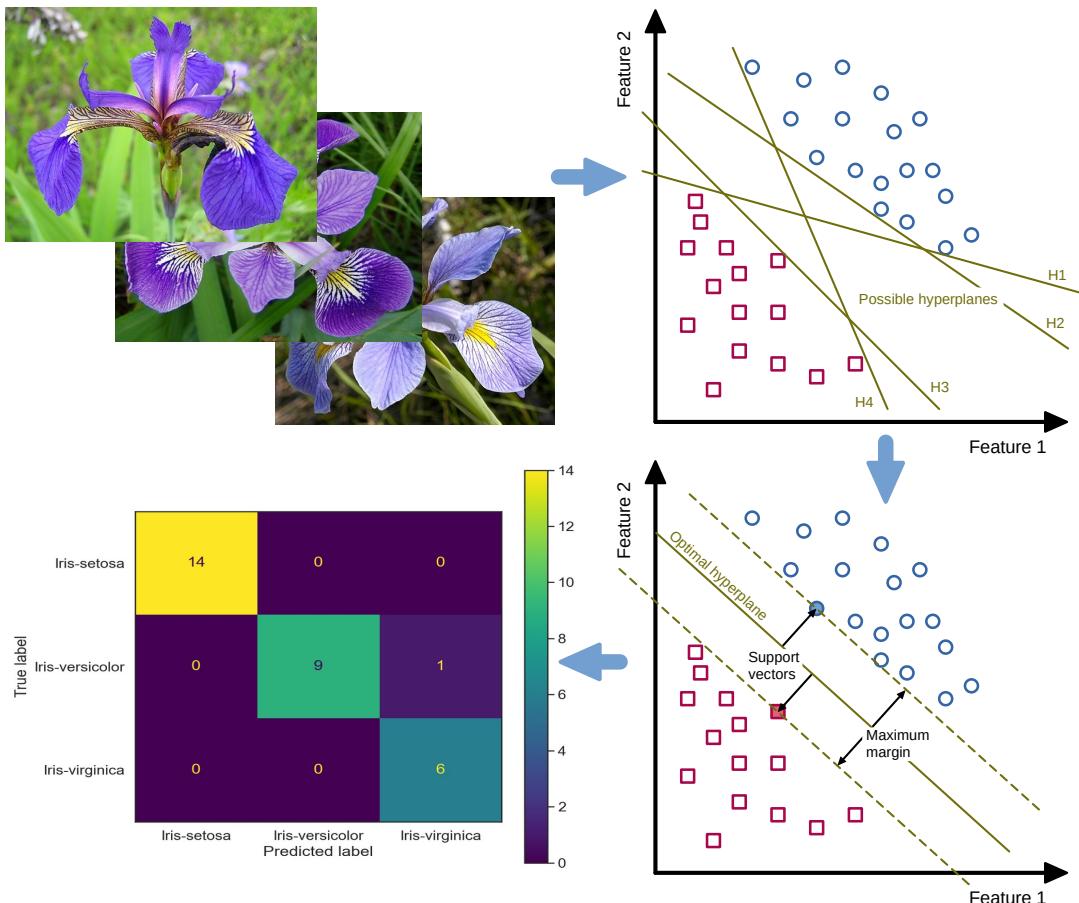


Getting started with Machine Learning (ML) and Support Vector Classifiers (SVC) - A systematic step-by-step approach

Dipl.-Ing. Björn Kasper (kasper.bjoern@bgetem.de)

Test and Certification Body for Electrical Engineering at BG ETEM

September 9, 2022



Anyone who wants to seriously deal with the emerging topic of our time “Artificial Intelligence (AI)” cannot avoid dealing with the basic mathematical models and algorithms from the field of “Machine Learning (ML)” as a subset of AI. However, someone who opens the door for the first time to this equally very exciting as well as arbitrarily complex and, at first glance, confusing world will very quickly be overwhelmed. Here, it is a good idea to consult introductory and systematic tutorials. Therefore, this Getting Started tutorial systematically demonstrates the typical ML work process step-by-step using the very powerful and performant “Support Vector Classifier (SVC)” and the widely known and exceptionally beginner-friendly “Iris Dataset”. Furthermore, the selection of the “correct” SVC kernel and its parameters are described and their effects on the classification result are shown.



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License \(CC BY-SA 4.0\)](#).

Contents

1	Introduction	3
1.1	English introduction	3
1.2	German introduction	6
1.3	Steps of the systematic ML process	8
2	STEP 0: Select hardware and software suitable for ML	8
2.1	Hardware	8
2.1.1	Training system	9
2.1.2	Application system	9
2.2	Software	10
2.2.1	Programming languages	10
2.2.2	Python packages	12
2.2.3	Programming IDEs	14
2.2.4	Operating systems	18
2.3	Community Support	19
2.4	Import Python packages	20
3	STEP 1: Acquire the ML dataset	20
4	STEP 2: Explore the ML dataset	20
4.1	Goals of exploration	21
4.2	Clarify the origins history	21
4.3	Overview of the internal structure and organization of the data	21
4.3.1	Inspect structure of dataframe	22
4.3.2	Get data types	26
4.3.3	Get data ranges and distribution	26
4.4	Identify anomalies in the datasets	29
4.4.1	Find and repair gaps in dataset	29
4.4.2	Find and remove duplicates in dataset	36
4.4.3	Compare the edited dataset with the original dataset side-by-side	39
4.4.4	Save edited dataset to new CSV file	39
4.5	Avoidance of tendencies due to bias	39
4.5.1	Count occurrences of unique values	39
4.5.2	Display Histogram	40
4.6	First idea of correlations in dataset	41
4.6.1	Visualise data with correlation heatmap	41
4.6.2	Visualise data with scatter plot	43
4.6.3	Visualise data with pairs plot	44
5	STEP 3: Choose and create the ML model	45
5.1	Short overview of the AI world and its ML algorithms	45
5.1.1	Relationship between AI, ML and others	45
5.1.2	Taxonomy of machine learning	46
5.2	Decision graph for selecting an suitable algorithm	48
5.3	Reasons for choosing Support Vector Classifier (SVC)	48
5.4	Operating principal of SVC	48
5.5	Create the SVC model	49
6	STEP 4: Prepare the dataset for training	49
6.1	Standarization	50
6.2	Normalization	50
7	STEP 5: Carry out training, prediction and testing	50
7.1	Split the dataset	50
7.2	Train the SVC	51
7.3	Make predictions	51

8 STEP 6: Evaluate model's performance	52
8.1 Accuracy Score	52
8.2 Classification Report	52
8.3 Cross-validation score	52
8.4 Confusion matrix	53
8.4.1 Textual confusion matrix	53
8.4.2 Colored confusion matrix	53
9 STEP 7: Vary parameters of the ML model manually	54
9.1 Prepare dataset	54
9.1.1 Prepare datasets for parameter variation and plotting	54
9.1.2 Prepare dataset for prediction and evaluation	55
9.2 Plotting functions	55
9.3 Vary kernel of SVC	56
9.4 Vary gamma parameter	58
9.5 Vary C parameter	61
9.6 Vary degree parameter	64
10 STEP 8: Tune the ML model systematically	69
10.1 Finding a baseline	70
10.2 Grid search	72
10.3 Randomized search	74
11 Summary and outlook	77
11.1 English summary	77
11.2 German summary	77
12 Acknowledgments	77
13 References	77

1 Introduction

1.1 English introduction

In the **digitized work environment**, there is an increasing demand for **Work equipment** to be able to adapt independently and in a task-related manner to changing work situations. Depending on the strength of the degree of flexibility, this **situational adaptivity** can often only be realized by applying mathematical models and algorithms from the field of **Machine Learning (ML)** as a subset of **Artificial Intelligence (AI)**.

Examples of such AI applications in work environments can range from comparatively simple **voice assistance systems** (similar, for example, to Siri or Alexa from the private sphere) to partially or **highly automated systems**. The transition from **automation to autonomy** is currently the subject of much controversy among experts and can be viewed in terms of the transition of responsibility from humans to technical systems (Adler 2021; Adler 2019).

By definition, a system is called **autonomous** only when it can achieve a given goal **independently** and adapted to the situation **without human control** or detailed **programming** (Dumitrescu et al. 2018; Kadner et al. 2017).

However, the distinction between the degree of automation and the autonomy of a technical system is relatively vague and difficult to define, depending on the technical context and the degree of abstraction. Crucial for the classification are the degrees of **self-determination**, **independence** as well as the **freedom of decision or action** of a technical system towards **human intervention** or preprogrammed behavior patterns (vgl. Wikipedia: Autonomie 2022).

In contrast to highly automated systems, autonomous systems are only able to act autonomously, solve problems, and learn to constantly improve in the process through the use of AI algorithms (Kadner et al. 2017).

For example, **driverless transport systems (AGVs)** can navigate **autonomously** through larger industrial facilities using self-learned self-updated maps shared with other AGVs, and avoid location-changing obstacles by independently finding and optimizing suitable routes. However, at a higher level of abstraction, new logistics tasks are given to them by human operators, which is why AGVs tend to be **highly automated systems** from a human perspective.

In addition to the many very interesting advantages, e.g. in terms of economic efficiency and workload reduction, such highly automated systems and, depending on the point of view, autonomous subsystems are characterized by a very high level of technical complexity. This concerns both their **operating functions** (e.g. autonomous navigation through complex industrial environments with shared use of the roadways by other human-controlled vehicles) and their **safety functions** (e.g. evaluation of interlinked imaging and non-imaging safety sensors for monitoring the driving space to avoid collisions).

Very high requirements are placed on such autonomous systems and the AI algorithms used for this purpose with regard to **functional safety**. However, the requirements for safety evalability in terms of **transparency** (complete understanding of the system) and **explainability** of decisions made by AI are currently very difficult or impossible to achieve, especially when using AI algorithms from the field of **deep learning** (Kuhn and Liggesmeyer 2019).

Unlike automated systems, the functionality of AI-powered autonomous systems is not fully programmed out before operational use, but is created by applying algorithms with learning capabilities to data. This results in a model that is merely executed by the software at runtime. Due to its **inherent complexity**, the resulting model is generally **not comprehensible** to humans, which means that the **decisions** of an AI system are often **not transparent**. Although the requirements for the AI system typically cannot be fully described, it must still function reliably later at runtime in a very large application space (Heidrich, Schneider, and Jedlitschka 2021). This pushes today's established methods and techniques of systematic software design and testing of safety-related software to their limits (cf. **V model** according to DIN EN 61508-3 2011).

Furthermore, in terms of their **recognition rates** and thus the **reliability of their decisions**, today's AI algorithms very often do not meet the functional safety requirements to achieve higher safety levels, even under the most favorable conditions. For example, a software-based safety function with a performance level d (PL_d) typically required for machines in accordance with ISO 13849-1 may only fail dangerously with a probability of $10^{-7} - 10^{-6}$ per hour during continuous use (see table K.1 in DIN EN ISO 13849-1 2016).

Compared to traditional, fully programmed software, the relatively low robustness of data-driven algorithms from the field of deep learning is another challenge. This can cause **small changes** in the function-determining **training data** to cause **large and unpredictable changes** in system behavior under some circumstances. However, the **predictability** and **transparency** of the system behavior are elementary for a **safety verification** (Jürgensohn et al. 2021).

An appropriate assessment or even **testing** with regard to the required functional safety according to uniform and ideally standardized criteria has numerous consequences for the future orientation and organization of technical **occupational safety and health (OSH)** in Germany and in Europe. In addition to the currently still very difficult safety-related assessability, an important point is that the previous clear separation between **placing on the market law** (see e.g. Machinery Directive) and **occupational safety and health law** (see European Framework Directive for Occupational Safety and Health and German Ordinance on Occupational Safety and Health) can no longer be continued in this way. The reason for this is that **safety-related properties** will also change, especially of systems **continuously learning** at runtime, due to new or **adapted behaviors** learned during operation (Jürgensohn et al. 2021). From today's point of view, systems based on **learned-out** and at runtime **invariable models** are not affected by this.

For these reasons, especially the actors of **technical occupational safety and health** who will deal with the **evaluation** of such **systems capable of learning** or system components with AI algorithms in the future should familiarize themselves in depth with the software structures used for this purpose as early as possible. This is the only way to ensure that the rapid development of systems capable of learning can be accompanied by OSH and their testing authorities in a constructive, critical and technically appropriate manner. If this is omitted, it must be assumed on the basis of the experiences of recent years that the OSH system will be ruthlessly circumvented or undermined by the economic interests of globally operating software giants. This would have the consequence that serious or fatal

occupational accidents are more likely to occur **due to inadequately designed AI-based work systems.**

However, the safety-related evaluation of such learning-capable systems requires a more in-depth technical entry into the world of **machine learning** as a subfield of **artificial intelligence**. For this purpose, it is necessary to deal with the basic operation of typical ML algorithms, corresponding software tools, libraries and programming systems.

However, someone who opens the door for the first time to this equally very exciting as well as arbitrarily complex and, at first glance, confusing world will very quickly be overwhelmed. In addition to reading general technical literature, it is advisable to consult introductory and systematic tutorials.

This Getting Started tutorial has exactly this goal, demonstrating systematically and step-by-step the typical ML workflow using the very powerful **Support Vector Classifier (SVC)** as an example.

This tutorial will be presented in the context of a workshop at the **Conference “Artificial Intelligence”**, hosted by the German Social Accident Insurance (DGUV), probably in November 2022 in Dresden. The workshop addresses interested ML novices in the technical occupational safety and health of the social accident insurance institutions.

Besides the **deep neural networks**, which are very present in the media, there is a very rich diversity of other very powerful ML algorithms - suitable for the particular use case. For a more generally comprehensible introduction, the SVC algorithm was deliberately chosen for the target audience of the workshop. Its operating principles are easy to convey to ML novices as well as in the time frame given for the workshop - quite in contrast to the entry into the world of deep neural networks.

The following main sections will demonstrate the typical ML workflow step-by-step. In **step 0**, specific guidance is provided for selecting hardware and software suitable for machine learning. To allow an ML novice to first familiarize themselves with the ML algorithms, tools, libraries, and programming systems, the ready-made and very beginner-friendly **Iris dataset** is involved in **step 1**. Only after a comprehensive acquaintance with the application of ML tools would it make sense to examine one's own environment for ML-suitable applications and to obtain suitable datasets from them. However, this is beyond the scope of this introductory tutorial.

One of the most important steps in the entire ML process is **step 2**, in which the dataset included in step 1 is examined using typical data analysis tools. In addition to exploring the **data structure** and **internal correlations** in the dataset, errors such as gaps, duplications, or obvious misentries must also be found and corrected where possible. This is enormously important so that the classification can later provide plausible results.

After exploring the dataset, in **step 3** one has to decide on a specific ML algorithm based on certain selection criteria. Among other ML algorithms suitable for the Iris dataset (such as the decision-tree-based **random-forests classifier**), the reasoned choice here in the tutorial falls on the **support vector classifier**. A dedicated SVC model is now being implemented.

In **step 4** the dataset is prepared for the actual classification by SVC. Depending on the selected ML algorithm as well as the data structure, it may be necessary to prepare the data before training (e.g., by standardization, normalization, or binarization based on thresholds). After splitting the dataset into a training and test dataset, the SVC model is trained with the training dataset in **step 5**. Subsequently, classification predictions are made with the trained SVC model based on the test data. In **step 6**, the quality of the classification result is evaluated using known **metrics** such as the **confusion matrix**.

Since the classification in step 5 was initially performed with standard parameters (so-called **hyper-parameters**), their meaning is explained in **step 7** and then their effect on the classification result is demonstrated by manually varying the individual hyper-parameters.

In the final **step 8**, two approaches to systematic hyper-parameter search are presented: **Grid Search** and **Randomized Search**. While the former exhaustively considers all parameter combinations for given values, the latter selects a number of candidates from a parameter space with a particular random distribution.

1.2 German introduction

Von den **Arbeitsmitteln** in der **digitalisierten Arbeitswelt** wird immer stärker gefordert, dass sie sich selbstständig und aufgabenbezogen an sich ändernde Arbeitssituationen anpassen können. Diese **situative Adaptivität** kann je nach Stärke des Flexibilisierungsgrades oft nur durch die Anwendung mathematischer Modelle und Algorithmen aus dem Bereich des **Maschinellen Lernens (ML)** als Teilmenge der **Künstlichen Intelligenz (KI)** realisiert werden.

Beispiele für solche KI-Anwendungen in der Arbeitswelt reichen von vergleichsweise einfachen **Sprachassistentensystemen** (ähnlich z. B. Siri oder Alexa aus dem privaten Umfeld) bis hin zu teil- oder **hochautomatisierten Systemen**. Der Übergang von **Automatisierung zu Autonomie** wird derzeit in der Fachwelt sehr kontrovers diskutiert und kann unter dem Aspekt des Übergangs der Verantwortung vom Menschen zum technischen System betrachtet werden (Adler 2021; Adler 2019).

Definitionsgemäß wird ein System erst dann als **autonom** bezeichnet, wenn es **ohne menschliche Steuerung** oder detaillierte **Programmierung** ein vorgegebenes Ziel **selbstständig** und an die Situation angepasst erreichen kann (Dumitrescu et al. 2018; Kadner et al. 2017).

Allerdings ist die Unterscheidung des Grades der Automatisierung bis hin zur Autonomie eines technischen Systems relativ fließend und je nach fachlichem Kontext und Abstraktionsgrad nur schwer zu definieren. Maßgeblich für die Einordnung sind die Grade der **Selbstbestimmtheit**, die **Unabhängigkeit** sowie die **Entscheidungs- bzw. Handlungsfreiheit** eines technischen Systems gegenüber **menschlichem Eingriff** oder vorprogrammierter Verhaltensmuster (vgl. Wikipedia: Autonomie 2022).

Im Gegensatz zu hochautomatisierten Systemen sind autonome Systeme nur durch Einsatz von KI-Algorithmen in der Lage, eigenständig zu agieren, Probleme zu lösen und dabei zu lernen, sich ständig zu verbessern (Kadner et al. 2017).

Beispielsweise können **fahrerlose Transportsysteme (FTS)** anhand selbst erlerner, selbstständig aktualisierter und mit anderen FTS geteilter Karten **autonom** durch größere Industrieanlagen navigieren und ortsveränderlichen Hindernissen ausweichen, indem sie selbstständig geeignete Routen finden und optimieren. Jedoch werden ihnen in einer höheren Abstraktionsebene neue Logistikaufträge durch menschliche Bediener vorgegeben, weswegen es sich bei FTS aus menschlicher Perspektive eher um **hochautomatisierte Systeme** handelt.

Neben den vielen sehr interessanten Vorteilen z. B. bzgl. Wirtschaftlichkeit und Arbeitserleichterung kennzeichnet solche hochautomatisierten und je nach Betrachtung autonomen Teilsysteme eine sehr hohe technische Komplexität. Diese betrifft sowohl ihre **Betriebsfunktionen** (z. B. autonome Navigation durch komplexe industrielle Umgebungen bei gemeinsamer Nutzung der Fahrwege durch andere menschlich gesteuerte Fahrzeuge) als auch ihre **Sicherheitsfunktionen** (z. B. Auswertung miteinander verknüpfter bildgebender und nicht-bildgebender Sicherheitssensorik zur Überwachung des Fahrraums zur Kollisionsvermeidung).

An solche autonomen Systeme und die hierfür eingesetzten KI-Algorithmen werden sehr hohe Anforderungen hinsichtlich der **funktionalen Sicherheit** gestellt. Jedoch sind die Anforderungen für eine sicherheitstechnische Bewertbarkeit bezüglich der **Transparenz** (vollständiges Systemverständnis) und **Erklärbarkeit** der durch KI getroffenen Entscheidungen insbesondere bei Einsatz von KI-Algorithmen aus dem Bereich des **Deep Learnings** derzeit nur sehr schwer oder gar nicht erreichbar (Kuhn and Liggesmeyer 2019).

Im Gegensatz zu automatisierten Systemen wird die Funktionalität KI-gestützter autonomer Systeme nicht vor der betrieblichen Verwendung vollständig ausprogrammiert, sondern durch das Anwenden lernfähiger Algorithmen auf Daten erstellt. Dadurch entsteht ein Modell, das von der Software zur Laufzeit lediglich ausgeführt wird. Das resultierende Modell ist aufgrund seiner **inhärenten Komplexität** im Allgemeinen **für den Menschen nicht verständlich**, wodurch die **Entscheidungen** eines KI-Systems oft **nicht transparent** sind. Obwohl die Anforderungen an das KI-System typischerweise nicht vollständig beschrieben werden können, muss es später zur Laufzeit in einem sehr großen Anwendungsbereich trotzdem verlässlich funktionieren (Heidrich, Schneider, and Jedlitschka 2021). Dadurch kommen die heute etablierten Methoden und Techniken des systematischen Softwareentwurfes und -testens sicherheitsgerichteter Software an ihre Grenzen (vgl. **V-Modell** nach DIN EN 61508-3 2011).

Weiterhin erfüllen heutige KI-Algorithmen hinsichtlich ihrer erreichbaren **Erkennungsraten** und damit der **Zuverlässigkeit ihrer Entscheidungen** selbst unter günstigsten Bedingungen sehr oft nicht die

Anforderungen an die funktionale Sicherheit, um höhere Safety-Level zu erreichen. Beispielsweise darf eine software-gestützte Sicherheitsfunktion mit einem für Maschinen typischerweise geforderten Performance Level d (PL_d) nach ISO 13849-1 bei kontinuierlicher Nutzung nur mit einer Wahrscheinlichkeit von $10^{-7} - 10^{-6}$ pro Stunde gefährlich ausfallen (siehe Tabelle K.1 in DIN EN ISO 13849-1 2016).

Im Vergleich zu traditioneller, vollständig ausprogrammierter Software ist bei datengetriebenen Algorithmen aus dem Bereich des Deep Learnings die verhältnismäßig geringe Robustheit eine weitere Herausforderung. Diese kann dazu führen, dass **kleine Änderungen** in den funktionsbestimmenden **Trainingsdaten** unter Umständen **große und unvorhersehbare Veränderungen** des Systemverhaltens bewirken. Jedoch sind die **Vorhersehbarkeit** und **Nachvollziehbarkeit** des Systemverhaltens für einen **Sicherheitsnachweis** elementar (Jürgensohn et al. 2021).

Eine hinsichtlich der geforderten funktionalen Sicherheit angemessene Bewertung oder gar **Prüfung** nach einheitlichen und idealerweise genormten Maßstäben hat viele Konsequenzen für die zukünftige Ausrichtung und Gestaltung des **technischen Arbeitsschutzes** in Deutschland und in Europa. Neben der derzeit noch sehr schwierigen sicherheitstechnischen Bewertbarkeit von KI-Algorithmen ist ein wichtiger Punkt, dass die bisherige klare Trennung zwischen **Inverkehrbringensrecht** (siehe z. B. Maschinenrichtlinie) und **betrieblichem Arbeitsschutzrecht** (siehe Arbeitsschutz-Rahmenrichtlinie und Betriebssicherheitsverordnung) so nicht mehr aufrechterhalten werden kann. Grund hierfür ist, dass sich auch die **sicherheitsrelevanten Eigenschaften** insbesondere von zur Laufzeit **weiterlernenden Systemen** durch während des Betriebs erlernte, neue oder **angepasste Verhaltensweisen** verändern werden (Jürgensohn et al. 2021). Systeme auf Basis **ausgelernter** und zur Laufzeit **unveränderlicher Modelle** sind aus heutiger Sicht hiervon nicht betroffen.

Aus diesen Gründen sollten sich insbesondere die Akteure des **technischen Arbeitsschutzes**, die sich zukünftig mit der **Prüfung** solcher **lernfähigen Systeme** oder Systemkomponenten mit KI-Algorithmen befassen werden, möglichst frühzeitig mit den hierfür eingesetzten Software-Strukturen vertieft auseinandersetzen. Nur dadurch lässt sich erreichen, dass die stürmische Entwicklung lernfähiger Systeme durch den Arbeitsschutz und dessen Prüfinstitute konstruktiv, kritisch und fachlich angemessen begleitet werden kann. Wird dies versäumt, muss aufgrund der Erfahrungen der vergangenen Jahre davon ausgegangen werden, dass das Arbeitsschutzsystem durch die wirtschaftlichen Interessen global agierender Softwaregiganten skrupellos umgangen oder ausgehebelt werden wird. Dies hätte die Folge, dass schwere oder tödliche **Arbeitsunfälle wegen unzulänglich gestalteter KI-basierter Arbeitssysteme** wahrscheinlicher werden.

Allerdings erfordert die sicherheitstechnische Bewertung solcher lernfähigen Systeme einen tiefer gehenden fachlichen Einstieg in die Welt des **maschinellen Lernens** als Teilgebiet der **künstlichen Intelligenz**. Hierzu muss sich mit den grundlegenden Funktionsweisen typischer ML-Algorithmen, entsprechenden Software-Werkzeugen, Bibliotheken und Programmiersystemen auseinander gesetzt werden.

Wer jedoch zum ersten Mal die Tür zu dieser ebenso spannenden wie beliebig komplexen und auf den ersten Blick verwirrenden Welt öffnet, wird sehr schnell überfordert sein. Hier empfiehlt es sich neben dem Lesen allgemeiner Fachliteratur, einführende und systematische Anleitungen zu Rate zu ziehen.

Genau dieses Ziel verfolgt das vorliegende Getting-Started-Tutorial, indem systematisch und Schritt-für-Schritt der typische ML-Arbeitsablauf am Beispiel des sehr leistungsfähigen **Support Vector Classifier (SVC)** demonstriert wird.

Dieses Tutorial wird im Rahmen eines Workshops auf der **Fachtagung "Künstliche Intelligenz"**, ausgerichtet durch die Deutsche Gesetzliche Unfallversicherung (DGUV), voraussichtlich im November 2022 in Dresden vorgestellt. Der Workshop richtet sich an interessierte ML-Neulinge im technischen Arbeitsschutz der gesetzlichen Unfallversicherungsträger.

Neben den medial sehr präsenten **tiefen neuronalen Netzen** gibt es eine sehr reichhaltige Auswahl anderer sehr leistungsfähiger ML-Algorithmen - passend für den jeweiligen Anwendungsfall. Für einen allgemein verständlicheren Einstieg wurde für die Zielgruppe des Workshops der SVC-Algorithmus bewusst gewählt. Dessen Arbeitsweise ist sowohl für ML-Neulinge als auch in dem für den Workshop vorgegebenen Zeitrahmen leicht vermittelbar - ganz im Gegensatz zum Einstieg in die Welt der tiefen neuronalen Netze.

Die folgenden Hauptabschnitte demonstrieren den typischen ML-Arbeitsablauf Schritt-für-Schritt. Im **Schritt 0** werden konkrete Hinweise für die Auswahl der für das maschinelle Lernen geeigneten Hardware und Software gegeben. Damit sich ein ML-Neuling zunächst mit den ML-Algorithmen, Werkzeugen,

Bibliotheken und Programmiersystemen vertraut machen kann, wird im **Schritt 1** der fertige und sehr einsteigerfreundliche **Iris-Datensatz** hinzugezogen. Erst nach einer umfassenden Einarbeitung in die Anwendung der ML-Werkzeuge wäre es sinnvoll, die eigene Umgebung auf ML-taugliche Anwendungen hin zu untersuchen und daraus geeignete Datensätze zu gewinnen. Dies geht jedoch über den Rahmen dieses einführenden Tutorials hinaus.

Mit der wichtigste Schritt im gesamten ML-Prozess ist **Schritt 2**, in dem der in Schritt 1 einbezogene Datensatz mit Hilfe typischer Datenanalyse-Werkzeuge untersucht wird. Neben der Erkundung der **Datenstruktur** sowie **innerer Zusammenhänge** im Datensatz müssen auch Fehler wie z. B. Lücken, Dopplungen oder offensichtliche Fehleingaben gefunden und nach Möglichkeit behoben werden. Dies ist enorm wichtig, damit die Klassifikation später plausible Ergebnisse liefern kann.

Nach der Erkundung des Datensatzes muss man sich im **Schritt 3** anhand bestimmter Auswahlkriterien für einen konkreten ML-Algorithmus entscheiden. Neben anderen für den Iris-Datensatz passenden ML-Algorithmen (wie z. B. der entscheidungsbaum-basierte **Random-forests-Classifier**) fällt die begründete Auswahl hier im Tutorial auf den **Support-Vector-Classifier**. Ein entsprechendes SVC-Modell wird nun implementiert.

Im **Schritt 4** wird der Datensatz für die eigentliche Klassifikation per SVC vorbereitet. Je nach gewähltem ML-Algorithmus sowie der Datenstruktur kann es erforderlich sein, dass die Daten vor dem Training aufbereitet werden müssen (z. B. durch Standardisierung, Normalisierung oder Binärisierung anhand von Schwellwerten). Nach der Aufteilung des Datensatzes in einen Trainings- und Testdatensatz, wird das SVC-Modell im **Schritt 5** mit dem Trainingsdatensatz trainiert. Anschließend werden mit dem trainierten SVC-Modell anhand der Testdaten Klassifikationsvorhersagen getroffen. Im **Schritt 6** wird die Güte des Klassifikationsergebnisses anhand bekannter **Metriken** wie z. B. der **Konfusionsmatrix** evaluiert.

Da die Klassifikation im Schritt 5 zunächst mit Standard-Parametern (den sogenannte **Hyper-Parametern**) durchgeführt wurde, wird ihre Bedeutung im **Schritt 7** erklärt und danach ihr Einfluss auf das Klassifikationsergebnis durch manuelle Variation der einzelnen Hyper-Parameter demonstriert.

Im abschließenden **Schritt 8** werden zwei Ansätze zur systematischen Hyper-Parameter-Suche vorgestellt: **Grid Search** und **Randomized Search**. Während bei ersterer für gegebene Werte erschöpfend alle Parameterkombinationen betrachtet werden, wird beim zweiten Ansatz eine Anzahl von Kandidaten aus einem Parameterraum mit einer bestimmten zufälligen Verteilung ausgewählt.

1.3 Steps of the systematic ML process

The following **steps of the systematic ML process** are covered in the next main sections:

- **STEP 0: Select hardware and software suitable for ML**
- **STEP 1: Acquire the ML dataset**
- **STEP 2: Explore the ML dataset**
- **STEP 3: Choose and create the ML model**
- **STEP 4: Prepare the dataset for training**
- **STEP 5: Carry out training, prediction and testing**
- **STEP 6: Evaluate model's performance**
- **STEP 7: Vary parameters of the ML model manually**
- **STEP 8: Tune the ML model systematically**

2 STEP 0: Select hardware and software suitable for ML

In this step, specific guidance is provided for selecting hardware and software suitable for machine learning.

2.1 Hardware

When considering hardware requirements, two systems and their use cases must be taken into account: the **training system** and the **application system**.

2.1.1 Training system

The **training phase** requires a lot of **computational power** and **memory (RAM)**, depending on the **amount of data** to be processed and the **ML algorithm (so-called estimator)** chosen.

Depending on the estimator model, highly parallel processing on a **Graphics Processing Unit (GPU)** can provide significant **speed advantages** over processing on a **Central Processing Unit (CPU)** (e.g., when training deep neural networks in the area of **deep learning**). To take advantage of this speed benefit, the AI application must be suitable in terms of **parallelizability** of the estimator model used as well as **GPU support** through special driver layers, the so-called **Operating System Abstraction Layer (OSAL)** (Wikipedia: OSAL 2022).

Such GPUs are installed on powerful **3D graphics cards**. However, these must be explicitly qualified for the application for AI - not every game-suitable graphics card from any manufacturer can be used. The manufacturer **Nvidia** offers GPUs suitable for AI in its high-performance graphics cards with **CUDA architecture**. **CUDA** stands for “Compute Unified Device Architecture” and is a **programming interface** (API) developed by Nvidia, with which program parts can be processed by the graphics processor (Wikipedia: CUDA 2022). A GPU with its several tens of thousands of threads can process highly parallelizable tasks that require only little data communication between the memory areas significantly more performantly than conventional CPUs. This speed advantage can be considerable despite currently available CPU technologies like **Multicore** with **Hyper-Threading** with Intel CPUs!

Nvidia graphics cards with CUDA-supporting GPUs are ranked based on their **compute capability** (NVIDIA.Developer 2022).

However, it should be mentioned that currently only the manufacturer Nvidia offers 3D graphics cards with CUDA implementation, since CUDA is a **proprietary framework**. In addition, there is also the much less well-known **open source** alternative **OpenCL**, which has now been implemented by a large number of graphics card manufacturers (Wikipedia: OpenCL 2022). Since OpenCL is an **open industry standard**, Intel and AMD chips and their GPUs, ATI Radeon cards of the 5, 6, 7 and R9 series as well as various Nvidia GeForce cards are supported, for example.

Regarding the **code execution performance** of both alternatives in direct comparison, there are different statements in the technical literature. The 2011 paper **A Performance Comparison of CUDA and OpenCL** sees the CUDA implementation as the clear favorite (Karimi, Dickson, and Hamze 2011). More recent publications point out the strong dependence of performance on **code quality**, **algorithm type** and the **GPU hardware** used, among other things - see e.g. here: **CUDA vs OpenCL: Which to Use for GPU Programming** (Exterman 2021).

It is therefore recommended that the decision for **CUDA or OpenCL** should depend on the extent to which most of the applications employed and the GPU hardware used are better supported by one of the two approaches in each case.

The **state of the art** should be also taken into account when selecting the rest of the training system's hardware. Otherwise, seemingly (price-wise) inexpensive components could very quickly nullify the speed advantage of the GPU. In addition to a mainboard suitable for one (or more) high-performance graphics cards with a correspondingly powerful BUS system (e.g. PCI Express), the RAM should be as large as possible (min. 64 GB) and fast. A large RAM allows, for example, the **virtualization** of several parallel systems in the form of **virtual machines** and thus a significantly better utilization of the available computing capacity (Wikipedia: VM 2022). The permanent memory should also be as large and fast as possible - high-performance **solid-state drives (SSDs)** should be clearly preferred over classic hard disks (HDDs).

2.1.2 Application system

In the **application phase** of the trained estimator model, considerably less computing power and RAM are usually required. If the concrete application does not require **continuous learning during operation**, significantly less expensive systems (in terms of acquisition costs, power consumption, etc.) can also be used. Such application-specific **embedded systems** have only one CPU (usually in **ARM architecture**), comparatively limited RAM (e.g. 1 - 8 GB) and usually no GPU. A popular **embedded computer** that is very well supported in terms of ML software is the **Raspberry Pi** (Wikipedia: Raspi 2022). In addition to its ARM CPU, the Raspberry Pi also has a GPU installed on the same processor

in the so-called **System on a Chip** design (**SoC**). However, the SoC manufacturer **Broadcom** does not support the CUDA API.

There are references in the technical literature that the open source alternative **OpenCL** can be installed on the Raspberry Pi and that the AI framework **TensorFlow** (see section “Software”) can be compiled with **SYCL** support, where SYCL stands for “Single Source OpenCL” (Wikipedia: SYCL 2022). However, a first rough review gives the impression that support for this approach is still very experimental at the moment. Therefore, parallelizing the AI application on the GPU of the Raspberry Pi does not seem to be an option (yet). Here are some links for further reading:

- Deep learning with Raspberry Pi and alternatives in 2022 (Politiek 2022a)
- Benchmarking Machine Learning on the New Raspberry Pi 4, Model B (Allan 2019)
- Portable Computer Vision: TensorFlow 2.0 on a Raspberry Pi (Johnson 2019)
- Install OpenCL on Raspberry Pi 3 B+ (Politiek 2022b)
- Does TensorFlow Support OpenCL? (IndianTechWarrior 2022)
- TensorFlow for OpenCL using SYCL (Iwanski 2016)

2.2 Software

2.2.1 Programming languages

The comparison of **advantages and disadvantages** of the various programming languages and the evaluation of their suitability for ML was inspired by the following articles, among others:

- What Is the Best Language for Machine Learning? (Gupta 2021)
- Is Octave Good for Machine Learning? (Adhikari 2021)

In summary, there is **no best language for machine learning**, each is good where it fits best.

However, there are definitely some programming languages that are better suited for machine learning tasks than others (Gupta 2021). On the one hand, this is due to whether the programming language is fundamentally well suited to **implement complex mathematical and statistical tasks** in efficient algorithms.

On the other hand, when deciding for or against a programming language, it should definitely also be taken into account whether it contains sufficient **basic functionalities for data analysis and its processing**, as well as very diverse **extension libraries** (so-called **packages**) that are well supported by the community are available. By using these libraries, it is possible to concentrate on the concrete task when creating an ML application and not have to constantly solve the same trivial problems anew in every new application (e.g. the efficient **handling of datasets** or the execution of **matrices calculations**).

Following trend chart shows how the **popularity of selected programming languages** suitable for machine learning has evolved since 2008:

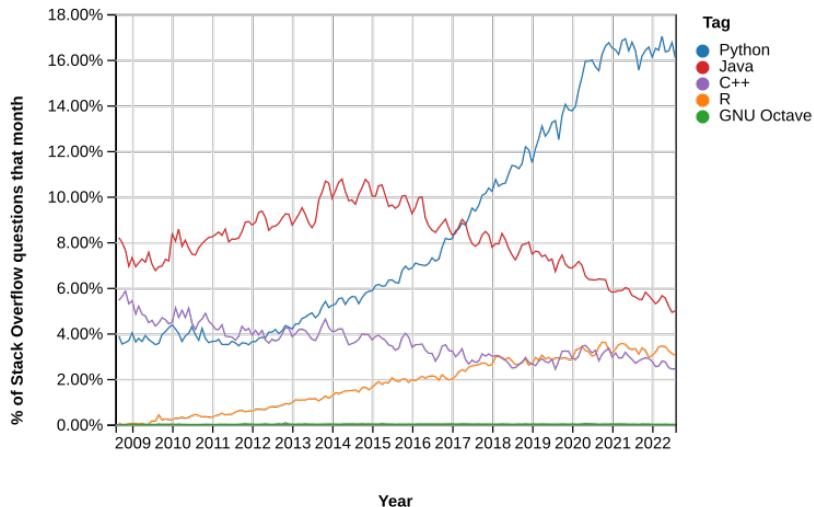


Figure 1: Trend chart shows popularity of programming languages for ML (source: [Stack Overflow Trends](#), license: CC BY-SA)

Python It is a high-level, **general-purpose** programming language where its design philosophy emphasizes **code readability**. The **variable types** in Python are **dynamic** and **memory is automatically managed** to create and delete data objects (see [garbage collection](#)).

Pros:

- Python offers simple, concise, and **readable code** for allowing to write robust and reliable programs.
- It lets you focus on solving the ML problem instead of getting lost in the language's technical nuances.
- Python has **extensive libraries for ML**, e.g. Scikit-learn, Pandas, TensorFlow or Keras have become standard libraries for various ML tasks.
- The language has been around for decades and has developed a large and helpful community.
- Besides extensive online documentation, there are thousands of question-answers and community guides for various functionalities of the language (this is also very well reflected in the trend graph on the popularity of programming languages).

Cons:

- (unknown drawbacks so far ...)

R It is a programming language for **statistical computing** and **graphics** supported by the **R Core Team** and the **R Foundation for Statistical Computing**. Created by statisticians Ross Ihaka and Robert Gentleman, R is used among data miners, bioinformaticians and statisticians for data analysis and developing statistical software.

Pros:

- After Python, R is the recommended ML programming language.
- R is a flexible and cross-platform compatible language.
- It has a growing, supportive community.
- R is well suited for data visualization and **statistics**, often making it the language of choice for applications with a large amount of statistical data.
- It is considered a powerful choice for **machine learning**, offering a variety of machine learning techniques (e.g., data visualization, data sampling, data analysis and supervised and unsupervised machine learning models) via post-installable libraries.

Cons:

- R is often reported to be laggier and slower as compared to Python when dealing with large-scale datasets.

- It has a **significantly lower community support** when answering questions or giving guidance **compared to Python** (see trend chart on popularity of programming languages).
- The **learning curve** for the basic entry into R and the application in more complex projects for data analysis or machine learning is significantly **steeper than with Python**.

Java It is a high-level, **class-based, object-oriented** programming language that is designed to have **as few implementation dependencies** as possible. It is a **general-purpose** programming language intended that compiled Java code can run on all platforms that support Java without the need to recompile.

Pros:

- Using Java for machine learning is especially popular among developers with a Java background, as it skips the need to learn another programming language such as Python or R.
- Like Python and R, Java also has a variety of third-party machine learning libraries, e.g. **JavaML** is a built-in library with a collection of algorithms implemented in Java for ML.
- Scalability is an important feature for many ML projects, which is well supported by Java.
- **Java Virtual Machine (JVM)** enables the development of ML applications for multiple platforms.
- Java is very well suited for speed-critical ML projects.

Cons:

- Java has a much lower community support in answering questions or giving guidance compared to Python - but a better one than R (see trend chart on popularity of programming languages).

GNU Octave It is a high-level programming language that's **designed for numerical computations** (Adhikari 2021).

Pros:

- With Octave, **linear and non-linear numerical problems** can be solved quickly.
- Octave is syntactically very similar to **MATLAB** and mostly **compatible with MATLAB**. If no MATLAB-specific functions are used, the program code also runs in Octave. In addition, Octave even has some language functions and a syntax diversity that MATLAB lacks.

Cons:

- However, Octave is not a good programming language for machine learning in a production environment.
- It doesn't have the same functionality as other languages used for ML, due to **missing libraries** and frameworks to speed up ML tasks.
- It's not as flexible, simple, and feature-rich as other programming languages.
- Compared to Python, R and Java, Octave has almost **no community support** when it comes to answering questions or providing guidance (compare trend chart on popularity of programming languages).

2.2.2 Python packages

The **mathematics** and the **numerical implementation** of various algorithms for data analysis and machine learning are usually **very complex** and often only comprehensible for ML experts (Gupta 2021). For a broad and praxis-oriented **usability**, better **reusability of code** and a successful **integration** into a concrete ML application, the functional relationships should be **encapsulated in libraries** (so-called "packages").

From the user's point of view, when selecting libraries for the respective task, attention should be paid not only to functionality but also to the **comprehensibility of the user interface supported by good documentation**. Furthermore, the **size of the community** behind the library, consisting of active developers as well as technical experts for supporting the users in the event of questions or problems arising, should be decisive in the selection.

Following trend chart shows how the **popularity of selected python packages** suitable for **data analysis, data visualization and machine learning** has evolved since 2008:

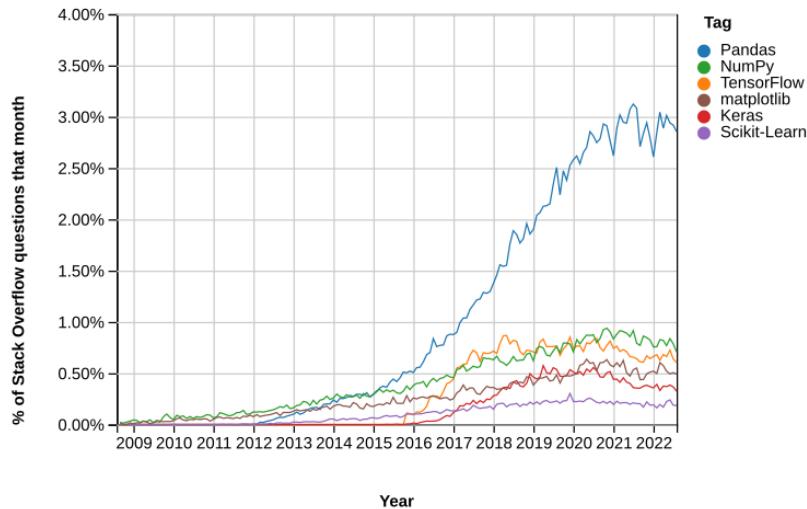


Figure 2: Trend chart shows popularity of selected python packages for data analysis, data visualization and machine learning (source: [Stack Overflow Trends](#), license: CC BY-SA)

In the scientific research and systematic improvement of ML algorithms, a very dynamic progression can be observed in recent years. The latest scientific findings are regularly compared with each other in “Machine Learning Competitions” using known and **freely available datasets** (see benchmarking competitions of ML algorithms on platforms such as <https://www.kaggle.com/competitions>). At the same time, the corresponding ML libraries are revised, extended and made available to general users by the scientific community. Therefore, this **scientific transfer** ideally takes place in the context of **open source developments**.

Due to the superior advantages of **Python** (see previous section), a selection of **open source** packages available for this programming language usable for **data analysis**, **data visualization** and **machine learning** are presented in this section.

Data analysis

- **NumPy** is a Python library that provides a **multidimensional array object**, various derived objects (such as masked arrays and **matrices**), and an assortment of routines for **fast operations on arrays**, including mathematical, logical, shape manipulation, sorting, selecting, discrete Fourier transformations, basic linear algebra, basic statistical operations, random simulation and much more.
- **Pandas** is a Python package providing fast, flexible and expressive data structures designed to work with **relational or labeled** datasets. It provides two primary data structures: **pandas.Series** (1-dimensional time series) and **pandas.DataFrame** (2-dimensional spreadsheets). The data structure **pandas.DataFrame** offers the same functionality as the structure **data.frame** known from the programming language R and much more.

Data visualization

- **Matplotlib** is a library for making **2D plots of arrays** in Python. Although it has its origins in **emulating the MATLAB graphics commands**, it is independent of MATLAB, and can be used in a Pythonic, object oriented way. Although Matplotlib is written primarily in pure Python, it makes heavy use of NumPy and other extension code to provide good performance even for large arrays (Hunter 2007).
- **Seaborn** is a library for making **statistical graphics** in Python. It builds **on top of matplotlib** and integrates closely with **pandas data structures**. Seaborn helps to explore and understand the data. Its plotting functions operate on **dataframes** and **arrays** containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots (Waskom 2021).

Machine learning

- **Scikit-Learn** is a free software machine learning library for Python. It features various **classification**, **regression** and **clustering** algorithms including **support-vector machines**, **random forests**, **gradient boosting** and **k-means**. It is designed to interoperate with the Python numerical and scientific libraries **NumPy** and **SciPy**. Scikit-Learn will be used in the next steps of this of this getting started tutorial.
- **TensorFlow** offers, among other things, the possibility to create and train **artificial neural networks (ANN)** based on **Google AI**. It is an open source software library for **machine learning** and **artificial intelligence**. It can be used across a range of tasks but has a particular focus on training and inference of **deep neural networks**. However, the installation and usage is very much beyond the scope of this beginner tutorial.
- **Keras** is an open source software library for **deep learning** that provides a Python interface for **ANNs**. Keras acts as an **general interface** for several **backends**, such as **TensorFlow**, **Microsoft Cognitive Toolkit** and **Theano**. Keras will also not be used in this beginner tutorial.

2.2.3 Programming IDEs

Integrated development environments (IDE) are software applications that provide comprehensive features to computer programmers for **software development**. An IDE typically consists of a **source code editor**, automated **build tools** for compiling or an **interpreter** for scripting languages, a front end to the **version control system** like e.g. **Git** and a **debugger** (Wikipedia: **IDE 2022**).

Following trend chart shows how the **popularity of selected IDEs** suitable for ML programming languages has evolved since 2008:

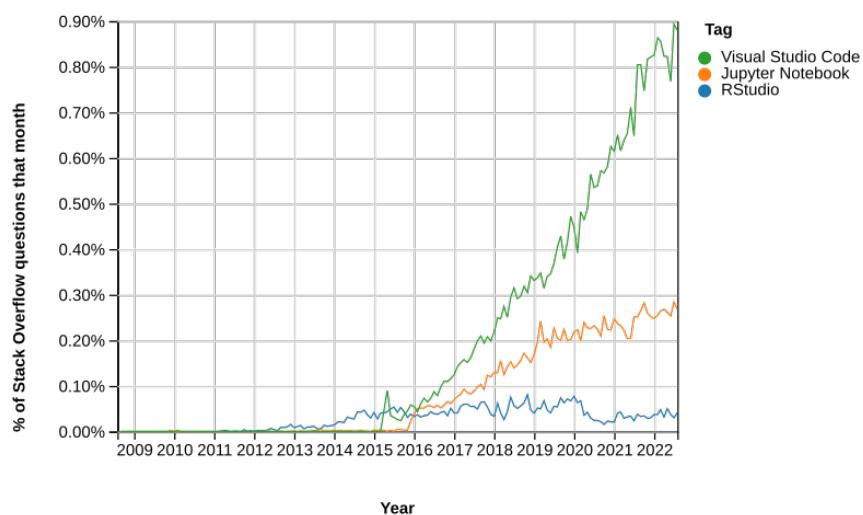


Figure 3: Trend chart shows popularity of selected IDEs for ML programming languages (source: [Stack Overflow Trends](#), license: CC BY-SA)

Visual Studio Code (VSC) It is an IDE made by **Microsoft** for **Windows**, **Linux** and **macOS**. Features include support for **debugging**, **syntax highlighting** for many different programming languages, intelligent **code completion** and embedded **version control system** Git. Users can change the theme, keyboard shortcuts, preferences, and install **extensions** from a huge repository that add additional functionality. Despite of its platform independence, VSC is **not open source** - in fact it is released under a traditional **Microsoft product license**.

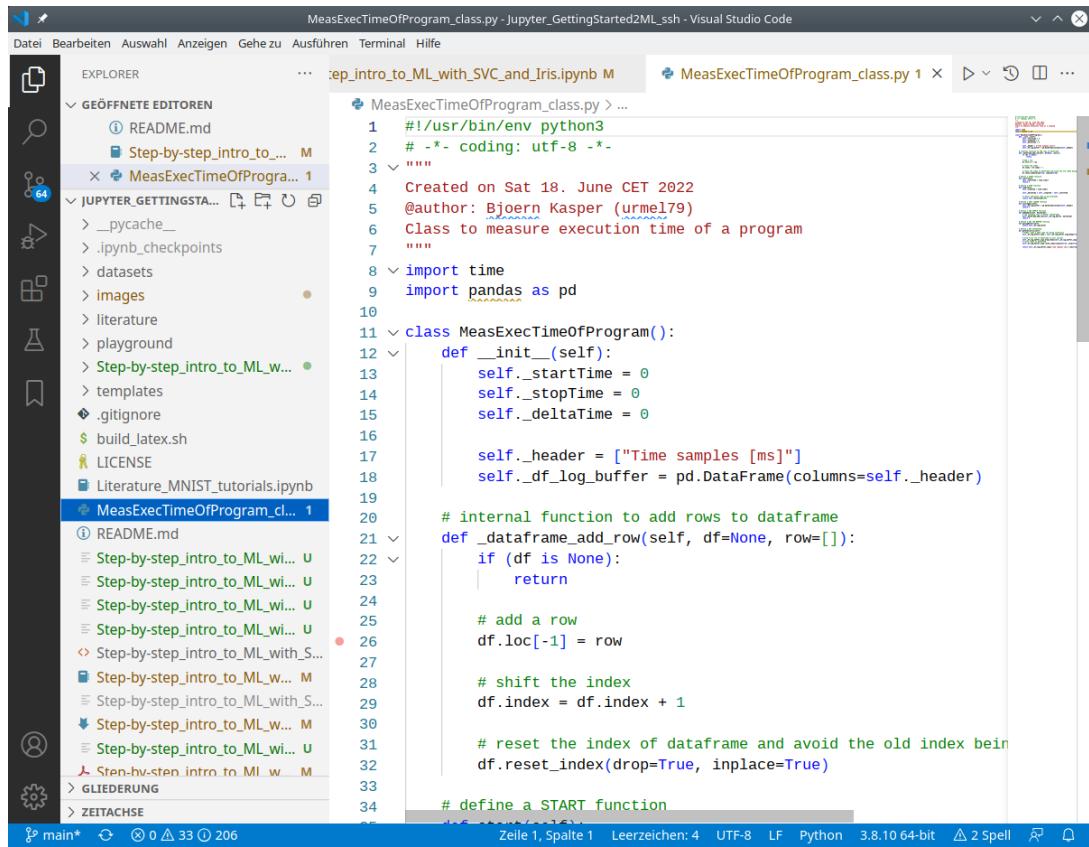


Figure 4: Screenshot of IDE *Visual Studio Code* (source: Kasper, license: CC BY-SA)

JupyterLab It is the successor product for the web-based interactive environment **Jupyter Notebook**. Within this IDE, Jupyter Notebook documents can be created, edited, and executed interactively. The notebooks consist of **input and output cells**, each of which can contain program code, formatted text in **Markdown** format and live plots generated from the code.

Jupyter is a new **open source** alternative to the proprietary numerical software **Mathematica** from **Wolfram Research** that is well on the way to becoming a **standard for exchanging research results** (Somers 2018; Romer 2018).

Originally Jupyter was intended as an IDE for the programming languages **Julia** and **Python**. Besides that it is also possible to install other interpreter kernels, such as the **IRkernel** for R. This can be interesting if the IDE **RStudio Desktop** is not available on the target platform used. For example, it is very difficult to install RStudio on the ARM-based embedded computer **Raspberry Pi** due to many technical dependencies. In contrast, using the R kernel in JupyterLab on the Raspberry Pi works very well and performant.

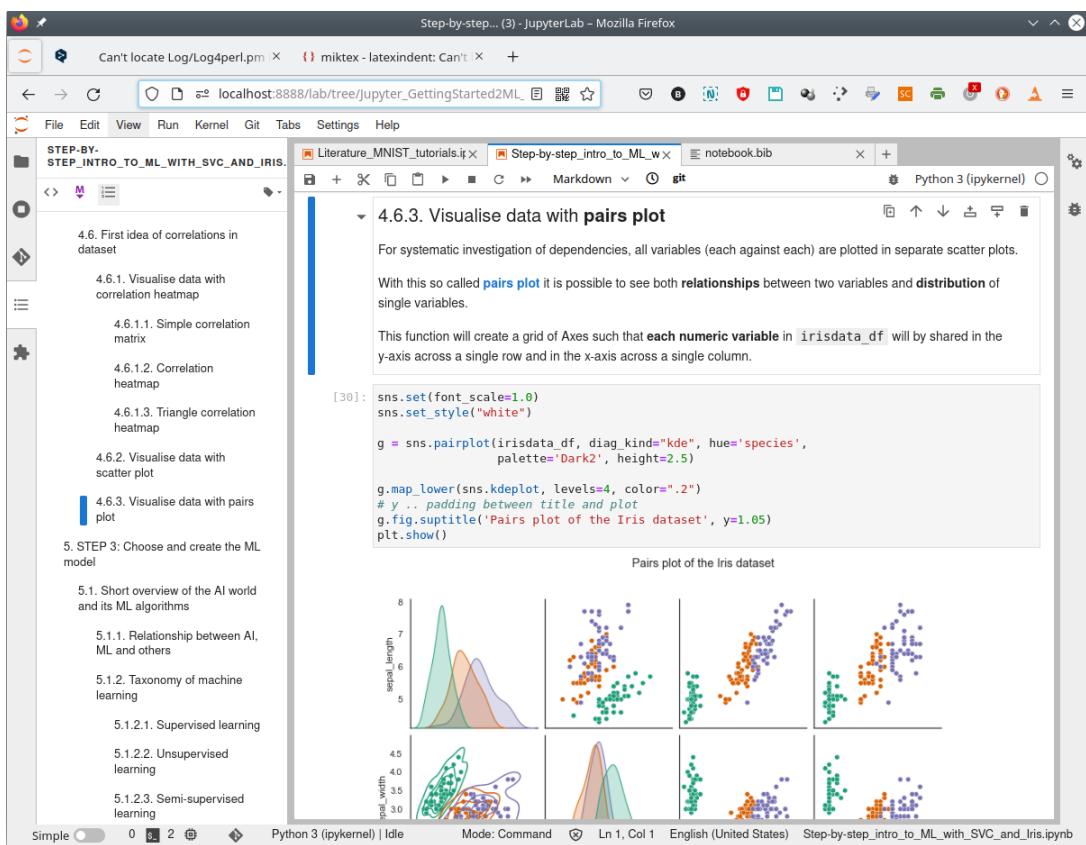
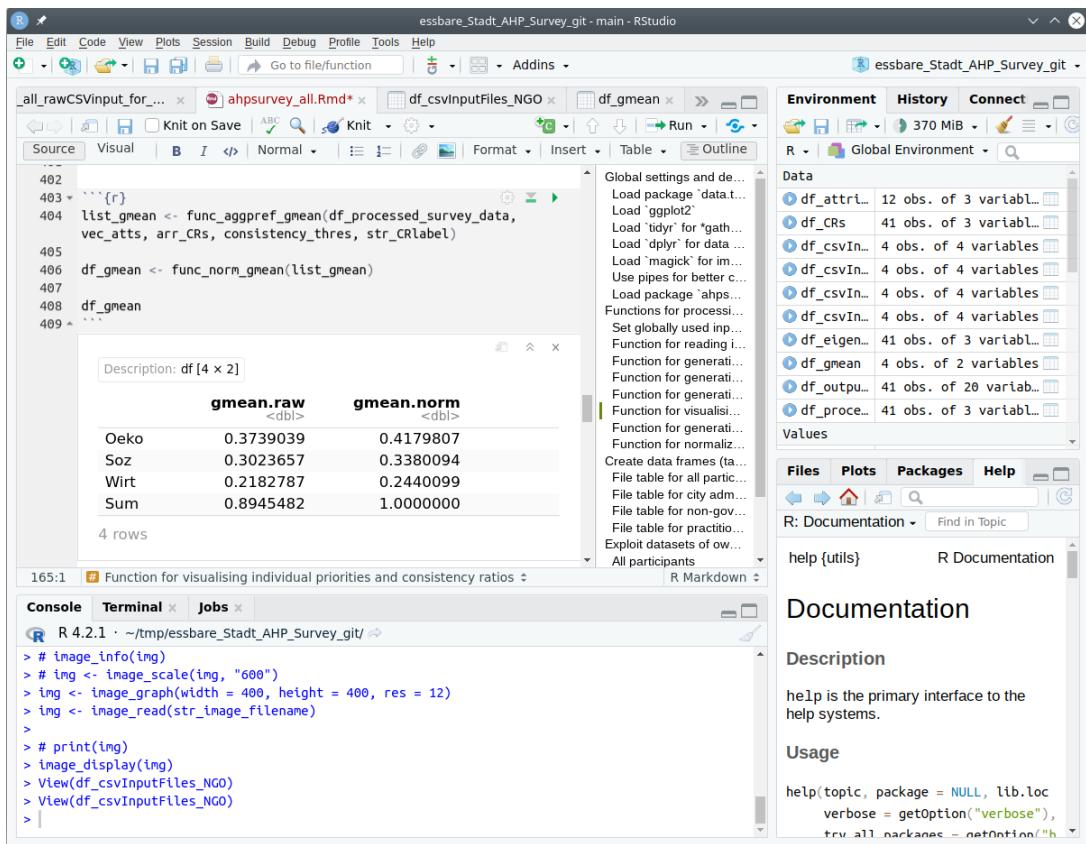


Figure 5: Screenshot of IDE *JupyterLab* (source: Kasper, license: CC BY-SA)

RStudio It is an IDE and graphical user interface for the statistical programming language **R** offered by **RStudio, Inc.** and is made available in two formats. **RStudio Desktop** is a regular desktop application while **RStudio Server** runs on a remote server and allows accessing RStudio using a web browser. Both software products are available in **open source** and **commercial** versions, each with different functionalities.

The program editor in RStudio allows **autocomplete**, **automatic indentation**, **syntax highlighting**, **code folding** as well as **integrated help** and information about functions and objects in the working environment. There is the ability to view and edit the contents of variables and datasets. To facilitate collaboration, scripts, data and other files can be combined into projects (.Rproj) and versioned with **Git**.

Figure 6: Screenshot of IDE *RStudio* (source: Kasper, license: CC BY-SA)

GNU Octave (GUI) It is the official graphical user interface for the **GNU Octave** programming language and is available for Windows, macOS, Linux and BSD under **Open Source** licensing.

If the command line interpreter (CLI) starts instead of the graphical user interface (GUI) when `octave` is called, this can be forced via the `octave --gui` option.

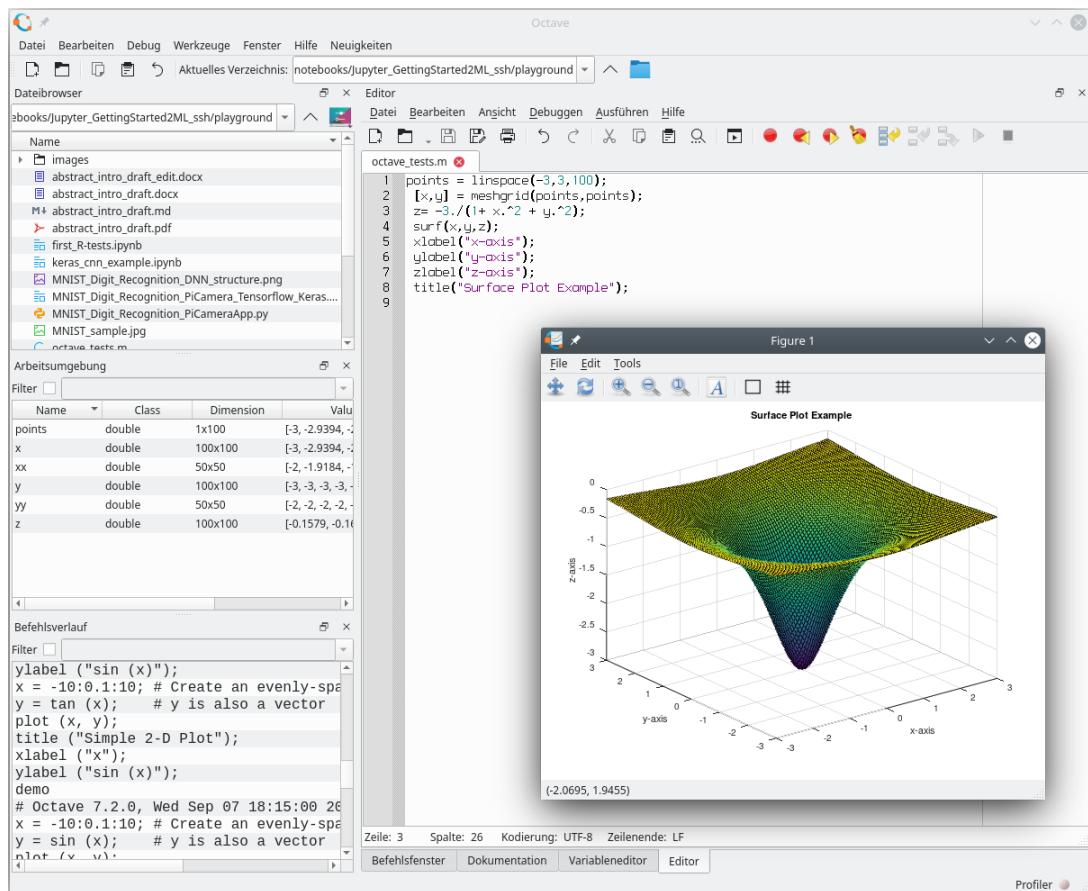


Figure 7: Screenshot of IDE *GNU Octave* (source: Kasper, license: CC BY-SA)

2.2.4 Operating systems

The **programming languages**, **Python libraries** and **development environments** presented in the previous sections are available for different operating systems, such as **Linux**, **Windows** and **macOS**. Therefore, the decision for or against an operating system may **depend on the technical background** of the ML developer.

Nevertheless, the following general **requirements** can be specified for an operating system **suitable for software development**:

- **Openness:** availability of very good interface documentation and ideally open source software
- **Self-administration:** user has full installation and configuration rights
- **Communication capability:** unfiltered and bidirectional communication in the local network as well as to the Internet on all necessary protocols possible
- **Extensibility:**
 - automated software installation and update management via central package management systems such as `apt`, `pip` or `conda`
 - possible integration of additional software libraries or external sensor hardware

Following trend chart shows how the **popularity of selected operating systems** used by **data analysts** and **ML developers** has evolved since 2008:

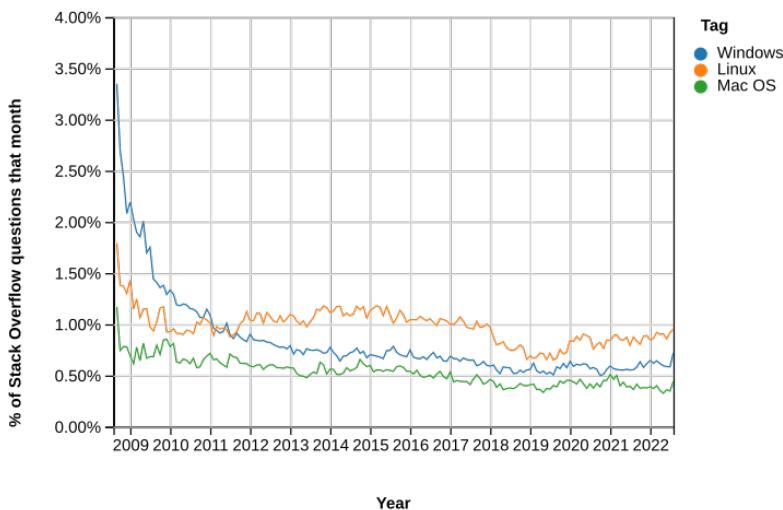


Figure 8: Trend chart shows popularity of selected operating systems used by **data analysts** and **ML developers** (source: [Stack Overflow Trends](#), license: CC BY-SA)

For security reasons, the **IT departments** of many employers massively **restrict installation and configuration rights**. Furthermore, very restrictive firewall settings severely **restrict** unfiltered and bidirectional **communication** in the local network and to the Internet. Automated **software installations** via package managers are often **not possible** or only possible with difficulty due to blocked protocols.

To deal with these challenges, two possible solutions are presented below.

Virtual machine To be able to install, configure and update the required software (IDEs, programming languages and ML packages) independently, the use of a **Virtual Machine (VM)** could be a possible alternative.

However, there are also significant disadvantages here:

- The **communication problem** is **not solved**, because the VM shares the access to the internet with the host system.
- The **access to 3D graphics cards** is usually **not possible** due to virtualization.
- This solution has only **low application performance**, as regular business computers are often only very sparsely equipped in terms of RAM and processor performance for cost reasons.

Separate lab computer All the problems mentioned in the previous section can only be solved satisfactorily by acquiring a **separate laboratory computer** with **its own internet access** (e.g. via an **LTE-capable Wifi router**).

This laboratory computer can be configured according to your own requirements, depending on the available budget in terms of hardware and software.

However, it should be noted here that the **IT departments** of many employers do **not offer any support** for this solution. You are usually responsible for software installation, maintenance and troubleshooting yourself!

2.3 Community Support

When selecting and deciding for or against the use of certain hardware and software components, in addition to purely technical or financial characteristics, significant attention should be paid to broad **support from a well-networked community**. This community should consist of a balanced share of **manufacturers** of hardware components (e.g. GPU suppliers, manufacturers of embedded systems or

sensors), **software developers** ideally from the **open source** ecosystem, and an active **user community** (e.g. for reporting hardware and software bugs or providing help in forums).

The author's many years of development experience show that the technically best hardware or software component is worthless if you are (apparently) the only user. This impression arises either because the component is actually very exotic and has only a few users or because the development takes place "behind closed doors", i.e. in the company's internal **closed source** domain.

Without the support of an active community, you are (almost) on your own when it comes to questions or problems. Progress in the development and maintenance of an AI application is therefore very difficult! The clear recommendation is therefore: Go for the (technically, price-wise, etc.) **second-best alternative** but with an even bigger **community**.

2.4 Import Python packages

The aim of this section is to import globally used Python packages for data analysis and ML, such as **Pandas**, **NumPY**, **matplotlib** and **Scikit-Learn**.

```
[2]: import time

from IPython.display import HTML

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, metrics
import seaborn as sns
%matplotlib inline
```

```
/home/bk/anaconda3/envs/PyLab/lib/python3.9/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version
of SciPy (detected version 1.23.1
    warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

3 STEP 1: Acquire the ML dataset

To allow an ML novice to first familiarize themselves with the ML algorithms, tools, libraries, and programming systems, the ready-made and very beginner-friendly **Iris dataset** is involved in this step. Only after a comprehensive acquaintance with the application of ML tools would it make sense to examine one's own environment for ML-suitable applications and to obtain suitable datasets from them. However, this is beyond the scope of this introductory tutorial.

Several details, for example, on the history of the creation of the **Iris flower dataset** can be found e.g. on Wikipedia (see Wikipedia: Iris dataset 2022).

It can be downloaded on [Kaggle: Iris Flower Dataset](#) (Srinagar 2018). Furthermore, the dataset is available via Python in the machine learning package **Scikit-learn**, so that users can access it without having to find a special source for it.

```
[3]: # import Iris dataset for exploration
irisdata_df = pd.read_csv('./datasets/IRIS_flower_dataset_kaggle.csv')
```

4 STEP 2: Explore the ML dataset

One of the most important steps in the entire ML process is this step, in which the dataset included in Step 1 is examined using typical data analysis tools. In addition to exploring the **data structure** and **internal correlations** in the dataset, errors such as **gaps**, **duplications**, or obvious **misentries** must

also be found and corrected where possible. This is enormously important so that the classification can later provide plausible results.

4.1 Goals of exploration

The objectives of the exploration of the dataset are as follows:

1. Clarify the **origins history**:
 - Where did the data come from? → Contact persons and licensing permissions?
 - Who obtained the data and with which (measurement) methods? → Did systematic errors occur during the acquisition?
 - What were they originally intended for? → Can they be used for my application?
2. Overview of the internal **structure and organisation** of the data:
 - Which columns are there? → With which methods can they be read in (e.g. import of CSV files)?
 - What do they contain for (physical) measured variables? → Which technical or physical correlations exist?
 - Which data formats or types are there? → Do they have to be converted?
 - In which value ranges do the measurement data vary? → Are normalizations necessary?
3. Identify **anomalies** in the datasets:
 - Do the data have **gaps** or **duplicates**? → Does the dataset need to be cleaned?
 - Are there obvious erroneous entries or measurement outliers? → Does (statistical) filtering have to be carried out?
4. Avoidance of **tendencies due to bias**:
 - Are all possible classes included in the dataset and equally distributed? → Does the dataset need to be enriched with additional data for balance?
5. Find a first rough **idea of which correlations** could be in the dataset

4.2 Clarify the origins history

The **Iris flower datasets** is a multivariate dataset introduced by the British statistician and biologist *Ronald Fisher* in his paper “[The use of multiple measurements in taxonomic problems](#)” (*Fisher 1936*). It is sometimes called *Anderson’s Iris dataset* because Edgar Anderson collected the data to quantify the morphologic variation of Iris flowers of three related species (Wikipedia: Iris dataset [2022](#)).

The dataset is published in Public Domain with a [CC0-License](#).

This dataset became a typical test case for many statistical classification techniques in machine learning such as **support vector machines**.

[..] measurements of the flowers of fifty plants each of the two species *Iris setosa* and *I. versicolor*, found **growing together in the same colony** and measured by Dr E. Anderson (*Fisher 1936*)

[..] *Iris virginica*, differs from the two other samples in **not being taken from the same natural colony** (*ibidem*)

4.3 Overview of the internal structure and organization of the data

The dataset consists of 50 samples from each of three species of Iris: *Iris setosa*, *Iris virginica* and *Iris versicolor*, so there are 150 samples in total (Wikipedia: Iris setosa [2022](#), Wikipedia: Iris virginica [2022](#) and Wikipedia: Iris versicolor [2022](#)).

Four features were measured from each sample: the length and the width of the **sepals** and **petals**, in centimetres (Wikipedia: Sepal [2022](#) and Wikipedia: Petal [2022](#)). Here you can see a principle illustration of a flower in which, among other things, the sepals and petals are shown:

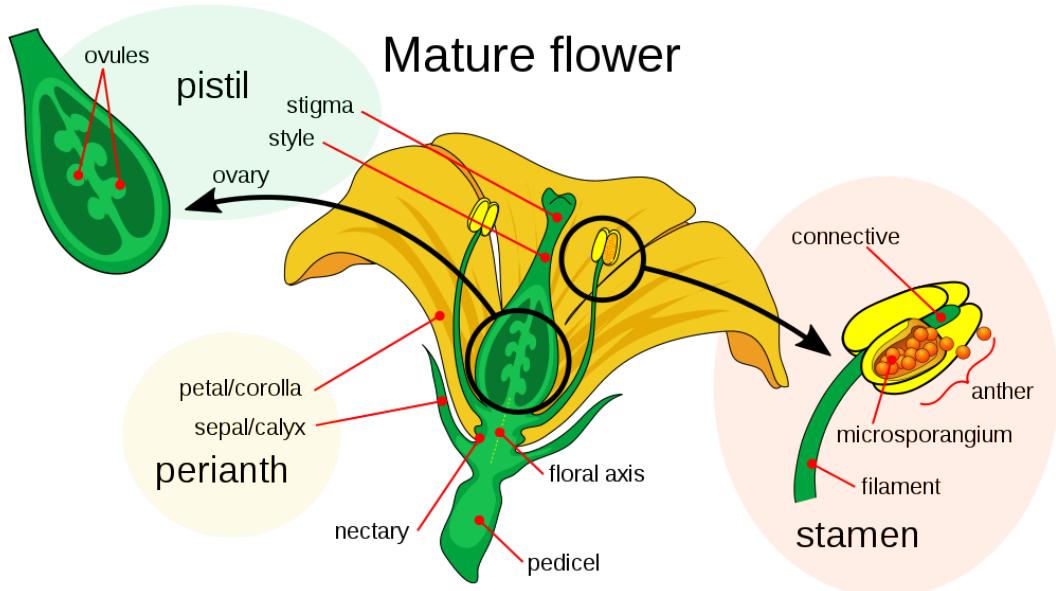


Figure 9: Principle illustration of a flower with sepals and petals (source: [Mature_flower_diagram.svg](#), license: public domain)

Here are pictures of the three different Iris species (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Given the dimensions of the flower, it will be possible to predict the class of the flower.



Figure 10: Left: *Iris setosa* (source: [Irissetosa1.jpg](#), license: public domain); middle: *Iris versicolor* (source: [Iris_versicolor_3.jpg](#), license: CC-SA 3.0); right: *Iris virginica* (source: [Iris_virginica.jpg](#), license: CC-SA 2.0)

4.3.1 Inspect structure of dataframe

Print first or last 10 rows of dataframe:

```
[4]: irisdata_df.head(10)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

```
[5]: irisdata_df.tail()
```

```
[5]:    sepal_length  sepal_width  petal_length  petal_width      species
145         6.7          3.0         5.2          2.3 Iris-virginica
146         6.3          2.5         5.0          1.9 Iris-virginica
147         6.5          3.0         5.2          2.0 Iris-virginica
148         6.2          3.4         5.4          2.3 Iris-virginica
149         5.9          3.0         5.1          1.8 Iris-virginica
```

While printing a dataframe - only an abbreviated view of the dataframe is shown :(
Default setting in the pandas library makes it to display only 5 lines from head and from tail.

```
[6]: irisdata_df
```

```
[6]:    sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5         1.4          0.2 Iris-setosa
1           4.9          3.0         1.4          0.2 Iris-setosa
2           4.7          3.2         1.3          0.2 Iris-setosa
3           4.6          3.1         1.5          0.2 Iris-setosa
4           5.0          3.6         1.4          0.2 Iris-setosa
...
145          6.7          3.0         5.2          2.3 Iris-virginica
146          6.3          2.5         5.0          1.9 Iris-virginica
147          6.5          3.0         5.2          2.0 Iris-virginica
148          6.2          3.4         5.4          2.3 Iris-virginica
149          5.9          3.0         5.1          1.8 Iris-virginica
```

[150 rows x 5 columns]

To print all rows of a dataframe, the option `display.max_rows` has to set to `None` in pandas:

```
[92]: pd.set_option('display.max_rows', None)
irisdata_df
```

```
[92]:    sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5         1.4          0.2 Iris-setosa
1           4.9          3.0         1.4          0.2 Iris-setosa
2           4.7          3.2         1.3          0.2 Iris-setosa
3           4.6          3.1         1.5          0.2 Iris-setosa
4           5.0          3.6         1.4          0.2 Iris-setosa
5           5.4          3.9         1.7          0.4 Iris-setosa
6           4.6          3.4         1.4          0.3 Iris-setosa
7           5.0          3.4         1.5          0.2 Iris-setosa
8           4.4          2.9         1.4          0.2 Iris-setosa
9           4.9          3.1         1.5          0.1 Iris-setosa
10          5.4          3.7         1.5          0.2 Iris-setosa
11          4.8          3.4         1.6          0.2 Iris-setosa
12          4.8          3.0         1.4          0.1 Iris-setosa
13          4.3          3.0         1.1          0.1 Iris-setosa
14          5.8          4.0         1.2          0.2 Iris-setosa
15          5.7          4.4         1.5          0.4 Iris-setosa
16          5.4          3.9         1.3          0.4 Iris-setosa
17          5.1          3.5         1.4          0.3 Iris-setosa
18          5.7          3.8         1.7          0.3 Iris-setosa
19          5.1          3.8         1.5          0.3 Iris-setosa
20          5.4          3.4         1.7          0.2 Iris-setosa
21          5.1          3.7         1.5          0.4 Iris-setosa
22          4.6          3.6         1.0          0.2 Iris-setosa
```

23	5.1	3.3	1.7	0.5	Iris-setosa
24	4.8	3.4	1.9	0.2	Iris-setosa
25	5.0	3.0	1.6	0.2	Iris-setosa
26	5.0	3.4	1.6	0.4	Iris-setosa
27	5.2	3.5	1.5	0.2	Iris-setosa
28	5.2	3.4	1.4	0.2	Iris-setosa
29	4.7	3.2	1.6	0.2	Iris-setosa
30	4.8	3.1	1.6	0.2	Iris-setosa
31	5.4	3.4	1.5	0.4	Iris-setosa
32	5.2	4.1	1.5	0.1	Iris-setosa
33	5.5	4.2	1.4	0.2	Iris-setosa
34	4.9	3.1	1.5	0.1	Iris-setosa
35	5.0	3.2	1.2	0.2	Iris-setosa
36	5.5	3.5	1.3	0.2	Iris-setosa
37	4.9	3.1	1.5	0.1	Iris-setosa
38	4.4	3.0	1.3	0.2	Iris-setosa
39	5.1	3.4	1.5	0.2	Iris-setosa
40	5.0	3.5	1.3	0.3	Iris-setosa
41	4.5	2.3	1.3	0.3	Iris-setosa
42	4.4	3.2	1.3	0.2	Iris-setosa
43	5.0	3.5	1.6	0.6	Iris-setosa
44	5.1	3.8	1.9	0.4	Iris-setosa
45	4.8	3.0	1.4	0.3	Iris-setosa
46	5.1	3.8	1.6	0.2	Iris-setosa
47	4.6	3.2	1.4	0.2	Iris-setosa
48	5.3	3.7	1.5	0.2	Iris-setosa
49	5.0	3.3	1.4	0.2	Iris-setosa
50	7.0	3.2	4.7	1.4	Iris-versicolor
51	6.4	3.2	4.5	1.5	Iris-versicolor
52	6.9	3.1	4.9	1.5	Iris-versicolor
53	5.5	2.3	4.0	1.3	Iris-versicolor
54	6.5	2.8	4.6	1.5	Iris-versicolor
55	5.7	2.8	4.5	1.3	Iris-versicolor
56	6.3	3.3	4.7	1.6	Iris-versicolor
57	4.9	2.4	3.3	1.0	Iris-versicolor
58	6.6	2.9	4.6	1.3	Iris-versicolor
59	5.2	2.7	3.9	1.4	Iris-versicolor
60	5.0	2.0	3.5	1.0	Iris-versicolor
61	5.9	3.0	4.2	1.5	Iris-versicolor
62	6.0	2.2	4.0	1.0	Iris-versicolor
63	6.1	2.9	4.7	1.4	Iris-versicolor
64	5.6	2.9	3.6	1.3	Iris-versicolor
65	6.7	3.1	4.4	1.4	Iris-versicolor
66	5.6	3.0	4.5	1.5	Iris-versicolor
67	5.8	2.7	4.1	1.0	Iris-versicolor
68	6.2	2.2	4.5	1.5	Iris-versicolor
69	5.6	2.5	3.9	1.1	Iris-versicolor
70	5.9	3.2	4.8	1.8	Iris-versicolor
71	6.1	2.8	4.0	1.3	Iris-versicolor
72	6.3	2.5	4.9	1.5	Iris-versicolor
73	6.1	2.8	4.7	1.2	Iris-versicolor
74	6.4	2.9	4.3	1.3	Iris-versicolor
75	6.6	3.0	4.4	1.4	Iris-versicolor
76	6.8	2.8	4.8	1.4	Iris-versicolor
77	6.7	3.0	5.0	1.7	Iris-versicolor
78	6.0	2.9	4.5	1.5	Iris-versicolor
79	5.7	2.6	3.5	1.0	Iris-versicolor

80	5.5	2.4	3.8	1.1	Iris-versicolor
81	5.5	2.4	3.7	1.0	Iris-versicolor
82	5.8	2.7	3.9	1.2	Iris-versicolor
83	6.0	2.7	5.1	1.6	Iris-versicolor
84	5.4	3.0	4.5	1.5	Iris-versicolor
85	6.0	3.4	4.5	1.6	Iris-versicolor
86	6.7	3.1	4.7	1.5	Iris-versicolor
87	6.3	2.3	4.4	1.3	Iris-versicolor
88	5.6	3.0	4.1	1.3	Iris-versicolor
89	5.5	2.5	4.0	1.3	Iris-versicolor
90	5.5	2.6	4.4	1.2	Iris-versicolor
91	6.1	3.0	4.6	1.4	Iris-versicolor
92	5.8	2.6	4.0	1.2	Iris-versicolor
93	5.0	2.3	3.3	1.0	Iris-versicolor
94	5.6	2.7	4.2	1.3	Iris-versicolor
95	5.7	3.0	4.2	1.2	Iris-versicolor
96	5.7	2.9	4.2	1.3	Iris-versicolor
97	6.2	2.9	4.3	1.3	Iris-versicolor
98	5.1	2.5	3.0	1.1	Iris-versicolor
99	5.7	2.8	4.1	1.3	Iris-versicolor
100	6.3	3.3	6.0	2.5	Iris-virginica
101	5.8	2.7	5.1	1.9	Iris-virginica
102	7.1	3.0	5.9	2.1	Iris-virginica
103	6.3	2.9	5.6	1.8	Iris-virginica
104	6.5	3.0	5.8	2.2	Iris-virginica
105	7.6	3.0	6.6	2.1	Iris-virginica
106	4.9	2.5	4.5	1.7	Iris-virginica
107	7.3	2.9	6.3	1.8	Iris-virginica
108	6.7	2.5	5.8	1.8	Iris-virginica
109	7.2	3.6	6.1	2.5	Iris-virginica
110	6.5	3.2	5.1	2.0	Iris-virginica
111	6.4	2.7	5.3	1.9	Iris-virginica
112	6.8	3.0	5.5	2.1	Iris-virginica
113	5.7	2.5	5.0	2.0	Iris-virginica
114	5.8	2.8	5.1	2.4	Iris-virginica
115	6.4	3.2	5.3	2.3	Iris-virginica
116	6.5	3.0	5.5	1.8	Iris-virginica
117	7.7	3.8	6.7	2.2	Iris-virginica
118	7.7	2.6	6.9	2.3	Iris-virginica
119	6.0	2.2	5.0	1.5	Iris-virginica
120	6.9	3.2	5.7	2.3	Iris-virginica
121	5.6	2.8	4.9	2.0	Iris-virginica
122	7.7	2.8	6.7	2.0	Iris-virginica
123	6.3	2.7	4.9	1.8	Iris-virginica
124	6.7	3.3	5.7	2.1	Iris-virginica
125	7.2	3.2	6.0	1.8	Iris-virginica
126	6.2	2.8	4.8	1.8	Iris-virginica
127	6.1	3.0	4.9	1.8	Iris-virginica
128	6.4	2.8	5.6	2.1	Iris-virginica
129	7.2	3.0	5.8	1.6	Iris-virginica
130	7.4	2.8	6.1	1.9	Iris-virginica
131	7.9	3.8	6.4	2.0	Iris-virginica
132	6.4	2.8	5.6	2.2	Iris-virginica
133	6.3	2.8	5.1	1.5	Iris-virginica
134	6.1	2.6	5.6	1.4	Iris-virginica
135	7.7	3.0	6.1	2.3	Iris-virginica
136	6.3	3.4	5.6	2.4	Iris-virginica

137	6.4	3.1	5.5	1.8	Iris-virginica
138	6.0	3.0	4.8	1.8	Iris-virginica
139	6.9	3.1	5.4	2.1	Iris-virginica
140	6.7	3.1	5.6	2.4	Iris-virginica
141	6.9	3.1	5.1	2.3	Iris-virginica
142	5.8	2.7	5.1	1.9	Iris-virginica
143	6.8	3.2	5.9	2.3	Iris-virginica
144	6.7	3.3	5.7	2.5	Iris-virginica
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

4.3.2 Get data types

[96]: `irisdata_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   sepal_length  150 non-null   float64 
 1   sepal_width   150 non-null   float64 
 2   petal_length  150 non-null   float64 
 3   petal_width   150 non-null   float64 
 4   species       150 non-null   object  
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

[97]: `irisdata_df.describe()`

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

4.3.3 Get data ranges and distribution

Histograms Histograms are useful to explore the frequency distribution for each feature in univariate plots:

[98]:

```
sns.set_context("notebook", font_scale=1.3, rc={"lines.linewidth": 2.0})
sns.set_style("whitegrid")

n_bins = 10
fig, axs = plt.subplots(2, 2, figsize=(8, 6))

axs[0,0].hist(irisdata_df['sepal_length'], bins = n_bins);
axs[0,0].set_title('Sepal Length');
```

```

axs[0,1].hist(irisdata_df['sepal_width'], bins = n_bins);
axs[0,1].set_title('Sepal Width');

axs[1,0].hist(irisdata_df['petal_length'], bins = n_bins);
axs[1,0].set_title('Petal Length');

axs[1,1].hist(irisdata_df['petal_width'], bins = n_bins);
axs[1,1].set_title('Petal Width');

# add some spacing between subplots
fig.tight_layout(pad=2.0);

```

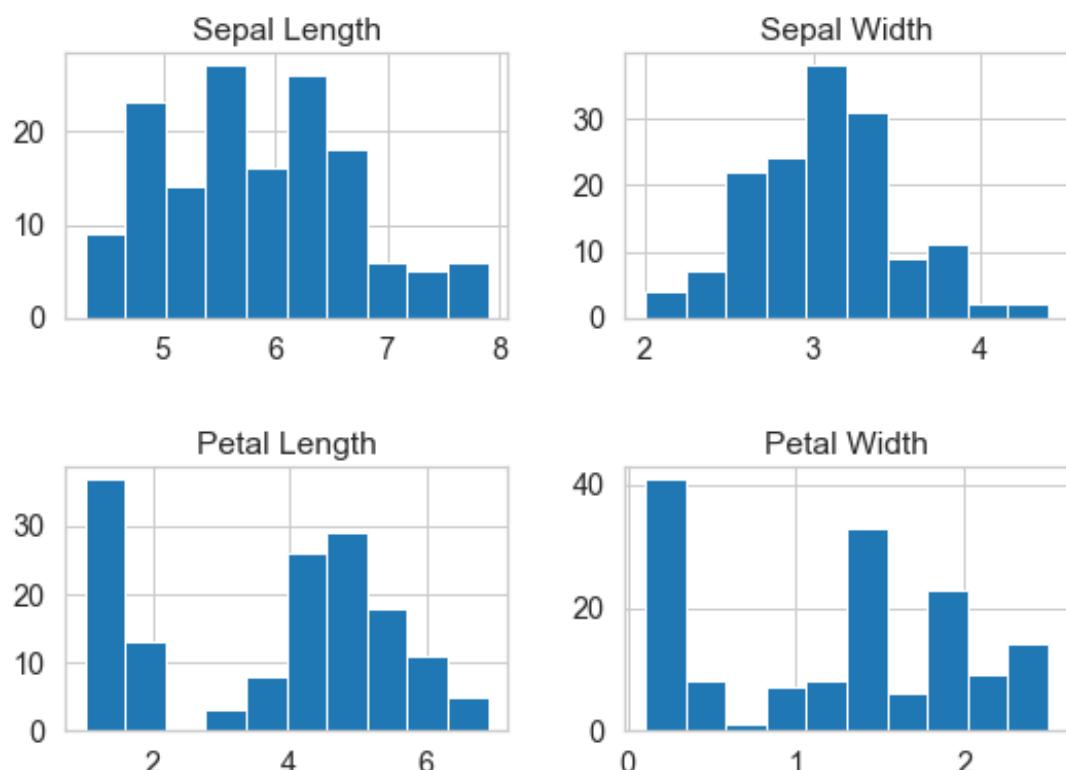


Figure 11: Histograms used to explore the frequency distribution of the 4 features in the Iris dataset

Boxplots Boxplots can be used to explore the **data ranges** in the dataset. These also provide information about **outliers**.

In the following code example, the 4 variables of the Iris dataset are displayed side-by-side in individual boxplots:

```

[99]: sns.set_context("notebook", font_scale=1.3, rc={"lines.linewidth": 2.0})
sns.set_style("whitegrid")
#sns.set_style("white")

fig, axs = plt.subplots(2, 2, figsize=(12, 10))

cn = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

# x, y: names of variables in data or vector data
# data: dataset for plotting

```

```
# order: order to plot the categorical levels in
# ax: assignment of the plot to the matplotlib subplot
box1 = sns.boxplot(x = 'species', y = 'sepal_length',
                    data = irisdata_df, order = cn, ax = axs[0,0])
box2 = sns.boxplot(x = 'species', y = 'sepal_width',
                    data = irisdata_df, order = cn, ax = axs[0,1])
box3 = sns.boxplot(x = 'species', y = 'petal_length',
                    data = irisdata_df, order = cn, ax = axs[1,0])
box4 = sns.boxplot(x = 'species', y = 'petal_width',
                    data = irisdata_df, order = cn, ax = axs[1,1])

# add some spacing between subplots
fig.tight_layout(pad=2.0)

plt.show()
```

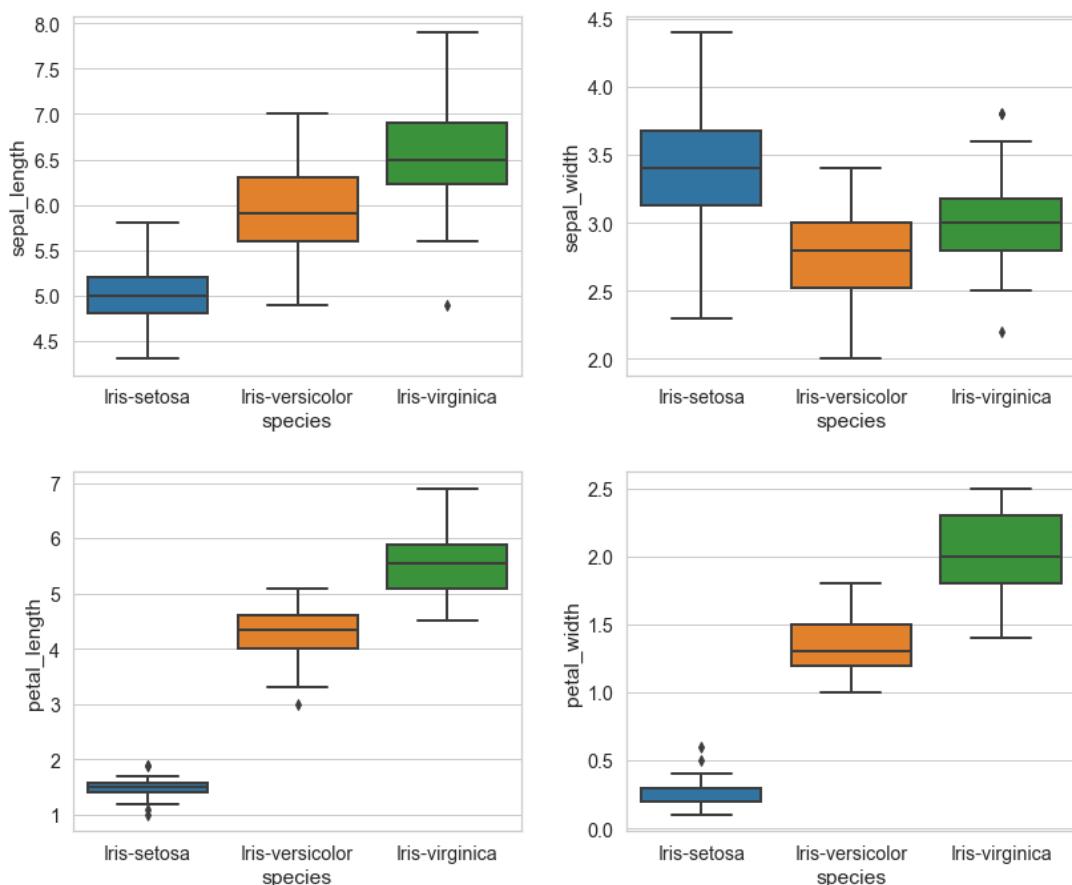


Figure 12: Boxplots used to explore the data ranges in the Iris dataset

Violin plots Another type of visualization is the **violin plot**, which **combines** the advantages of both the **histogram** and the **box plot**:

```
[100]: sns.set_context("notebook", font_scale=1.3, rc={"lines.linewidth": 2.0})
sns.set_style("whitegrid")

fig, axs = plt.subplots(2, 2, figsize=(12, 10))

cn = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

```

violin1 = sns.violinplot(x='species', y='sepal_length',
                         data=irisdata_df, order = cn, ax = axs[0,0])
violin2 = sns.violinplot(x='species', y='sepal_width',
                         data=irisdata_df, order = cn, ax = axs[0,1])
violin3 = sns.violinplot(x='species', y='petal_length',
                         data=irisdata_df, order = cn, ax = axs[1,0])
violin4 = sns.violinplot(x='species', y='petal_width',
                         data=irisdata_df, order = cn, ax = axs[1,1])
# add some spacing between subplots
fig.tight_layout(pad=2.0)

plt.show()

```

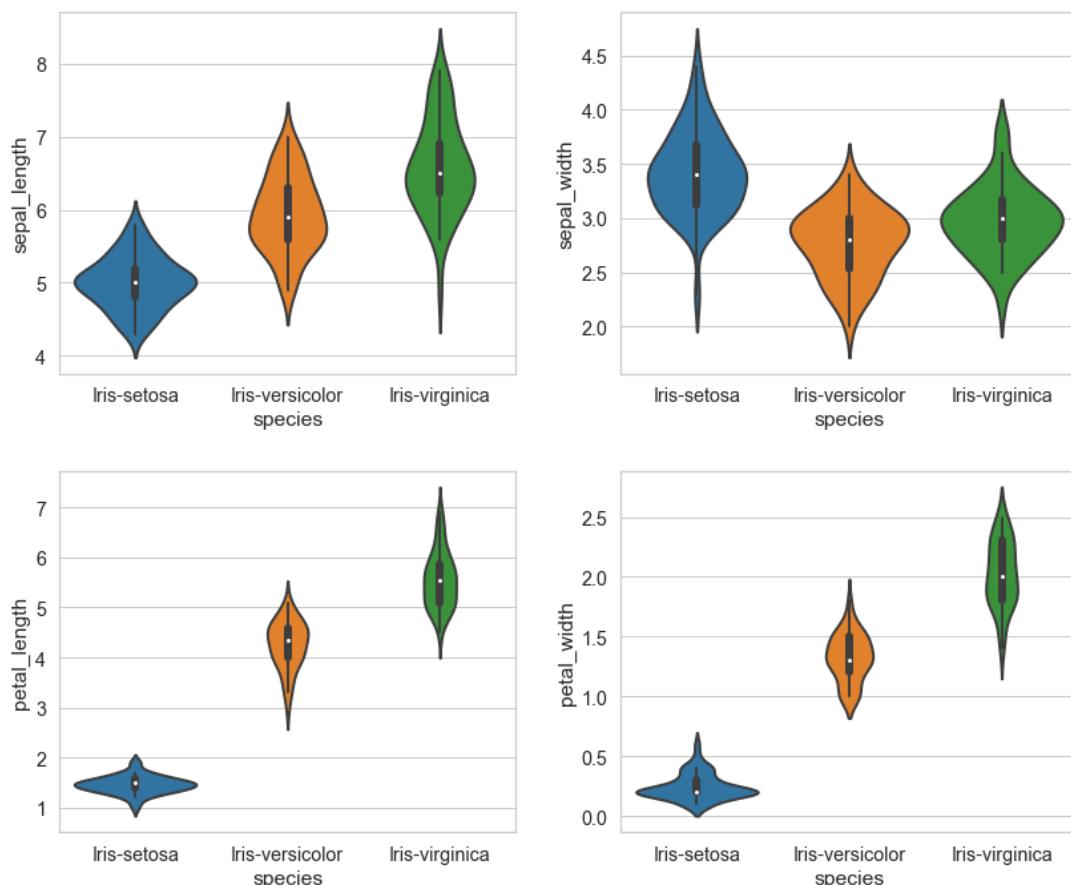


Figure 13: Violin plots combine histograms and box plots

4.4 Identify anomalies in the datasets

4.4.1 Find and repair gaps in dataset

This section was inspired by [Working with Missing Data in Pandas](#).

Check for missing values using `isnull()` In order to check for missing values in Pandas DataFrame, we use the function `isnull()`. This function returns a dataframe of Boolean values which are True for NaN values.

```
[101]: pd.set_option('display.max_rows', 40)
pd.set_option('display.min_rows', 30)
```

```
[102]: irisdata_df.isnull()
```

```
[102]:    sepal_length  sepal_width  petal_length  petal_width  species
 0        False       False       False       False       False
 1        False       False       False       False       False
 2        False       False       False       False       False
 3        False       False       False       False       False
 4        False       False       False       False       False
 5        False       False       False       False       False
 6        False       False       False       False       False
 7        False       False       False       False       False
 8        False       False       False       False       False
 9        False       False       False       False       False
 10       False      False       False       False       False
 11       False      False       False       False       False
 12       False      False       False       False       False
 13       False      False       False       False       False
 14       False      False       False       False       False
 ..
 ..
 135      False      False       False       False       False
 136      False      False       False       False       False
 137      False      False       False       False       False
 138      False      False       False       False       False
 139      False      False       False       False       False
 140      False      False       False       False       False
 141      False      False       False       False       False
 142      False      False       False       False       False
 143      False      False       False       False       False
 144      False      False       False       False       False
 145      False      False       False       False       False
 146      False      False       False       False       False
 147      False      False       False       False       False
 148      False      False       False       False       False
 149      False      False       False       False       False
```

[150 rows x 5 columns]

Show only the gaps:

```
[103]: irisdata_df_gaps = irisdata_df[irisdata_df.isnull().any(axis=1)]
irisdata_df_gaps
```

```
[103]: Empty DataFrame
Columns: [sepal_length, sepal_width, petal_length, petal_width, species]
Index: []
```

Fine - this dataset seems to be complete :)

So let's look for something else for exercise: [employees.csv](#)

```
[145]: # import data to dataframe from csv file
employees_df = pd.read_csv("./datasets/employees_edit.csv")

# highlight cells with NaN values
# HINT: Set to 'False' when compiling to PDF!
highlight = False

employees_df_hl = employees_df
```

```
if highlight:
    employees_df_hl = employees_df.style.highlight_null('yellow')

employees_df_hl
```

[145]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	\
0	Douglas	Male	8/6/1993	12:42 PM	97308	6945.00	
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.17	
2	Maria	Female	4/23/1993	11:17 AM	130590	11858.00	
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.34	
4	Larry	Male	1/24/1998	4:47 PM	101004	1389.00	
5	Dennis	Male	4/18/1987	1:35 AM	115163	10125.00	
6	Ruby	Female	8/17/1987	4:20 PM	65476	10012.00	
7	NaN	Female	7/20/2015	10:43 AM	45906	11598.00	
8	Angela	Female	11/22/2005	6:29 AM	95570	18523.00	
9	Frances	Female	8/8/2002	6:51 AM	139852	7524.00	
10	Louise	Female	8/12/1980	9:01 AM	63241	15132.00	
11	Julie	Female	10/26/1997	3:19 PM	102508	12637.00	
12	Brandon	Male	12/1/1980	1:08 AM	112807	17492.00	
13	Gary	Male	1/27/2008	11:40 PM	109831	5831.00	
14	Kimberly	Female	1/14/1999	7:13 AM	41426	14543.00	
...	
989	Stephen	NaN	7/10/1983	8:10 PM	85668	1909.00	
990	Donna	Female	11/26/1982	7:04 AM	82871	17999.00	
991	Gloria	Female	12/8/2014	5:08 AM	136709	10331.00	
992	Alice	Female	10/5/2004	9:34 AM	47638	11209.00	
993	Justin	NaN	2/10/1991	4:58 PM	38344	3794.00	
994	Robin	Female	7/24/1987	1:35 PM	100765	10982.00	
995	Rose	Female	8/25/2002	5:12 AM	134505	11051.00	
996	Anthony	Male	10/16/2011	8:35 AM	112769	11625.00	
997	Tina	Female	5/15/1997	3:53 PM	56450	19.04	
998	George	Male	6/21/2013	5:47 PM	98874	4479.00	
999	Henry	NaN	11/23/2014	6:09 AM	132483	16655.00	
1000	Phillip	Male	1/31/1984	6:30 AM	42392	19675.00	
1001	Russell	Male	5/20/2013	12:39 PM	96914	1421.00	
1002	Larry	Male	4/20/2013	4:45 PM	60500	11985.00	
1003	Albert	Male	5/15/2012	6:24 PM	129949	10169.00	

	Senior Management		Team
0	True		Marketing
1	True		NaN
2	False		Finance
3	True		Finance
4	True	Client Services	
5	False		Legal
6	True		Product
7	NaN		Finance
8	True		Engineering
9	True	Business Development	
10	True		NaN
11	True		Legal
12	True	Human Resources	
13	False		Sales
14	True		Finance
...	
989	False		Legal

```

990          False      Marketing
991          True       Finance
992         False   Human Resources
993         False        Legal
994         True    Client Services
995         True      Marketing
996         True       Finance
997         True    Engineering
998         True      Marketing
999         False   Distribution
1000        False       Finance
1001        False        Product
1002        False Business Development
1003        True        Sales

```

[1004 rows x 8 columns]

Show only the gaps from this gappy dataset again:

```

[146]: employees_df_gaps = employees_df[employees_df.isnull().any(axis=1)]

# highlight cells with NaN values
# HINT: Set to 'False' when compiling to PDF!
highlight = False

if highlight:
    employees_df_gaps = employees_df_gaps.style.highlight_null('yellow')

employees_df_gaps

```

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	\
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.17	
7	NaN	Female	7/20/2015	10:43 AM	45906	11598.00	
10	Louise	Female	8/12/1980	9:01 AM	63241	15132.00	
20	Lois	NaN	4/22/1995	7:18 PM	64714	4934.00	
22	Joshua	NaN	3/8/2012	1:58 AM	90816	18816.00	
23	NaN	Male	6/14/2012	4:19 PM	125792	5042.00	
25	NaN	Male	10/8/2012	1:12 AM	37076	18576.00	
27	Scott	NaN	7/11/1991	6:58 PM	122367	5218.00	
31	Joyce	NaN	2/20/2005	2:40 PM	88657	12752.00	
32	NaN	Male	8/21/1998	2:27 PM	122340	6417.00	
39	NaN	Male	1/29/2016	2:33 AM	122173	7797.00	
41	Christine	NaN	6/28/2015	1:08 AM	66582	11308.00	
49	Chris	NaN	1/24/1980	12:13 PM	113590	3055.00	
51	NaN	NaN	12/17/2011	8:29 AM	41126	14009.00	
53	Alan	NaN	3/3/2014	1:28 PM	40341	17578.00	
..	
916	Joe	Male	12/8/1998	10:28 AM	126120	1.02	
927	Irene	NaN	2/28/1991	10:23 PM	135369	4.38	
929	NaN	Female	8/23/2000	4:19 PM	95866	19388.00	
941	Aaron	NaN	1/22/1986	7:39 PM	63126	18424.00	
942	Mark	NaN	9/9/2006	12:27 PM	44836	2657.00	
943	Ralph	NaN	7/28/1995	6:53 PM	70635	2147.00	
949	Gerald	NaN	4/15/1989	12:44 PM	93712	17426.00	
950	NaN	Female	9/15/1985	1:50 AM	133472	16941.00	
951	NaN	Male	7/30/2012	3:07 PM	107351	5329.00	
955	NaN	Female	9/14/2010	5:19 AM	143638	9662.00	
965	Antonio	NaN	6/18/1989	9:37 PM	103050	3.05	

```

976    Victor      NaN  7/28/2006        2:49 PM  76381  11159.00
989    Stephen     NaN  7/10/1983       8:10 PM  85668  1909.00
993    Justin      NaN  2/10/1991      4:58 PM  38344  3794.00
999    Henry       NaN  11/23/2014     6:09 AM  132483  16655.00

   Senior Management          Team
1            True           NaN
7            NaN          Finance
10           True           NaN
20           True          Legal
22           True  Client Services
23            NaN           NaN
25            NaN  Client Services
27          False          Legal
31          False          Product
32            NaN           NaN
39            NaN  Client Services
41           True  Business Development
49          False          Sales
51            NaN           Sales
53           True          Finance
..
..            ...
916          False           NaN
927          False  Business Development
929            NaN           Sales
941          False  Client Services
942          False  Client Services
943          False  Client Services
949           True  Distribution
950            NaN  Distribution
951            NaN  Marketing
955            NaN           NaN
965          False          Legal
976           True          Sales
989          False          Legal
993          False          Legal
999          False  Distribution

```

[237 rows x 8 columns]

Fill in missing string values with `fillna()` Now all null values (NaN) in the column “Gender” of the data type String are filled with “No gender”.

Warning: We are doing that directly in this dataframe with `inplace = True` - we don't make a deep copy!

```

[147]: # filling a null values using fillna()
employees_df["Gender"].fillna("No Gender", inplace = True)

# switch to apply highlight style to dataframe
# HINT: Set to 'False' when compiling to PDF!
highlight = False

employees_df_filled = employees_df

if highlight:
    # highlight cells by condition

```

```
employees_df_filled = employees_df.style.apply(lambda x:
                                                ["background: yellow"
                                                 if v == 'No Gender'
                                                 else "" for v in x],
                                                axis = 1)
```

```
employees_df_filled
```

```
[147]:   First Name    Gender Start Date Last Login Time    Salary    Bonus % \
0      Douglas     Male  8/6/1993  12:42 PM    97308  6945.00
1      Thomas     Male  3/31/1996   6:53 AM    61933   4.17
2      Maria   Female  4/23/1993  11:17 AM   130590 11858.00
3      Jerry     Male  3/4/2005   1:00 PM   138705   9.34
4      Larry     Male  1/24/1998  4:47 PM   101004 1389.00
5      Dennis     Male  4/18/1987  1:35 AM   115163 10125.00
6      Ruby   Female  8/17/1987  4:20 PM    65476 10012.00
7       NaN   Female  7/20/2015 10:43 AM    45906 11598.00
8      Angela  Female 11/22/2005  6:29 AM    95570 18523.00
9      Frances  Female  8/8/2002  6:51 AM   139852  7524.00
10     Louise  Female  8/12/1980  9:01 AM    63241 15132.00
11     Julie   Female 10/26/1997  3:19 PM   102508 12637.00
12     Brandon     Male 12/1/1980  1:08 AM   112807 17492.00
13      Gary     Male  1/27/2008 11:40 PM   109831  5831.00
14    Kimberly  Female  1/14/1999  7:13 AM    41426 14543.00
...
989     ...     ...
990     Stephen No Gender 7/10/1983  8:10 PM    85668 1909.00
990     Donna   Female 11/26/1982  7:04 AM   82871 17999.00
991     Gloria  Female 12/8/2014  5:08 AM   136709 10331.00
992     Alice   Female 10/5/2004  9:34 AM   47638 11209.00
993     Justin  No Gender 2/10/1991  4:58 PM   38344 3794.00
994     Robin   Female  7/24/1987  1:35 PM   100765 10982.00
995     Rose   Female  8/25/2002  5:12 AM   134505 11051.00
996     Anthony     Male 10/16/2011  8:35 AM   112769 11625.00
997     Tina   Female  5/15/1997  3:53 PM   56450   19.04
998     George     Male  6/21/2013  5:47 PM   98874 4479.00
999     Henry  No Gender 11/23/2014  6:09 AM   132483 16655.00
1000    Phillip     Male 1/31/1984  6:30 AM   42392 19675.00
1001    Russell     Male  5/20/2013 12:39 PM   96914 1421.00
1002    Larry   Male  4/20/2013  4:45 PM   60500 11985.00
1003    Albert     Male  5/15/2012  6:24 PM 129949 10169.00
```

	Senior Management	Team
0	True	Marketing
1	True	NaN
2	False	Finance
3	True	Finance
4	True	Client Services
5	False	Legal
6	True	Product
7	NaN	Finance
8	True	Engineering
9	True	Business Development
10	True	NaN
11	True	Legal
12	True	Human Resources
13	False	Sales
14	True	Finance

```

...
989      False          Legal
990      False        Marketing
991      True          Finance
992      False   Human Resources
993      False          Legal
994      True  Client Services
995      True        Marketing
996      True          Finance
997      True      Engineering
998      True        Marketing
999      False    Distribution
1000     False          Finance
1001     False          Product
1002     False  Business Development
1003      True          Sales

```

[1004 rows x 8 columns]

Fill in missing *numerical* values with median values Missing integer or float values can be filled with the **median values of the corresponding column**.

@TODO:

Incorporate section “4.1.3 Fehlende Werte ergänzen” of the book `mitp_Praxishandbuch_Machine_Learning_Python_Scikit-learn_TensorFlow_2018_AnM_bk.pdf` (see Raschka and Mirjalili 2018).

- <https://www.statology.org/pandas-fillna-with-median/>
- <https://stackoverflow.com/questions/18689823/pandas-dataframe-replace-nan-values-with-average-of-columns>

Drop missing values using `dropna()` In order to drop null values from a dataframe, we use `dropna()` function. This function drops rows or columns of datasets with NaN values in different ways.

Default is to drop rows with at least 1 null value (NaN). Giving the parameter `how = 'all'` the function drops rows with all data missing or contain null values (NaN).

[11]: # making a new dataframe with dropped NaN values
`employees_df_dropped = employees_df.dropna(axis = 0, how = 'any')`

[11]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	\
0	Douglas	Male	8/6/1993	12:42 PM	97308	6945.00	
2	Maria	Female	4/23/1993	11:17 AM	130590	11858.00	
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.34	
4	Larry	Male	1/24/1998	4:47 PM	101004	1389.00	
5	Dennis	Male	4/18/1987	1:35 AM	115163	10125.00	
...	
999	Henry	No Gender	11/23/2014	6:09 AM	132483	16655.00	
1000	Phillip	Male	1/31/1984	6:30 AM	42392	19675.00	
1001	Russell	Male	5/20/2013	12:39 PM	96914	1421.00	
1002	Larry	Male	4/20/2013	4:45 PM	60500	11985.00	
1003	Albert	Male	5/15/2012	6:24 PM	129949	10169.00	

	Senior Management	Team
0	True	Marketing
2	False	Finance

```

3          True      Finance
4          True    Client Services
5         False       Legal
...
999        ...       ...
1000       False      Finance
1001       False      Product
1002      False  Business Development
1003       True       Sales

[903 rows x 8 columns]

```

Finally we compare the sizes of dataframes so that we learn how many rows had at least 1 Null value.

```
[17]: print("Old data frame length:", len(employees_df))
print("New data frame length:", len(employees_df_dropped))
print("Number of rows with at least 1 NaN value: ",
      (len(employees_df)-len(employees_df_dropped)))
```

```

Old data frame length: 1004
New data frame length: 903
Number of rows with at least 1 NaN value: 101

```

4.4.2 Find and remove duplicates in dataset

This section was inspired by: - [How to Find Duplicates in Pandas DataFrame \(With Examples\)](#) - [How to Drop Duplicate Rows in a Pandas DataFrame](#)

Check for duplicate values using duplicated() In order to check for duplicate values in Pandas DataFrame, we use a function `duplicated()`. This function can be used in two ways: - find duplicate rows across all **columns** with `duplicateRows = df[df.duplicated()]` - find duplicate rows across **specific columns** `duplicateRows = df[df.duplicated(subset=['col1', 'col2'])]`

Find duplicate rows across **all columns**:

```
[12]: # import (again) data to dataframe from csv file
employees_df = pd.read_csv("./datasets/employees_edit.csv")
```

```
[13]: # find duplicate rows across all columns
duplicateRows = employees_df[employees_df.duplicated()]
duplicateRows
```

```
[13]:   First Name Gender Start Date Last Login Time Salary Bonus % \
112      Karen Female  11/30/1999      7:46 AM 102488 17653.0
127      Linda Female  5/25/2000       5:45 PM 119009 12506.0
296     Brandon    NaN  11/3/1997      8:17 PM 121333 15295.0
580    Nicholas   Male  3/1/2013      9:26 PM 101036 2826.0
```

	Senior Management		Team
112	True		Product
127	True	Business Development	
296	False	Business Development	
580	True	Human Resources	

```
[14]: # argument keep='last' displays the first duplicate rows instead of the last
duplicateRows = employees_df[employees_df.duplicated(keep='last')]
duplicateRows
```

```
[14]:   First Name Gender Start Date Last Login Time Salary Bonus % \
 55      Karen Female 11/30/1999      7:46 AM 102488 17653.0
 92      Linda Female 5/25/2000      5:45 PM 119009 12506.0
153     Brandon    NaN 11/3/1997      8:17 PM 121333 15295.0
442    Nicholas   Male 3/1/2013      9:26 PM 101036 2826.0
```

	Senior Management	Team
55	True	Product
92	True	Business Development
153	False	Business Development
442	True	Human Resources

Find duplicate rows across **specific columns**:

```
[15]: # identify duplicate rows across 'First Name' and 'Last Login Time' columns
duplicateRows = employees_df[employees_df.duplicated(
    subset=['First Name', 'Last Login Time'])]
duplicateRows
```

```
[15]:   First Name Gender Start Date Last Login Time Salary Bonus % \
 112      Karen Female 11/30/1999      7:46 AM 102488 17653.0
 127      Linda Female 5/25/2000      5:45 PM 119009 12506.0
 296     Brandon    NaN 11/3/1997      8:17 PM 121333 15295.0
 577      NaN Female 1/13/2009      1:01 PM 118736 7421.0
 580    Nicholas   Male 3/1/2013      9:26 PM 101036 2826.0
 632      NaN    NaN 9/2/1988      12:49 PM 147309 1702.0
 881      NaN   Male 9/5/1980      7:36 AM 114896 13823.0
 929      NaN Female 8/23/2000      4:19 PM 95866 19388.0
 934     Nancy Female 9/10/2001      11:57 PM 85213 2386.0
 973      Linda Female 2/4/2010      8:49 PM 44486 17308.0
```

	Senior Management	Team
112	True	Product
127	True	Business Development
296	False	Business Development
577	NaN	Client Services
580	True	Human Resources
632	NaN	Distribution
881	NaN	Client Services
929	NaN	Sales
934	True	Marketing
973	True	Engineering

```
[16]: # argument keep='last' displays the first duplicate rows instead of the last
duplicateRows = employees_df[employees_df.duplicated(
    subset=['First Name', 'Last Login Time'], keep='last')]
duplicateRows
```

```
[16]:   First Name Gender Start Date Last Login Time Salary Bonus % \
 23      NaN   Male 6/14/2012      4:19 PM 125792 5042.00
 37      Linda Female 10/19/1981      8:49 PM 57427 9557.00
 55      Karen Female 11/30/1999      7:46 AM 102488 17653.00
 66      Nancy Female 12/15/2012      11:57 PM 125250 2672.00
 92      Linda Female 5/25/2000      5:45 PM 119009 12506.00
153     Brandon    NaN 11/3/1997      8:17 PM 121333 15295.00
222      NaN Female 6/17/1991      12:49 PM 71945      5.56
269      NaN   Male 2/4/2005      1:01 PM 40451 16044.00
```

```
442    Nicholas    Male   3/1/2013      9:26 PM 101036  2826.00
778        NaN  Female  6/18/2000     7:36 AM 106428  10867.00
```

```
Senior Management          Team
23            NaN           NaN
37            True          Client Services
55            True           Product
66            True          Business Development
92            True          Business Development
153           False         Business Development
222           NaN           Marketing
269           NaN           Distribution
442           True          Human Resources
778           NaN           NaN
```

Drop duplicate values using drop_duplicates() In order to drop duplicate values from a dataframe, we use `drop_duplicates()` function.

This function can be used in two ways: - remove duplicate rows across **all columns** with `df.drop_duplicates()` - find duplicate rows across **specific columns** `df.drop_duplicates(subset=['col1', 'col2'])`

Warning: We are doing that directly in this dataframe with `inplace = True` - we don't make a deep copy!

Remove duplicate rows across **all columns**:

```
[17]: # remove duplicate rows across all columns
employees_df.drop_duplicates(inplace=True)
employees_df
```

```
[17]:   First Name Gender Start Date Last Login Time Salary Bonus %
0      Douglas  Male   8/6/1993  12:42 PM  97308  6945.00
1      Thomas  Male   3/31/1996  6:53 AM   61933   4.17
2      Maria  Female  4/23/1993  11:17 AM  130590  11858.00
3      Jerry  Male   3/4/2005   1:00 PM   138705   9.34
4      Larry  Male   1/24/1998  4:47 PM   101004  1389.00
...
999     Henry  NaN   11/23/2014  6:09 AM  132483  16655.00
1000    Phillip  Male  1/31/1984  6:30 AM  42392  19675.00
1001    Russell  Male  5/20/2013  12:39 PM  96914  1421.00
1002    Larry  Male  4/20/2013   4:45 PM  60500  11985.00
1003    Albert  Male  5/15/2012  6:24 PM  129949  10169.00
```

```
Senior Management          Team
0            True          Marketing
1            True           NaN
2           False          Finance
3            True          Finance
4            True          Client Services
...
999           ...           ...
1000          False         Distribution
1001          False          Finance
1002          False           Product
1003          True           Sales
```

[1000 rows x 8 columns]

Remove duplicate rows across **specific columns**:

```
[18]: # remove duplicate rows across 'First Name' and 'Last Login Time' columns
employees_df.drop_duplicates(
    subset=['First Name', 'Last Login Time'], keep='last', inplace=True)
employees_df
```

```
[18]:      First Name  Gender  Start Date Last Login Time  Salary  Bonus % \
0        Douglas     Male   8/6/1993       12:42 PM  97308  6945.00
1        Thomas     Male   3/31/1996       6:53 AM   61933   4.17
2        Maria Female  4/23/1993      11:17 AM 130590 11858.00
3        Jerry     Male   3/4/2005       1:00 PM  138705   9.34
4        Larry     Male   1/24/1998      4:47 PM 101004 1389.00
...
999      Henry     NaN  11/23/2014       6:09 AM 132483 16655.00
1000     Phillip    Male  1/31/1984       6:30 AM  42392 19675.00
1001     Russell    Male  5/20/2013      12:39 PM  96914 1421.00
1002     Larry     Male  4/20/2013       4:45 PM  60500 11985.00
1003     Albert    Male  5/15/2012      6:24 PM 129949 10169.00

           Senior Management          Team
0            True             Marketing
1            True              NaN
2           False             Finance
3            True             Finance
4            True  Client Services
...
999           False            Distribution
1000          False             Finance
1001          False             Product
1002          False  Business Development
1003           True              Sales

[994 rows x 8 columns]
```

4.4.3 Compare the edited dataset with the original dataset side-by-side

@TODO:

Incorporate following sources: - [Compare two DataFrames and output their differences side-by-side](#) - [pandas compare two data frames and highlight the differences](#) - [How to Compare Two Pandas DataFrames and Get Differences](#)

4.4.4 Save edited dataset to new CSV file

@TODO:

Add explanation and python code here.

4.5 Avoidance of tendencies due to bias

The description of the Iris dataset says, that it consists of **50 samples from each of three species** of Iris (Iris setosa, Iris virginica and Iris versicolor), so there are **150 total samples**.

But how to prove it?

4.5.1 Count occurrences of unique values

To prove whether all possible classes included in the dataset and equally distributed, you can use the function `df.value_counts`.

Following parameters can be used for fine tuning:

- `dropna=False` causes that NaN values are included
- `normalize=True`: relative frequencies of the unique values are returned
- `ascending=False`: sort resulting classes descending

```
[19]: # import (again) data to dataframe from csv file
employees_df = pd.read_csv("./datasets/employees_edit.csv")
```



```
[20]: # count unique values without missing values in a column,
# ordered descending and normalized
irisdata_df['species'].value_counts(ascending=False, dropna=False, normalize=True)
```

```
[20]: Iris-setosa      0.333333
Iris-versicolor    0.333333
Iris-virginica     0.333333
Name: species, dtype: float64
```

```
[21]: # count unique values and missing values in a column,
# ordered descending and not absolute values
employees_df['Team'].value_counts(ascending=False, dropna=False, normalize=False)
```

```
[21]: Client Services      106
Business Development    103
Finance                  102
Marketing                 98
Product                   96
Sales                      94
Engineering                 92
Human Resources            92
Distribution                 90
Legal                      88
NaN                        43
Name: Team, dtype: int64
```

4.5.2 Display Histogram

This section was inspired by: [Pandas Histogram – DataFrame.hist\(\)](#).

Histograms represent **frequency distributions** graphically. This requires the separation of the data into classes (so-called **bins**).

These classes are represented in the histogram as rectangles of equal or variable width. The height of each rectangle then represents the (relative or absolute) **frequency density**.

```
[22]: employees_df.hist(column=['Salary'])
plt.show()
```

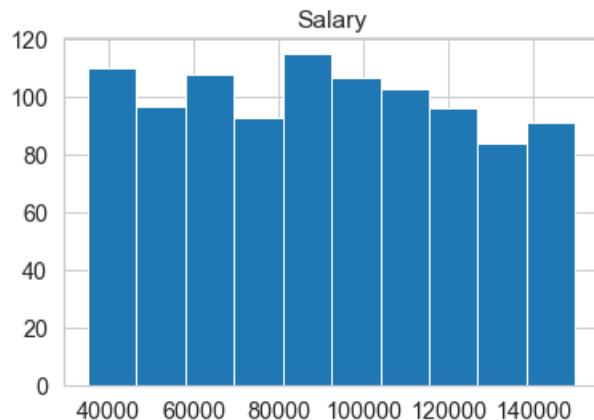


Figure 14: Histogram for frequency distribution of the salary

```
[23]: employees_df.hist(column='Salary', by='Gender')
plt.show()
```

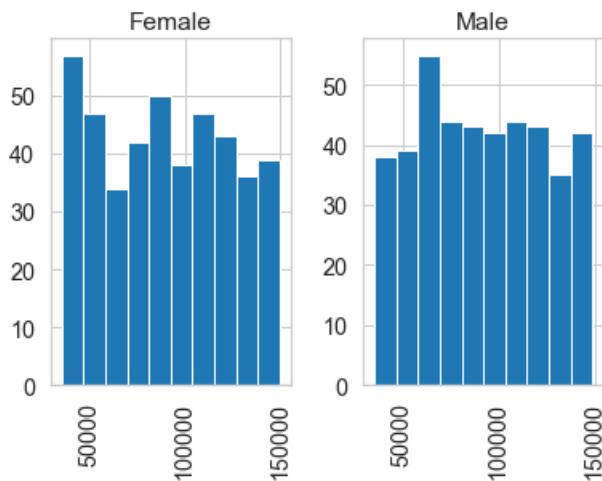


Figure 15: Histogram for the frequency distribution of the salary in comparison between men and women

4.6 First idea of correlations in dataset

To get a rough idea of the **dependencies** and **correlations** in the dataset, it can be helpful to visualize the whole dataset in a **correlation heatmap**. They show in a glance which variables are correlated, to what degree and in which direction.

Later, 2 particularly well correlated variables are selected from the dataset and plotted in a **scatterplot**.

4.6.1 Visualise data with correlation heatmap

This section was inspired by [How to Create a Seaborn Correlation Heatmap in Python?](#).

Correlation matrices are an **essential tool of exploratory data analysis**. Correlation heatmaps contain the same information in a visually appealing way. What more: they show in a glance which variables are correlated, to what degree, in which direction, and alerts us to potential multicollinearity problems (source: ibidem).

Simple correlation matrix Because **string values can never be correlated**, the class names (species) have to be converted first:

```
[24]: # encoding the class column
irisdata_df_enc = irisdata_df.replace({"species": {"Iris-setosa":0,
                                                 "Iris-versicolor":1,
                                                 "Iris-virginica":2}})

#irisdata_df_enc
```

```
[25]: irisdata_df_enc.corr()
```

```
[25]:      sepal_length  sepal_width  petal_length  petal_width  species
sepal_length      1.000000   -0.109369    0.871754    0.817954  0.782561
sepal_width     -0.109369      1.000000   -0.420516   -0.356544 -0.419446
petal_length      0.871754   -0.420516      1.000000    0.962757  0.949043
petal_width       0.817954   -0.356544     0.962757     1.000000  0.956464
species          0.782561   -0.419446     0.949043     0.956464  1.000000
```

Correlation heatmap Choose the color sets from color map.

```
[26]: # increase the size of the heatmap
plt.figure(figsize=(16, 6))

# store heatmap object in a variable to easily access it
# when you want to include more features (such as title)
# set the range of values to be displayed on the colormap from -1 to 1,
# and set 'annotation=True' to display the correlation values on the heatmap
heatmap = sns.heatmap(irisdata_df_enc.corr(), vmin=-1, vmax=1,
                      annot=True, cmap='PRGn_r')

# give a title to the heatmap
# 'pad=12' defines the distance of the title from the top of the heatmap
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=16)
plt.show()
```

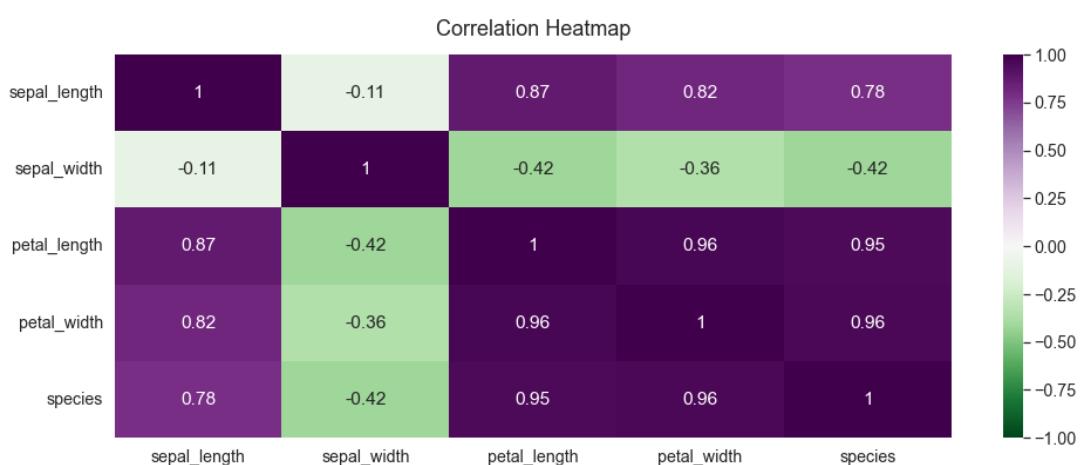


Figure 16: Correlation heatmap to explore coherences between single variables in the iris dataset

Triangle correlation heatmap When looking at the correlation heatmaps above, you would not lose any information by **cutting away half of it along the diagonal** line marked by 1-s.

The **numpy** function `np.triu()` can be used to isolate the upper triangle of a matrix while turning all the values in the lower triangle into 0.

```
[27]: np.triu(np.ones_like(irisdata_df_enc.corr()))
```

```
[27]: array([[1., 1., 1., 1., 1.],
       [0., 1., 1., 1., 1.],
       [0., 0., 1., 1., 1.],
       [0., 0., 0., 1., 1.],
       [0., 0., 0., 0., 1.]])
```

Use this mask to cut the heatmap along the diagonal:

```
[28]: plt.figure(figsize=(16, 6))

# define the mask to set the values in the upper triangle to 'True'
mask = np.triu(np.ones_like(irisdata_df_enc.corr(), dtype=bool))

heatmap = sns.heatmap(irisdata_df_enc.corr(), mask=mask,
                      vmin=-1, vmax=1, annot=True, cmap='PRGn_r')

heatmap.set_title('Triangle Correlation Heatmap', fontdict={'fontsize':18}, pad=16)
plt.show()
```

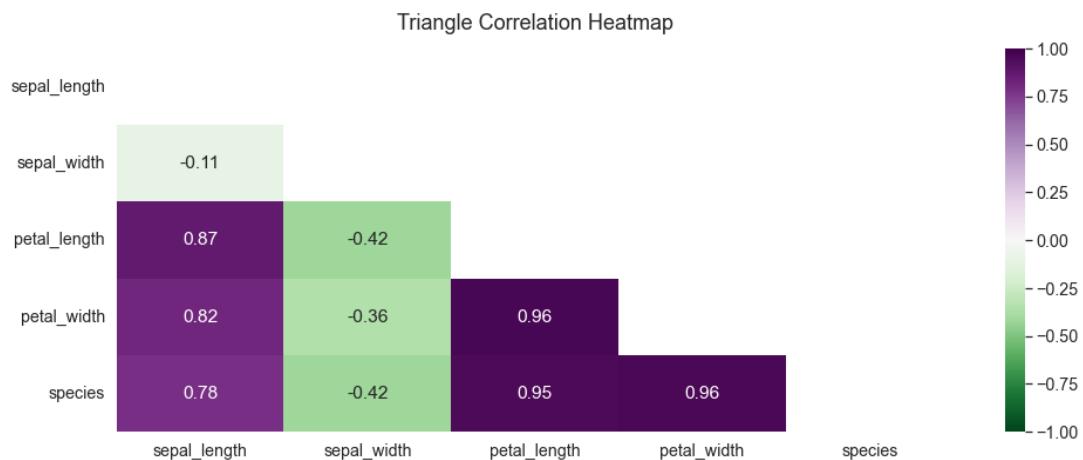


Figure 17: Correlation heatmap, which was cut at its main diagonal without losing any information

As a result from the **heatmaps** we can see, that the shape of the **petals** are the **most correlated columns** (0.96) with the **type of flowers** (species classes).

Somewhat lower correlates **sepal length** with **petal length** (0.87).

4.6.2 Visualise data with scatter plot

In the following, **Seaborn** is applied which is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures.

To investigate whether there are dependencies (e.g. correlations) in **irisdata_df** between individual variables in the dataset, it is advisable to plot them in a **scatter plot**.

```
[29]: # There are five preset seaborn themes: darkgrid, whitegrid, dark, white, and ticks.
sns.set_style("whitegrid")
# set scale of fonts
sns.set_context("notebook", font_scale=1.3, rc={"lines.linewidth": 2.5})

# 'sepal_length', 'petal_length' are iris feature data
```

```
# 'height' used to define height of graph
# 'hue' stores the class/label of iris dataset
sns.FacetGrid(irisdata_df, hue ="species",
               height = 7).map(plt.scatter,
                               'petal_width',
                               'petal_length').add_legend()

plt.title('Scatterplot of petal length and width')
plt.show()
```

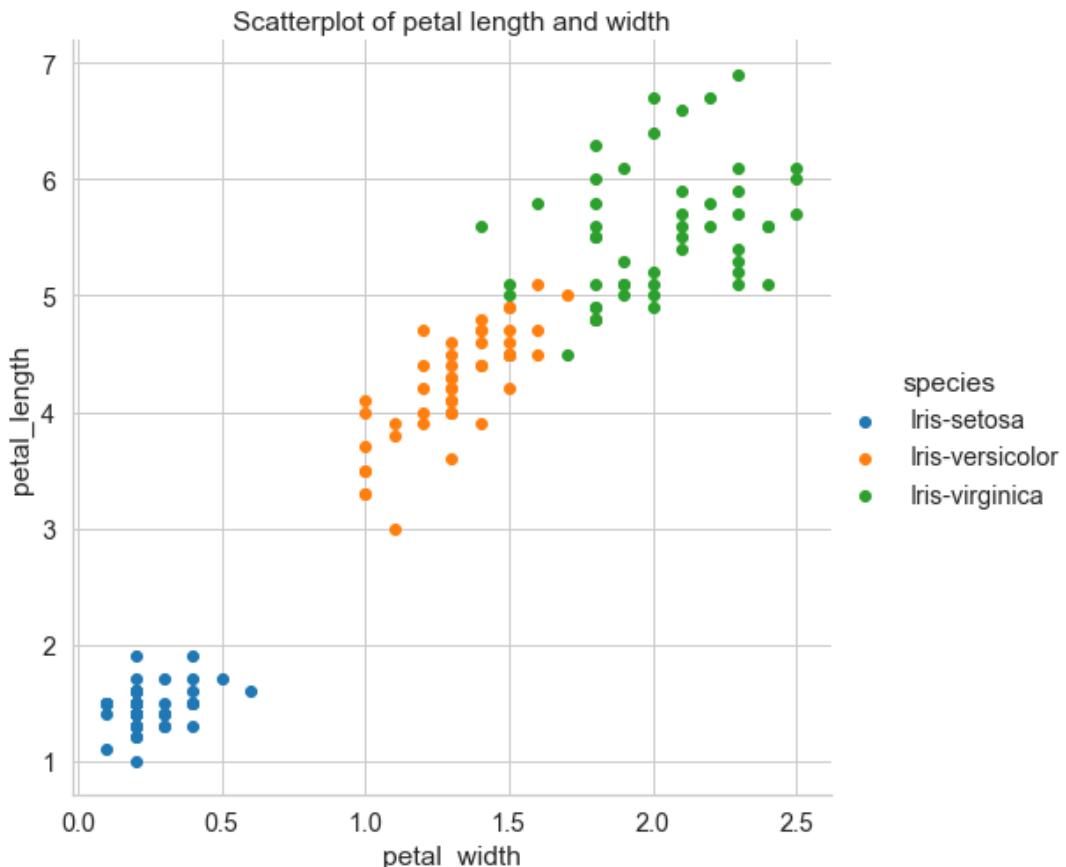


Figure 18: Plotting two individual variables of the iris dataset in the scatterplot to explore the relationships between these two

4.6.3 Visualise data with pairs plot

For systematic investigation of dependencies, all variables (each against each) are plotted in separate scatter plots.

With this so called **pairs plot** it is possible to see both **relationships** between two variables and **distribution** of single variables.

This function will create a grid of Axes such that **each numeric variable** in `irisdata_df` will be shared in the y-axis across a single row and in the x-axis across a single column.

```
[30]: sns.set(font_scale=1.0)
sns.set_style("white")

g = sns.pairplot(irisdata_df, diag_kind="kde", hue='species',
                 palette='Dark2', height=2.5)
```

```

g.map_lower(sns.kdeplot, levels=4, color=".2")
# y .. padding between title and plot
g.fig.suptitle('Pairs plot of the Iris dataset', y=1.05)
plt.show()

```

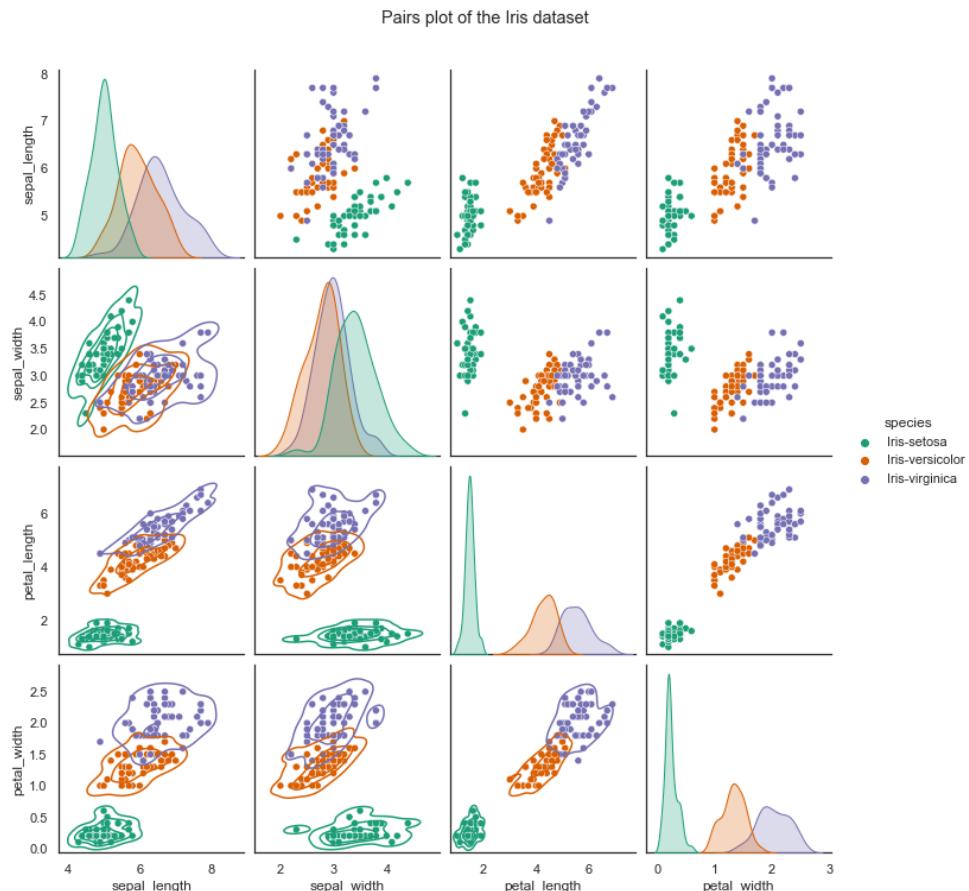


Figure 19: Plot all individual variables of the Iris dataset in pairs plot to see both the relationships between two variables and the distribution of the individual variables

5 STEP 3: Choose and create the ML model

After exploring the dataset, in this step one has to decide on a specific ML algorithm based on certain selection criteria.

However, since the AI or ML world is so huge and impossible for a ML novice to overlook, a brief description of the **relationship between AI and ML** is given in the following sections. Furthermore, a **taxonomy** of the different **learning types** is presented by also providing some example algorithms.

5.1 Short overview of the AI world and its ML algorithms

5.1.1 Relationship between AI, ML and others

@TODO: Include in this section the presentation IFA_Steimers_KI_Grundlagen_Neuronaler_Netze_2021-03-22.pdf by Prof. Steimers (IFA): - slides 6-7 “Definition of KI” - slides 10-12 “ML: Categories based on the data, task and algorithms”

In the **science world**, the term **artificial intelligence (AI)** refers to machines and systems that are capable of performing tasks that are characteristic of human intelligence.

In the **business world**, on the other hand, AI typically refers to mechanisms that perceive environmental factors and take autonomous actions. This is seen as an opportunity to achieve **predefined goals** with maximum success - without human intervention. Ultimately, this view is a mapping of **input information** to controlled **output actions** of a system. This expectation of AI-driven systems is thus hardly higher than what can be expected from today's modern automation systems.

Machine Learning (ML), on the other hand, addresses the mathematical models and algorithms that enable a computer system to recognize (new) correlations in huge amounts of sample data from various sources by inferring them independently. For scientists, machine learning is a subset of AI.

The umbrella term AI covers a very large research area. It includes a number of techniques that enable computers to learn independently and solve complex problems:

- Computer-Vision (CV)
- Supervised and Unsupervised Learning
- Reinforcement Learning and Genetic Algorithms
- Computational Linguistics
- Robotics
- etc.

The following Venn diagram shows the relationship between Artificial Intelligence (AI), Machine Learning (ML) and other integrated technologies. The quantities that do not belong to the main category represent techniques that can function as stand-alone techniques and do not necessarily fall into the artificial intelligence group in all cases (for further details see [Emerging technologies based on artificial intelligence to assess quality and consumer preference of beverages](#)).

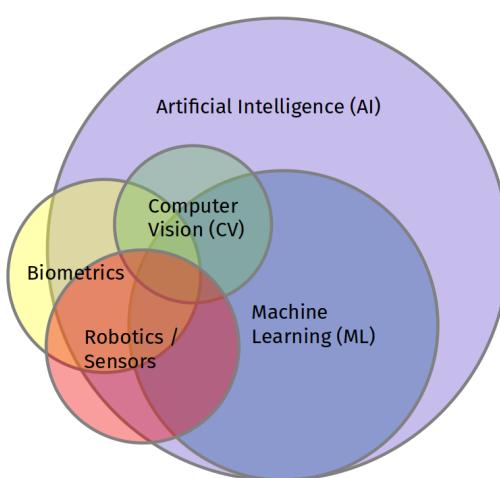


Figure 20: Venn diagram showing the relationship between Artificial Intelligence (AI), Machine Learning (ML) and other integrated technologies (source: Kasper, adapted from [Emerging technologies based on artificial intelligence to assess quality and consumer preference of beverages](#), license: CC-BY-SA 4.0)

5.1.2 Taxonomy of machine learning

The field of machine learning can be divided into the following **types of learning**:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

Here are some further sources:

- [Taxonomy of machine learning algorithms](#)
- [Comprehensive Survey of Machine Learning Approaches in Cognitive Radio-Based Vehicular Ad Hoc Networks](#)
- [A Taxonomy of Machine Learning Techniques](#)

- [ML Algorithms: One SD](#)
- [Machine Learning Map](#)

Supervised learning The goal of **supervised learning (SL)** is to learn a **function** that maps a **input to an output**, based on example input-output pairs. This involves inferring a relationship describable by a mathematical function from **labeled training data** consisting of a set of training examples (see [Supervised Learning](#)).

A few well-known algorithms from the field of **supervised learning** are mentioned here:

- Naive Bayes
- Linear Regression
- Logistic Regression
- Artificial Neural Networks (ANN)
- Support Vector Classifier (SVC)
- Decision Trees
- Random Forests

Unsupervised learning The algorithms of this category look for internal structures in the data of a dataset, such as **grouping or clustering of data points**. These algorithms can thus learn relationships from test data that have not been labeled, classified, or categorized. Rather than responding to feedback (as in supervised learning), unsupervised learning algorithms detect **commonalities in the data** and respond based on the presence or absence of such commonalities in each new dataset (see [Unsupervised learning](#)).

Here are some algorithms from the field of **unsupervised learning**:

- K-means Clustering
- Spectral Clustering
- Hierarchical Clustering
- Principal Component Analysis (PCA)

Semi-supervised learning This type of learning falls between **unsupervised** learning (without any labeled training data) and **supervised** learning (with completely labeled training data). Some of the training examples are missing training labels, yet many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce a considerable improvement in learning accuracy (source: [Semi-supervised learning](#)).

Reinforcement learning This is an area of machine learning concerned with how **intelligent agents** ought to **take actions in an environment** in order to maximize the notion of cumulative **reward**. Due to its generality, the field is studied in many other disciplines, such as **game theory** and **control theory**.

Reinforcement learning differs from supervised learning in **not needing labeled input/output pairs** be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on **finding a balance** between **exploration** (of uncharted territory) and **exploitation** (of current knowledge) (source: [Reinforcement learning](#)).

Here are some algorithms from the field of **reinforcement learning**:

- Iterative Policy
- Q-Learning
- SARSA
- Learning Classifiers
- Stochastic Gradient
- Genetic Algorithm

5.2 Decision graph for selecting an suitable algorithm

Now that the iris dataset has been analyzed in terms of its data structure and internal correlations, the most difficult task on the way to solving a problem using machine learning arises: finding the “right” ML algorithm (also called **estimator**).

The diverse estimators available are more or less well qualified for the respective problems with their partly very different data types. The good news is that the ML software package **Scikit-Learn** provides the following **flowchart** as a rough **guide** in choosing the right estimator for the particular task (see: [Choosing the right estimator](#)).

However, it must also be emphasized that a considerable **level of experience** through systematic trial and error is crucial to be successful in finding an “optimal” estimator.

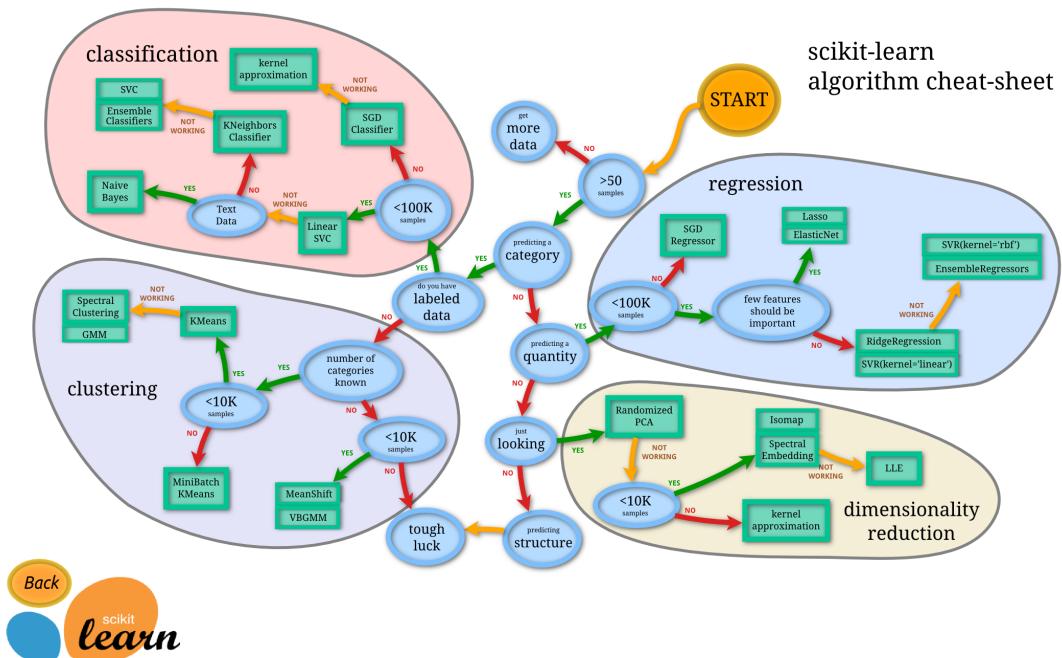


Figure 21: Decision graph for choosing an appropriate ML algorithm (source: [Choosing the right estimator](#), license: unknown)

5.3 Reasons for choosing Support Vector Classifier (SVC)

Among other ML algorithms suitable for the Iris dataset (such as the decision-tree-based **random-forests classifier**), the reasoned choice here in this tutorial falls on the **support vector classifier (SVC)**.

The following **reasons** led to the decision for the **Support Vector Classifier (SVC)**:

- the aim is to predict the species using unlabeled test data, so the task is to **classify**
- the iris dataset is **fully labeled** (by designating the iris species)
- the dataset contains significantly **less than 100k samples**

But the most important reason is that it is **easy to understand** how it works - so it is exactly suitable for a beginner tutorial ;)

5.4 Operating principal of SVC

Support Vector Classifiers (SVC) try to **find the best hyperplane to separate** the different classes by maximizing the distance between sample points and the hyperplane (source: [In Depth: Parameter tuning for SVC](#)).

The following figure shows the operating principal of the SVC algorithm: the hyperplanes H_1 till H_4 (left graphic) do separate the classes. A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier (source: [Support-vector machine](#)).

The right graphic shows the optimal hyperplane characterized by maximizing the margin between the classes. The perpendicular distance of the closest data points to the hyperplane determines their position and orientation. These perpendicular distances are the **support vectors** of the hyperplane - this is how the algorithm got its name.

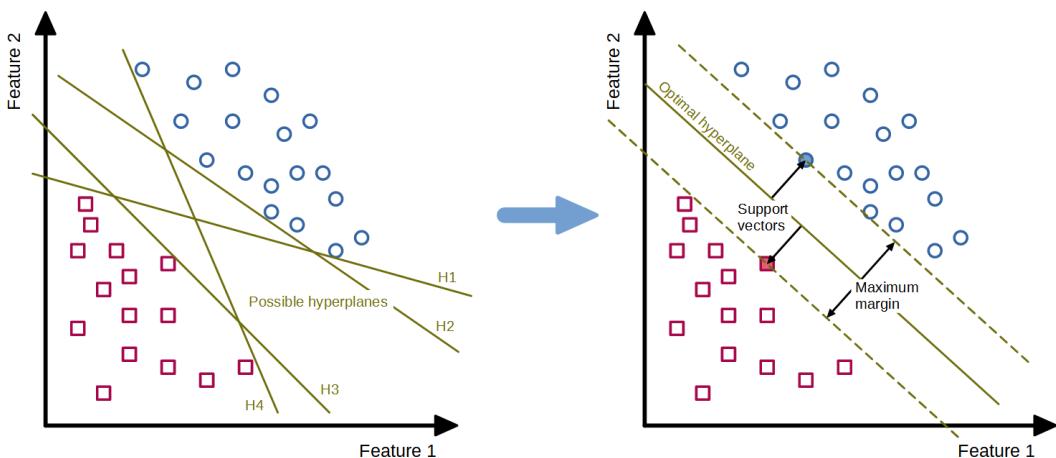


Figure 22: Support Vector Classifiers (SVC) separate the data points in classes by finding the best hyperplane by maximizing the margin to its support vectors (source: Kasper, license: CC-BY-SA 4.0)

5.5 Create the SVC model

In this step we create the SVC model choosing a **linear kernel** with default parameters.

```
[31]: from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
```

6 STEP 4: Prepare the dataset for training

In this step the dataset is prepared for the actual classification by SVC. Depending on the selected ML algorithm as well as the data structure, it may be necessary to prepare the data before training (e.g., by **standardization**, **normalization**, or **binarization** based on thresholds). Furthermore, errors in the dataset (e.g. **data gaps**, **duplicates** or obvious **misentries**) should be corrected now at the latest.

Through the intensive exploration of the data in ([STEP 2: Explore the ML dataset](#)), we know that special **preparation** of the data is **not necessary**. The values are complete and without gaps and there are no duplicates. The values are in similar ranges, which **does not require normalization** of the data.

Furthermore, we know that the **classes** are very **evenly distributed** and thus bias tendencies should be avoided.

For further details about **Standardization** and **Normalization** read here: [What are standardization and normalization? Test with iris data set in Scikit-learn](#).

```
[32]: # import Iris dataset for exploration (again)
irisdata_df = pd.read_csv('./datasets/IRIS_flower_dataset_kaggle.csv')
```

6.1 Standardization

Standardize the feature values by computing the **mean**, subtracting the mean from the data points, and then dividing by the **standard deviation**.

@TODO:

Incorporate section “Skalieren von Merkmalen” of the book `OReilly_Praxiseinstieg_Machine_Learning_Scikit-Learn` (see Géron 2018).

```
[39]: from sklearn.preprocessing import StandardScaler
```

```
#scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)
irisdata_df
```

```
#X_train
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
..
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

[150 rows x 5 columns]

6.2 Normalization

7 STEP 5: Carry out training, prediction and testing

7.1 Split the dataset

In the next very important step, the dataset is split into **2 subsets**: a **training dataset** and a **test dataset**. As the names suggest, the training dataset is used to train the ML algorithm. The test dataset is then used to check the quality of the trained ML algorithm (here the **recognition rate**). For this purpose, the **class labels** are **removed** from the training dataset - after all, these are to be predicted.

Typically, the **test dataset** should contain about **20%** of the entire dataset.

In particular, to **avoid bias** in the sorted iris dataset due to splitting, the **order** of the data rows must be **randomized**. This is done with the parameter `shuffle=True`.

```
[35]: from sklearn.model_selection import train_test_split

X = irisdata_df.drop('species', axis=1)
y = irisdata_df['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, shuffle=True)
```

Check that the split datasets are still balanced and that no **bias** has been created by the splitting.

For this test, the previously separated labels `y_train` must be added back to the training dataset `X_train`.

```
[37]: # make a deep copy of 'X_train'
X_train_bias_test_df = X_train.copy(deep=True)

# add list of labels to test dataframe
X_train_bias_test_df['species'] = y_train

# count unique values without missing values in a column,
# ordered descending and normalized
X_train_bias_test_df['species'].value_counts(ascending=False, dropna=False, normalize=True)
```

```
-----  
TypeError                                     Traceback (most recent call last)  
Input In [37], in <cell line: 2>()  
      1 # make a deep copy of 'X_train'  
----> 2 X_train_bias_test_df = X_train.copy(deep=True)  
      4 # add list of labels to test dataframe  
      5 X_train_bias_test_df['species'] = y_train  
  
TypeError: copy() got an unexpected keyword argument 'deep'
```

For training, do not use only the variables that correlate best with each other, but all of them.

Otherwise, the result of the prediction would be significantly worse. Maybe this is already an indication of **overfitting** of the ML model.

```
[38]: # DO NOT USE THIS!
X_train, X_test, y_train, y_test = train_test_split(X[['sepal_length',
                                                       'sepal_width']],
                                                    y, test_size = 0.20)
```

7.2 Train the SVC

In this step the SVC is trained with the training data. Training means to **fit** the SVC to the **training data**.

```
[39]: # fit the model for the data
classifier.fit(X_train, y_train)
```

```
[39]: SVC(kernel='linear', random_state=0)
```

7.3 Make predictions

In this step the aim is to **predict the species** using unlabeled test data.

```
[40]: y_pred = classifier.predict(X_test)
#X_test
#y_pred
```

8 STEP 6: Evaluate model's performance

Subsequently to the training of the SVC model and the classification predictions made based on the test data, this step evaluates the **quality of the classification result** using known **metrics** such as the **accuracy score** as well as the **confusion matrix**.

8.1 Accuracy Score

In a multilabel classification (such as the Iris dataset), this **Accuracy classification score** computes the subset accuracy. For further details see [sklearn.metrics.accuracy_score](#).

```
[41]: from sklearn.metrics import accuracy_score

acc_score = accuracy_score(y_test, y_pred)

print("Accuracy score: {:.2f} %".format(acc_score.mean()*100))
```

Accuracy score: 80.00 %

8.2 Classification Report

The classification report shows a representation of the main **classification metrics on a per-class basis**. This gives a deeper intuition of the classifier behavior over global accuracy which can mask functional weaknesses in one class of a multiclass problem (see [Classification Report](#)).

```
[42]: from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	5
Iris-versicolor	0.86	0.75	0.80	16
Iris-virginica	0.64	0.78	0.70	9
accuracy			0.80	30
macro avg	0.83	0.84	0.83	30
weighted avg	0.81	0.80	0.80	30

8.3 Cross-validation score

The function `cross_val_score()` from the Scikit-learn package **trains and tests a model over multiple folds** of your dataset. This cross validation method gives a better **understanding of model performance** over the whole dataset instead of just a single train/test split (see [Using cross_val_score in sklearn, simply explained](#)).

@TODO:

Incorporate section “Bessere Auswertung mittels Kreuzvalidierung” of the book [OReilly_Praxiseinstieg_Machine_Learning_Scikit-Learn_TensorFlow_2018_Anm_bk.pdf](#).

```
[43]: from sklearn.model_selection import cross_val_score

accuracies = cross_val_score(estimator = classifier, X = X_train,
                             y = y_train, cv = 10)

print("Cross-validation score: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Cross-validation score: 82.50 %
 Standard Deviation: 14.65 %

8.4 Confusion matrix

The **confusion matrix** measures the quality of predictions from a classification model by looking at how many **predictions** are **True** and how many are **False** (see [What the Confusion Matrix Measures?](#)..

8.4.1 Textual confusion matrix

For checking the accuracy of the model, the **confusion matrix** can be used for the **cross validation**.

By using the function `sklearn.metrics.confusion_matrix()` a confusion matrix of the true iris class labels versus the predicted class labels is plotted.

```
[25]: cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[ 6  0  0]
 [ 0  9  1]
 [ 0  0 14]]
```

8.4.2 Colored confusion matrix

The function `sklearn.metrics.ConfusionMatrixDisplay()` plots a colored confusion matrix.

```
[44]: sns.set_style("white")

# print colored confusion matrix
cm_colored = metrics.ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

cm_colored.figure_.suptitle("Colored Confusion Matrix")
cm_colored.figure_.set_figwidth(8)
cm_colored.figure_.set_figheight(7)

cm_colored.confusion_matrix

# save figure as PNG
plt.tight_layout()
plt.savefig('images/confusion_matrix.png', dpi=150, pad_inches=5)
plt.show()
```

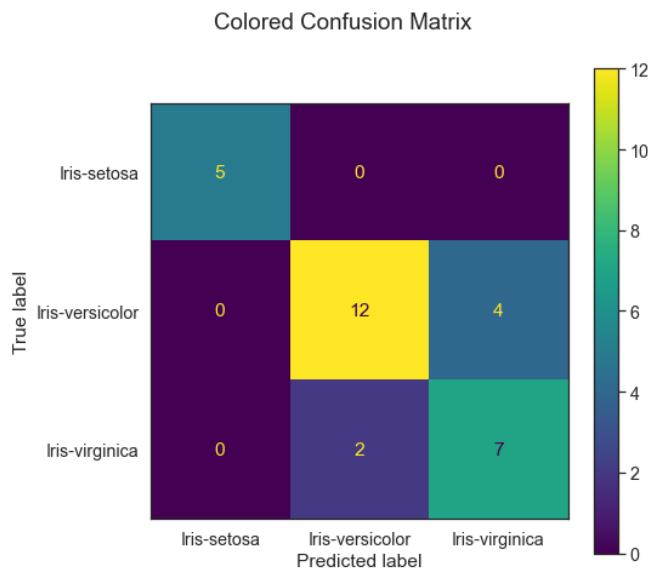


Figure 23: Checking the accuracy of the model by using the confusion matrix for cross-validation

9 STEP 7: Vary parameters of the ML model manually

This section was inspired by [In Depth: Parameter tuning for SVC](#)

In this section, the 4 SVC parameters `kernel`, `gamma`, `C` and `degree` will be introduced one by one. Furthermore, their influence on the classification result by varying these single parameters will be shown.

Disclaimer: In order to show the effects of varying the individual parameters in 2D graphs, only the best correlating variables `petal_length` and `petal_width` are used to train the SVC.

9.1 Prepare dataset

```
[ ]: from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
import numpy as np

# import iris dataset again
irisdata_df = pd.read_csv('./datasets/IRIS_flower_dataset_kaggle.csv')

# encode the class column from class strings to integer equivalents
irisdata_df_enc = irisdata_df.replace({"species": {"Iris-setosa":0,
                                                    "Iris-versicolor":1,
                                                    "Iris-virginica":2}})

#irisdata_df_enc
```

9.1.1 Prepare datasets for parameter variation and plotting

These datasets will be used for parameter variation and plotting only. In particular, for later **2D plotting** of the effects of parameter variation, only **2 variables** of the iris dataset can be used.

However, as seen in the previous section, this selection is very much at the expense of detection accuracy. Therefore, it is not useful to make predictions with this subset of data - it is not necessary to divide it into a training and a test dataset.

```
[ ]: # copy only 2 feature columns
# and convert pandas dataframe to numpy array
X_plot = irisdata_df_enc[['petal_length', 'petal_width']].to_numpy(copy=True)
#X_plot = irisdata_df_enc[['sepal_length', 'sepal_width']].to_numpy(copy=True)
#X_plot
```

```
[ ]: # convert pandas dataframe to numpy array
# and get a flat 1D copy of 2D numpy array
y_plot = irisdata_df_enc[['species']].to_numpy(copy=True).flatten()
#y_plot
```

9.1.2 Prepare dataset for prediction and evaluation

To evaluate the recognition accuracy by parameter variation, the complete iris dataset with all variables must be used. To make predictions with test data, the dataset is again divided into a training and a test dataset.

```
[ ]: X = irisdata_df.drop('species', axis=1)
y = irisdata_df['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, shuffle=True)
```

9.2 Plotting functions

This function helps to visualize the modifications by varying the individual SVC parameters:

```
[ ]: def plotSVC(title, svc, X, y, xlabel, ylabel):
    # create a mesh to plot in
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # prevent division by zero
    if x_min == 0.0:
        x_min = 0.1

    h = (x_max / x_min)/1000
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    plt.subplot(1, 1, 1)
    Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.6)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.xlim(xx.min(), xx.max())
    plt.title(title)
    plt.show()
```

This function cares for cross validation:

```
[ ]: def crossValSVC(X_train, y_train, kernel='rbf', gamma='scale', C=1.0, degree=3):
    # train the SVC
    svc = svm.SVC(kernel=kernel,
                  gamma=gamma,
```

```

        C=C,
        degree=degree).fit(X_train, y_train)
# calculate accuracies
accuracies = cross_val_score(estimator = svc, X = X_train,
                             y = y_train, cv = 10)

accuracy = accuracies.mean()*100
return accuracy

```

This function plots the variation of the SVC parameters against the prediction accuracy to show the effect of variation and its limits regarding the phenomenon **overfitting**:

```
[ ]: def plotParamsAcc(param_list, acc_list, param_name, log_scale=False):
    fig, ax = plt.subplots(figsize=(10,6))
    title_str = 'Variation of {} parameter '.format(param_name) \
                +'and its effect to prediction accuracy'
    plt.title(title_str)
    ax.plot(param_list, accuracy_list)
    if log_scale:
        # set the X axis scale to logarithmic
        ax.set_xscale('log')
    plt.xlabel(param_name)
    plt.ylabel('accuracy [%]')
    plt.grid()
    plt.show()
```

9.3 Vary kernel of SVC

The `kernel` parameter selects the type of hyperplane that is used to separate the data. Using `linear` (`linear classifier`) kernel will use a linear hyperplane (a line in the case of 2D data). The `rbf` (`radial basis function kernel`) and `poly` (`polynomial kernel`) kernel use non linear hyperplanes. The `default` is `kernel=rbf`.

```
[ ]: kernels = ['linear', 'rbf', 'poly', 'sigmoid']

xlabel = 'Petal length'
ylabel = 'Petal width'

for kernel in kernels:
    svc_plot = svm.SVC(kernel=kernel).fit(X_plot, y_plot)
    accuracy = crossValSVC(X_train, y_train, kernel=kernel)
    title_str = 'kernel: \''+str(kernel)+'\', '+'Acc. prediction: {:.2f}%'.format(accuracy)
    plotSVC(title_str, svc_plot, X_plot, y_plot, xlabel, ylabel)
```

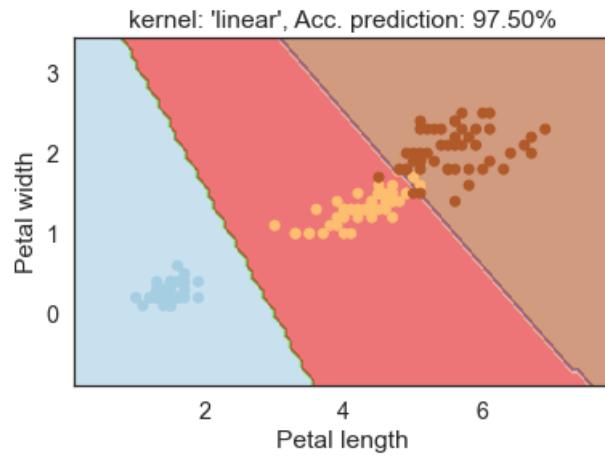


Figure 24: This group of images shows the effect on the classification by the choice of the different SVC kernels ('linear', 'rbf', 'poly' and 'sigmoid')

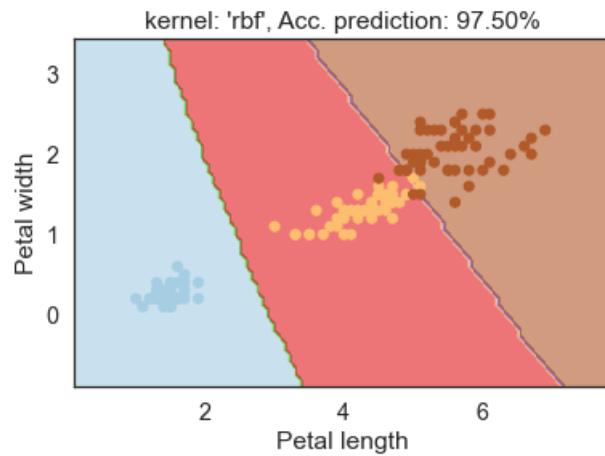


Figure 25: This group of images shows the effect on the classification by the choice of the different SVC kernels ('linear', 'rbf', 'poly' and 'sigmoid')

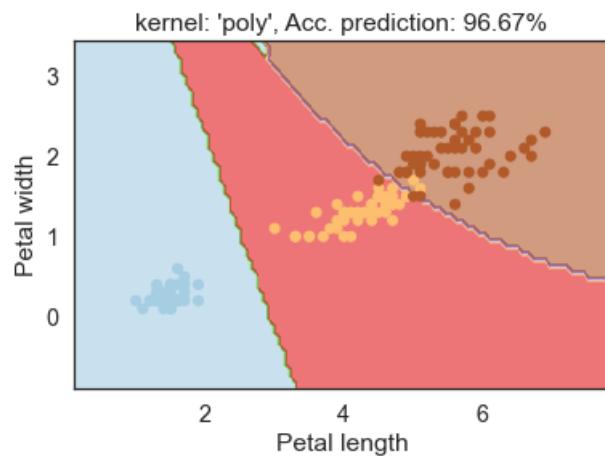


Figure 26: This group of images shows the effect on the classification by the choice of the different SVC kernels ('linear', 'rbf', 'poly' and 'sigmoid')

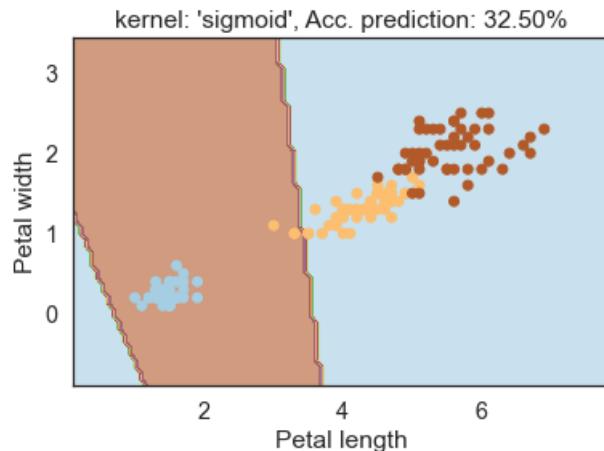


Figure 27: This group of images shows the effect on the classification by the choice of the different SVC kernels ('linear', 'rbf', 'poly' and 'sigmoid')

9.4 Vary gamma parameter

The `gamma` parameter is used for **non linear hyperplanes**. The higher the `gamma` float value it tries to exactly fit the training dataset. The **default** is `gamma='scale'`.

```
[ ]: gammas = [0.1, 1, 10, 100, 200]

xlabel = 'Petal length'
ylabel = 'Petal width'

for gamma in gammas:
    svc_plot = svm.SVC(kernel='rbf', gamma=gamma).fit(X_plot, y_plot)
    accuracy = crossValSVC(X_train, y_train, kernel='rbf', gamma=gamma)
    title_str = 'gamma: \''+str(gamma)+'\', ' \
               +'Acc. prediction: {:.2f}%'.format(accuracy)
    plotSVC(title_str, svc_plot, X_plot, y_plot, xlabel, ylabel)
```

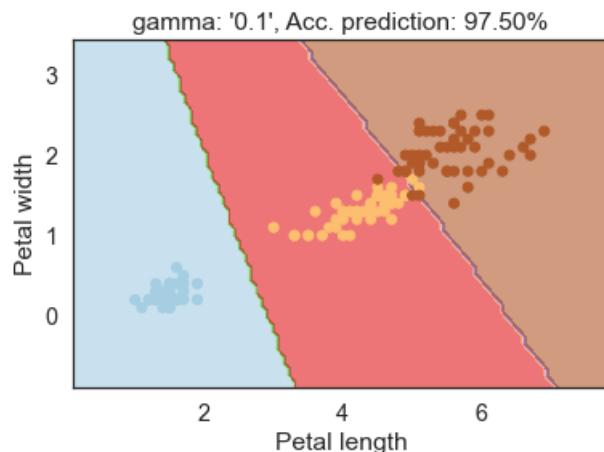


Figure 28: This group of images shows the effect on the classification by the variation of the parameter 'gamma' of the 'rbf' kernel

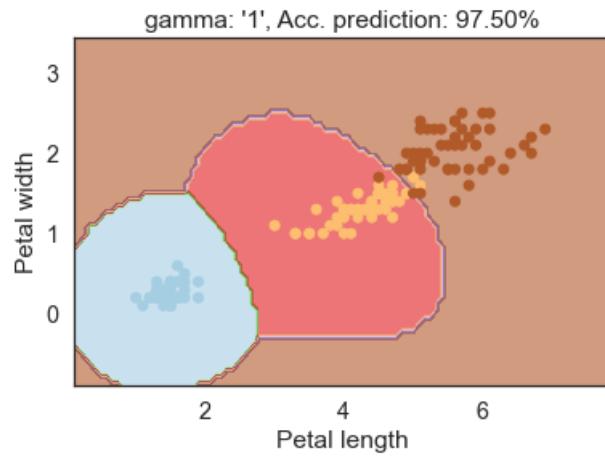


Figure 29: This group of images shows the effect on the classification by the variation of the parameter 'gamma' of the 'rbf' kernel

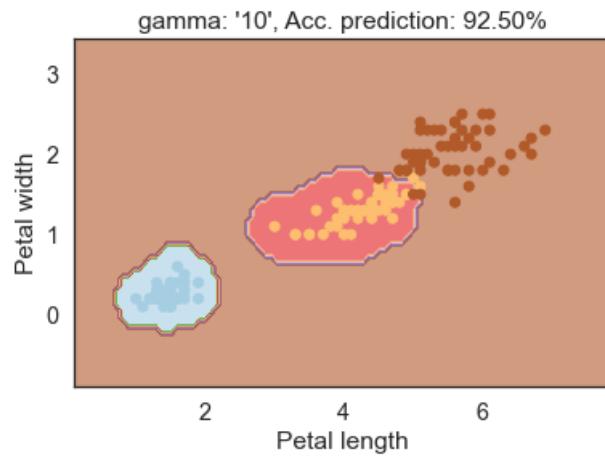


Figure 30: This group of images shows the effect on the classification by the variation of the parameter 'gamma' of the 'rbf' kernel

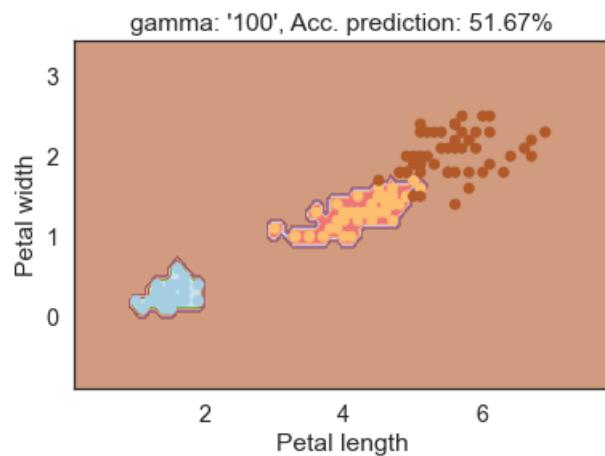


Figure 31: This group of images shows the effect on the classification by the variation of the parameter 'gamma' of the 'rbf' kernel

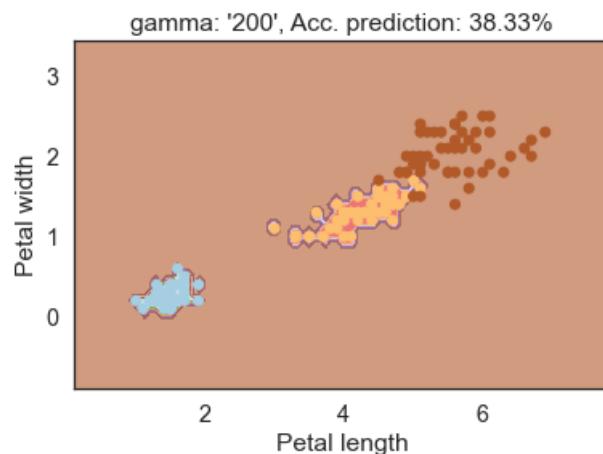


Figure 32: This group of images shows the effect on the classification by the variation of the parameter 'gamma' of the 'rbf' kernel

Show the variation of the SVC parameter `gamma` against the **prediction accuracy**.

As we can see, increasing `gamma` leads to **overfitting** as the classifier tries to perfectly fit the training data.

```
[ ]: gammas = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 10, 100, 200]

accuracy_list = []
for gamma in gammas:
    accuracy = crossValSVC(X_train, y_train, kernel='rbf', gamma=gamma)
    accuracy_list.append(accuracy)

plotParamsAcc(gammas, accuracy_list, 'gamma', log_scale=True)
```

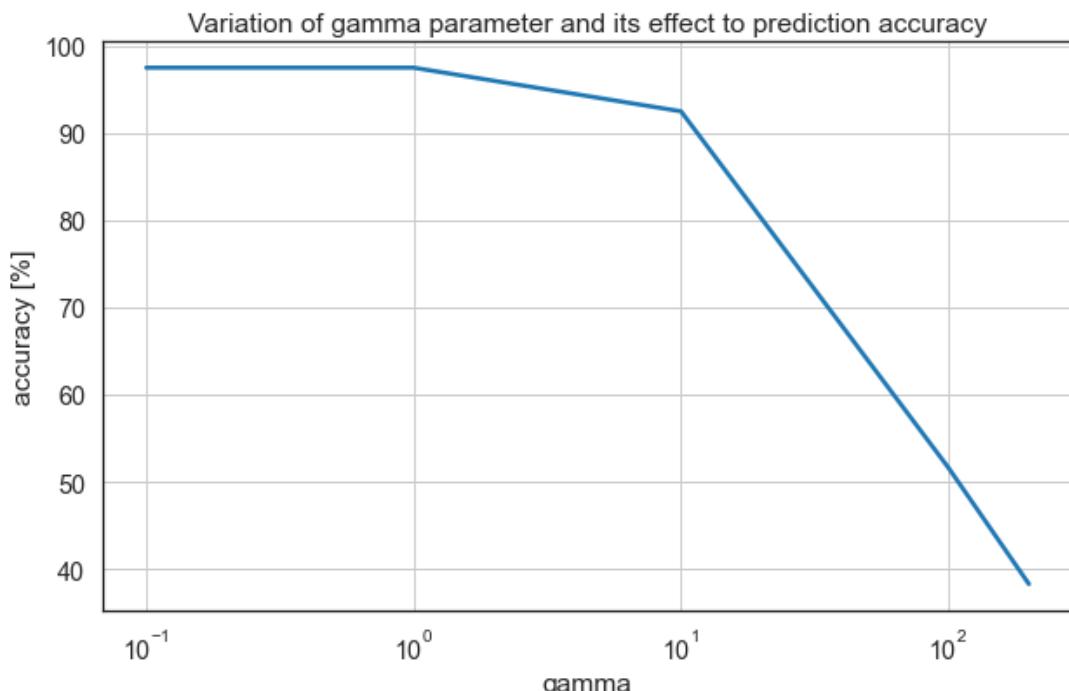


Figure 33: The plot shows the variation of the SVC parameter 'gamma' against the prediction accuracy

9.5 Vary C parameter

The C parameter is the **penalty** of the error term. It controls the trade off between smooth decision boundary and classifying the training points correctly. The **default** is C=1.0.

```
[ ]: cs = [0.1, 1, 5, 10, 100, 1000, 10000]

xlabel = 'Petal length'
ylabel = 'Petal width'

for c in cs:
    svc_plot = svm.SVC(kernel='rbf', C=c).fit(X_plot, y_plot)
    accuracy = crossValSVC(X_train, y_train, kernel='rbf', C=c)
    title_str = 'C: \''+str(c)+'\', '
    + 'Acc. prediction: {:.2f}%'.format(accuracy)
    plotSVC(title_str, svc_plot, X_plot, y_plot, xlabel, ylabel)
```

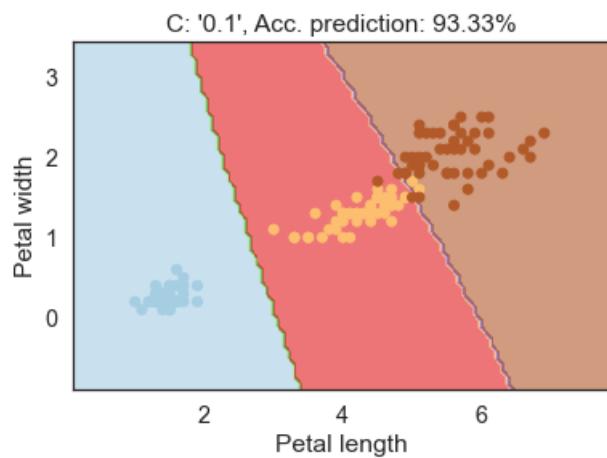


Figure 34: This group of images shows the effect on the classification by the variation of the parameter 'C' of the 'rbf' kernel

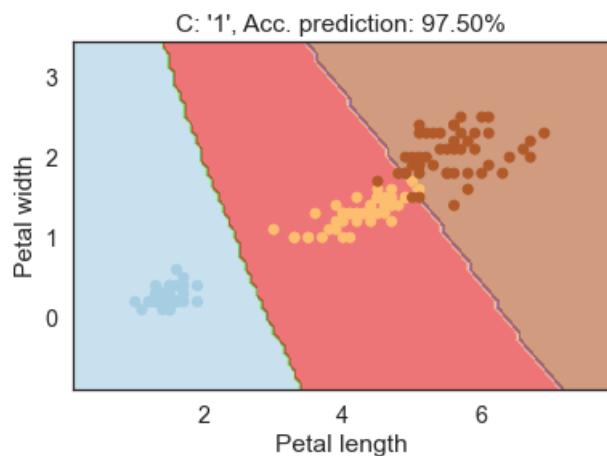


Figure 35: This group of images shows the effect on the classification by the variation of the parameter 'C' of the 'rbf' kernel

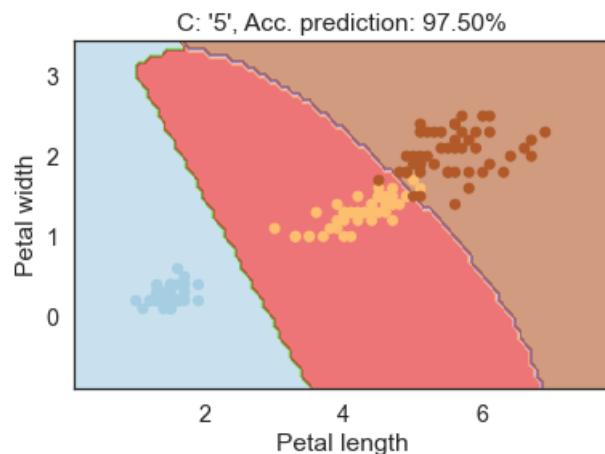


Figure 36: This group of images shows the effect on the classification by the variation of the parameter 'C' of the 'rbf' kernel

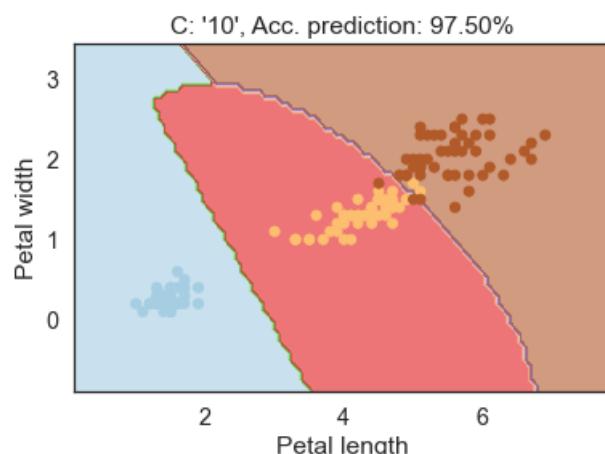


Figure 37: This group of images shows the effect on the classification by the variation of the parameter 'C' of the 'rbf' kernel

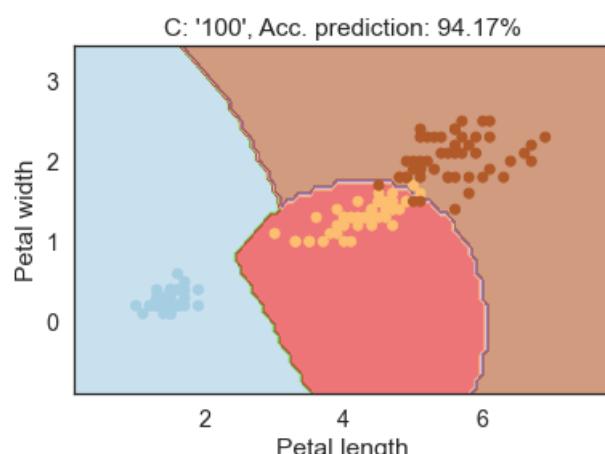


Figure 38: This group of images shows the effect on the classification by the variation of the parameter 'C' of the 'rbf' kernel

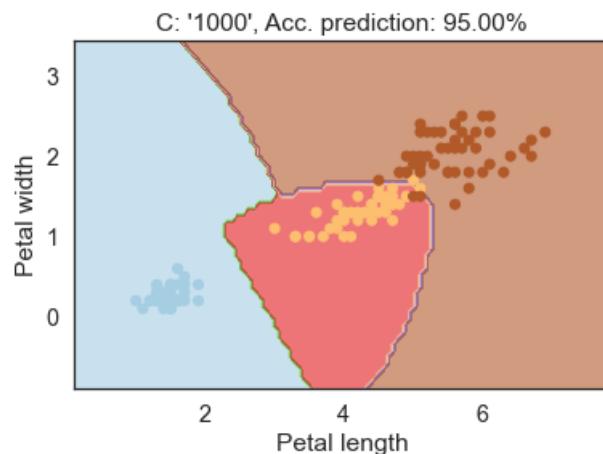


Figure 39: This group of images shows the effect on the classification by the variation of the parameter 'C' of the 'rbf' kernel

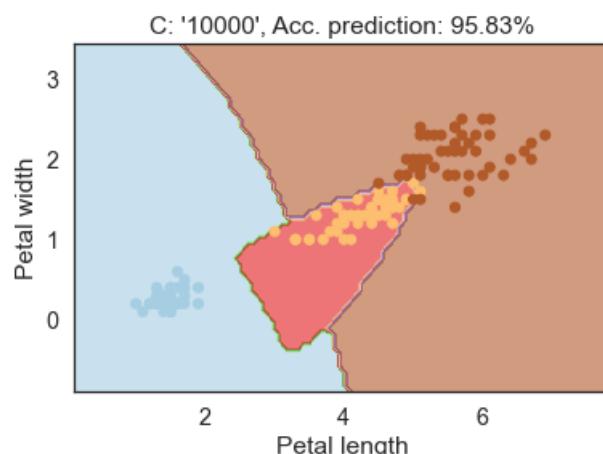


Figure 40: This group of images shows the effect on the classification by the variation of the parameter 'C' of the 'rbf' kernel

Show the variation of the SVC parameter C against the **prediction accuracy**.

But be careful: to high C values may lead to **overfitting** the training data.

```
[ ]: cs = [0.1, 1, 5, 6, 7, 8, 10, 100, 1000, 10000]

accuracy_list = list()
for c in cs:
    accuracy = crossValSVC(X_train, y_train, kernel='rbf', C=c)
    accuracy_list.append(accuracy)

plotParamsAcc(cs, accuracy_list, 'C', log_scale=True)
```

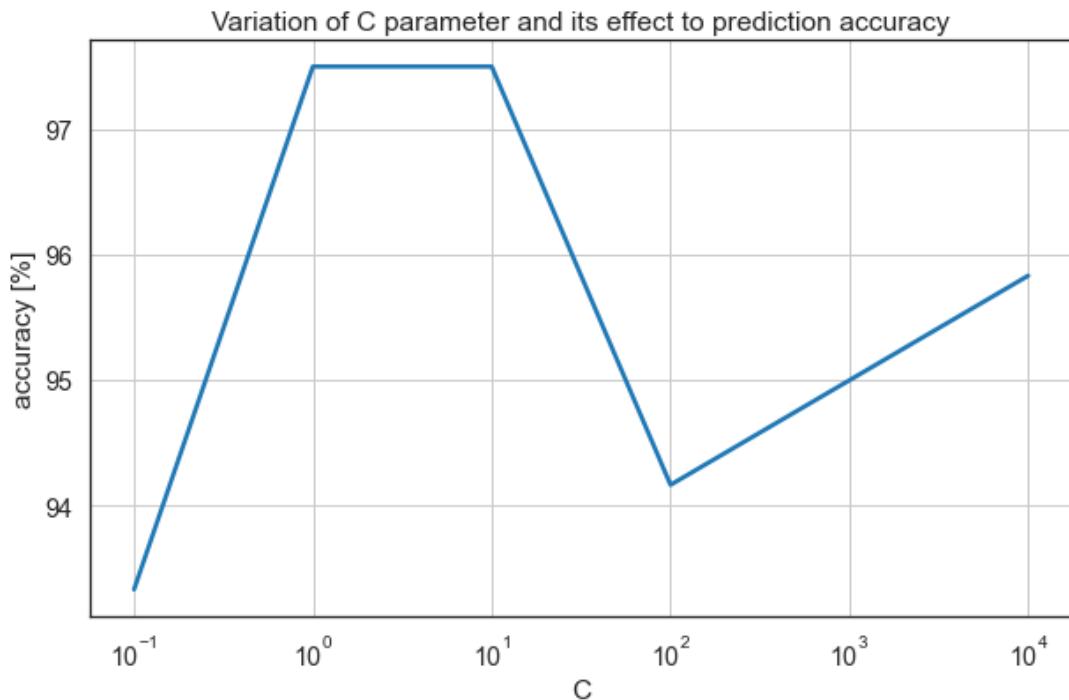


Figure 41: The plot shows the variation of the SVC parameter 'C' against the prediction accuracy

9.6 Vary degree parameter

The `degree` parameter is used when the `kernel` is set to `poly` and is ignored by all other kernels. It's basically the **degree of the polynomial** used to find the hyperplane to split the data. The **default** is `degree=3`.

Using `degree = 1` is the same as using a `linear` kernel. Also, increasing this parameters leads to **higher training times**.

```
[ ]: degrees = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

xlabel = 'Petal length'
ylabel = 'Petal width'

for degree in degrees:
    svc_plot = svm.SVC(kernel='poly', degree=degree).fit(X_plot, y_plot)
    accuracy = crossValSVC(X_train, y_train, kernel='poly', degree=degree)
    title_str = 'degree: \''+str(degree)+'\', '\ \
               +'Acc. prediction: {:.2f}%'.format(accuracy)
    plotSVC(title_str, svc_plot, X_plot, y_plot, xlabel, ylabel)
```

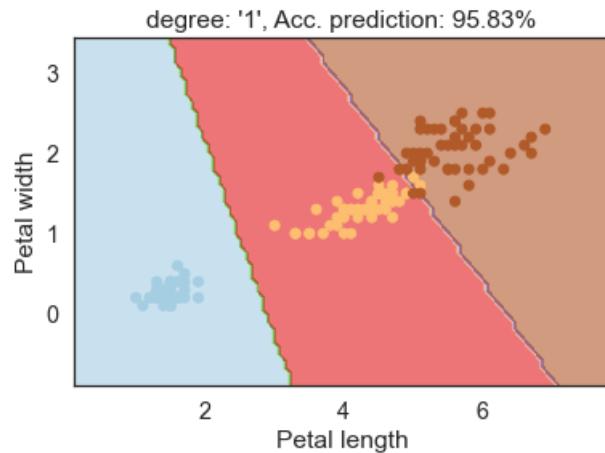


Figure 42: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

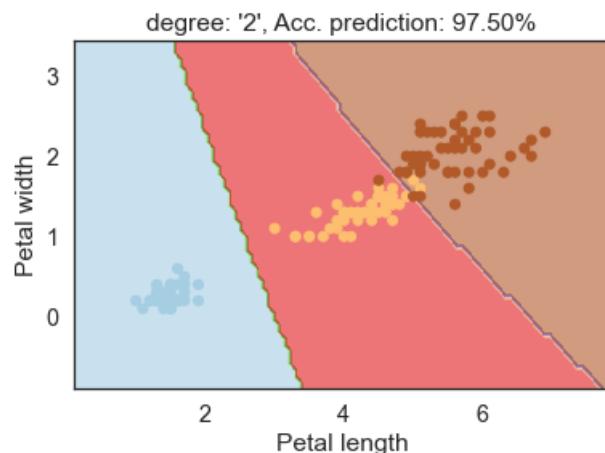


Figure 43: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

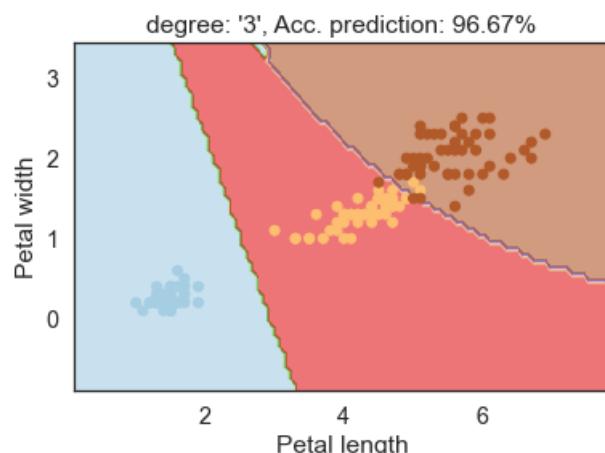


Figure 44: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

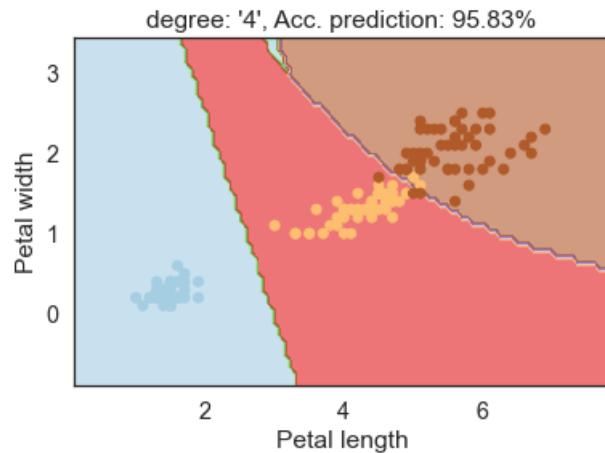


Figure 45: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

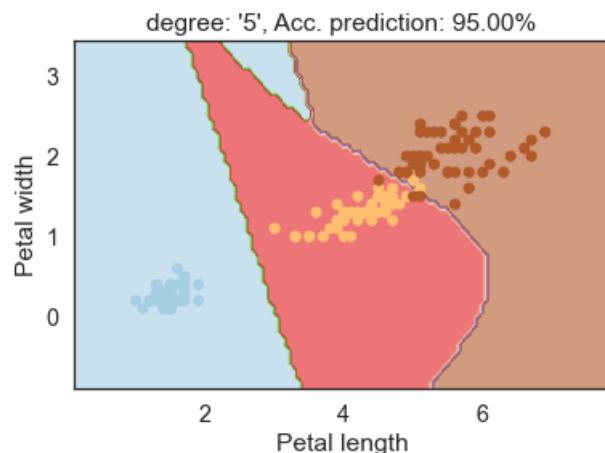


Figure 46: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

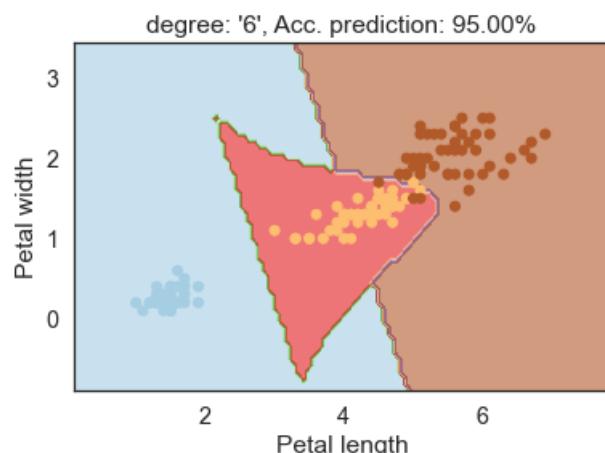


Figure 47: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

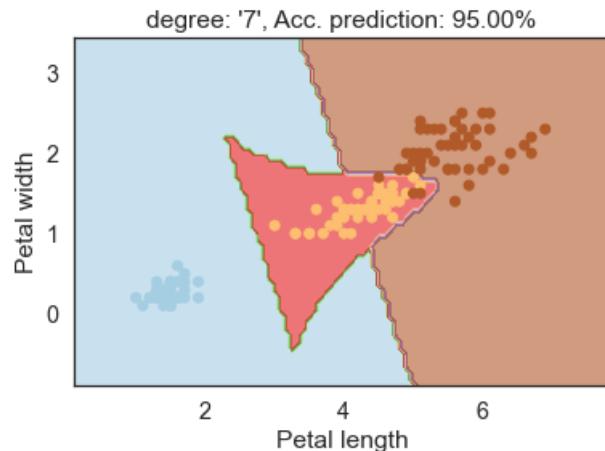


Figure 48: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

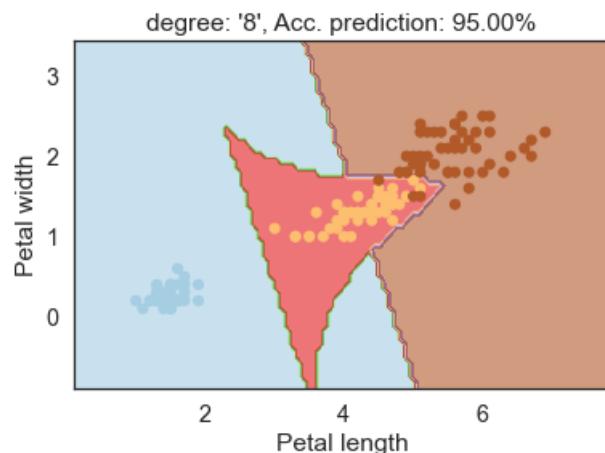


Figure 49: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

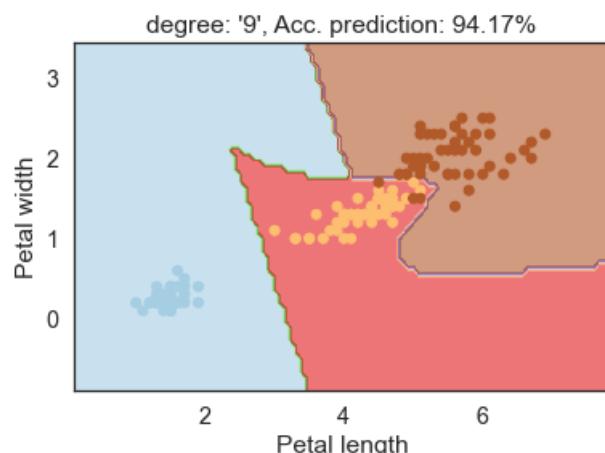


Figure 50: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

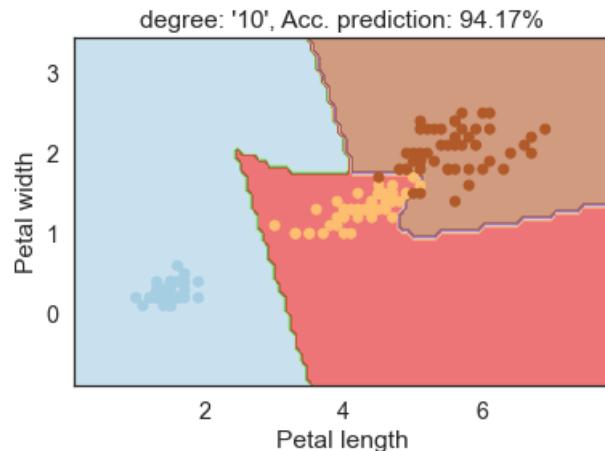


Figure 51: This group of images shows the effect on the classification by the variation of the parameter 'degree' of the 'poly' kernel

Show the variation of the SVC parameter degree against the **prediction accuracy**.

As we can see, increasing the **degree** of the polynomial hyperplane leads to **overfitting** the training data.

```
[ ]: degrees = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

accuracy_list = list()
for degree in degrees:
    accuracy = crossValSVC(X_train, y_train, kernel='poly', degree=degree)
    accuracy_list.append(accuracy)

plotParamsAcc(degrees, accuracy_list, 'degree', log_scale=False)
```

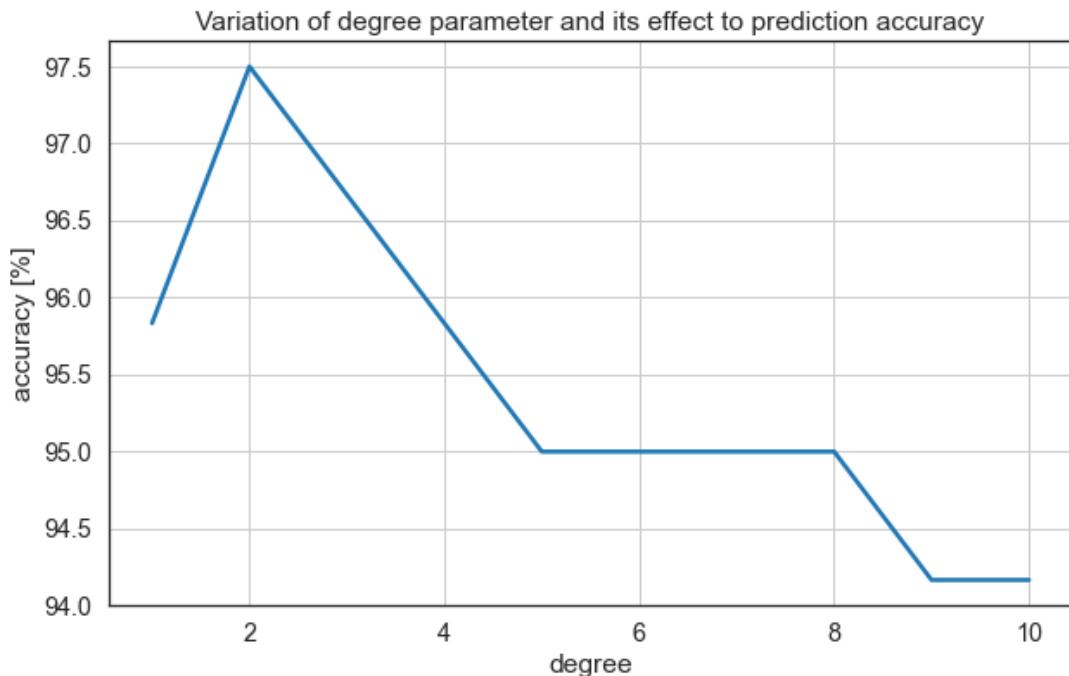


Figure 52: The plot shows the variation of the SVC parameter 'degree' against the prediction accuracy

10 STEP 8: Tune the ML model systematically

In the final step, two approaches to systematic hyper-parameter search are presented: **Grid Search** and **Randomized Search**. While the former exhaustively considers all parameter combinations for given values, the latter selects a number of candidates from a parameter space with a particular random distribution.

Sources:

- 3.2. Tuning the hyper-parameters of an estimator
 - `sklearn.model_selection.GridSearchCV`
 - `sklearn.model_selection.RandomizedSearchCV`
- Introduction to hyperparameter tuning with scikit-learn and Python
 - Abalone Dataset
- Hyperparameter tuning using Grid Search and Random Search: A Conceptual Guide

Import the necessary packages:

```
[47]: # general packages
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
#from sklearn.svm import SVC
from sklearn import svm, metrics
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# additional packages for grid search
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV

# additional packages for randomized search
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedKFold

# import class MeasExecTimeOfProgram from python file MeasExecTimeOfProgramclass.py
from MeasExecTimeOfProgram_class import MeasExecTimeOfProgram
```

Set path and columns of the Iris dataset for import:

```
[2]: # specify the path of the dataset
CSV_PATH = "./datasets/IRIS_flower_dataset_kaggle.csv"
```

Load dataset and split it into subsets for training and testing in the ratio 80% to 20%:

```
[23]: # load the dataset, separate the features and labels, and perform a
# training and testing split using 80% of the data for training and
# 20% for evaluation
irisdata_df = pd.read_csv(CSV_PATH)

X = irisdata_df.drop('species', axis=1)
y = irisdata_df['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, shuffle=True)
```

Check that the split datasets are still balanced and that no **bias** has been created by the splitting.

For this test, the previously separated labels `y_train` must be added back to the training dataset `X_train`.

```
[24]: # make a deep copy of 'X_train'
X_train_bias_test_df = X_train.copy(deep=True)

# add list of labels to test dataframe
X_train_bias_test_df['species'] = y_train

# count unique values without missing values in a column,
# ordered descending and normalized
X_train_bias_test_df['species'].value_counts(ascending=False, dropna=False, normalize=True)
```

```
[24]: Iris-versicolor    0.358333
Iris-virginica      0.333333
Iris-setosa         0.308333
Name: species, dtype: float64
```

Standardize the feature values by computing the **mean**, subtracting the mean from the data points, and then dividing by the **standard deviation**:

```
[ ]: scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#X_train
```

10.1 Finding a baseline

The aim of this sub-step is to establish a baseline on the Iris dataset by training a **Support Vector Classifier (SVC)** with no hyperparameter tuning.

Train the model with **no tuning of hyperparameters** to find the baseline for later improvements:

```
[54]: classifier = svm.SVC(kernel = 'linear', random_state = 0)

# initiate measuring execution time
execTime = MeasExecTimeOfProgram()
execTime.start()

classifier.fit(X_train, y_train)

# print time delta
print('Execution time: {:.4f} ms'.format(execTime.stop()))
```

Execution time: 1.6954 ms

Evaluate our model using accuracy score:

```
[55]: # predict labels
y_pred = classifier.predict(X_test)
```

```
[56]: # calculate cross validation score
# HINT: do NOT use the accuracy score - it's too inaccurate!
accuracies = cross_val_score(estimator = classifier, X = X_train,
                             y = y_train, cv = 10)
```

```
print("Cross-validation score: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Cross-validation score: 97.50 %
 Standard Deviation: 3.82 %

```
[57]: # print classification report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	13
Iris-versicolor	1.00	0.86	0.92	7
Iris-virginica	0.91	1.00	0.95	10
accuracy			0.97	30
macro avg	0.97	0.95	0.96	30
weighted avg	0.97	0.97	0.97	30

```
[58]: sns.set_style("white")

# print colored confusion matrix
cm_colored = metrics.ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

cm_colored.figure_.suptitle("Colored Confusion Matrix")
cm_colored.figure_.set_figwidth(8)
cm_colored.figure_.set_figheight(7)

cm_colored.confusion_matrix

plt.tight_layout()
plt.show()
```

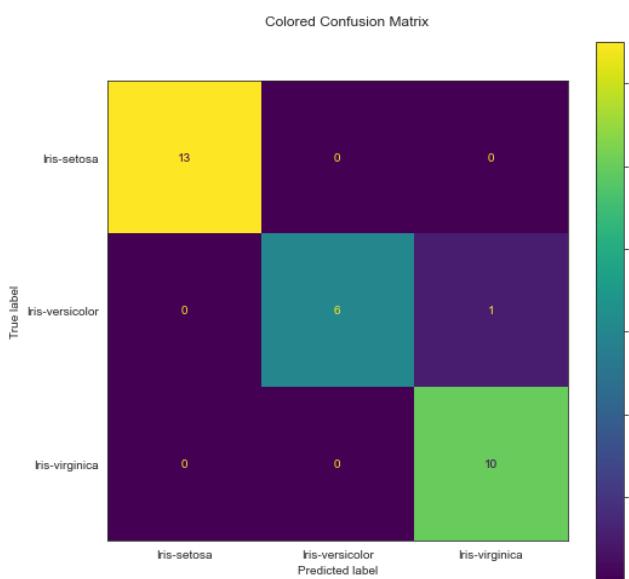


Figure 53: Confusion matrix for cross-validation of the baseline

```
[42]: classifier.get_params()
```

```
[42]: {'C': 1.0,
      'break_ties': False,
      'cache_size': 200,
      'class_weight': None,
      'coef0': 0.0,
      'decision_function_shape': 'ovr',
      'degree': 3,
      'gamma': 'scale',
      'kernel': 'linear',
      'max_iter': -1,
      'probability': False,
      'random_state': 0,
      'shrinking': True,
      'tol': 0.001,
      'verbose': False}
```

10.2 Grid search

Initialize the SVC model and define the **space of the hyperparameters** to perform the **grid search** over:

```
[45]: classifier = svm.SVC()

kernels = ["linear", "rbf", "sigmoid", "poly"]
gammas = [0.1, 1, 10, 100, 200]
cs = [0.1, 1, 5, 10, 100, 1000, 10000]

# reduce the possible polynomial degrees to reasonable values,
# since with higher degrees the calculation time increases exponentially
degrees = [1, 2, 3, 4, 5]

grid = dict(kernel=kernels, gamma=gammas, C=cs, degree=degrees)
```

Initialize a **cross-validation fold** and **perform a grid search** to tune the hyperparameters:

```
[59]: cvFold = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

gridSearch = GridSearchCV(estimator=classifier, param_grid=grid, n_jobs=-1,
                          cv=cvFold, scoring="accuracy")

# initiate measuring execution time
execTime = MeasExecTimeOfProgram()
execTime.start()

searchResults = gridSearch.fit(X_train, y_train)

# print time delta
print('Execution time: {:.2f} s'.format(execTime.stop()/1000))
```

Execution time: 39.64 s

Extract the best model and evaluate it:

```
[61]: # predict labels by best model
bestModel = searchResults.best_estimator_
```

```
y_pred = bestModel.predict(X_test)
```

```
[62]: # calculate cross validation score from the best model
# HINT: do NOT use the accuracy score - it's too inaccurate!
accuracies = cross_val_score(estimator = bestModel, X = X_train,
                             y = y_train, cv = 10)

print("Cross-validation score: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Cross-validation score: 98.33 %

Standard Deviation: 3.33 %

```
[63]: from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	13
Iris-versicolor	1.00	0.86	0.92	7
Iris-virginica	0.91	1.00	0.95	10
accuracy			0.97	30
macro avg	0.97	0.95	0.96	30
weighted avg	0.97	0.97	0.97	30

```
[64]: sns.set_style("white")

# print colored confusion matrix
cm_colored = metrics.ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

cm_colored.figure_.suptitle("Colored Confusion Matrix")
cm_colored.figure_.set_figwidth(8)
cm_colored.figure_.set_figheight(7)

cm_colored.confusion_matrix

plt.tight_layout()
plt.show()
```

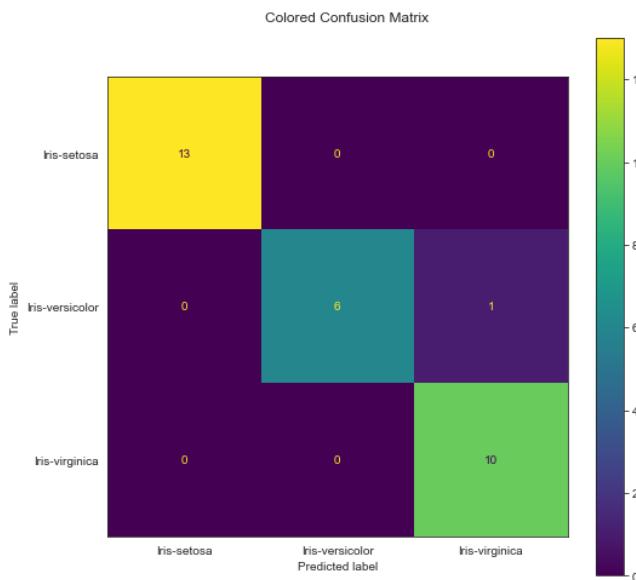


Figure 54: Confusion matrix for cross-validation after the grid search has been performed

```
[ ]: bestModel.get_params()
```

```
[ ]: {'C': 5,
      'break_ties': False,
      'cache_size': 200,
      'class_weight': None,
      'coef0': 0.0,
      'decision_function_shape': 'ovr',
      'degree': 1,
      'gamma': 0.1,
      'kernel': 'poly',
      'max_iter': -1,
      'probability': False,
      'random_state': None,
      'shrinking': True,
      'tol': 0.001,
      'verbose': False}
```

10.3 Randomized search

Initialize the SVC model and define the **space of the hyperparameters** to perform the **randomized search** over:

```
[72]: classifier = svm.SVC()

kernels = ["linear", "rbf", "sigmoid", "poly"]
gammas = [0.1, 1, 10, 100, 200]
cs = [0.1, 1, 5, 10, 100, 1000, 10000]

# reduce the possible polynomial degrees to reasonable values,
# since with higher degrees the calculation time increases exponentially
degrees = [1, 2, 3, 4, 5]

grid = dict(kernel=kernels, gamma=gammas, C=cs, degree=degrees)
```

Initialize a **cross-validation fold** and perform a **randomized search** to tune the hyperparameters:

```
[73]: cvFold = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

randomSearch = RandomizedSearchCV(estimator=classifier, n_jobs=-1,
                                    cv=cvFold, param_distributions=grid,
                                    scoring="accuracy")

# initiate measuring execution time
execTime = MeasExecTimeOfProgram()
execTime.start()

searchResults = randomSearch.fit(X_train, y_train)

# print time delta
print('Execution time: {:.3f} s'.format(execTime.stop()/1000))
```

Execution time: 0.720 s

Extract the best model and evaluate it:

```
[74]: # predict labels by best model
bestModel = searchResults.best_estimator_

y_pred = bestModel.predict(X_test)
```

```
[75]: # calculate cross validation score from the best model
# HINT: do NOT use the accuracy score - it's too inaccurate!
accuracies = cross_val_score(estimator = bestModel, X = X_train,
                             y = y_train, cv = 10)

print("Cross-validation score: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Cross-validation score: 97.50 %

Standard Deviation: 3.82 %

```
[76]: from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	13
Iris-versicolor	1.00	0.86	0.92	7
Iris-virginica	0.91	1.00	0.95	10
accuracy			0.97	30
macro avg	0.97	0.95	0.96	30
weighted avg	0.97	0.97	0.97	30

```
[77]: sns.set_style("white")

# print colored confusion matrix
cm_colored = metrics.ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

cm_colored.figure_.suptitle("Colored Confusion Matrix")
cm_colored.figure_.set_figwidth(8)
```

```
cm_colored.figure_.set_figheight(7)

cm_colored.confusion_matrix

plt.tight_layout()
plt.show()
```

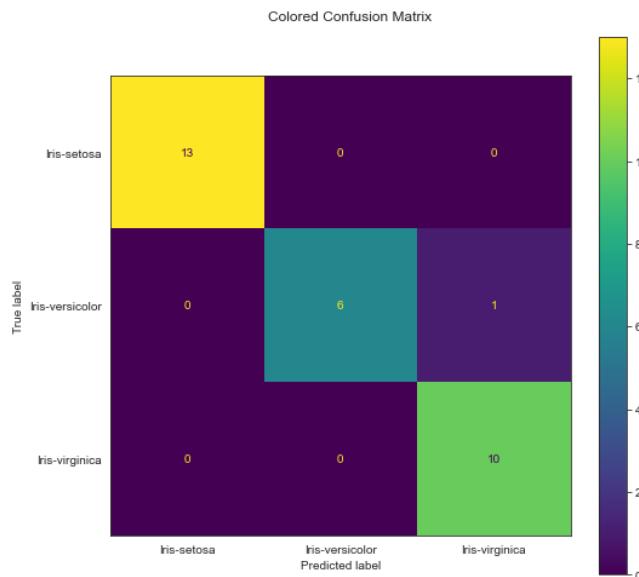


Figure 55: Confusion matrix for cross-validation after the randomized search has been performed

[78]: bestModel.get_params()

```
[78]: {'C': 10,
'break_ties': False,
'cache_size': 200,
'class_weight': None,
'coef0': 0.0,
'decision_function_shape': 'ovr',
'degree': 1,
'gamma': 0.1,
'kernel': 'rbf',
'max_iter': -1,
'probability': False,
'random_state': None,
'shrinking': True,
'tol': 0.001,
'verbose': False}
```

11 Summary and outlook

11.1 English summary

11.2 German summary

12 Acknowledgments

13 References

Online references

- Adhikari, Daisy (Mar. 1, 2021). *Is Octave Good for Machine Learning?* English. Data Science Nerd. URL: <https://datasciencenerd.com/is-octave-good-for-machine-learning/> (visited on 09/08/2022) (cit. on pp. 10, 12).
- Adler, Rasmus (Sept. 6, 2019). *Autonom oder vielleicht doch nur hochautomatisiert? Was ist z.B. der Unterschied zwischen autonomem Fahren und hochautomatisiertem Fahren?* German. Ed. by Fraunhofer IESE. URL: <https://www.iese.fraunhofer.de/blog/autonom-oder-vielleicht-doch-nur-hochautomatisiert-was-ist-eigentlich-der-unterschied/> (visited on 09/06/2022) (cit. on pp. 3, 6).
- (June 10, 2021). *Autonome Systeme: großes Potenzial für die digitale Zukunft.* German. Ed. by Fraunhofer IESE. URL: <https://www.iese.fraunhofer.de/blog/autonome-systeme/> (visited on 09/06/2022) (cit. on pp. 3, 6).
- Allan, Alasdair (June 24, 2019). *Benchmarking Machine Learning on the New Raspberry Pi 4, Model B.* English. URL: <https://www.hackster.io/news/benchmarking-machine-learning-on-the-new-raspberry-pi-4-model-b-88db9304ce4> (visited on 08/31/2022) (cit. on p. 10).
- Exterman, Dori (June 7, 2021). *CUDA vs OpenCL: Which to Use for GPU Programming.* English. URL: <https://www.incredibuild.com/blog/cuda-vs-opencl-which-to-use-for-gpu-programming> (visited on 08/31/2022) (cit. on p. 9).
- Gupta, Sakshi (Oct. 6, 2021). *What is the best language for machine learning?* English. Springboard. URL: <https://www.springboard.com/blog/data-science/best-language-for-machine-learning/> (visited on 09/08/2022) (cit. on pp. 10, 12).
- Heidrich, Jens, Daniel Schneider, and Andreas Jedlitschka (Mar. 4, 2021). *Dependable AI / Verlässliche KI – Ein Überblick.* German. Ed. by Fraunhofer IESE. URL: <https://www.iese.fraunhofer.de/blog/dependable-ai/> (visited on 09/06/2022) (cit. on pp. 4, 6).
- IndianTechWarrior, ed. (June 7, 2022). *Does TensorFlow Support OpenCL?* English. IndianTechWarrior. URL: <https://indiantechwarrior.com/does-tensorflow-support-opencl/> (visited on 08/31/2022) (cit. on p. 10).
- Iwanski, Luke (June 1, 2016). *TensorFlow for OpenCL using SYCL.* English. Codeplay Software Ltd. URL: <https://www.codeplay.com/portal/blogs/2016/06/01/tensorflow-for-opencl-using-sycl.html> (visited on 08/31/2022) (cit. on p. 10).
- Johnson, Leigh (June 24, 2019). *Portable Computer Vision: TensorFlow 2.0 on a Raspberry Pi.* English. URL: <https://towardsdatascience.com/portable-computer-vision-tensorflow-2-0-on-a-raspberry-pi-part-1-of-2-84e318798ce9> (visited on 08/31/2022) (cit. on p. 10).
- NVIDIA.Developer (July 20, 2022). *CUDA GPUs - Compute Capability.* English. Ed. by NVIDIA Corporation. URL: <https://developer.nvidia.com/cuda-gpus> (visited on 08/31/2022) (cit. on p. 9).
- Politiek, Rients (Aug. 16, 2022a). *Deep learning with Raspberry Pi and alternatives in 2022.* English. Q-engineering. URL: <https://qengineering.eu/deep-learning-with-raspberry-pi-and-alternatives.html> (visited on 08/31/2022) (cit. on p. 10).
- (June 10, 2022b). *Install OpenCL on Raspberry Pi 3 B+.* English. Q-engineering. URL: <https://qengineering.eu/install-opencl-on-raspberry-pi-3.html> (visited on 08/31/2022) (cit. on p. 10).
- Romer, Paul (Apr. 13, 2018). *Jupyter, Mathematica, and the Future of the Research Paper.* English. URL: <https://paulromer.net/jupyter-mathematica-and-the-future-of-the-research-paper/> (visited on 09/08/2022) (cit. on p. 15).
- Somers, James (Apr. 5, 2018). *The Scientific Paper Is Obsolete.* English. The Atlantic. URL: <https://www.theatlantic.com/science/archive/2018/04/the-scientific-paper-is-obsolete/556676/> (visited on 09/08/2022) (cit. on p. 15).

- Srinagar (Mar. 22, 2018). *Iris Flower Dataset*. Iris flower data set used for multi-class classification. English. URL: <https://www.kaggle.com/datasets/arshid/iris-flower-dataset> (visited on 08/31/2022) (cit. on p. 20).
- Wikipedia: Autonomie (June 4, 2022). *Autonomie*. German. Ed. by Wikipedia, The Free Encyclopedia. URL: <https://de.wikipedia.org/w/index.php?title=Autonomie&oldid=224750608> (visited on 09/06/2022) (cit. on pp. 3, 6).
- Wikipedia: CUDA (Aug. 28, 2022). *CUDA*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: <https://en.wikipedia.org/w/index.php?title=CUDA&oldid=1107147697> (visited on 08/30/2022) (cit. on p. 9).
- Wikipedia: IDE (Sept. 1, 2022). *Integrated development environment*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=Integrated_development_environment&oldid=1107871547 (visited on 09/08/2022) (cit. on p. 14).
- Wikipedia: Iris dataset (Aug. 8, 2022). *Iris flower data set*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=Iris_flower_data_set&oldid=1103056245 (visited on 08/31/2022) (cit. on pp. 20, 21).
- Wikipedia: Iris setosa (Aug. 5, 2022). *Iris setosa*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=Iris_setosa&oldid=1102525458 (visited on 09/01/2022) (cit. on p. 21).
- Wikipedia: Iris versicolor (June 4, 2022). *Iris versicolor*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=Iris_versicolor&oldid=1091474628 (visited on 09/01/2022) (cit. on p. 21).
- Wikipedia: Iris virginica (June 21, 2022). *Iris virginica*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=Iris_virginica&oldid=1094181148 (visited on 09/01/2022) (cit. on p. 21).
- Wikipedia: OpenCL (Aug. 21, 2022). *OpenCL*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: <https://en.wikipedia.org/w/index.php?title=OpenCL&oldid=1105615914> (visited on 08/30/2022) (cit. on p. 9).
- Wikipedia: OSAL (Apr. 28, 2022). *OSAL*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=Operating_system_abstraction_layer&oldid=1085096210 (visited on 09/01/2022) (cit. on p. 9).
- Wikipedia: Petal (May 15, 2022). *Petal*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: <https://en.wikipedia.org/w/index.php?title=Petal&oldid=1087986268> (visited on 09/01/2022) (cit. on p. 21).
- Wikipedia: Raspi (Aug. 28, 2022). *Raspberry Pi*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=Raspberry_Pi&oldid=1107184087 (visited on 08/31/2022) (cit. on p. 9).
- Wikipedia: Sepal (May 23, 2022). *Sepal*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: <https://en.wikipedia.org/w/index.php?title=Sepal&oldid=1089457643> (visited on 09/01/2022) (cit. on p. 21).
- Wikipedia: SYCL (Aug. 1, 2022). *SYCL*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: <https://en.wikipedia.org/w/index.php?title=SYCL&oldid=1101810571> (visited on 08/31/2022) (cit. on p. 10).
- Wikipedia: VM (Aug. 17, 2022). *Virtual machine*. English. Ed. by Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=Virtual_machine&oldid=1104946655 (visited on 08/31/2022) (cit. on p. 9).

Books, technical reports and others

- DIN EN 61508-3 (Feb. 2011). *Funktionale Sicherheit sicherheitsbezogener elektronischer/elektronischer/programmierbarer elektronischer Systeme – Teil 3: Anforderungen an Software (IEC 61508-3:2010); Deutsche Fassung EN 61508-3:2010*. deutsch. Tech. rep. DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE. URL: <https://www.beuth.de/de/norm/din-en-61508-3/135505701> (visited on 09/06/2022) (cit. on pp. 4, 6).
- DIN EN ISO 13849-1 (June 2016). *Sicherheit von Maschinen – Sicherheitsbezogene Teile von Steuerungen – Teil 1: Allgemeine Gestaltungsleitsätze (ISO 13849-1:2015); Deutsche Fassung EN ISO 13849-1:2015*. deutsch. Tech. rep. DIN Deutsches Institut für Normung e. V. URL: <http://www.beuth.de/de/norm/din-en-iso-13849-1/230387878> (visited on 09/06/2022) (cit. on pp. 4, 7).

- Dumitrescu, Roman et al. (Jan. 28, 2018). *Studie "Autonome Systeme"*. Tech. rep. Expertenkommission Forschung und Innovation (EFI), Berlin. URL: <https://www.econstor.eu/handle/10419/175555> (visited on 09/06/2022) (cit. on pp. 3, 6).
- Fisher, R. A. (Sept. 1936). “The use of multiple measurements in taxonomic problems”. In: vol. 7. 2, pp. 179–188. DOI: <https://doi.org/10.1111/j.1469-1809.1936.tb02137.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x> (visited on 08/31/2022) (cit. on p. 21).
- Géron, Aurélien (2018). *Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. Konzepte, Tools und Techniken für intelligente Systeme*. German. Trans. English by Kristian Rother. 1. Auflage. Heidelberg: O'Reilly, p. 552. ISBN: 978-3-96010-114-7. URL: <https://katalog.ub.uni-heidelberg.de/cgi-bin/titel.cgi?katkey=68275614> (visited on 08/30/2022) (cit. on p. 50).
- Hunter, J. D. (June 18, 2007). “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: [10.1109/MCSE.2007.55](https://doi.org/10.1109/MCSE.2007.55). URL: <https://matplotlib.org/stable/users/project/history.html> (visited on 09/09/2022) (cit. on p. 13).
- Jürgensohn, Thomas et al. (Apr. 30, 2021). *Rechtliche Rahmenbedingungen für die Bereitstellung autonomer und KI-Systeme*. German. Tech. rep. BAuA, Bundesanstalt für Arbeitsschutz und Arbeitsmedizin. DOI: [doi:10.21934/baua:bericht20210423](https://doi.org/10.21934/baua:bericht20210423). URL: <http://www.baua.de/dok/8859246> (visited on 09/06/2022) (cit. on pp. 4, 7).
- Kadner, Susanne et al. (Mar. 20, 2017). *Fachforum autonome Systeme – Chancen und Risiken für Wirtschaft, Wissenschaft und Gesellschaft*. Tech. rep. acatech – Deutsche Akademie der Technikwissenschaften. URL: <https://www.acatech.de/publikation/fachforum-autonome-systeme-chancen-und-risiken-fuer-wirtschaft-wissenschaft-und-gesellschaft-abschlussbericht/> (visited on 09/06/2022) (cit. on pp. 3, 6).
- Karimi, Kamran, Neil G. Dickson, and Firas Hamze (May 16, 2011). “A Performance Comparison of CUDA and OpenCL”. In: DOI: <https://doi.org/10.48550/arXiv.1005.2581>. URL: <https://arxiv.org/abs/1005.2581> (visited on 08/31/2022) (cit. on p. 9).
- Kuhn, Thomas and Peter Liggesmeyer (Jan. 30, 2019). “Autonome Systeme – Potenziale und Herausforderungen: Perspektiven, Herausforderungen und Grenzen der Künstlichen Intelligenz in der Arbeitswelt”. In: *Autonome Systeme und Arbeit*. transcript Verlag, pp. 27–46. ISBN: 978-3-8376-4395-4. DOI: [10.14361/9783839443958-003](https://doi.org/10.14361/9783839443958-003). URL: https://www.researchgate.net/publication/338265062_Autonome_Systeme_Potenzziale_und_Herausforderungen_Perspektiven_Herausforderungen_und_Grenzen_der_Kunstlichen_Intelligenz_in_der_Arbeitswelt (visited on 09/06/2022) (cit. on pp. 4, 6).
- Raschka, Sebastian and Vahid Mirjalili (2018). *Machine Learning mit Python und Scikit-learn und TensorFlow. Das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics*. German. Trans. English by Knut Lorenzen. 2nd ed. mitp Verlags GmbH & Co. KG, Frechen, p. 577. ISBN: 978-3-95845-733-1. URL: <https://katalog.ub.uni-heidelberg.de/cgi-bin/titel.cgi?katkey=68206280> (visited on 08/30/2022) (cit. on p. 35).
- Waskom, Michael L. (Apr. 6, 2021). “seaborn: statistical data visualization”. In: *Journal of Open Source Software* 6.60, p. 3021. DOI: [10.21105/joss.03021](https://doi.org/10.21105/joss.03021). URL: <https://joss.theoj.org/papers/10.21105/joss.03021> (visited on 09/09/2022) (cit. on p. 13).