menschen klassifiziert und bewertet, z. B. bei Bewerbungen? Das hat nämlich oft nichts mit Objektivität zu tun, wie Sie noch sehen werden, wenn wir in Kapitel 3, »Das künstliche neuronale Netz«, über Korrelation und Kausalität sprechen. Diese sind nur einige Punkte, welche die Gefahren aufzeigen sollen.

Nichtsdestotrotz darf man die Augen nicht verschließen und denken, dass das alles nur eine Modeerscheinung ist. Die Entwicklung wird sich nicht aufhalten lassen und bietet große Chancen. Die KI ist mittlerweile in allen Industriezweigen angekommen. Konzentrieren wir uns auf die positiven Aspekte. Das Wissen in diesem Bereich wird auch Ihr Urteilsvermögen über diese Technologien stärken.

## 1.4 Verwendete Werkzeuge

Es stehen viele Programmiersprachen, Bibliotheken und Tools zur Verfügung, um KIs zu entwickeln. Hier soll eine kurze Vorstellung über die Auswahl erfolgen, die in diesem Buch Anwendung findet. Es wird noch nichts installiert oder programmiert.

### 1.4.1 Python

Die Programmiersprache Python ist eine sehr einfach zu erlernende, aber gleichzeitig sehr mächtige Sprache. Schon bei der Entwicklung wurde Wert auf Lesbarkeit, Plattform-unabhängigkeit und einfache Strukturen gelegt. Auch wenn Sie unter Windows programmieren, ist das Programm anschließend auf einem Linux-Rechner oder macOS lauffähig. Diese Vorteile haben dazu geführt, dass sie mittlerweile eine der beliebtesten Programmiersprachen überhaupt ist. In der Welt der Data Science und KI hat sie sich als De-facto-Standard durchgesetzt. Wie schon erwähnt, soll dieses Buch keine Einführung in die Programmiersprache Python sein. Für uns ist sie nur Mittel zum Zweck. Die folgenden Python-Anweisungen sollen die Einfachheit verdeutlichen. Wir werden mit der Programmierung erst beginnen, nachdem im nächsten Kapitel die Installation erfolgt ist.

Das obligatorische »Hallo Welt«-Beispiel sieht folgendermaßen aus:

```
print("Hallo Welt!")
```

Mit der `print`-Funktion können Daten in der Konsole ausgegeben werden, in diesem Falle »Hallo Welt!«. Wir können diese Zeichenkombination (Datentyp String) auch erst in einer Variable speichern und danach ausgeben:

```
ausgabe = "Hallo Welt!"
print(ausgabe)
```

#### Datentypen

Die elementarsten Datentypen in Python sind

- ▶ Strings (Zeichenketten), welche in einfachen oder doppelten Anführungszeichen geschrieben werden (z. B. "Hallo Welt!"),
- ▶ Integer (Ganzzahlen, z. B. 72) und
- ▶ Float (Fließkommazahlen, z. B. 3.14).

Je nach Datentyp sind bestimmte Operationen möglich. String-Daten wie »Hallo« und »Welt« können zu »Hallo Welt« zusammengefügt werden. Integer- oder Float-Daten können miteinander verrechnet werden, z. B.  $3 \times 8$ .





Weitere Datentypen werden wir erst vorstellen, wenn sie zur Lösung einer bestimmten Aufgabenstellung benötigt werden.

Dieselbe Variable kann auch für eine Ganzzahl (Datentyp Integer) oder Fließkommazahl (Datentyp Float) wiederverwendet werden. In Python-Variablen können Daten beliebigen Typs gespeichert werden.

```
ausgabe = "Hallo Welt!"  
print(ausgabe)
```

```
ausgabe = 42  
print(ausgabe)
```

```
ausgabe = 42.2  
print(ausgabe)
```

Hier wird nacheinander »Hallo Welt!«, 42 und 42.2 ausgegeben. Sehen wir uns nun ein vermeintlich komplizierteres Programm an:

```
# Festlegung des Alters auf 20  
alter = 20  
# Abfrage, ob eine Volljährigkeit vorliegt  
if alter >= 18:  
    print("Du bist volljährig!")  
else:  
    print("Du bist noch nicht volljährig!")
```

Das #-Zeichen leitet Kommentare ein, die beim Lesen des Codes helfen können; diese Zeilen werden nicht ausgeführt. In der Variable `alter` wird die Zahl 20 gespeichert. Danach kommt eine Abfrage, ob das Alter größer oder gleich 18 ist. Trifft das zu (wie in diesem Beispiel der Fall), erfolgt die entsprechende Ausgabe. Der `else`-Zweig wird übersprungen und das Programm beendet. Klären wir folgende Fragen:

1. Welche Auswirkung hätte es, wenn in der Variable `alter` der Wert 16 gespeichert werden würde? Antwort: Der `if`-Zweig würde übersprungen und der `else`-Zweig ausgeführt werden. Die Ausgabe wäre »Du bist noch nicht volljährig!«
2. In manchen Ländern ist man erst mit 21 volljährig. Welche Änderung ist an diesem Programm dafür notwendig? Antwort: Bei der `if`-Abfrage erfolgt der Vergleich mit der Zahl 21 statt mit 18.
3. Kann man die oben durchgeführte Änderung für die Abfrage der Volljährigkeit als »Programmierung« bezeichnen? Oder als »Konfiguration«? Darüber kann man streiten, wenn man nichts Besseres zu tun hat.

Wie Sie sehen, ist es in Python recht einfach, den Quellcode nachzuvollziehen.

### 1.4.2 Jupyter Notebook

Wir benötigen eine Software, in welche wir die Python-Befehle eintippen können. Mithilfe dieser Software soll programmiert, ausgeführt und getestet werden. Software dieser Art nennt man *integrierte Entwicklungsumgebung* (*IDE, Integrated Development Environment*). Für Python gibt es eine Reihe von sehr guten Entwicklungsumgebungen. In Data Science und KI-Entwicklung haben sich jedoch einfache Jupyter Notebooks (open source) etabliert. Hier wird im Browser programmiert, meist besteht das Programm aus einer einzigen Datei. Um Python sowie Jupyter Notebook (und bei Bedarf viele weitere Tools) zu installieren und zu konfigurieren, werden wir die kostenlose, für diesen Einsatzzweck entwickelte Anaconda-Distribution verwenden. Diese gibt es für Windows, Linux und macOS. Mit Anaconda können Sie einfache Entwicklungsumgebungen lokal am PC oder zentral auf einem Server einrichten. Der Vorteil der Servervariante ist, dass man von mehreren Clients aus einfach per Browser mit Jupyter Notebook programmieren kann. An diesen Clients ist keine weitere Installation notwendig. Wenn Sie hingegen keinen eigenen Server benutzen wollen, sondern einen von Anaconda in der Cloud zur Verfügung gestellten, kostet es Geld.

#### Entwicklungsumgebung

Eine integrierte Entwicklungsumgebung (IDE, Integrated Development Environment) stellt wichtige Werkzeuge zur Verfügung, welche Sie bei der Programmierung unterstützen. Dazu gehören das automatische Erkennen und Vervollständigen von Befehlen, Autokorrektur sowie hilfreiche Hinweise bei Programmierfehlern.



### 1.4.3 Orange

Die Software Orange verfolgt einen ganz anderen Ansatz: Die Programmierung erfolgt mittels grafischer Bausteine, welche per Drag-and-Drop in den Arbeitsbereich gezogen, miteinander verbunden und konfiguriert werden. Man nennt das auch visuelle Programmierung. Orange Data Science (meist einfach *Orange* genannt) wurde von der University of Ljubljana in Slowenien entwickelt. Die Software ist open source, kostenlos und plattformunabhängig.

In Abbildung 1.1 sehen Sie ein Programm mit einem künstlichen neuronalen Netz, welches mit Trainingsdaten trainiert und mit Testdaten getestet wird. Der Aufbau ist sehr übersichtlich und gut nachvollziehbar.

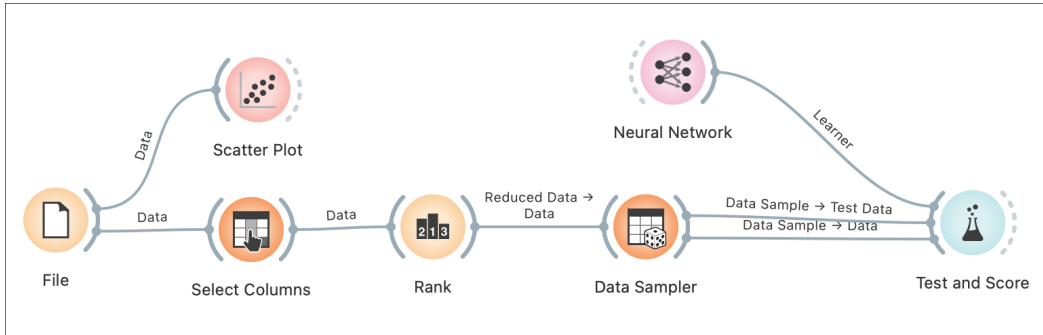


Abbildung 1.1 Ein Programm mit einem künstlichen neuronalen Netz

### 1.4.4 KNIME

KNIME ist, wie Orange auch, ein Tool zur visuellen Programmierung, hat aber einen größeren Funktionsumfang. Dafür bedarf es auch einer längeren Einarbeitungszeit. KNIME basiert auf Eclipse, einer weit verbreiteten Entwicklungsumgebung für verschiedenste Programmiersprachen. Auch diese Software ist open source, kostenlos und plattform-unabhängig.

Das folgende Beispiel zeigt ein Programm, welches die Inhalte einer Excel-Datei ausliest und die Zahlenwerte in einem Grafen plottet (siehe Abbildung 1.2).

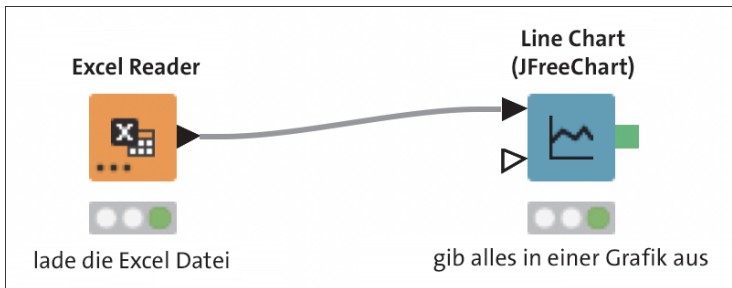


Abbildung 1.2 Laden einer Excel-Datei und Plotten der Inhalte

Die Ausgabe des Bausteins »Line Chart« sehen Sie in Abbildung 1.3: Dazu müssen Sie nach dem Ausführen einen Rechtsklick auf den Baustein ausführen und auf VIEW: LINE CHART klicken.

Ist die Datenquelle nicht eine Excel-Datei, sondern z. B. eine Datenbank oder CSV-Datei, müssen Sie nur den Baustein EXCEL READER ersetzen.

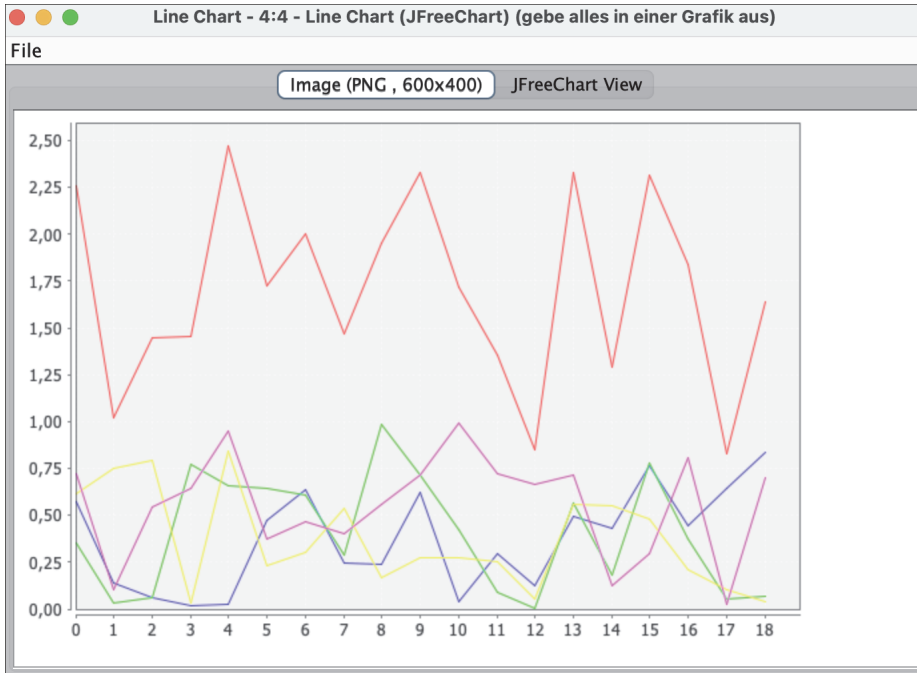


Abbildung 1.3 Plotten der Inhalte

### 1.4.5 ChatGPT und Gemini

Fragen wir einfach ChatGPT oder Gemini selbst, wer oder was es ist ...

**User:** Wer oder was bist du? Erkläre es mir so, als wäre ich 12 Jahre alt.

**Antwort:** Okay! Stell dir vor, ich bin eine richtig schlaue Art Roboter – aber einer, der nicht herumlaufen kann. Ich lebe nur auf Computern oder im Internet.

Mein Name ist **ChatGPT**.

Ich bin wie ein riesiges Buch, in dem du alles Mögliche nachschlagen kannst – nur dass du mir direkt Fragen stellen kannst, und ich antworte dir in meinen eigenen Worten.

Ich habe ganz viel gelernt, indem ich riesige Mengen an Texten gelesen habe, z. B. Bücher, Artikel, Webseiten. Deshalb kann ich dir bei Hausaufgaben helfen, Geschichten erzählen, Witze machen oder sogar Dinge erklären, die schwierig wirken.

Aber: Ich weiß nicht alles. Ich denke mir nichts einfach aus, wenn ich es nicht sicher weiß (obwohl ich manchmal aus Versehen Fehler machen kann). Ich kann auch nicht selber etwas fühlen oder etwas wirklich erleben – ich tue nur so, als könnte ich.

**Kurz gesagt:** *Ich bin ein sehr kluges Computerprogramm, das dir hilft, Antworten auf deine Fragen zu finden und mit dir zu reden, fast so wie ein freundlicher Mitschüler oder Lehrer.*

Auf den Webseiten von ChatGPT bzw. Gemini können Sie diese Chatbots ausprobieren. Diese beeindruckenden Tools werden wir uns noch genauer anschauen.

### 1.4.6 DALL-E und Imagen

Diese Tools basieren auf den Modellen von ChatGPT bzw. Gemini, welche für die Text-zu-Bild-Synthese angepasst wurden. Sie sind mittlerweile in ChatGPT und Gemini integriert. Der Name ist von Salvador Dali (bedeutender surrealer Künstler) bzw. dem englischen Wort »Image«) abgeleitet.

Mit diesen Tools können Sie Bilder aus textuellen Beschreibungen generieren lassen (siehe Abbildung 1.4).

**User:** *Generiere folgendes Bild: Ein Programmierer sitzt in einem Passagierflugzeug am Fensterplatz und arbeitet konzentriert an seinem Laptop. Auf dem Bildschirm sind Codezeilen sichtbar. Die Szene ist realistisch, modern und detailreich dargestellt, mit ruhigen Farben und sanftem Tageslicht, das durch das Flugzeugfenster fällt. Der Programmierer trägt Freizeitkleidung.*



**Abbildung 1.4** Das generierte Bild von ChatGPT (bzw. DALL-E)

Das Bild wirkt, als sei eine reale Person im Flugzeug fotografiert worden. Auch mit DALL-E und Imagen werden wir uns noch näher beschäftigen.

# Kapitel 2

## Installation von Anaconda

*Bevor wir mit der eigentlichen Entwicklung beginnen, werden wir Anaconda installieren und testen.*

### Worum geht es in diesem Kapitel?

Die wichtigsten Werkzeuge, die Sie für die Arbeit mit diesem Buch benötigen, werden installiert und getestet. Weitere Tools und Module werden wir an geeigneter Stelle installieren, wenn wir sie benötigen.

Die Einrichtung eines Entwicklungsrechners kann sehr zeitaufwendig sein. Nehmen Sie sich diese Zeit! Wir wollen uns später bei der Entwicklung nicht mit kuriosen Fehlermeldungen herumschlagen.

### Betriebssystemversionen

Die folgenden Installationsanweisungen wurden auf diesen Maschinen getestet:

- ▶ Windows 11
- ▶ macOS Sequoia 15.3.2 (Apple M2)
- ▶ Ubuntu LTS Noble Numbat 24.04



Mithilfe der Anaconda-Distribution werden wir Python sowie Jupyter Notebook installieren und die Installationen testen. Die Installationsdatei für die Betriebssysteme Windows, macOS und Linux können Sie auf der Website [www.anaconda.com/products/distribution](http://www.anaconda.com/products/distribution) herunterladen. Über den Browser wird das Betriebssystem erkannt und die richtige Installationsdatei heruntergeladen.

## 2.1 Windows und macOS

Führen Sie die Installationsdatei aus, und folgen Sie den Anweisungen auf dem Bildschirm. Die empfohlenen Standardeinstellungen können beibehalten werden. Nach der Installation können Sie den Anaconda-Navigator starten (siehe Abbildung 2.1).

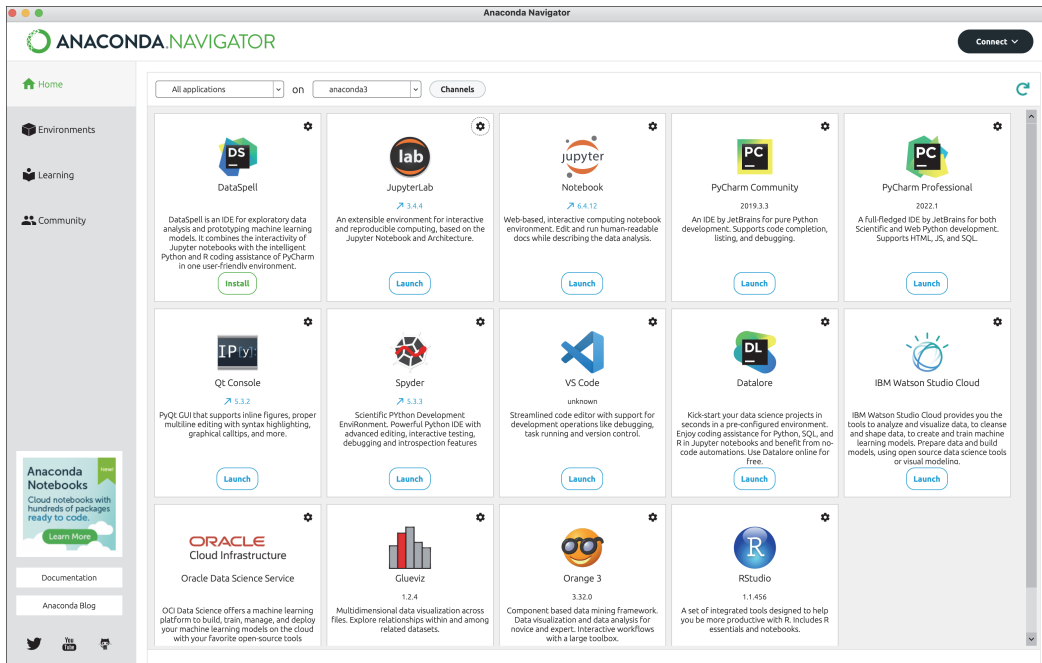


Abbildung 2.1 Home-Bildschirm des Anaconda-Navigators

Nun können wir uns der noch notwendigen Konfiguration des Entwicklungsrechners widmen.

## 2.2 Linux



### Installationsanleitung für Anaconda

Eine sehr gute Installationsanleitung für Anaconda finden Sie unter [wiki.ubuntuusers.de/anaconda](http://wiki.ubuntuusers.de/anaconda).

Nachdem die Datei heruntergeladen und die Installationsroutine durchgeführt wurde, öffnen Sie ein neues Terminalfenster (damit die neuen Umgebungsvariablen wirken) und geben folgenden Befehl ein, um den Anaconda-Navigator zu starten:

```
anaconda-navigator
```

Damit ist die Installation schon abgeschlossen. Sie müssen aber noch einige Einstellungen vornehmen.

## 2.3 Konfiguration und Test

Wir werden ein neues Environment für die Beispiele im Buch anlegen. Wenn Sie im Anaconda-Navigator auf ENVIRONMENT klicken, erhalten Sie eine Übersicht über alle vorhandenen Environments (momentan nur *base*, welcher standardmäßig bei der Installation angelegt wurde). Hier können Sie auch Module installieren oder deinstallieren (siehe Abbildung 2.2).

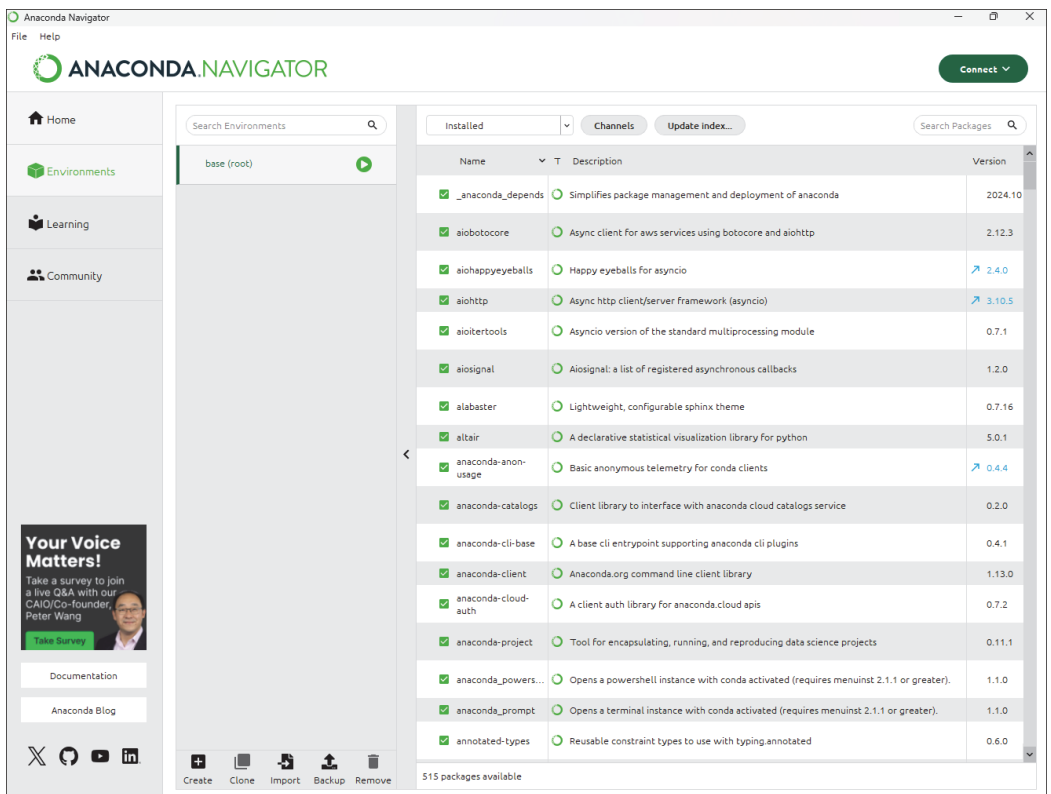


Abbildung 2.2 Übersicht über die Environments

In dieser Ansicht klicken Sie links unten auf CREATE und legen ein neues Environment an. Im Pop-up-Fenster müssen Sie einen Namen vergeben und eine Python-Version auswählen (siehe Abbildung 2.3).



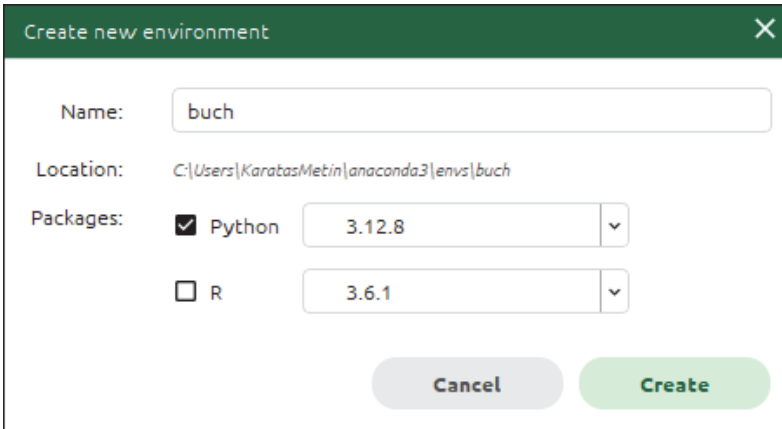


Abbildung 2.3 Environment anlegen

Ich habe hier zuerst die Version 3.13 installiert. Leider konnte ich dann einige Module wie TensorFlow nicht installieren, die Prozedur wurde mit einer Fehlermeldung abgebrochen. Daher habe ich mich schlussendlich für die Version 3.12.8 entschieden. Damit funktionieren alle im Buch vorgestellten Programme.



### Anaconda und Environments

In Python sind Environments isolierte Umgebungen, in denen Python-Quellcode ausgeführt werden kann. In jedem Environment können unterschiedliche Versionen von Paketen installiert sein. Der Vorteil vom getrennten Environment ist, dass die Installationen in einem Environment nicht die anderen Environments beeinflussen.

Wenn Sie ein wenig Erfahrung mit Python bzw. Anaconda haben, können Sie ein eigenes Environment für Jupyter Notebook anlegen und die Pakete darin installieren. Sie können aber auch das Base-Environment verwenden.

Nach erfolgreicher Installation können Sie im Home-Bereich des Anaconda-Navigators verschiedene Anwendungen über den jeweiligen Launch-Button starten (achten Sie bitte darauf, dass Sie das richtige Environment, hier »buch«, selektiert haben). Wenn Sie nun hierüber Jupyter Notebook starten, kommen Sie auf die Jupyter-Notebook-Homepage (siehe Abbildung 2.4).

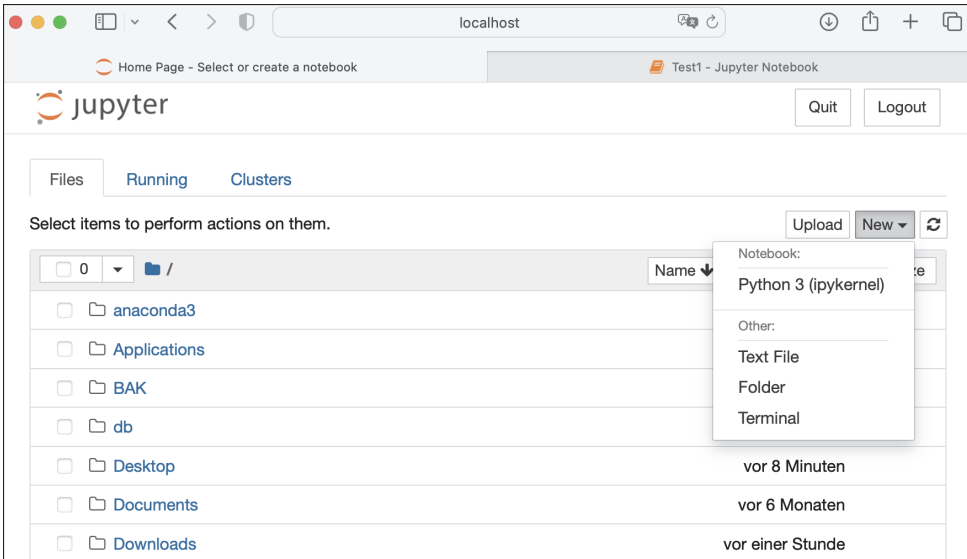


Abbildung 2.4 Jupyter-Notebook-Homepage

Hier gehen wir auf NEW • PYTHON3 (IPYKERNEL) und starten ein neues Notebook (siehe Abbildung 2.5).

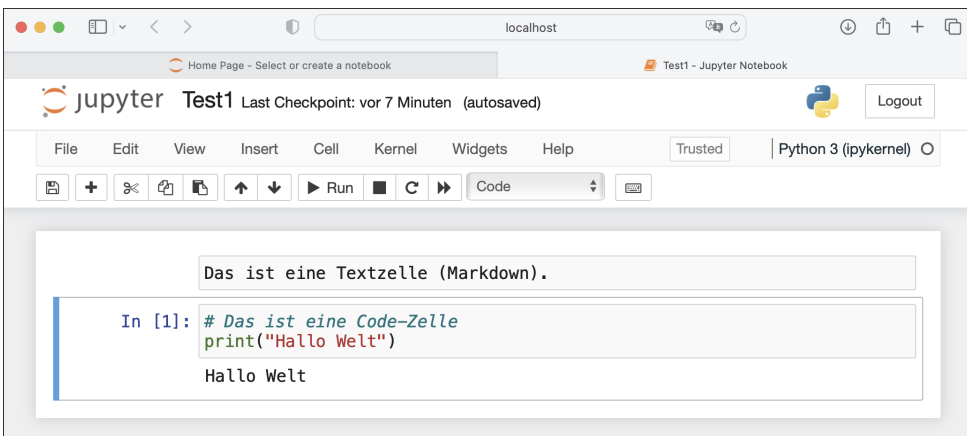


Abbildung 2.5 Erstes Programm in Jupyter Notebook

Das Programm ist in verschiedenen Zellen organisiert. Mit dem Button + können Sie eine neue Zelle hinzufügen und mit den Pfeil-Buttons die selektierte Zelle jeweils nach oben oder unten verschieben. Im Dropdown-Menü können Sie einstellen, ob die Zelle nur zur Dokumentation dient (Markdown) oder aber Quellcode enthält (Code). Der Play-

Button führt die selektierte Codezelle, der Forward-Button alle Zellen aus. Mit FILE • RE-NAME können Sie die Datei umbenennen, mit FILE • SAVE AS angeben, wo die Datei gespeichert werden soll.

Im Anaconda-Navigator können Sie alternativ JupyterLab starten (siehe Abbildung 2.6), was ebenfalls Notebooks ausführen kann, aber mehr Funktionalität bietet.

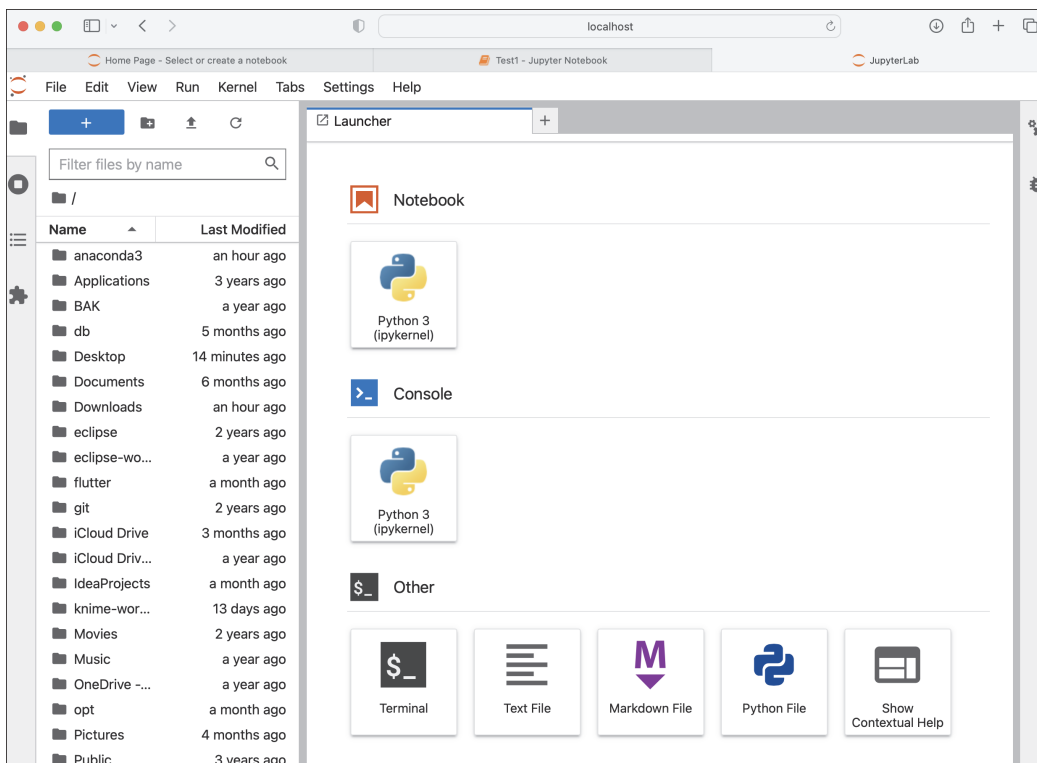


Abbildung 2.6 JupyterLab Launcher

Hier können Sie mit NOTEBOOK • PYTHON 3 (IPYKERNEL) ein neues Notebook starten (siehe Abbildung 2.7) oder aber z. B. die oben erstellte Datei *Test1.ipynb* öffnen. Die Verzeichnisse und Dateien sind links aufgelistet.

Bei dieser Gelegenheit werden wir einige benötigte Module installieren. Mit der Installation von Python haben Sie nicht sofort Zugriff auf alle Funktionalitäten, die diese Programmiersprache bietet. Vielmehr ist Python so aufgebaut, dass Sie weitere benötigte Module bei Bedarf nachinstallieren können. Dieser modulare Aufbau sorgt dafür, dass das Programm nicht unnötig viel Speicherplatz einnimmt.

# Kapitel 11

## Visuelle Programmierung mit KNIME

*Visuelle Datenanalyse oder KI – ohne Programmierung oder mit optionalem Code: KNIME bietet beides.*

### Worum geht es in diesem Kapitel?

- ▶ Installation
- ▶ Einführung in die visuelle Programmierung
- ▶ Regression und Klassifizierung mit künstlichen neuronalen Netzen
- ▶ Regression und Klassifizierung mit XGBoost
- ▶ Bildklassifizierung mit vortrainierten Modellen
- ▶ Transfer Learning
- ▶ Autoencoder
- ▶ Textklassifizierung
- ▶ automatische Erstellung von KI-Modellen mit AutoML
- ▶ Clusteranalyse
- ▶ Zeitreihenanalyse

Mit KNIME können Sie verschiedenste KI-Modelle mithilfe von grafischen Bausteinen programmieren. Einer der Vorteile von visueller Programmierung ist die Übersichtlichkeit. Selbst wenn das Programm von jemand anderem entwickelt worden ist, können Sie sich relativ schnell einarbeiten, der Datenfluss ist einfach nachvollziehbar. Dabei ist KNIME alles andere als eine Spielerei. Viele Firmen und Forschende nutzen diese Software für professionelle Aufgaben.

Mit KNIME ist es auch relativ einfach, mit Daten aus verschiedenen Quellen zu arbeiten, wengleich eine längere Einarbeitungszeit als bei Orange benötigt wird. Bei Programmiersprachen wie Python müssen Sie, je nach Datenquelle, verschiedene Module importieren und mithilfe dieser die Daten einlesen. Für Dateiformate wie Excel oder CSV gibt es eigene Module, die jeweils andere Funktionen zur Verfügung stellen. Wenn die Daten aus einer Datenbank eingelesen werden sollen, müssen Sie sich mit der Datenbanksprache SQL (Structured Query Language) vertraut machen. Bei KNIME gibt es jeweils verschiedene Bausteine für die Datenquellen, wie z. B. die Bausteine EXCEL READER oder CSV READER. Sogar für Datenbanken gibt es Bausteine, sodass Sie ohne

SQL-Kenntnisse die Daten einlesen können. Nachdem die Daten aus der jeweiligen Quelle eingelesen worden sind, kann man sie auf eine einheitliche Art und Weise verarbeiten und für die Entwicklung von KI-Modellen nutzen. Als KI-Entwicklerin bzw. KI-Entwickler müssen Sie sich nicht erst aufwändig mit der Anbindung der Datenquelle oder dem Import der Daten beschäftigen. Sie können auch Python-Quellcode einbinden, wo es Ihnen sinnvoll erscheint.

Die Programme, die in diesem Buch vorgestellt werden, können Sie selbst programmieren oder aber die zur Verfügung gestellten Programme importieren. Nach dem Import muss der Pfad zur jeweiligen Datenquelle (CSV-Datei) aktualisiert werden. Dann wollen wir mal loslegen.

## 11.1 Installation

Die Software *KNIME Analytics Plattform* können Sie unter [www.knime.com/downloads](http://www.knime.com/downloads) für das bevorzugte Betriebssystem herunterladen. Bei Windows und macOS führen Sie das Installationsprogramm aus, die empfohlenen Einstellungen können beibehalten werden. Bei Linux müssen Sie nur das Archiv entpacken (in Ihrem Home-Verzeichnis).



### Darstellung mit Linux Wayland

Beim Start von KNIME bekommen Sie einen Hinweis, dass Sie für eine optimale Darstellung Xorg nutzen sollten. Sie können es trotzdem mit Wayland probieren. Wenn die Darstellung nicht zufriedenstellend sein sollte, können Sie sich abmelden und im Anmeldefenster auf Xorg umstellen.

Nach dem Start des Programmes erscheint ein Fenster mit dem Hinweis, den Workspace (in dem Ihre Programme und Konfigurationen gespeichert werden) auszuwählen (siehe Abbildung 11.1). Setzen Sie den Haken bei **USE THIS AS THE DEFAULT AND DO NOT ASK AGAIN**, und klicken Sie auf **LAUNCH**.

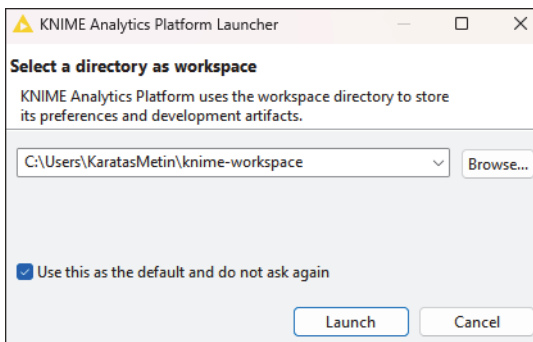


Abbildung 11.1 Auswahl des Workspace

Nachdem Sie den Workspace eingestellt bzw. bestätigt haben, erscheint der Home-Bildschirm von KNIME (siehe Abbildung 11.2).

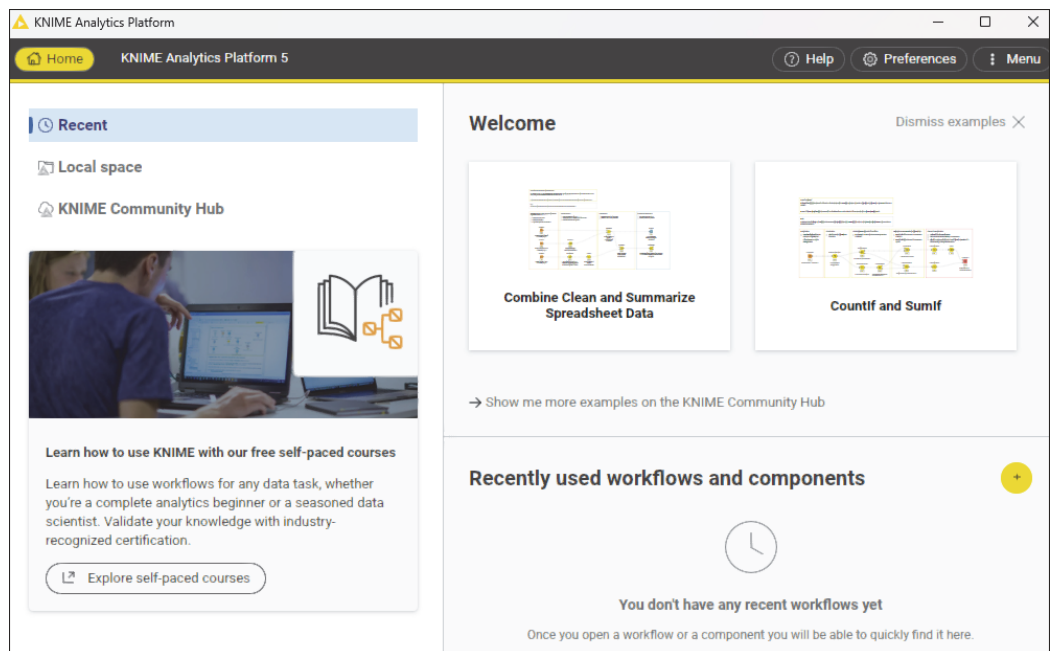


Abbildung 11.2 Home-Bildschirm von KNIME

Installieren wir noch weitere benötigte Pakete. Gehen Sie dazu in der Menüleiste auf MENU • INSTALL EXTENSIONS. Bei WORK WITH stellen Sie ein, dass auf allen Seiten gesucht werden soll. Suchen Sie anschließend nach Python, und selektieren Sie die Optionen, wie in Abbildung 11.3 gezeigt:

- ▶ SHOW ONLY THE LATEST VERSION OF AVAILABLE SOFTWARE (nur neueste Versionen anzeigen)
- ▶ HIDE ITEMS THAT ARE ALREADY INSTALLED (nur anzeigen, was noch nicht installiert wurde)
- ▶ GROUP ITEMS BY CATEGORY (nach Kategorie sortieren)

Dann wählen Sie zur Installation aus:

- ▶ KNIME Conda Integration
- ▶ KNIME Python Integration
- ▶ KNIME Python Scripting extension

- ▶ KNIME Deep Learning – Keras Integration
- ▶ KNIME Deep Learning – TensorFlow Integration
- ▶ KNIME Deep Learning – TensorFlow2 Integration
- ▶ KNIME Text Processing
- ▶ KNIME Image Processing
- ▶ KNIME XGBoost Integration

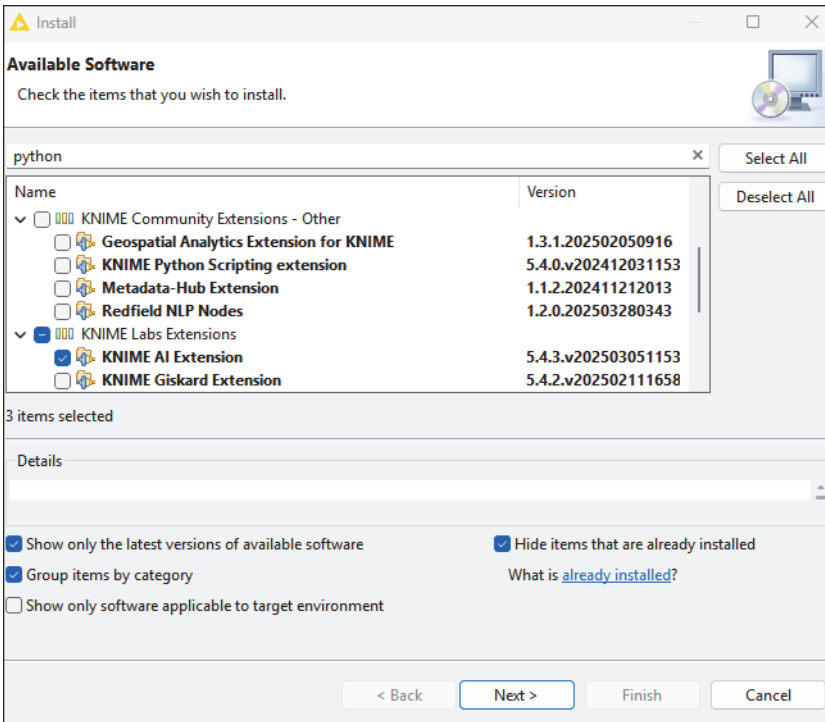


Abbildung 11.3 Installation weiterer Pakete

Klicken Sie auf NEXT, akzeptieren Sie die Lizenzbedingungen, und bestätigen Sie mit FINISH. Die selektierten Pakete werden heruntergeladen und installiert.

Wir müssen noch einige Einstellungen für Python, Keras und TensorFlow vornehmen. Um den Pfad zur Anaconda-Installation zu überprüfen, gehen Sie über die Menüleiste auf PREFERENCES • KNIME • CONDA (siehe Abbildung 11.4)

Falls der Pfad zu Ihrer Anaconda-Installation nicht automatisch ermittelt werden kann, müssen Sie diesen selbst explizit setzen. Als Nächstes folgt die Einstellung für Python (siehe Abbildung 11.5).

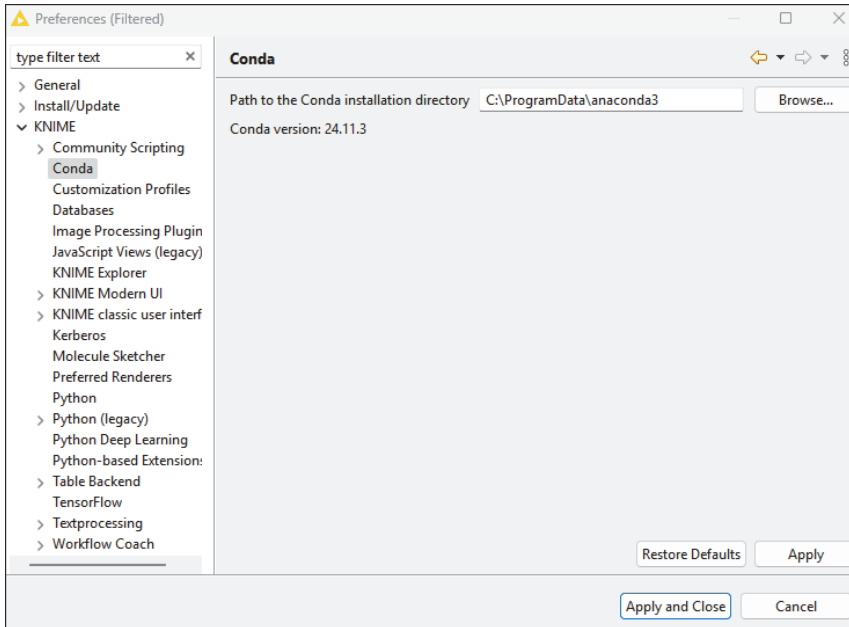


Abbildung 11.4 Einstellung für Conda

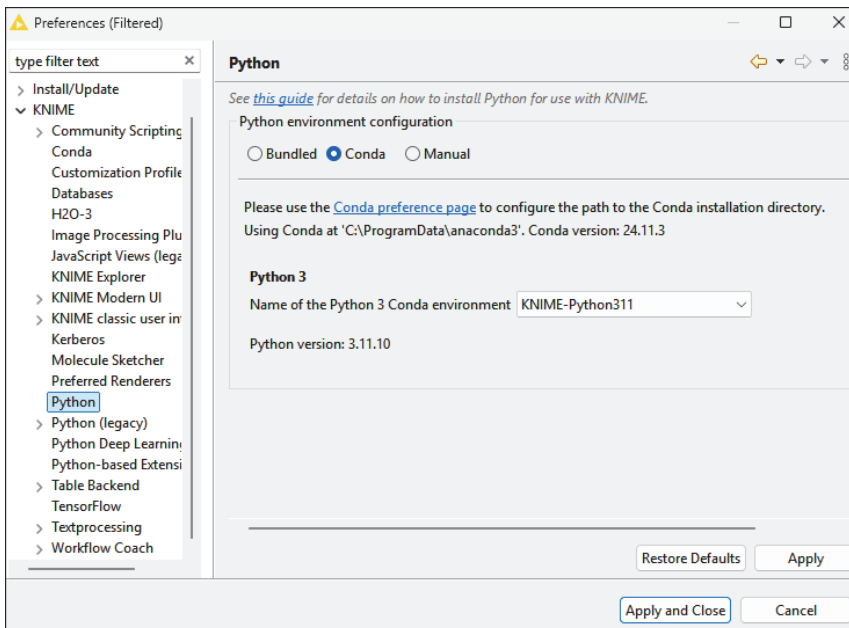


Abbildung 11.5 Einstellung für Python



Hier selektieren Sie CONDA, und warten kurz, bis KNIME alle Informationen zur Installation gesammelt hat. Wahrscheinlich wird eine Warnung angezeigt, dass einige Pakete fehlen. Daher werden wir nun ein Environment anlegen, das alle benötigten Pakete enthält. Es wird hier auch angezeigt, wie das zu realisieren ist. In Kapitel 2, »Installation von Anaconda«, haben wir bereits den Anaconda-Navigator installiert und ein Environment angelegt. Unter ENVIRONMENTS selektieren Sie dort z. B. BASE und öffnen über den PLAY-Button ein Terminal. Hier geben Sie genau den Befehl ein, der Ihnen in KNIME angezeigt wird, wobei Sie <ENV\_NAME> durch einen Namen ihrer Wahl ersetzen. Da Version 3.9 ziemlich veraltet ist, habe ich es mit Version 3.12 versucht – das führte leider zu einem Fehler. Mit Version 3.11 hatte ich aber Erfolg. Meine Eingabe lautet:

```
conda create -name KNIME-Python311 -c knime -c conda-forge knime-python-scripting python=3.11
```



### Apple-Silicon-Prozessor

Setzen Sie hier das Environment auf »buch«. Bitte beachten Sie auch den Hinweis am Ende dieses Abschnitts.

Nachdem das neue Environment erstellt worden ist, können Sie dieses in KNIME selektieren. Als Letztes müssen wir die Einstellung für Python Deep Learning konfigurieren. Übernehmen Sie dazu die Einstellungen von Abbildung 11.6, wobei wir noch je ein neues Environment für Keras und TensorFlow 2 anlegen müssen. Diese Environments müssen Sie aber nicht über den Anaconda-Navigator erstellen, sondern können das relativ einfach mit einem Klick auf NEW ENVIRONMENT umsetzen. Anschließend wird ein neues Fenster geöffnet und ein Name für das neue Environment vorgeschlagen, welchen Sie übernehmen können. Klicken Sie in diesem Fenster bitte auf CREATE NEW ENVIRONMENT. Führen Sie dies bitte sowohl für KERAS als auch für TENSORFLOW 2 durch.



### Apple-Silicon-Prozessor

Leider ist es aktuell nicht möglich, bei diesen Prozessoren die Einstellungen für Deep Learning wie oben beschrieben durchzuführen, da einige Pakete zur Erstellung von künstlichen neuronalen Netzen nicht unterstützt werden. Daher können Sie diesen Schritt überspringen. Ich werde Ihnen zeigen, wie Sie in KNIME Python-Bausteine verwenden können. Für die Datenverarbeitung kommen somit grafische Bausteine zum Einsatz, das künstliche neuronale Netz wird in einem Python-Baustein mit ein paar Zeilen Code aufgebaut.

In der Einstellung zu Python können sie das Environment einfach auf »buch« setzen, da wir hier schon alle benötigten Pakete installiert haben.

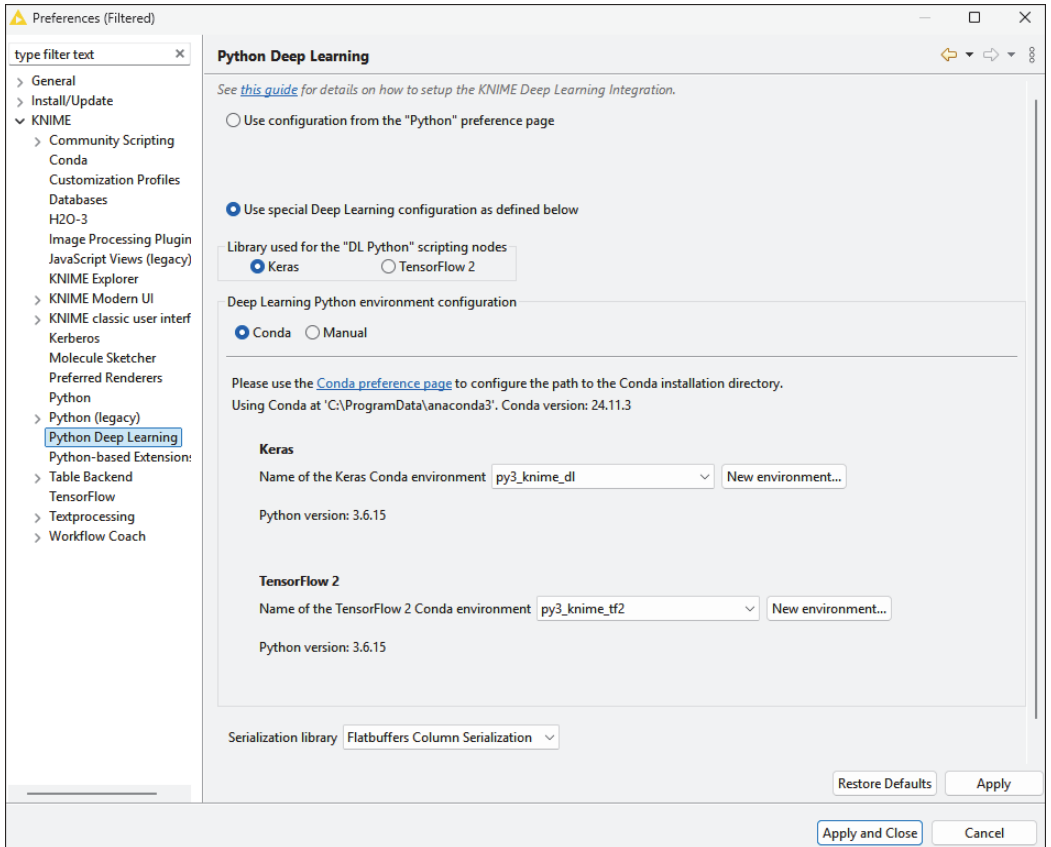


Abbildung 11.6 Einstellung für Python Deep Learning

## 11.2 Einfache künstliche neuronale Netze

Wenden wir das Erlernete über künstliche neuronale Netze (KNN) hier an. Vergleichen Sie die KNIME-Programme jeweils mit den Python-Programmen, die wir bereits in vorherigen Kapiteln erstellt haben.

Im Home-Bildschirm, Abschnitt LOCAL SPACE, können Sie Programme (Workflows) neu anlegen oder vorhandene importieren. Wenn Sie vorher noch nicht mit KNIME gearbeitet haben, schlage ich vor, dass Sie eines der zur Verfügung gestellten Programme importieren, anstatt ein Programm von Grund auf neu zu erstellen.

### 11.2.1 Klassifizierung

Der erste Workflow soll mithilfe von einem KNN Schwertlilien klassifizieren. Anhand der Länge und Breite der Blätter soll die Unterart der Schwertlilie bestimmt werden. Für diese Problemstellung haben wir bereits mit KNN und XGBoost KI-Modelle in der Programmiersprache Python entwickelt. Bei diesem ersten Programm werden wir uns die verwendeten Bausteine (*Nodes*) genau anschauen (siehe Abbildung 11.7).

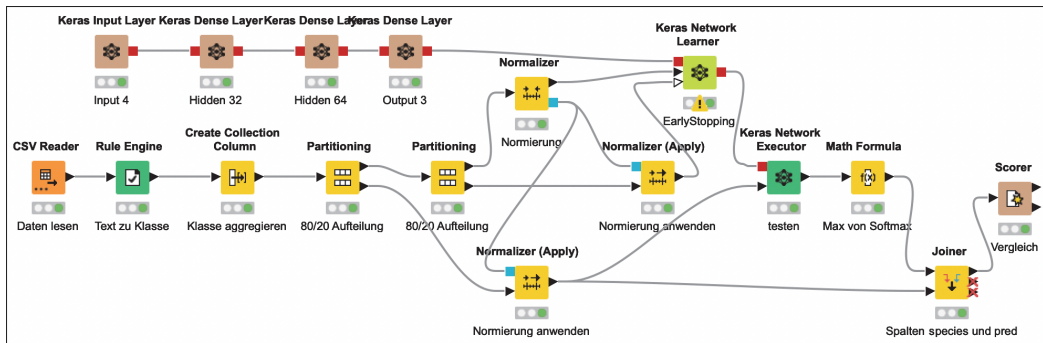


Abbildung 11.7 Das Programm »01-iris.knwf«

In der Mitte sehen Sie das strukturierte Programm (Workflow). Hier können Sie die Bausteine platzieren, konfigurieren und miteinander verbinden. Der linke Bereich ist in Registern aufgebaut.

- ▶ Info: Wenn Sie einen Baustein im Programm selektieren, werden hier die Informationen dazu angezeigt.
- ▶ Nodes: Das ist die Bausteinbibliothek. Hier können Sie nach Bausteinen suchen und diese in den Workflow ziehen, um Ihr Programm zu erstellen.
- ▶ Explorer: Hier haben Sie eine Übersicht über Ihre gespeicherten Programme.
- ▶ K-AI: Dies ist ein Chatbot, den Sie bei Bedarf um Rat fragen können.
- ▶ Monitor: Bei Warnungen und Fehlermeldungen können Sie hier die Details dazu einsehen.

Im unteren Bereich befindet sich der Node-Monitor. Hier können Sie sehen, welche Daten ein Baustein ausgibt bzw. dem nachfolgenden Baustein übergibt.

## Datenvorbereitung

Fangen wir mit der Datenquelle an. Sie können den Baustein CSV-READER aus dem Nodes-Bereich (Bausteinbibliothek) links in den Arbeitsbereich ziehen und mit Rechtsklick und Auswahl von CONFIGURE die gewünschten Einstellungen vornehmen. Alternativ können Sie auch die CSV-Datei in den Workflow-Editor per Drag-and-Drop ablegen. Dann wird automatisch der passende Baustein eingefügt, und es erscheint das Einstellungsfenster (siehe Abbildung 11.8).

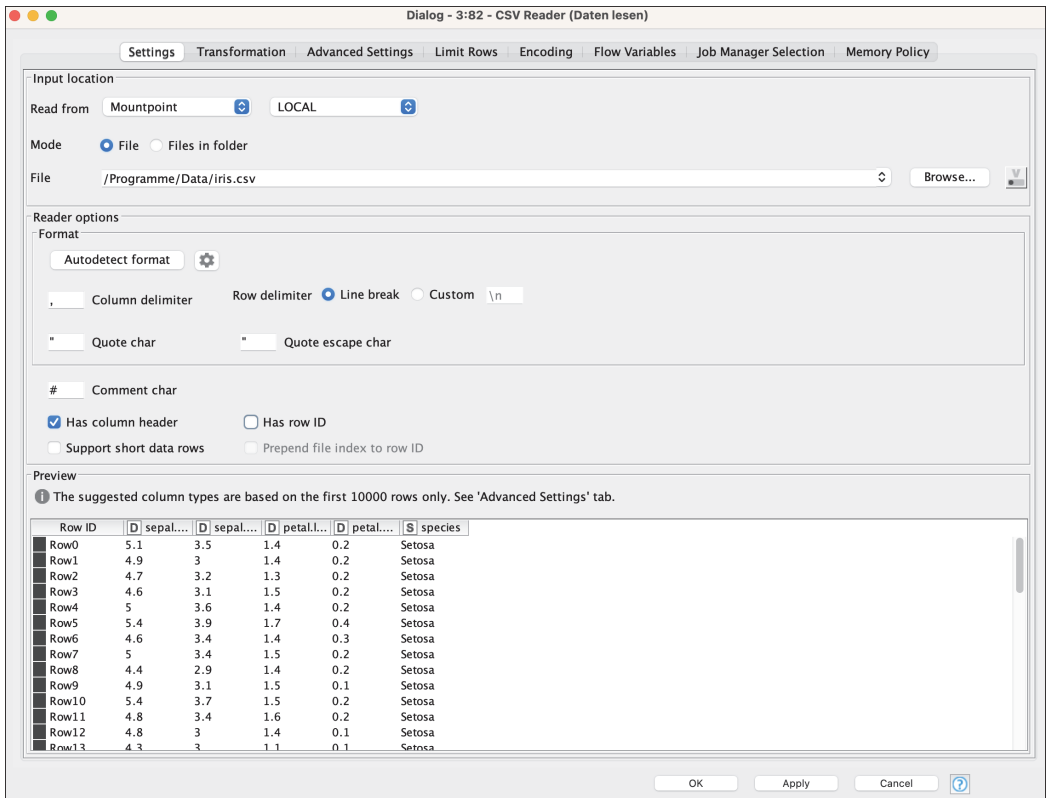


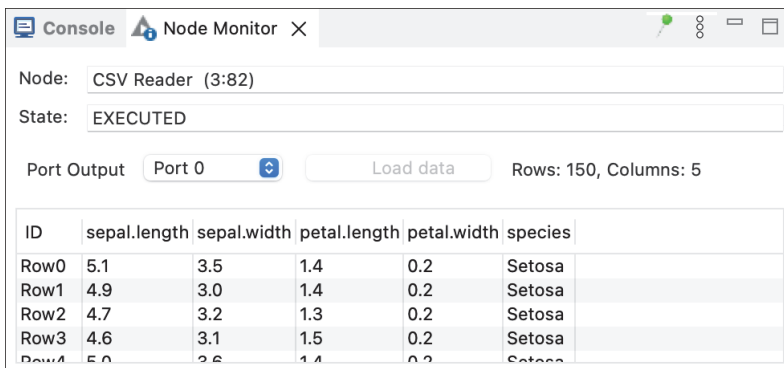
Abbildung 11.8 Das Einstellungsfenster für die CSV-Datei

Die wichtigsten Einstellungen, die vorzunehmen sind:

- ▶ der Pfad zur Datei (FILE)
- ▶ das Trennzeichen zwischen den Spalten (COLUMN DELIMITER)
- ▶ Angabe, ob die Tabelle Spaltenüberschriften hat (HAS COLUMN HEADER)

- Angabe, ob die Tabelle eine Spalte zur Indexierung hat (HAS ROW ID). Wenn die Tabelle selbst eine Spalte z. B. mit aufsteigender Nummerierung hat, um die Zeilen eindeutig zu kennzeichnen, kann diese Spalte verwendet werden. Sonst wird eine neue, temporäre Spalte mit eindeutigen Indizes erstellt.

Die Statusanzeige des Bausteins ist nach der Konfiguration gelb. Mit einem Rechtsklick und Auswahl von EXECUTE können Sie den Baustein ausführen, dann wechselt die Statusanzeige auf grün. Alternativ kann der Baustein selektiert und oben in der Menüleiste der EXECUTE-Button geklickt werden. Im Node-Monitor sehen Sie die Ausgabewerte des Bausteins (siehe Abbildung 11.9).



ID	sepal.length	sepal.width	petal.length	petal.width	species
Row0	5.1	3.5	1.4	0.2	Setosa
Row1	4.9	3.0	1.4	0.2	Setosa
Row2	4.7	3.2	1.3	0.2	Setosa
Row3	4.6	3.1	1.5	0.2	Setosa
Row4	5.0	2.8	1.4	0.2	Setosa

Abbildung 11.9 Ausgabe des Node-Monitors zu »CSV-Reader«

Mit den Daten am Ausgang des Bausteins CSV READER werden wir weiterarbeiten. Ab hier ist es egal, aus welcher Datenquelle die Daten stammen, sie liegen im Ergebnis immer in Tabellenform vor. Auf diese Art und Weise und mit den entsprechenden Bausteinen können Sie Daten z. B. von Excel-Dateien oder Datenbanken auslesen. An der restlichen Programmstruktur ändert sich nichts. Das ist eines der großen Vorteile von KNIME.

Damit das KNN mit den Eingangsdaten rechnen kann, müssen alle dafür benötigten, nichtnumerischen Spalten in Zahlen transformiert werden. Dies betrifft hier nur die Spalte *species*. In diesem Beispiel werden wir dafür den Baustein RULE ENGINE verwenden. Die Ausdrücke, die in diesem Baustein eingegeben werden können, haben eine ungewöhnliche Syntax, wie Sie in Abbildung 11.10 sehen.

Die ersten Zeilen enthalten einige auskommentierte Beispiele. Im linken Fenster sehen Sie die Variablen, die Sie verwenden können. Wichtig sind hier die Spaltenüberschriften, insbesondere *species*.

```

$species$ = "Iris-setosa" => 0
$species$ = "Iris-virginica" => 1
TRUE => 2

```

Listing 11.1 Transformation der Daten mit »RuleEngine«

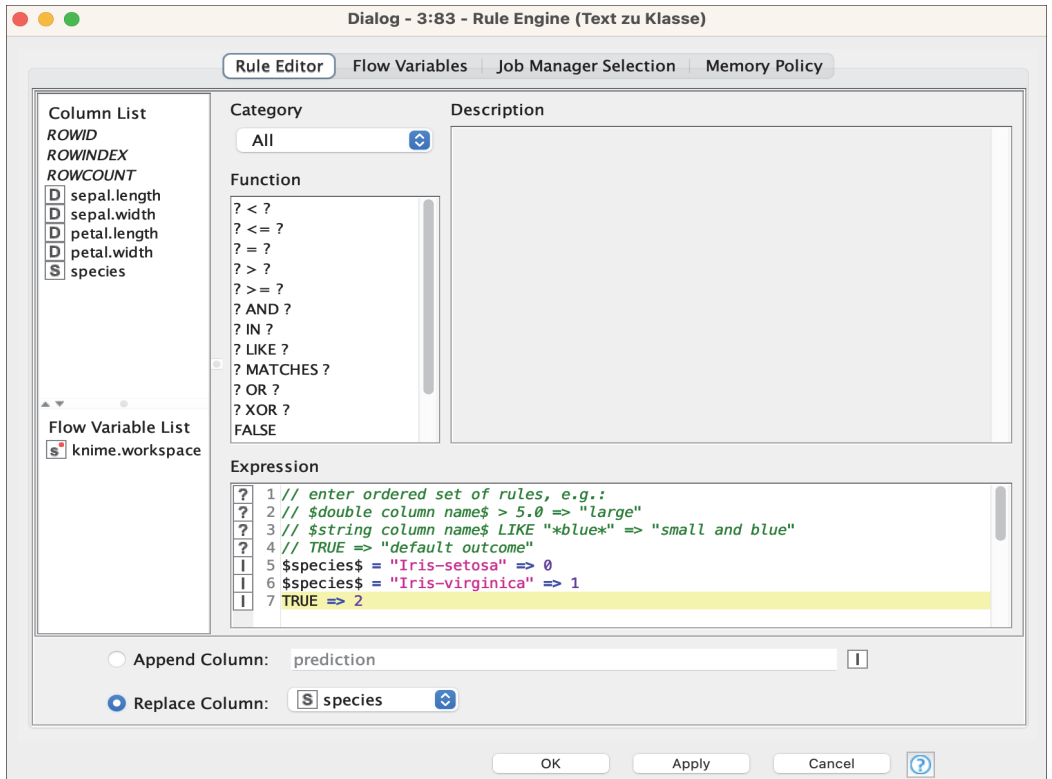


Abbildung 11.10 Das Einstellungsfenster von »Rule Engine«

Die erste Zeile bedeutet: Ist der Inhalt der Variable `$species$` »Iris-setosa«, wird die Zahl 0 verwendet. Die zweite Zeile ist entsprechend für die nächste Unterart zu interpretieren. Auch die dritte Zeile könnte genauso für »Iris-versicolor« formuliert werden. Aber hier sehen Sie eine andere Möglichkeit: Bei allen anderen Variableninhalten werden die Werte durch die Zahl 2 ersetzt. Sie können noch einstellen, ob der Inhalt der Spalte `species` überschrieben oder eine neue Spalte mit den Zahlenwerten erstellt werden soll. Nach dem Ausführen des Bausteins ist im NODE MONITOR zu sehen, dass die Transformation erfolgreich durchgeführt wurde. Beachten Sie in der Abbildung 11.11 die Spalte `species`.

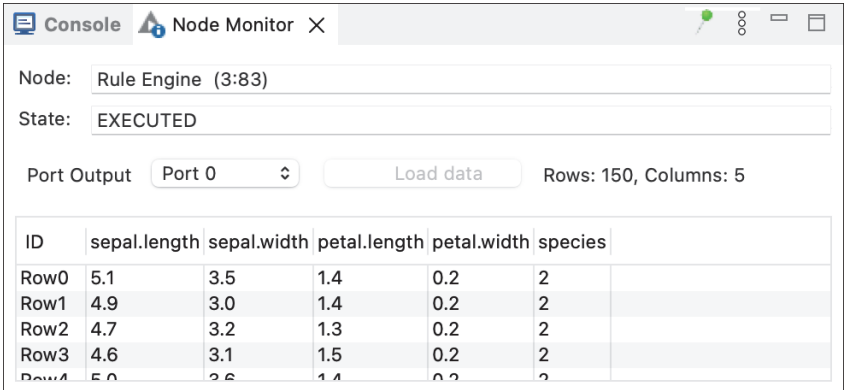


Abbildung 11.11 Ausgabe des Node-Monitors zu »Rule Engine«

Der Baustein für KNN erwartet als Ausgangsdaten (Zielspalte) Listen mit Zahlenwerten, wenn die Softmax-Funktion als Aktivierungsfunktion verwendet werden soll. Dies wird mit dem Baustein CREATE COLLECTION COLUMN realisiert, dessen Einstellungsfenster Abbildung 11.12 zeigt.

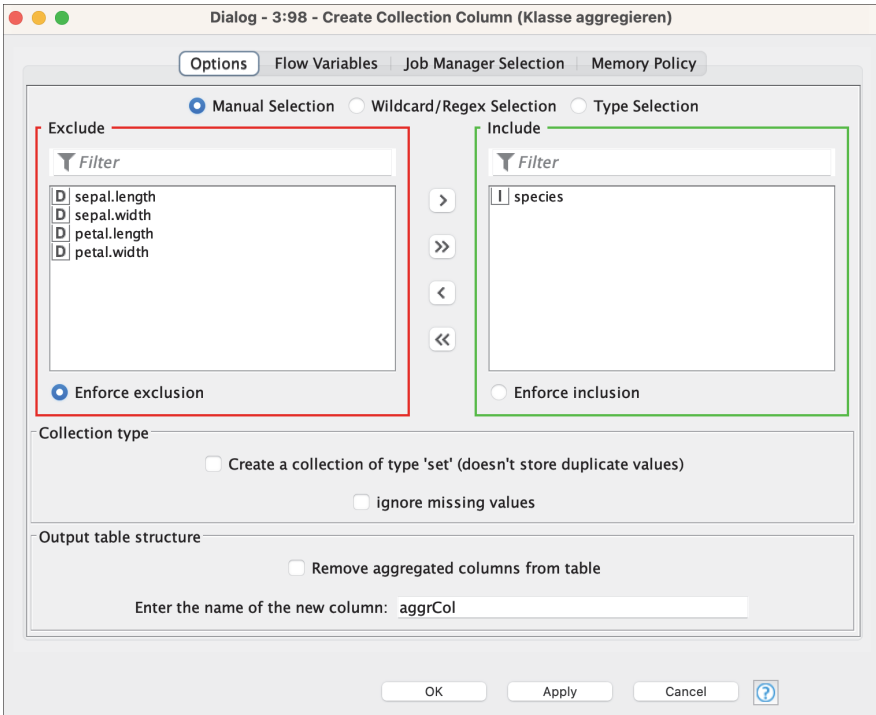
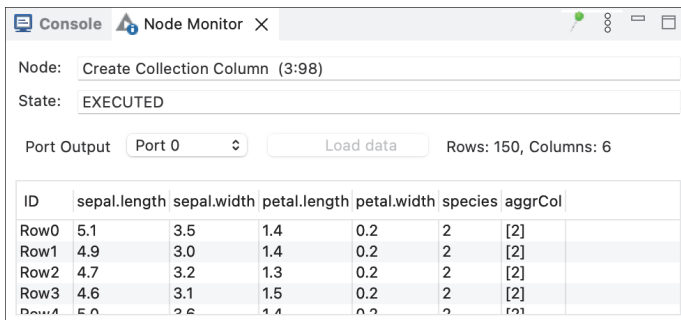


Abbildung 11.12 Das Einstellungsfenster von »Create Collection Column«

Mit diesem Baustein kann man mehrere Spalten in eine Liste zusammenführen. Wir müssen nur die einzelnen Elemente in eine Liste »packen«, damit wir später die Softmax-Funktion verwenden können. Dieser Schritt ist also nur notwendig, weil der Baustein für das KNN mit Softmax als Aktivierungsfunktion diesen Datentyp erwartet. Im Abschnitt INCLUDE werden die Spalten, welche zusammengeführt werden sollen, aufgelistet (hier nur *species*). Weiterhin ergänzen wir die Tabelle, wie in Abbildung 11.13 zu sehen, um die Spalte *aggrCol*, in welcher die Ergebnisse der Datentypumwandlung (Liste mit jeweils einem Element) gespeichert werden.



ID	sepal.length	sepal.width	petal.length	petal.width	species	aggrCol
Row0	5.1	3.5	1.4	0.2	2	[2]
Row1	4.9	3.0	1.4	0.2	2	[2]
Row2	4.7	3.2	1.3	0.2	2	[2]
Row3	4.6	3.1	1.5	0.2	2	[2]
Row4	5.0	2.6	1.1	0.2	2	[2]

Abbildung 11.13 Ausgabe des Node-Monitors zu »Create Collection Column«

Wenn Sie den Baustein ausführen, sehen Sie im Node-Monitor, dass die Zahlenwerte dieser neuen Spalte in eckigen Klammern stehen (siehe Abbildung 11.13). Dabei wird z. B. aus der Zahl 0 die Liste [0].

Die Daten sind nun eingelesen und vorbereitet. Jetzt können die Daten mit den Bausteinen PARTITIONING in Trainings-, Test- und Validierungsdaten aufgeteilt werden (siehe Abbildung 11.14). 80 % der gemischten Daten sollen am oberen Ausgabeknoten ausgegeben werden (Trainingsdaten), der Rest am unteren. Die Daten am unteren Ausgang werden nochmals in Test- und Validierungsdaten aufgeteilt (80 % bzw. 20 %). Wir stellen auch ein, dass die Daten reproduzierbar gemischt werden sollen (USE RANDOM SEED).

Bei Bausteinen mit mehreren Ausgängen können Sie im Node-Monitor einstellen, welche Daten angezeigt werden sollen (PORT OUTPUT).

Mit den Trainingsdaten wird das KNN trainiert, und somit werden die Gewichte aktualisiert. Mit den Testdaten wird die Korrektklassifizierungsrate nach jeder Epoche während der Trainingsphase ermittelt. Die Trainingsdaten dienen bei diesem Programm dazu zu erkennen, ob eine Stagnation auftritt und eine weitere Optimierung der Gewichte ausbleibt. In diesem Fall wird die Trainingsprozedur abgebrochen, um ein Overfitting zu verhindern. Am Ende wird mit den neuen, unbekanntenen Evaluierungsdaten die Korrektklassifizierungsrate ermittelt.



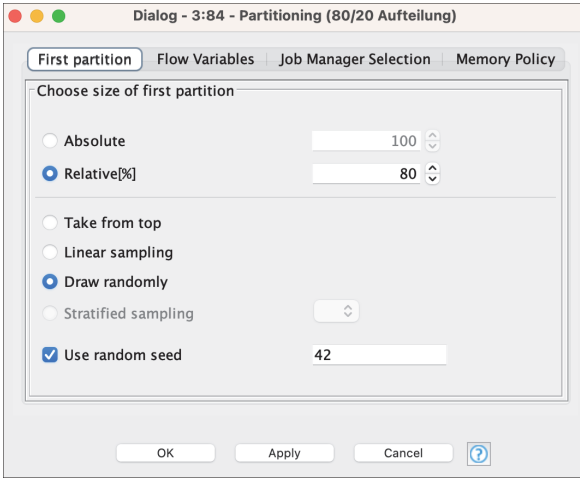


Abbildung 11.14 Das Einstellungsfenster von »Partitioning«

Wie Sie bereits aus vorherigen Kapiteln wissen, können KNN mit normierten Daten bessere Vorhersagen treffen, da sonst die verschiedenen Attribute durch sehr unterschiedliche Wertebereiche der Daten zu einer Verzerrung bei der Aktualisierung der Gewichte führen. Die Eingabedaten werden hier mittels Z-Score normiert (siehe Abbildung 11.15, entspricht dem Standardscaler von sklearn).

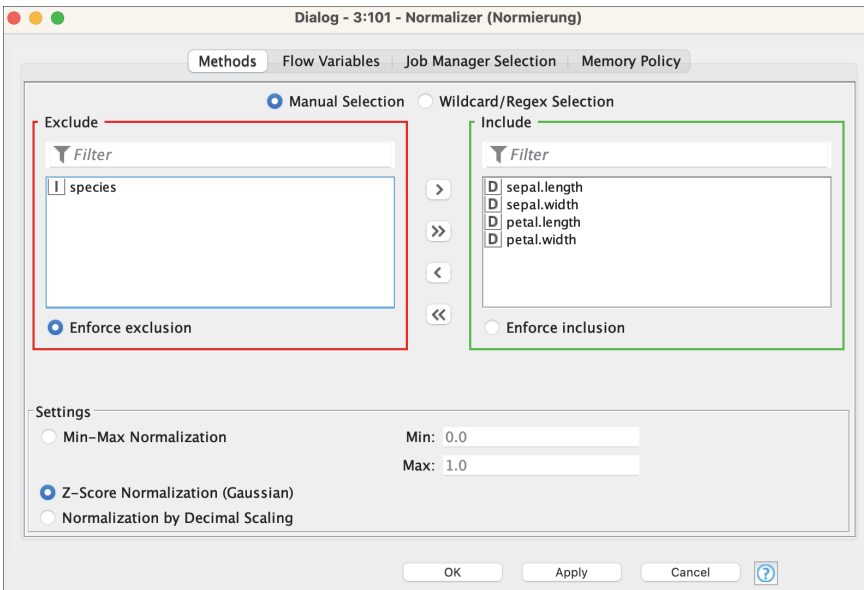
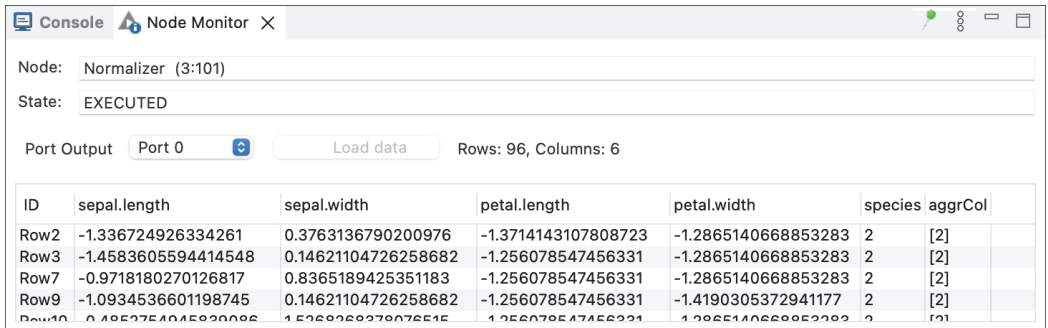


Abbildung 11.15 Das Einstellungsfenster von »Normalizer«

Der Baustein wird nur auf die Trainingsdaten angewandt. Im NODE MONITOR können Sie wieder die Ausgabedaten des Bausteins betrachten:



Node: Normalizer (3:101)  
State: EXECUTED  
Port Output: Port 0  
Load data  
Rows: 96, Columns: 6

ID	sepal.length	sepal.width	petal.length	petal.width	species	aggrCol
Row2	-1.336724926334261	0.3763136790200976	-1.3714143107808723	-1.2865140668853283	2	[2]
Row3	-1.4583605594414548	0.14621104726258682	-1.256078547456331	-1.2865140668853283	2	[2]
Row7	-0.9718180270126817	0.8365189425351183	-1.256078547456331	-1.2865140668853283	2	[2]
Row9	-1.0934536601198745	0.14621104726258682	-1.256078547456331	-1.4190305372941177	2	[2]
Row10	0.485274045820086	1.5269268278076515	1.256078547456331	1.2865140668853283	2	[2]

Abbildung 11.16 Ausgabe des Node-Monitors zu »Normalizer«

Sie sehen, dass die Test- und Validierungsdaten bei diesem Programm jeweils mit dem Baustein NORMALIZER (APPLY) normiert werden und dass dieser Baustein mit NORMALIZER verbunden ist. Der Mittelwert und die Standardabweichung werden nur mithilfe der Trainingsdaten ermittelt. Die Test- sowie Validierungsdaten sollen neue, unbekannte Daten simulieren, daher wird die Normierung bei diesen Daten nur angewendet. Dies ist die korrekte Vorgehensweise bei der Normierung von Datensätzen, wenn für praktische Anwendungen programmiert werden soll. Testdaten sollen ja neue, unbekannte Daten simulieren, daher ist es nicht ratsam, diese Werte für die Berechnung von Mittelwert und Standardabweichung zu verwenden.

### Aufbau des KNN

Jetzt wird es Zeit, das KNN aufzubauen. Die Verbindungen der bisher verwendeten Bausteine zeigen den Datenfluss an, symbolisiert durch Pfeile. Die Verbindungen der Layer eines KNN sind einfache Linien, welche lediglich eine Verbindung ohne Datenfluss symbolisieren. Am Baustein KERAS NETWORK LEARNER werden wir einige zentrale Einstellungen für das ganze Netz vornehmen (siehe Abbildung 11.17). Dieser Baustein muss vorher mit KERAS INPUT LAYER und drei in Reihe geschalteten KERAS DENSE LAYER-Bausteinen verbunden werden. Der erste Layer hat vier Eingangsknoten. Danach folgen zwei Hidden Layer mit jeweils 32 bzw. 64 Knoten, als Aktivierungsfunktion wählen wir ReLu. Der Output Layer hat drei Knoten mit Softmax als Aktivierungsfunktion.

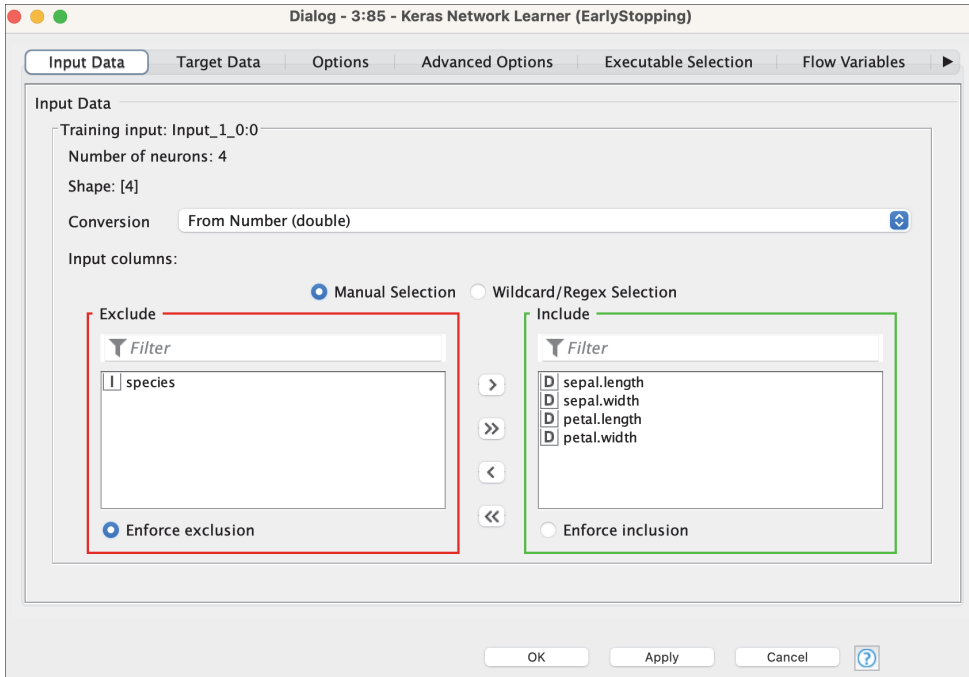


Abbildung 11.17 Das Einstellungsfenster von »Keras Network Learner«

Das Einstellungsfenster von KERAS NETWORK LEARNER ist in folgenden Registern organisiert; die ersten vier sind:

- ▶ **INPUT DATA:** Die vier Eingangsvariablen werden ausgewählt, unter **CONVERSION** wird **FROM NUMBER (DOUBLE)** eingestellt.
- ▶ **TARGET DATA:** Die Spalte *col* wird als Eingabespalte ausgewählt, unter **CONVERSION** wird **FROM COLLECTION OF NUMBER (INTEGER) TO ONE-HOT TENSOR** eingestellt. Zusätzlich wird **CATEGORICAL CROSS ENTROPY** als Verlustfunktion eingestellt.
- ▶ **OPTIONS:** Die Epochenzahl beträgt 40, **BATCH SIZE** ist 10. Wählen Sie mit **SHUFFLE TRAINING DATA BEFORE EACH EPOCH** noch aus, dass reproduzierbar gemischt werden soll. Sie müssen hier eine Zahl für **USE RANDOM SEED** eingeben, der Wert ist allerdings irrelevant.
- ▶ **ADVANCED OPTIONS:** Hier können Sie **Early Stopping** (über **VALIDATION LOSS**) einstellen. Wenn Sie sich dafür entscheiden, können Sie unter **OPTIONS** die Epochenzahl z. B. auf 100 setzen. In diesem Fall wird der Trainingsprozess abgebrochen, wenn keine Verbesserung mehr auftritt.

Der Baustein KERAS NETWORK LEARNER wird mit KERAS NETWORK EXECUTOR verbunden. Auch an diesem Baustein sind einige Einstellungen vorzunehmen, alle unter dem Register OPTIONS (siehe Abbildung 11.18):

- ▶ Die vier Eingangsvariablen werden ausgewählt, unter CONVERSION wird FROM NUMBER (DOUBLE) eingestellt.
- ▶ Mit ADD OUTPUT können Sie den Ausgang hinzufügen. Es muss ausgewählt werden, welcher der Layer als Ausgang definiert werden soll (OUTPUT\_1/SOFTMAX:0). Auch bei den Ausgangsdaten muss CONVERSION eingestellt werden, hier TO NUMBER (DOUBLE).

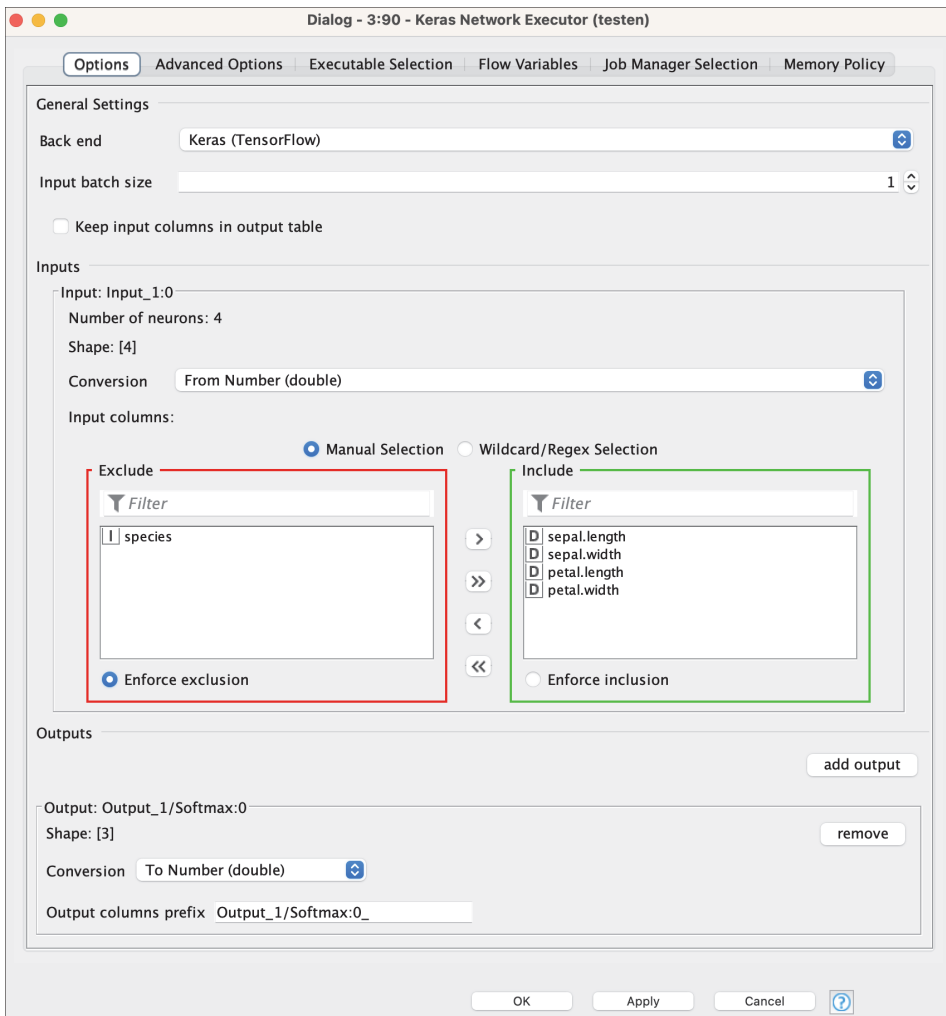


Abbildung 11.18 Das Einstellungsfenster von »Keras Network Executor«

## Training, Test und Validierung

Wie läuft das Verfahren für Training, Test und Validierung ab? Die Trainings- und Testdaten werden dem Baustein KERAS NETWORK LEARNER zugeführt. Mit den Trainingsdaten wird trainiert (also die Gewichte aktualisiert), mit den Testdaten nach jeder Iteration überprüft, wie gut die Vorhersage des KNN ist (die Gewichte werden dabei nicht aktualisiert). Wenn es zu keiner Verbesserung der Vorhersage bei den Testdaten kommt (Early Stopping), wird die Trainings- und Testprozedur abgebrochen. Anschließend werden dem Baustein KERAS NETWORK EXECUTOR die Validierungsdaten zugeführt. Im Feldeinsatz wäre hier Schluss, wir müssten den Vorhersagen des Bausteins vertrauen. Aber wir kennen ja zu den Validierungsdaten die Zielwerte, daher können wir die tatsächlichen Werte mit den Vorhersagen vergleichen und somit das KNN nochmals validieren.

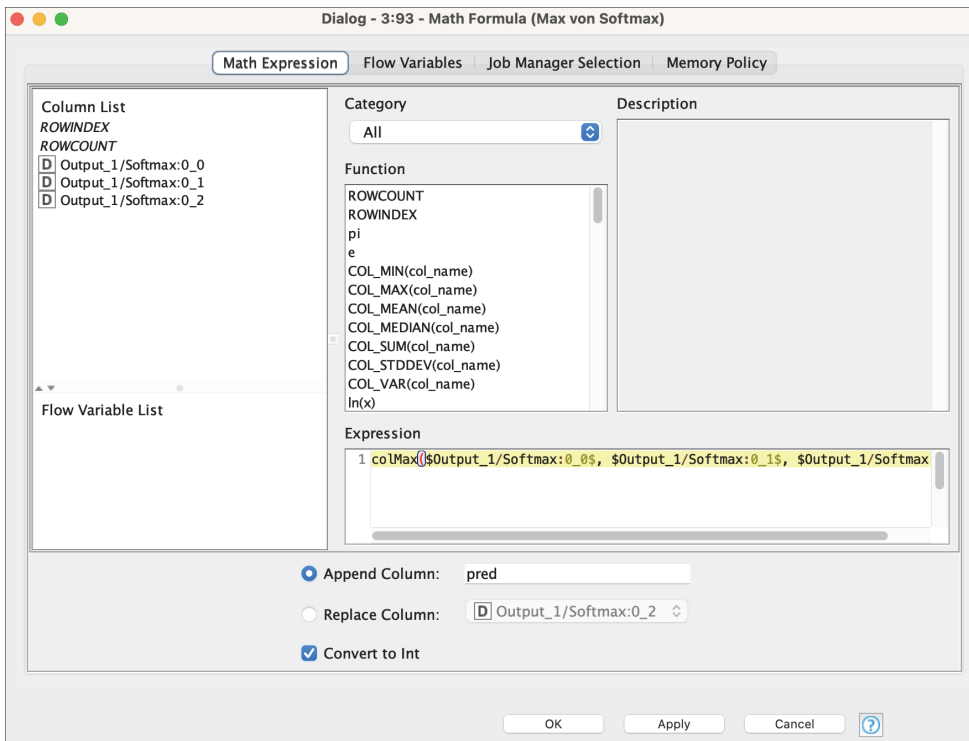


Abbildung 11.19 Das Einstellungsfenster von »Math Formula«

Der Baustein MATH FORMULA (siehe Abbildung 11.19) vergleicht die drei Werte der Softmax-Funktion des Bausteins KERAS NETWORK EXECUTOR und ermittelt den Index des

Maximalwertes. Dadurch sind als Ergebnisse die Zahlen 0 bis 2 möglich, welche den codierten Unterarten entsprechen.

Im Baustein ist die Funktion *colMax* definiert, welche den Maximalwert ermittelt:

```
colMax($Output_1/Softmax:0_0$, $Output_1/Softmax:0_1$, $Output_1/Softmax:0_2$)
```

**Listing 11.2** Ausdruck im Baustein »Math Formula«

Das Ergebnis wird in den Datentyp Integer umgewandelt und in der neuen Spalte *pred* gespeichert (siehe Abbildung 11.20).

ID	Output_1/Softmax:0_0	Output_1/Softmax:0_1	Output_1/Softmax:0_2	pred
Row1	1.100744975701673E-5	1.1369509593350813E-5	0.9999775886535645	2
Row6	6.1424061641446315E-6	7.389073743979679E-6	0.9999864101409912	2
Row30	1.284527843381511E-5	1.4026146345713641E-5	0.9999731779098511	2
Row32	1.231559053849196E-6	1.0824219316418748E-6	0.9999977350234985	2
Row40	1.002958778958572E-5	1.082501210052020E-5	0.0000701292742966	2

**Abbildung 11.20** Ausgabe des Node-Monitors zu »Math Formula«

Diese Spalte *pred* muss mit der Spalte *species* der Validierungsdaten verglichen werden. Mit dem Baustein JOINER wird eine neue Tabelle mit diesen beiden Spalten erstellt. Im Anschluss können wir die Werte dieser Tabelle verwenden, um Berechnungen durchzuführen, in diesem Fall, um die Korrektklassifizierungsrate zu berechnen (siehe Abbildung 11.21).

Jede Zeile einer Tabelle hat eine eindeutige RowID. Da beide Spalten ursprünglich aus einer einzigen Tabelle stammen, haben die Zeilen der übereinstimmenden Spalten dieselbe RowID. Diese Information aus beiden Tabellen wird als Grundlage für die Zusammenführung verwendet. Weiterhin wird die Schnittmenge (*Inner Join*) der beiden Spalten gebildet.

Nur RowIDs, die in beiden Tabellen enthalten sind, werden für die neue Tabelle verwendet. Hat eine Spalte mehr Einträge als die andere oder hat eine Spalte keinen »Gegenpart«, würde diese verworfen werden. Im Register COLUMN SELECTION des Bausteins JOINER wählen wir *pred* der einen und *species* der anderen Tabelle aus.

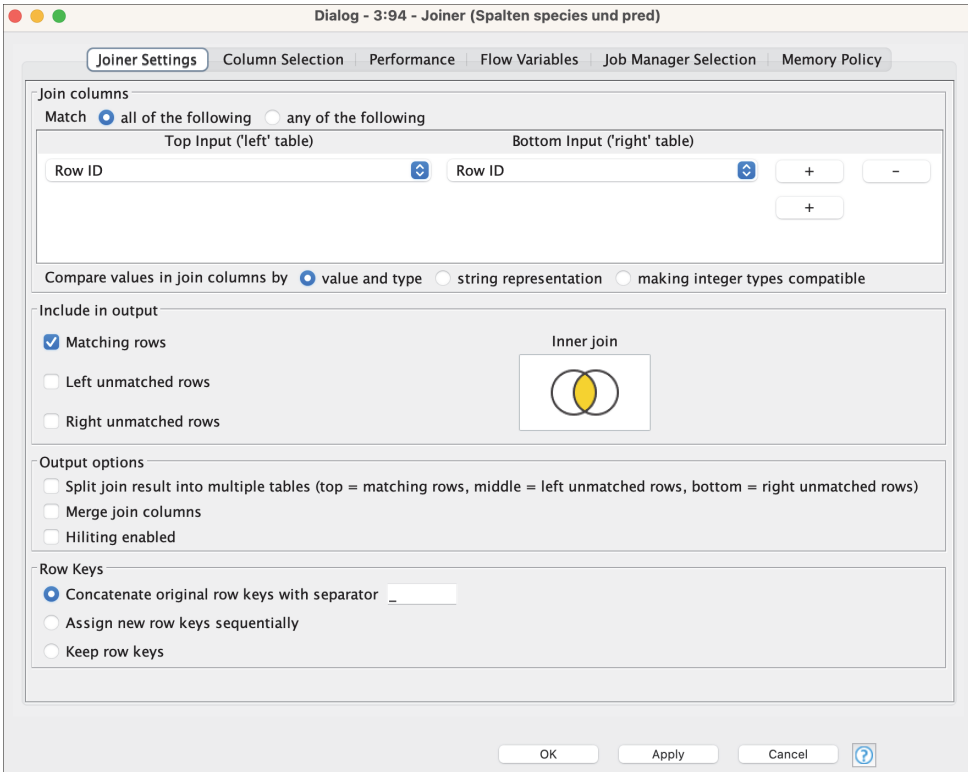


Abbildung 11.21 Das Einstellungsfenster von »Joiner«

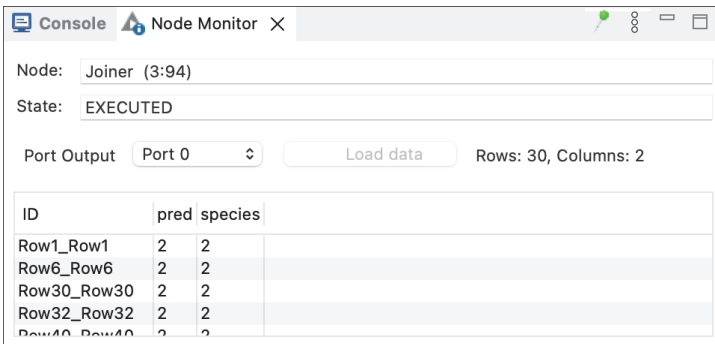


Abbildung 11.22 Ausgabe des Node-Monitors zu »Joiner«

Jetzt könnten wir diese neue Ergebnistabelle Zeile für Zeile durchgehen und uns anschauen, wo eventuell eine Fehlvorhersage generiert worden ist. Das ist aber sehr umständlich und nicht Sinn der Sache. Daher kommt als letzter Baustein SCORER zum Ein-

satz. Auch hier müssen *pred* und *species* ausgewählt werden. Selektieren Sie den Baustein, und lesen Sie die Beschreibung dafür im linken Fenster (NODE-BESCHREIBUNG) durch, um zu erfahren, welche Variable an welchen Eingang angeschlossen werden soll. Nach dem Ausführen erfolgt ein Rechtsklick und die Auswahl von VIEW: CONFUSION MATRIX (siehe Abbildung 11.23).

Confusion Matrix - 3:92 - Scorer (Vergleich)				
File	Hilite			
species \ pred	0	2	1	
0	7	0	0	
2	0	8	2	
1	0	2	11	

Correct classified: 26	Wrong classified: 4
Accuracy: 86,667%	Error: 13,333%

Abbildung 11.23 »Confusion Matrix«

Mit diesem ersten Entwurf wurde eine Korrektklassifizierungsrate von 87 % erreicht.

## 11.2.2 Klassifizierung mit Python Node

Sie können die Aufgabenstellung vom vorherigen Abschnitt auch mit Python-Bausteinen lösen (siehe Abbildung 11.24).

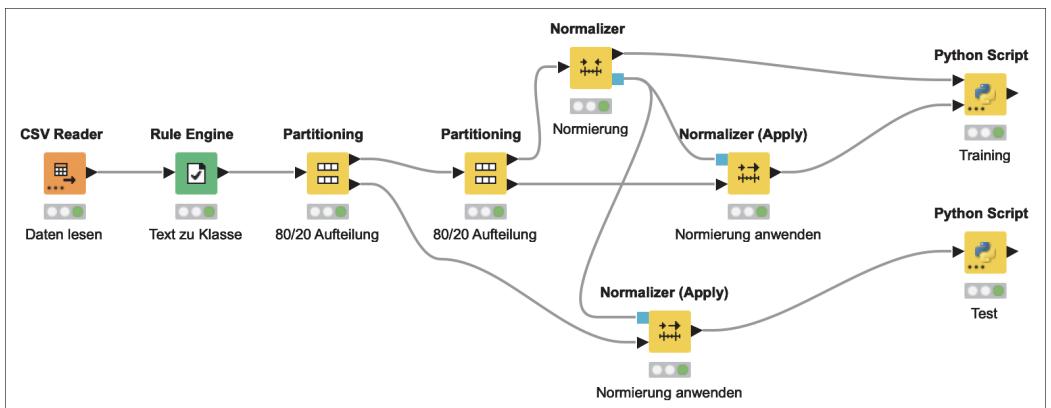


Abbildung 11.24 Das Programm »01-iris-py.knwf«

Der Aufbau des Programmes bleibt ähnlich, es fehlen lediglich die Keras-Bausteine. Dafür kommen zwei PYTHON SCRIPT-Bausteine für Training und Test zum Einsatz.



```

import knime.scripting.io as knio
import tensorflow as tf
import pandas as pd

# Trainingsdaten
train_col = knio.input_tables[0][4].to_pandas().astype('category')
train_data = knio.input_tables[0][0:4].to_pandas()

# Testdaten
test_col = knio.input_tables[1][4].to_pandas().astype('category')
test_data = knio.input_tables[1][0:4].to_pandas()

# Aufbau Modell
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(train_data.shape[1],)))
model.add(tf.keras.layers.Dense(32, activation=tf.nn.sigmoid))
model.add(tf.keras.layers.Dense(64, activation=tf.nn.sigmoid))
model.add(tf.keras.layers.Dense(3, activation=tf.nn.softmax))

# Konfiguration, Training und Test
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
cb_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(train_data, train_col, epochs=100,
                    validation_data=(test_data, test_col), callbacks=[cb_early])

# Modell speichern
model.save("modell1.keras")

# Ausgabe des Ergebnisses
out = [history.history['val_accuracy'][-1]]
df = pd.DataFrame(out, columns=['acc'])
knio.output_tables[0] = knio.Table.from_pandas(df)

```

**Listing 11.3** Python-Script für das Training des Modells

Der Quellcode sollte nach den vielen Modellen in Kapitel 3, »Das künstliche neuronale Netz«, leicht zu interpretieren sein, lediglich das Einlesen und Ausgeben der Daten kann ungewöhnlich erscheinen. Die Datentypen dieser Daten müssen jeweils umgewandelt werden, damit der Datenfluss zu anderen Bausteinen gewährleistet ist. Halten Sie sich

nicht zu lange mit dieser Konvertierung auf, Sie werden noch weitere Beispiele sehen, die Sie für Ihre zukünftigen Programme kopieren und verwenden können. Bei Interesse finden Sie auf <http://r-wrk.de/9763-input-output> eine ausführliche Dokumentation zur Verarbeitung bzw. Vorbereitung der Ein- und Ausgabedaten. Noch ein Hinweis: Mit `meine_liste[-1]` lesen Sie das letzte Element einer Liste aus. In diesem Programm wird nur der letzte Wert der Evaluierung ausgegeben.

```

Import knime.scripting.io as knio
import tensorflow as tf
import pandas as pd

# Validierungsdaten laden
val_col = knio.input_tables[0][4].to_pandas().astype('category')
val_data = knio.input_tables[0][0:4].to_pandas()

# Modell laden
loaded_model = tf.keras.saving.load_model("modell1.keras")

# Evaluation
val_loss, val_acc = loaded_model.evaluate(val_data, val_col)

# Ausgabe des Ergebnisses
df = pd.DataFrame([val_acc], columns=['acc'])
knio.output_tables[0] = knio.Table.from_pandas(df)

```

#### Listing 11.4 Python-Script für Test

Das gespeicherte Modell wird geladen und auf die Testdaten angewendet. Der Wert der Korrektklassifizierung wird ausgegeben, sodass Sie ihn im Node-Monitor sehen können.

### 11.2.3 Regression

Im nächsten Beispiel (Abbildung 11.25) soll wieder die Länge des Kelchblattes (Sepalum) ermittelt werden. Bei der Vorhersage einer stetigen Zahl mittels KNN führt ebenfalls die Normierung der Daten wie bei der Klassifizierung oft zu besseren Ergebnissen. Wir werden von dem KNN den mittleren absoluten Fehler berechnen lassen. Das gibt uns Aufschluss darüber, wie weit die Vorhersage vom tatsächlichen Wert entfernt ist.

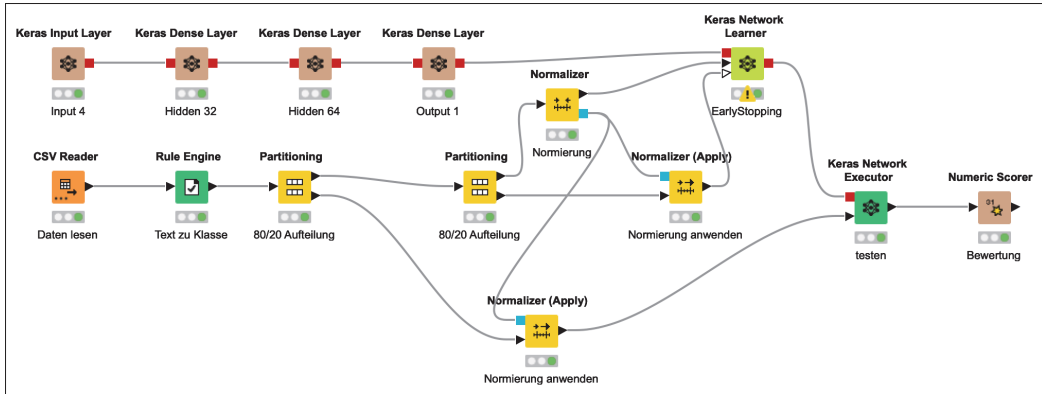


Abbildung 11.25 Das Programm »02-iris.knwf«

Um nicht den ganzen Workflow komplett neu zu erstellen, können Sie den vorherigen kopieren und umbenennen. Vieles bleibt gleich oder ähnlich. Schauen wir uns im Folgenden die Unterschiede an.

Die Zielspalte brauchen wir nicht zu aggregieren, da wir keine Softmax-Funktion benötigen. Unser Ziel ist nicht die Klassifizierung, sondern die Ermittlung des mittleren absoluten Fehlers.

Der Baustein NORMALIZER normiert die Trainingsdaten, die Spalte für die Länge des Kelchblattes (Zielspalte) ist von der Normierung ausgeschlossen. Sonst wären die ermittelten MAE-Werte (Mean Absolute Error) des KNN auch normiert und müssten zurückskaliert werden, um die Ergebnisse interpretieren zu können (siehe Abbildung 11.26).

Console Node Monitor

Node: Normalizer (4:101)  
State: EXECUTED

Port Output: Port 0 | Load data | Rows: 96, Columns: 5

ID	sepal.length	sepal.width	petal.length	petal.width	species
Row2	4.7	0.3763136790200976	-1.3714143107808723	-1.2865140668853283	-1.2288317380538267
Row3	4.6	0.14621104726258682	-1.256078547456331	-1.2865140668853283	-1.2288317380538267
Row7	5.0	0.8365189425351183	-1.256078547456331	-1.2865140668853283	-1.2288317380538267
Row9	4.9	0.14621104726258682	-1.256078547456331	-1.4190305372941177	-1.2288317380538267
Row10	5.1	1.528228278776515	1.256078547456331	1.2865140668853283	1.2288317380538267

Abbildung 11.26 Ausgabe des Node-Monitors zu »Normalizer«

Der Output Layer des KNN hat jetzt auch nur einen Knoten mit einer linearen Aktivierungsfunktion, Ein- und Ausgabewerte dieser Aktivierungsfunktion sind identisch.

Man könnte auch sagen, es wird gar keine Aktivierungsfunktion angewendet. Bei dem Baustein können wir die Aktivierungsfunktion nicht deaktivieren, aber mit der Auswahl dieser Funktion erreichen wir genau das.

Beim Baustein KERAS NETWORK LEARNER sind wieder einige Einstellungen bei folgenden Registern vorzunehmen (siehe Abbildung 11.27):

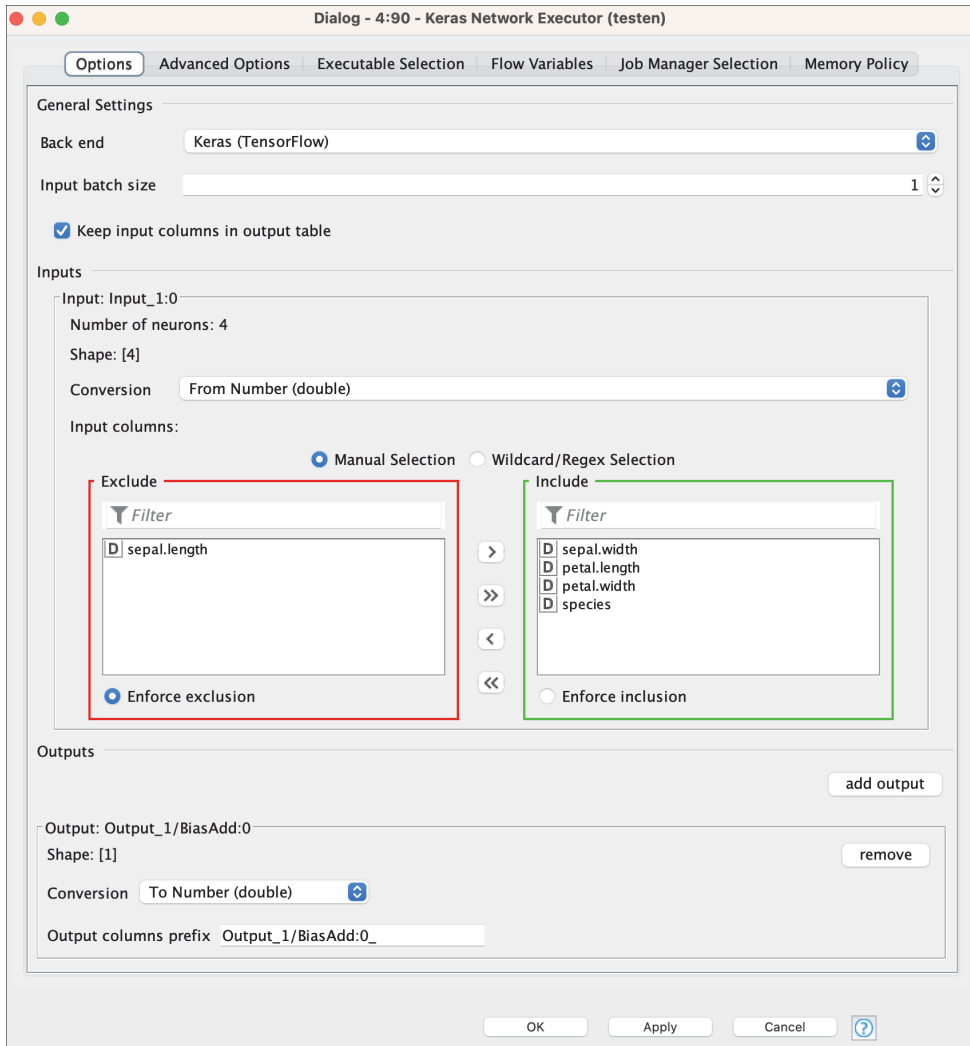


Abbildung 11.27 Das Einstellungsfenster von »Keras Network Executor«

- ▶ **INPUT DATA:** Bei **CONVERSION** muss **FROM NUMBER (DOUBLE)** eingestellt werden. Die Länge des Kelchblattes wird vom Datensatz ausgeschlossen.
- ▶ **TARGET DATA:** Bei **CONVERSION** muss wieder **FROM NUMBER (DOUBLE)** eingestellt werden. Vom Datensatz wird nur die Länge des Kelchblattes verwendet. Als Verlustfunktion wird **MEAN ABSOLUTE ERROR** eingestellt.
- ▶ **OPTIONS:** Die Epochenzahl wird auf 100 und die Batch Size auf 10 gesetzt. Die Daten sollen reproduzierbar gemischt werden. Die Wahrscheinlichkeit für Overfitting ist bei dieser Epochenzahl hoch. Aber im nächsten Punkt wird das Problem gelöst.
- ▶ **ADVANCED OPTIONS:** Early Stopping wird mithilfe von **TRAINING LOSS (TOTAL)** ausgewählt. Somit wird der Trainingsprozess abgebrochen, wenn keine Verbesserung mehr auftritt.

Bei dem Baustein **KERAS NETWORK EXECUTOR** müssen ebenfalls die entsprechen Eingangsdaten selektiert werden. Im Abschnitt **OUTPUT** wird der Ausgang des letzten Output Layers ausgewählt. Selektieren Sie auch **KEEP INPUT COLUMNS IN OUTPUT TABLE**. Dann stehen am Ausgang des Bausteins die Ist- und die Sollwerte weiterhin zur Verfügung. Somit brauchen Sie keinen Joiner, um diese Spalten für die Berechnung im Anschluss aus anderen Tabellen zusammenzuführen.

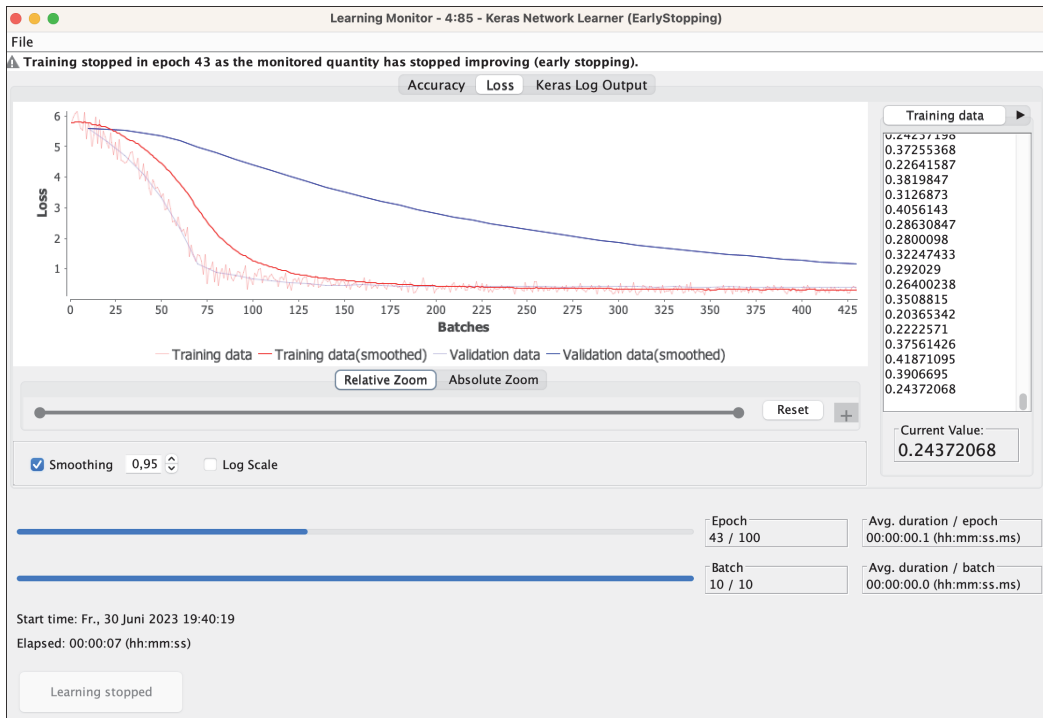


Abbildung 11.28 Learning Monitor

Der NUMERIC SCORER berechnet unter anderem den MAE, dafür müssen Sie lediglich die zwei Spalten für Ist- und Sollwerte selektieren. Nach dem Ablauf des Programmes können Sie wieder mit Rechtsklick und Auswahl von VIEW: STATISTICS unter anderem den mittleren absoluten Fehler einsehen (0,45).

Während des Ablaufs oder auch danach können Sie mit Rechtsklick auf KERAS NETWORK LEARNER und Auswahl von LEARNING MONITOR den Trainingsverlauf grafisch analysieren (siehe Abbildung 11.28).

Sie können den Kurvenverlauf glätten und bestimmte Bereiche mit der Maus auswählen, um diese zu vergrößern. Wenn der Trainingsprozess zu lange dauert, können Sie das Training mit dem Button STOP LEARNING beenden. Das führt zu keinem Fehler, der Rest des Programmes kann weiter ausgeführt werden.

### 11.2.4 Regression mit Python Node

Wenn Sie das Programm mit Python-Bausteinen realisieren wollen, müssen Sie lediglich das KNN selbst programmieren (siehe Abbildung 11.29).

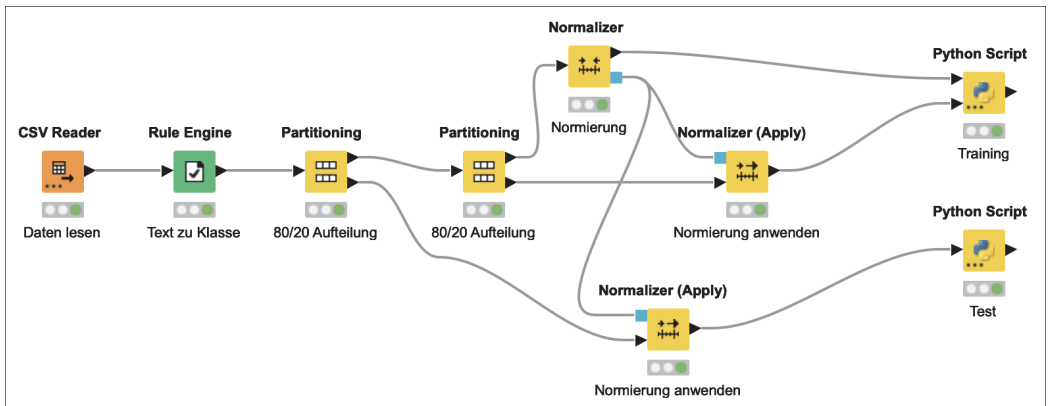


Abbildung 11.29 Das Programm »02-iris-py.knwf«

Betrachten wir zuerst den Code für das Training:

```
import knime.scripting.io as knio
import tensorflow as tf
import pandas as pd

# Trainingsdaten
train_col = knio.input_tables[0][0].to_pandas().astype('category')
train_data = knio.input_tables[0][1:5].to_pandas()
```

```

# Testdaten
test_col = knio.input_tables[1][0].to_pandas().astype('category')
test_data = knio.input_tables[1][1:5].to_pandas()

# Aufbau Modell
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(train_data.shape[1],)))
model.add(tf.keras.layers.Dense(32, activation=tf.nn.sigmoid))
model.add(tf.keras.layers.Dense(64, activation=tf.nn.sigmoid))
model.add(tf.keras.layers.Dense(1))

# Konfiguration, Training und Test
model.compile(optimizer='adam', loss='mae', metrics=['mae'])
cb_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(train_data, train_col, epochs=100,
                    validation_data=(test_data, test_col),callbacks=[cb_early])

# Modell speichern
model.save("model2.keras")

# Ausgabe des Ergebnisses
out = [history.history['val_mae'][-1]]
df = pd.DataFrame(out, columns=['mae'])
knio.output_tables[0] = knio.Table.from_pandas(df)

```

#### Listing 11.5 Python-Script für Training

Auch dieses Modell wird trainiert und gespeichert. Fühlen Sie sich bitte frei, die Hyperparameter zu verändern.

```

import knime.scripting.io as knio
import tensorflow as tf
import pandas as pd

# Validierungsdaten laden
val_col = knio.input_tables[0][0].to_pandas().astype('category')
val_data = knio.input_tables[0][1:5].to_pandas()

# Modell laden
loaded_model = tf.keras.saving.load_model("model2.keras")

```

```

# Evaluation
val_loss, val_mae = loaded_model.evaluate(val_data, val_col)

# Ausgabe des Ergebnisses
df = pd.DataFrame([val_mae], columns=['mae'])
knio.output_tables[0] = knio.Table.from_pandas(df)

```

Listing 11.6 Python-Script für Test

Die Testdaten werden ausgewertet und ausgegeben. Im Node-Monitor kann das Ergebnis betrachtet werden.

## 11.3 XGBoost

In diesem Abschnitt werden wir Regression und Klassifizierung mit XGBoost realisieren. Sie erinnern sich noch: KI-Modelle, die auf XGBoost basieren, sind übersichtlicher, und die Daten müssen nicht aufwändig vorbereitet werden. Auch erreicht man mit diesen Modellen auf einfache Art und Weise sehr gute Ergebnisse.

### 11.3.1 Klassifizierung

Mit dem Programm *03-iris* werden wieder Schwertlilien klassifiziert. Vergleichen Sie den Aufbau mit dem Programm *01-iris*. Sie sehen, wie einfach und übersichtlich das Programm mit XGBoost ist (siehe Abbildung 11.30).

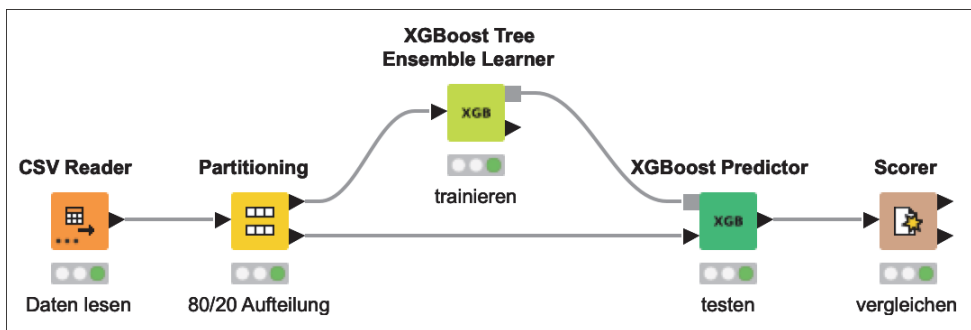


Abbildung 11.30 Das Programm »03-iris.knwf«

Bei dem Baustein XGBOOST TREE ENSEMBLE LEARNER müssen zentrale Einstellungen vorgenommen werden (siehe Abbildung 11.31).



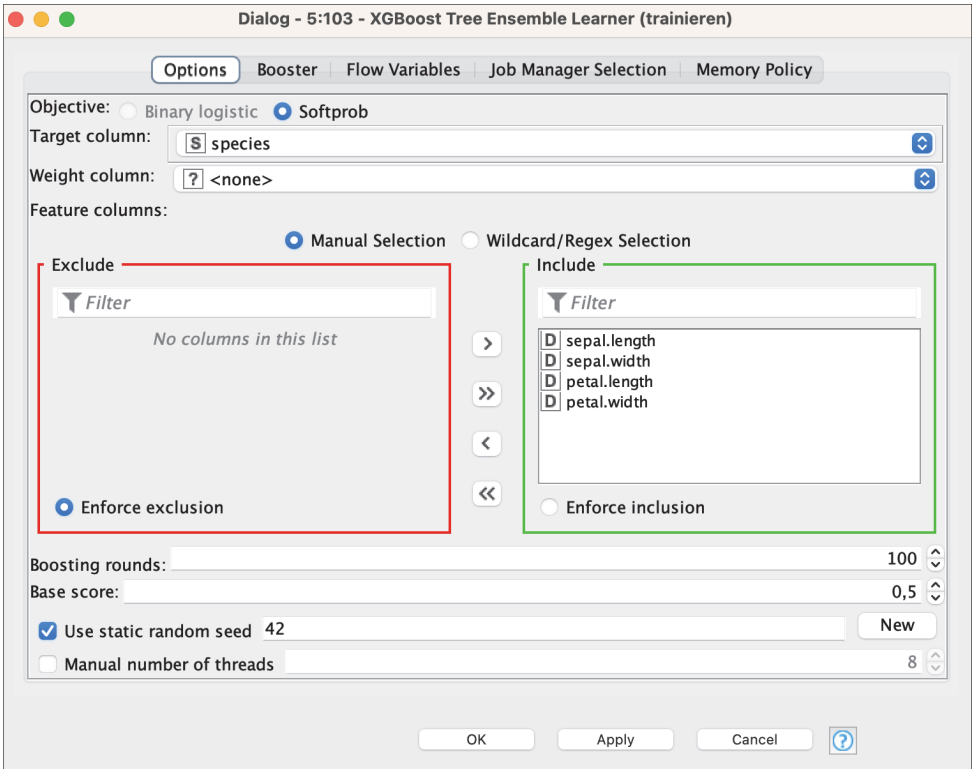


Abbildung 11.31 Das Einstellungsfenster von »XGBoost Tree Ensemble Learner«

Die wichtigsten Einstellungen sind die Auswahl von Eingabe- und Zielspalten. Mit USE STATIC RANDOM SEED werden die Bäume reproduzierbar aufgebaut, sodass ein wiederholter Aufruf auch die gleichen Ergebnisse liefert.

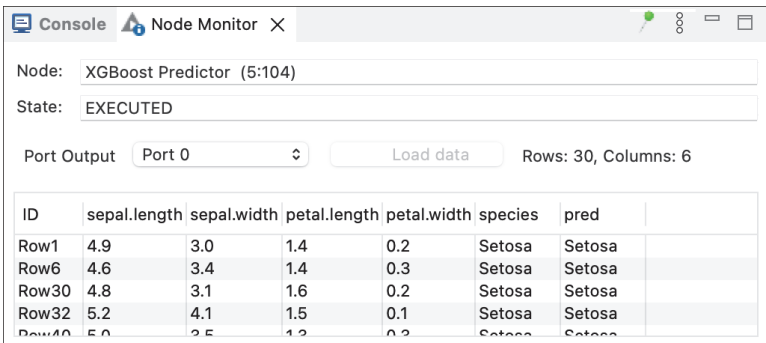


Abbildung 11.32 Ausgabe des Node-Monitors zu »XGBoost Predictor«

Bei dem Baustein XGBOOST PREDICTOR stellen wir ein, dass die Vorhersagen in der neuen Spalte *pred* gespeichert werden. Der Ausgang dieses Bausteins enthält alle Daten der ursprünglichen Tabelle, ergänzt um die Vorhersage (siehe Abbildung 11.32).

Mit dem letzten Baustein SCORER werden die Spalten *pred* und *species* miteinander verglichen. Dieses Programm erreicht eine Korrektklassifizierungsrate von über 93 %.

### 11.3.2 Deployment

In diesem Abschnitt werden wir nicht nur eine KI trainieren und testen (Klassifizierung von Schwertlilien), sondern auch das Modell (mit MODEL WRITER) und die Validierungsdaten (mit TABLE WRITER) speichern. Ein anderes Programm, das Sie in Abbildung 11.33 sehen, lädt diese Daten und erstellt Vorhersagen.

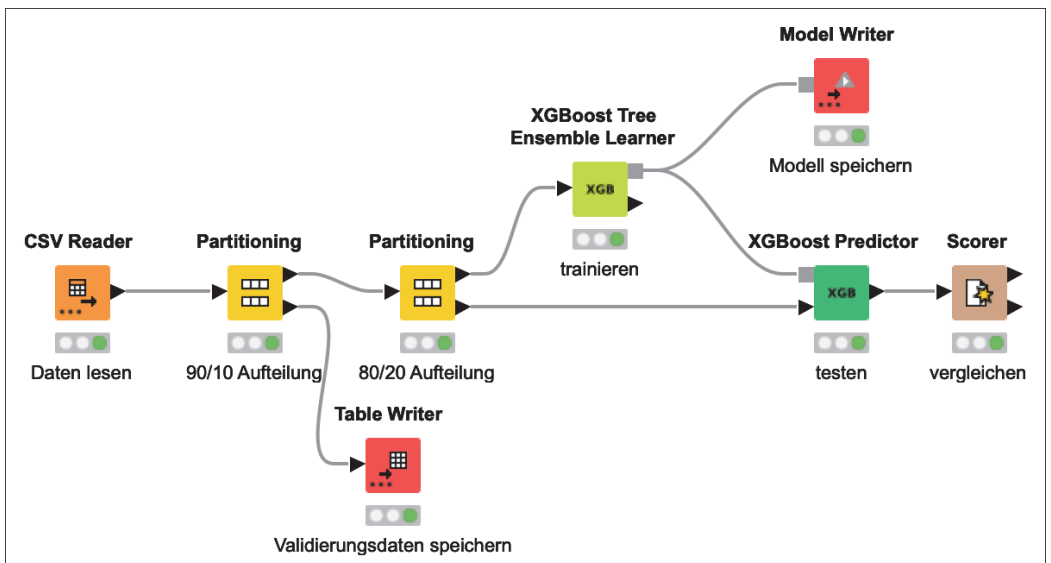


Abbildung 11.33 Das Programm »04a-iris.knwf«

10 % der Gesamtdaten werden als Validierungsdaten im Binärformat gespeichert, ebenso das Modell. Bei diesen Speicherbausteinen müssen Sie den Pfad sowie Speicheroptionen einstellen. Achten Sie darauf, dass die Einstellung `WRITE OPTIONS: OVERWRITE` gesetzt ist. Sonst generiert ein erneuter Ablauf des Programmes eine Fehlermeldung mit dem Hinweis, dass die Datei bereits existiert, wie in Abbildung 11.34 zu sehen.

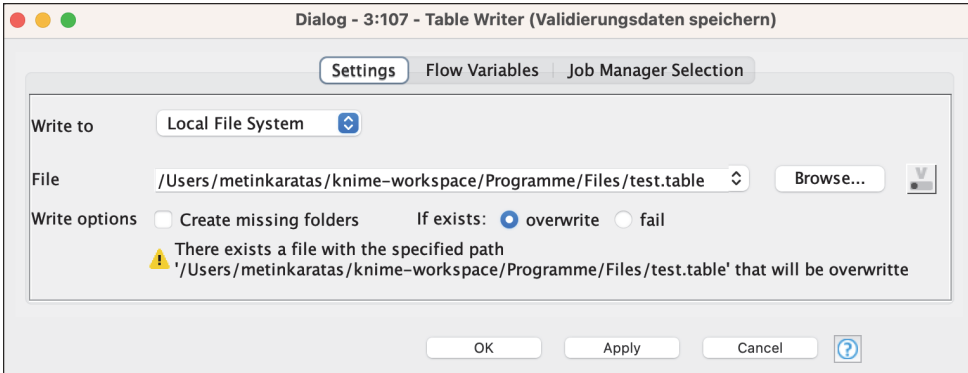


Abbildung 11.34 Das Einstellungsfenster von »Table Writer«

Was ist der große Unterschied zwischen Validierungsdaten und realen Daten beim produktiven Einsatz? Die Validierungsdaten enthalten die Zielspalte (also die Information, was vorhergesagt werden soll). Die KI ist eigentlich dafür da, diese Information zu ermitteln. Aber mit den Validierungsdaten können wir das Modell bewerten, indem wir die Vorhersage mit den tatsächlichen Werten vergleichen (siehe Abbildung 11.35).

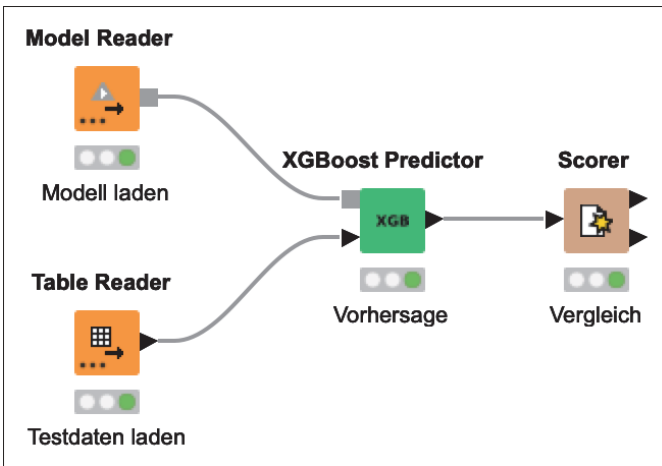


Abbildung 11.35 Das Programm »04b-iris.knwf«

Auch in diesem Programm legt der Baustein eine Spalte *pred* mit den Vorhersagen an, damit SCORER die Korrektklassifizierungsrate ermitteln kann.

Bei einer Vorhersage für reale Daten, in denen die Information der Unterart nicht enthalten ist, wäre der Baustein SCORER hinfällig. Sie müssten den Vorhersagen der KI vertrauen.

# Auf einen Blick

1	Einleitung .....	17
2	Installation von Anaconda .....	29
3	Das künstliche neuronale Netz .....	39
4	Entscheidungsbäume .....	95
5	Faltungsschichten, Bilder und mehr .....	123
6	Transfer Learning .....	151
7	Anomalieerkennung .....	163
8	Textklassifizierung .....	179
9	Clusteranalyse .....	197
10	Visuelle Programmierung mit Orange .....	211
11	Visuelle Programmierung mit KNIME .....	239
12	Reinforcement Learning .....	325
13	Genetische Algorithmen .....	343
14	ChatGPT und Gemini .....	359
15	DALL-E und Imagen .....	393
16	Ausblick .....	403

# Inhalt

Materialien zum Buch .....	15
<b>1 Einleitung</b> .....	<b>17</b>
<b>1.1 Was bietet dieses Buch?</b> .....	<b>18</b>
<b>1.2 Was ist eine »künstliche Intelligenz«?</b> .....	<b>19</b>
<b>1.3 Geschichte der KI – ein kurzer Überblick</b> .....	<b>21</b>
<b>1.4 Verwendete Werkzeuge</b> .....	<b>23</b>
1.4.1 Python .....	23
1.4.2 Jupyter Notebook .....	25
1.4.3 Orange .....	25
1.4.4 KNIME .....	26
1.4.5 ChatGPT und Gemini .....	27
1.4.6 DALL-E und Imagen .....	28
<b>2 Installation von Anaconda</b> .....	<b>29</b>
<b>2.1 Windows und macOS</b> .....	<b>29</b>
<b>2.2 Linux</b> .....	<b>30</b>
<b>2.3 Konfiguration und Test</b> .....	<b>31</b>
<b>3 Das künstliche neuronale Netz</b> .....	<b>39</b>
<b>3.1 Klassifizierung</b> .....	<b>40</b>
<b>3.2 Das Kochrezept</b> .....	<b>42</b>
3.2.1 Datenvorbereitung .....	42
3.2.2 Die KI aufbauen .....	44
3.2.3 Die KI trainieren .....	44

3.2.4	KI testen .....	45
3.2.5	KI einsetzen .....	45
<b>3.3</b>	<b>Aufbau künstlicher neuronaler Netze .....</b>	<b>46</b>
<b>3.4</b>	<b>Aufbau eines künstlichen Neurons .....</b>	<b>48</b>
<b>3.5</b>	<b>Feed Forward .....</b>	<b>50</b>
<b>3.6</b>	<b>Backpropagation .....</b>	<b>52</b>
<b>3.7</b>	<b>Aktualisierung der Gewichte .....</b>	<b>55</b>
<b>3.8</b>	<b>KNN für Klassifizierung .....</b>	<b>58</b>
<b>3.9</b>	<b>Hyperparameter und Overfitting .....</b>	<b>66</b>
<b>3.10</b>	<b>Umgang mit nichtnumerischen Daten .....</b>	<b>69</b>
<b>3.11</b>	<b>Umgang mit Datenlücken .....</b>	<b>71</b>
3.11.1	Leere Zellen mit Daten füllen .....	72
3.11.2	Zeilen mit leeren Zellen entfernen .....	72
<b>3.12</b>	<b>Korrelation versus Kausalität .....</b>	<b>73</b>
<b>3.13</b>	<b>Normierung der Daten .....</b>	<b>81</b>
<b>3.14</b>	<b>Regression .....</b>	<b>84</b>
<b>3.15</b>	<b>Deployment .....</b>	<b>87</b>
3.15.1	Trainieren, testen und speichern .....	87
3.15.2	KNN anwenden .....	90
<b>3.16</b>	<b>Übungen .....</b>	<b>93</b>
3.16.1	Übung 1: Hyperparameter-Optimierung für Klassifizierung .....	93
3.16.2	Übung 2: Hyperparameter-Optimierung für Regression .....	93
3.16.3	Übung 3: KNN für Klassifizierung .....	93
3.16.4	Übung 4: KNN für Regression .....	94
<b>4</b>	<b>Entscheidungsbäume .....</b>	<b>95</b>
<hr/>		
<b>4.1</b>	<b>Einfache Entscheidungsbäume .....</b>	<b>96</b>
4.1.1	Decision Tree Classifier .....	96
4.1.2	Decision Tree Regressor .....	103
4.1.3	Entscheidungswälder .....	106

4.1.4	Random Forest Classifier .....	106
4.1.5	Random Forest Regressor .....	107
<b>4.2</b>	<b>Boosting</b> .....	<b>108</b>
4.2.1	Gradient Boosting .....	108
4.2.2	XGBoost Classifier .....	111
4.2.3	Automatische Hyperparameter-Einstellung mit GridSearchCV .....	116
<b>4.3</b>	<b>XGBoost Regressor</b> .....	<b>118</b>
<b>4.4</b>	<b>Deployment</b> .....	<b>120</b>
<b>4.5</b>	<b>Übungen</b> .....	<b>121</b>
4.5.1	Übung 1: XGBoost für Klassifizierung .....	121
4.5.2	Übung 2: XGBoost für Regression .....	121
4.5.3	Übung 3: Automatische Hyperparameter-Optimierung .....	122
<b>5</b>	<b>Faltungsschichten, Bilder und mehr</b> .....	<b>123</b>
<b>5.1</b>	<b>Einfache Bildklassifizierung</b> .....	<b>125</b>
<b>5.2</b>	<b>Hyperparameter-Optimierung mit Early Stopping und KerasTuner</b> .....	<b>130</b>
<b>5.3</b>	<b>Convolutional Neural Network (CNN)</b> .....	<b>135</b>
<b>5.4</b>	<b>Bildklassifizierung mit CIFAR-10</b> .....	<b>142</b>
<b>5.5</b>	<b>Verwendung vortrainierter Netze</b> .....	<b>146</b>
<b>5.6</b>	<b>Übungen</b> .....	<b>150</b>
5.6.1	Übung 1: Hyperparameter-Optimierung für CIFAR-10 .....	150
5.6.2	Übung 2: Vortrainiertes Modell VGG19 .....	150
<b>6</b>	<b>Transfer Learning</b> .....	<b>151</b>
<b>6.1</b>	<b>Funktionsweise</b> .....	<b>154</b>
<b>6.2</b>	<b>Übungen</b> .....	<b>161</b>
6.2.1	Übung 1: Stein-Schere-Papier .....	161
6.2.2	Übung 2: Mensch oder Pferd .....	161

<b>7</b>	<b>Anomalieerkennung</b>	163
7.1	Unausgewogene Daten .....	164
7.2	Resampling .....	169
7.3	Autoencoder .....	171
7.4	Übungen .....	178
7.4.1	Übung 1: Anomaliedetektion mit XGBoost und Upsampling .....	178
7.4.2	Übung 2: Anomaliedetektion mit Autoencoder .....	178
<b>8</b>	<b>Textklassifizierung</b>	179
8.1	Embedding Layer .....	179
8.2	GlobalAveragePooling1D Layer .....	183
8.3	Text Vectorization .....	185
8.4	Analyse der Zusammenhänge .....	188
8.5	Klassifizierung großer Datenmengen .....	193
8.6	Übungen .....	196
8.6.1	Übung 1: Hyperparameter-Optimierung .....	196
8.6.2	Übung 2: Textklassifizierung .....	196
8.6.3	Übung 3: Textklassifizierung mit Upsampling .....	196
<b>9</b>	<b>Clusteranalyse</b>	197
9.1	Grafische Analyse der Daten .....	198
9.2	Der Algorithmus k-Means-Clustering .....	203
9.3	Das fertige Programm .....	206
9.4	Übungen .....	209
9.4.1	Übung 1: Gruppierung von Diamanten .....	209
9.4.2	Übung 2: Gruppierung von Pilzen .....	210



<b>10</b>	<b>Visuelle Programmierung mit Orange</b>	211
<b>10.1</b>	<b>Installation</b>	211
<b>10.2</b>	<b>Klassifizierung mit Entscheidungsbaum</b>	213
<b>10.3</b>	<b>Regression mit Entscheidungsbaum</b>	220
<b>10.4</b>	<b>Deployment</b>	221
<b>10.5</b>	<b>Klassifizierung mit KNN</b>	223
<b>10.6</b>	<b>Regression mit KNN</b>	227
<b>10.7</b>	<b>Bildklassifizierung mit XGBoost</b>	229
<b>10.8</b>	<b>Unüberwachte Klassifizierung von Bildern</b>	230
<b>10.9</b>	<b>Textklassifizierung mit XGBoost</b>	234
<b>10.10</b>	<b>Übungen</b>	236
10.10.1	Übung 1: Klassifizierung	236
10.10.2	Übung 2: Regression	237
10.10.3	Übung 3: Bildklassifizierung	237
10.10.4	Übung 4: Textklassifizierung	237
<b>11</b>	<b>Visuelle Programmierung mit KNIME</b>	239
<b>11.1</b>	<b>Installation</b>	240
<b>11.2</b>	<b>Einfache künstliche neuronale Netze</b>	245
11.2.1	Klassifizierung	246
11.2.2	Klassifizierung mit Python Node	259
11.2.3	Regression	261
11.2.4	Regression mit Python Node	265
<b>11.3</b>	<b>XGBoost</b>	267
11.3.1	Klassifizierung	267
11.3.2	Deployment	269
11.3.3	Regression	271
<b>11.4</b>	<b>Bildklassifizierung mit vortrainiertem Modell</b>	271
11.4.1	Bildklassifizierung mit Keras Node	271
11.4.2	Bildklassifizierung mit Python Node	276

<b>11.5</b>	<b>Transfer Learning</b> .....	277
11.5.1	Transfer Learning mit Keras Node .....	278
11.5.2	Transfer Learning mit Python Node .....	281
<b>11.6</b>	<b>Autoencoder</b> .....	284
11.6.1	Autoencoder mit Keras Node .....	284
11.6.2	Autoencoder mit Python Node .....	289
<b>11.7</b>	<b>Textklassifizierung</b> .....	292
11.7.1	Textklassifizierung mit Keras Node .....	293
11.7.2	Textklassifizierung mit Python Node .....	295
<b>11.8</b>	<b>AutoML</b> .....	297
11.8.1	Installation .....	297
11.8.2	Klassifizierung .....	299
<b>11.9</b>	<b>Clusteranalyse</b> .....	301
11.9.1	Manuelle Clustereinstellung .....	302
11.9.2	Clustereinstellung mit einem Loop .....	303
<b>11.10</b>	<b>Zeitreihenanalyse</b> .....	306
11.10.1	Rekurrente neuronale Netze .....	307
11.10.2	Long Short-Term Memory .....	308
11.10.3	Vorhersage des Energieverbrauchs (nächste Stunde) mit Keras Node .....	310
11.10.4	Vorhersage des Energieverbrauchs (nächste Stunde) mit Python Node .....	316
11.10.5	Vorhersage des Energieverbrauchs (nächste 500 Stunden) mit Keras Node .....	317
11.10.6	Vorhersage des Energieverbrauchs (nächste 500 Stunden) mit Python Node .....	319
<b>11.11</b>	<b>Weitere Hinweise zu KNIME</b> .....	322
<b>11.12</b>	<b>Übungen</b> .....	322
11.12.1	Übung 1: XGBoost für Klassifizierung, Pilze .....	322
11.12.2	Übung 2: XGBoost für Regression, Diamanten .....	322
11.12.3	Übung 3: Bildklassifizierung mit InceptionV3 .....	322
11.12.4	Übung 4: Transfer Learning, »Mensch oder Pferd« .....	322
11.12.5	Übung 5: Anomaliedetektion mit Autoencoder, EKG .....	323
11.12.6	Übung 6: Textklassifizierung .....	323
11.12.7	Übung 7: AutoML für Regression .....	323
11.12.8	Übung 8: Clusteranalyse .....	323
11.12.9	Übung 9: Zeitreihenanalyse .....	324

<b>12</b>	<b>Reinforcement Learning</b>	325
<b>12.1</b>	<b>Q-Learning</b>	327
<b>12.2</b>	<b>Erforderliche Python-Kenntnisse für das Spiel</b>	332
12.2.1	Listen	332
12.2.2	Verzweigungen	333
12.2.3	Schleifen	335
12.2.4	Random Choice	336
12.2.5	Funktionen	337
<b>12.3</b>	<b>Training</b>	337
<b>12.4</b>	<b>Test</b>	340
<b>12.5</b>	<b>Ausblick</b>	341
<b>12.6</b>	<b>Übungen</b>	342
12.6.1	Übung 1: Hyperparameter	342
12.6.2	Übung 2: Erweiterung des Spiels	342
<b>13</b>	<b>Genetische Algorithmen</b>	343
<b>13.1</b>	<b>Der Algorithmus</b>	344
13.1.1	Startgeneration	345
13.1.2	Selektion	346
13.1.3	Reproduktion	346
13.1.4	Mutation	347
13.1.5	Neue Generation	348
<b>13.2</b>	<b>Beispiel einer sortierten Liste</b>	348
<b>13.3</b>	<b>Beispiel für Gleichungssysteme</b>	351
<b>13.4</b>	<b>Beispielanwendung aus der Praxis</b>	353
<b>13.5</b>	<b>Übungen</b>	357
13.5.1	Übung 1: Hyperparameter-Optimierung	357
13.5.2	Übung 2: Gleichungssystem	357

<b>14</b>	<b>ChatGPT und Gemini</b>	359
<b>14.1</b>	<b>Prompt Engineering</b>	362
14.1.1	Inhalte generieren	362
14.1.2	Programmieren	368
14.1.3	GPTs	371
14.1.4	Analysieren und Zusammenfassen	372
14.1.5	Abschließende Fragen an den Chatbot	373
<b>14.2</b>	<b>Programmierschnittstelle</b>	375
14.2.1	API-Key und erstes Programm mit ChatGPT	376
14.2.2	API-Key und erstes Programm mit Gemini	378
14.2.3	Parameter	379
14.2.4	API-Kontext	381
14.2.5	Eingabefilter	383
14.2.6	Gedächtnis	386
14.2.7	OpenAI Playground und Google AI Studio	388
14.2.8	Speech to Text	390
<b>14.3</b>	<b>Übung</b>	392
14.3.1	Übung 1: Mathematik-Support	392
<b>15</b>	<b>DALL-E und Imagen</b>	393
<b>15.1</b>	<b>Bildgenerierung</b>	394
15.1.1	Prompt Engineering	394
15.1.2	Bearbeitung generierter Bilder	395
15.1.3	Bilderstellung mit der API	398
<b>15.2</b>	<b>Übung 1: API mit Moderation</b>	402
<b>16</b>	<b>Ausblick</b>	403

<b>Anhang</b>	405
<b>A</b>	<b>Lösungen</b> ..... 405
A.1	Kapitel 3 ..... 405
A.2	Kapitel 4 ..... 411
A.3	Kapitel 6 ..... 414
A.4	Kapitel 7 ..... 416
A.5	Kapitel 8 ..... 419
A.6	Kapitel 9 ..... 423
A.7	Kapitel 10 ..... 425
A.8	Kapitel 11 ..... 426
A.9	Kapitel 12 ..... 434
A.10	Kapitel 13 ..... 436
A.11	Kapitel 14 ..... 438
A.12	Kapitel 15 ..... 439
<b>B</b>	<b>Literaturhinweise</b> ..... 441
Index	..... 443

# Index

## A

---

Abweichung .....	53
accuracy .....	62–63
Agent .....	327
Aktivierungsfunktion .....	49
Alpha-Regularisierung .....	227
Anaconda-Distribution .....	25
Anomalieerkennung .....	163
API (Application Programming Interface) ...	375
API-Key .....	376
Artificial Intelligence .....	19
Autoencoder .....	171, 284
AutoML .....	297

## B

---

Backpropagation .....	52
Bagging .....	106
Bag of Words .....	235
batch_size .....	145
BatchNormalization .....	145
Bearbeitung generierter Bilder .....	395
Belohnung .....	328, 330
Bias .....	57, 75, 128
Bilderstellung .....	398
Bildklassifizierung mit vortrainiertem Modell .....	271
Bildtiefe .....	138
Blackbox .....	44
Blattknoten .....	96
Boolean .....	176
Boosting .....	108

## C

---

ChatGPT .....	27, 359
CIFAR-10-Datenbank .....	142
Classification → Klassifizierung	
Clusteranalyse .....	197, 301
Colaboraty .....	35
Color Manager .....	302

Column Auto Type Cast .....	276
Column Rename .....	304
Column Resorter .....	323
Column Selection .....	272
Confusion Matrix .....	166
Convolutional Layer (Faltungsschicht) .....	135
Convolutional Neural Network (CNN) .....	135
Counter Generation .....	304
Create Collection Column .....	250
CSV-Datei .....	59
CSV-Reader .....	247

## D

---

DALL-E .....	393
Datentyp .....	23
Datentyp Dictionary .....	117
Datenvorbereitung .....	42, 247
Decision Tree Classifier .....	96
Decision Tree Regressor .....	103
Decoder .....	172, 285
Deep Learning .....	47
Deep Q-Learning .....	341
Deployment .....	87, 269
Dictionary Replacer .....	294
Differenz .....	53
Dimension Swapper .....	273
Diskriminierung .....	76
Distortion Score .....	205
DL Python Network .....	280
Downsampling .....	169
Dropout .....	145

## E

---

Early Stopping .....	130
Eingabefilter .....	383
Embedding Layer .....	179
Empty Table Creator .....	304
Encoder .....	172, 285
Ensamble Learning .....	106, 108
Entropy Scorer .....	304

Entscheidungsbaum .....	96
Entscheidungswald .....	106
Evolutionszyklus .....	345

## F

Falsch-negativ .....	168
Falsch-positiv .....	168
Feed Forward .....	50
Fehler .....	53
Fehlerfunktion .....	55
File Reader .....	293
Fitnesswert .....	346
Flatten .....	140
Float .....	23
Führungsgröße (w) .....	355
Funktionen .....	337

## G

Gemini .....	359
Generative Pre-trained Transformer (GPT) .	359, 371
Genetischer Algorithmus .....	343–344
Gewicht .....	47
Gini Impurity .....	97
GlobalAveragePooling1D Layer .....	183
Gradient Boosting .....	108
GradientBoostingRegressor .....	110
Gradientenverfahren .....	55
Grayscale .....	125
GridSearchCV .....	116
Guess data type and format .....	313

## H

Hidden Layer .....	47
Hierarchical Clustering .....	232
Hyperparameter .....	66

## I

Image Calculator .....	271
Image Embedding .....	231
ImageNet .....	146

Inertia .....	205
Inhalt generieren .....	362
Inner Join .....	257
Integer .....	23
Integrated Development Environment (IDE)	25
Iris Dataset .....	40

## J

Joiner .....	257
JSON-Dateiformat .....	120
Jupyter Notebook .....	25, 29

## K

Kaggle .....	123
Kausalität .....	73
Keras Dense Layer .....	253
Keras Input Layer .....	253
Keras Network Executor .....	255
Keras Network Learner .....	253
Keras Network Reader .....	273
KerasTuner .....	130
Kernel .....	123, 136
k-fache Kreuzvalidierung .....	116
Klassifizierung .....	39, 246, 259, 267, 299
k-Means-Clustering .....	203, 302
KNIME Workbench .....	241
Knoten .....	47
Know Your Data .....	152
Kochrezept .....	42
Konfusionsmatrix .....	166
Konstruktor .....	103
Korrektklassifizierungsrate .....	46, 213
Korrelation .....	73
Kreuzvalidierung (Cross Validation) .....	116

## L

Labeln .....	20, 126
Lag Column .....	313
Layer .....	47
Leaf Node .....	96
Lemmatization .....	196
Lernrate .....	56, 330

Line Chart .....	26
Liste .....	60
Listen .....	332
Long Short-Term Memory (LSTM) .....	308
Loop End .....	305
loss .....	63

## M

Machine Learning .....	20
Many to One .....	275
Math Formula .....	256
Matrix .....	51
Matrizenmultiplikation .....	51
Max Pooling .....	139
Mean Absolute Error .....	221
Mean Squared Error (MSE) .....	103
Metanode .....	278
Min-Max-Skalierung .....	82
Missing Value .....	313
Mittlere absolute Abweichung .....	87
MNIST-Datenbank .....	125
Model Writer .....	269
Monochrom .....	125
Mutation .....	347

## N

Natural Language Processing (NLP) .....	179
Neuronale Netze mit KNIME .....	253
Node .....	246
Normalizer .....	253
Normierung .....	81
Numeric Scorer .....	265

## O

One Hot Encoding (OHE) .....	69
OpenCV-Python .....	137
Orange .....	25, 211
Overfitting .....	48, 66–67, 158

## P

Pandas DataFrame .....	60
Pandas Series .....	60
Partitioning .....	251
PID-Regler .....	354
Playground .....	388
Pooling Layer .....	139
Programmieren .....	368
Programmierschnittstelle (ChatGPT) .....	375
Programmierschnittstelle (DALL-E) .....	398
Prompt Engineering .....	362, 394
Punktprodukt .....	51
PyGAD .....	348
Python .....	23
Python Script (Baustein) .....	286

## Q

Q-Learning .....	327
Q-Tabelle .....	328

## R

Random Choice .....	336
Random Forest Classifier .....	106
Random Forest Regressor .....	107
range-Funktion .....	117
Recurrent Neural Network (RNN) .....	307
Regelgröße (x) .....	355
Regelkreis .....	354
Regression .....	40, 84, 261, 265, 271
Reinforcement Learning .....	20, 325
Reproduktion .....	346
Resampling .....	169
RGB-Farbmodell .....	141
Richtig-negativ .....	168
Richtig-positiv .....	168
Rollen .....	381
Root Node .....	96
RowID .....	288
Row Splitter .....	285
Rule Engine .....	248





# Eigene KI-Anwendungen programmieren

>create image

Künstliche Intelligenz in eigenen Projekten einsetzen – machen Sie mit! Tauchen Sie ein in Reinforcement Learning, Classifier, DALL-E und Co. Erfahren Sie, wie KI-Verfahren intern arbeiten und was das für die Praxis bedeutet. Mit vielen Hands-on-Anleitungen steigen Sie ohne Vorkenntnisse ein.



## Konzepte und Verfahren

KI ist nicht gleich KI. Lernen Sie jedes Verfahren Schritt für Schritt kennen – auch mit etwas Mathematik, vor allem aber mit praktischen Anwendungen.

## Erprobte Projekte mit starken Tools

Die Praxisprojekte nutzen leistungsfähige Frameworks wie TensorFlow und Keras. Mit Jupyter Notebooks und Python arbeiten Sie lokal oder legen ohne Installation direkt im Browser los.

## Datenanalyse mit KNIME und Orange

Verarbeiten Sie Daten aus verschiedenen Quellen, trainieren Sie Neuronale Netze und führen Sie professionelle Analysen durch – alles mit visueller Programmierung.



**Beispielcode, Datenmaterial und einsatzbereite Jupyter Notebooks zum Download**



Metin Karatas unterrichtet an einer Fachschule für berufliche Bildung u. a. das Fach »Künstliche Intelligenz«. Die Projekte in diesem Buch hat er eigens dafür entwickelt. Er verbindet wo immer möglich Theorie mit praktischer Erfahrung.

## Installation und Werkzeuge

- + Anleitungen für alle Tools
- + Anaconda und Python
- + Jupyter Notebooks
- + TensorFlow und Keras
- + KNIME und Orange

## Verfahren im Detail

- + Convolutional Neural Networks
- + Reinforcement Learning
- + Transfer Learning
- + Transformer wie ChatGPT

## Praktische Anwendungen

- + Anomalieerkennung
- + Zeitreihenanalysen
- + Texte generieren
- + Bilderkennung

