Machine Learning mit Python

EXXETA
CONSULTING AND TECHNOLOGIES

Schulung der GfU

Philipp Bongartz

10.-12. Mai 2021

GFU Cyrus AG

Biographie

- EXXETA, Mannheim
- PhD Bioinformatik, HITS, Heidelberg
- Mathematik-Diplom, Universität Bonn

Beratungskompetenz

- Softwareentwicklung Python, C
- Data Science & Deep Learning:Keras, Tensorflow, SciPy, Numpy, Matplotlib, Jupyter Notebook, Pandas, ScikitLearn
- DWH & ETL: T-SQL, SQL-Server, SSIS

Sprachen

- Deutsch
- Englisch

Dr. Philipp Bongartz

Consultant

Dr. Philipp Bongartz programmiert seit 2013 unterschiedlichste Anwendungen in Python mit einem Fokus auf performante Algorithmen und Machine Learning. Er sammelte im Rahmen seiner Promotion Erfahrung im Deep Learning, im klassischen Algorithm Design und in der statistischen Analyse großer Datenmengen. Bei EXXETA ist er neben den Themen Python Programmierung und Machine Learning auch im Data Warehousing aktiv.

Auszug relevante Projekterfahrung

Lead Programmer, Applikation zum Auslesen von Personalausweisen, Produktentwicklung

- Entwicklung der Python Codebase für eine ML-Pipeline zum Auslesen von Personalausweisen mit Komponenten aus Bilderkennung und -bearbeitung, OCR, Optimierung, Dynamic Programming, Clustering, sowie Deep Learning.

Lead Programmer, Applikation zum automatischen Matchen von Consultants und Stellenausschreibungen

- Betreuung und Weiterentwicklung der Python Codebase einer serverbasierten KI-Anwendung für die automatische Auswertung von Stellenausschreibungen

DWH Entwickler, Betreuung und Weiterentwicklung einer Softwarelösung für die Datenversorgung von über 90 Applikationen aus verschiedenen SAP Systemen

- Anforderungsaufnahme und Analyse sowie Anpassen von ETL-Strecken, Datenbanken und Reports
- Sicherstellung von Datenkonsistenz & Performanceoptimierung
- Automatisierte ETL-Streckenimplementierung durch Scripting
- Onboarding Schulungen & Enabling für Projektübergabe zu Kunden
- Technologien: MS SQL Server, SSIS, Theobald XTract IS (für SAP BW, -CRM, -ECC, -SCM), SAP SolMan

Data Scientist, Predictive Maintenance auf Basis von Sensordaten, Schienen-Logistik Firma

- Entwicklung eines Self-Supervised Deep Learning Models zur Klassifizierung von Zeitreihen
- Entwicklung einer regelbasierten Zeitreihenklassifikation für Batteriespannungsverläufe im Rahmen von Monitoring und Predictive Maintenance

AGENDA

- **Tag 1**
  - Machine Learning Einführung
  - Panda Basics
  - Data Preprocessing
  - Lineare Regression
  - Korrelation
  - Decision Trees
  - Clustering
- **Tag 2**
  - Deep Learning Einführung
  - Numpy Basics
  - Gradient Descent & Generalisierung
  - Convolutional Neural Networks
  - Natural Language Processing
- **Tag 3**
  - Visualisierung
  - ML Ops

Inhalt

Daten ziehen und aufbereiten

- Dateien im Verzeichnis ansteuern
- Standardverfahren zum Lesen/Schreiben von Text- und CSV-Dateien
- *SQL*-Datenbanken ansteuern
- Arbeiten mit Datenmatrizen: Einführung in *Numpy* & *Pandas*
- Daten inspizieren und beschreiben
- Fehlende Werte behandeln

Machine Learning Grundlagen

- Grundlagen, Varianten und Techniken des Machine Learnings.
- Arbeiten mit der Machine Learning Bibliothek scikit-learn.
- Einfache Zusammenhänge zwischen stetigen Variablen modellieren: Lineare Regression
- Klassifizieren mit Logistic Regression, Softmax und Support Vector Machine.
- Modellen evaluieren: Accuracy, Precision, Recall & Confusion matrix
- Unterteilen der Daten in Trainings- und Testdaten

Feature-Extraction

- Kategoriale Daten vorbereiten: *One-Hot Codierung*
- Standardisierung von Daten
- Dimensionsreduktion mit PCA (Principle Component Analysis)
- Aufbereitung von Textdaten: Tokenizer und Bag-of-words.

Machine Learning Workflow

- Daten aufarbeiten und zusammenführen (DataMapper)
- Pipelines einrichten
- Speichern und laden trainierter Klassifizierer

Beschreibung

Das Seminar gibt eine Einführung in Konzepte und in die Praxis des Machine Learnings. Es zeigt Einsatzoptionen, erklärt statistische Grundlagen und schult anhand praktischer Beispiele die Anwendung der gängigsten Lernalgorithmen aus der *Scikit-Learn* Bibliothek. Neben der Klassifizierung und Prognostizierung von Daten, liegt der Schwerpunkt auf der Vorverarbeitung der Daten und der Extraktion relevanter Variablen für den Anlernprozess.

Schulungsziel

Sie lernen die Grundlagen und die praktische Anwendung der gängigsten Lernalgorithmen aus der *Scikit-Learn* Bibliothek. Sie bekommen ein Gespür für den Wert ihrer Daten und dafür, wie Sie diese Daten in Ihrem Geschäftsfeld einsetzen können, um Automatisierungsprozesse voranzutreiben. Sie verstehen die verschiedenen Lernalgorithmen in Theorie und Praxis (lineare und logistische Regression, Support Vector Machine, Decision Tree, Naive Bayes) und üben die Anwendung anhand praktischer Beispiele ein. Die Konzepte werden anhand von Folien anschaulich erklärt, an Beispielen verdeutlicht und gemeinsam in *Python* umgesetzt. Passgenaue Aufgabenstellungen ergänzen den Lernprozess und ermöglichen es Ihnen, die verschiedenen Lernszenarien kennenzulernen und einzuüben. Am Ende des Seminars sind Sie in der Lage, Daten zielsicher zu extrahieren, Algorithmen anzulernen und zur Klassifizierung oder Prognose einzusetzen.

https://github.com/PhilippBongartz/GfU_ML_mit_Python

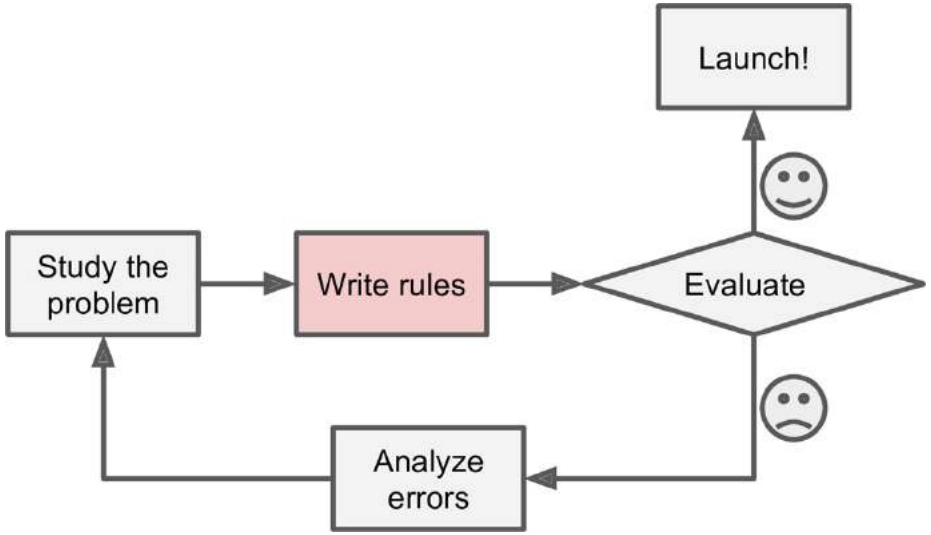
Vorstellungsrunde

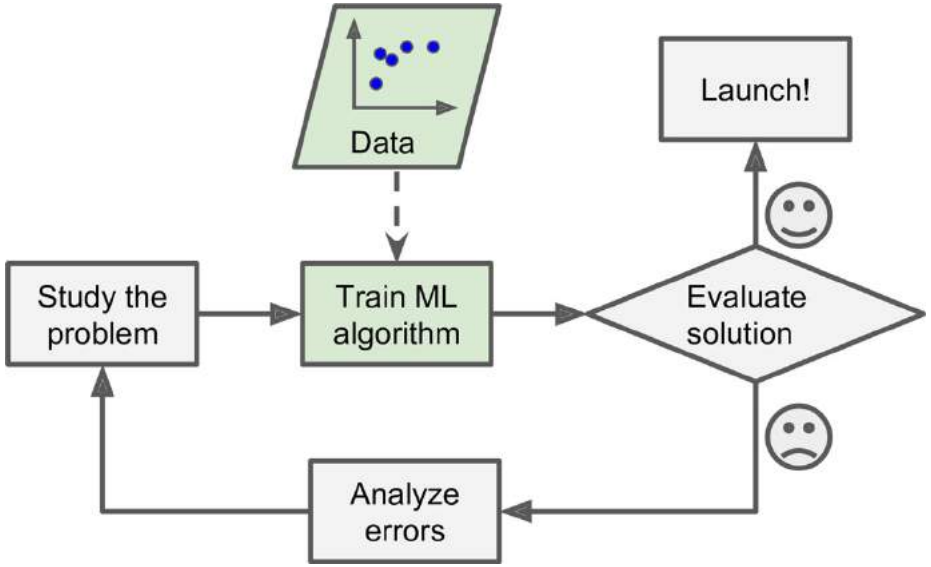
- Wie heißt Ihr?
- Was habt Ihr für einen Hintergrund?
- Was führt Euch hierher?
- Warum interessiert Euch Machine Learning?

Machine Learning Überblick

Was ist Machine Learning?

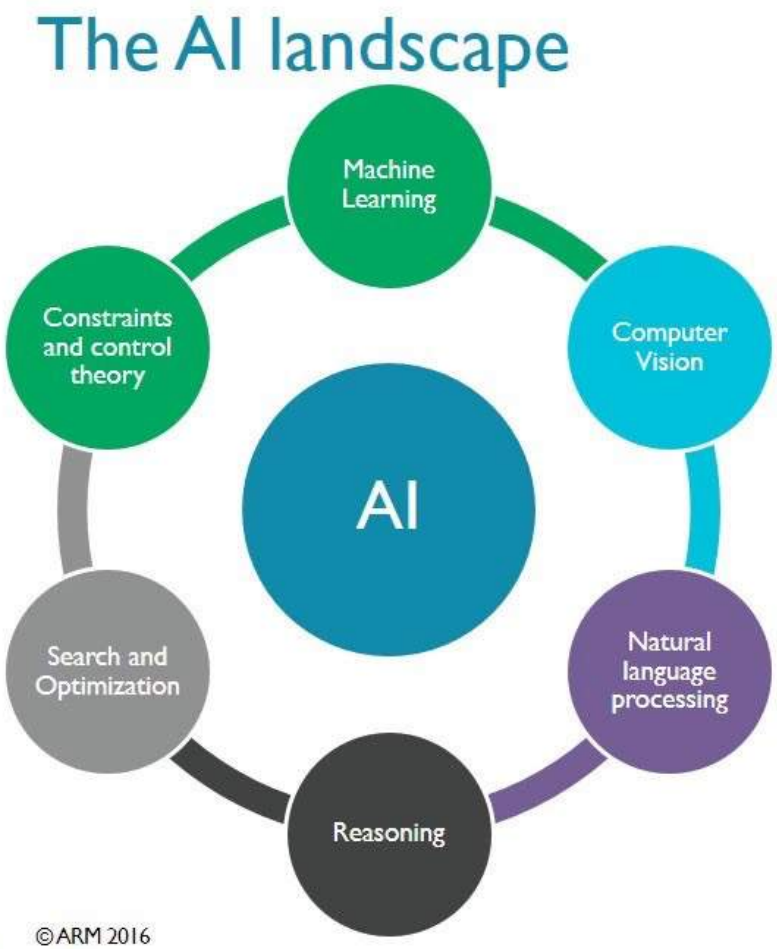
Algorithmen, die aus Daten lernen oder Struktur in Daten entdecken.

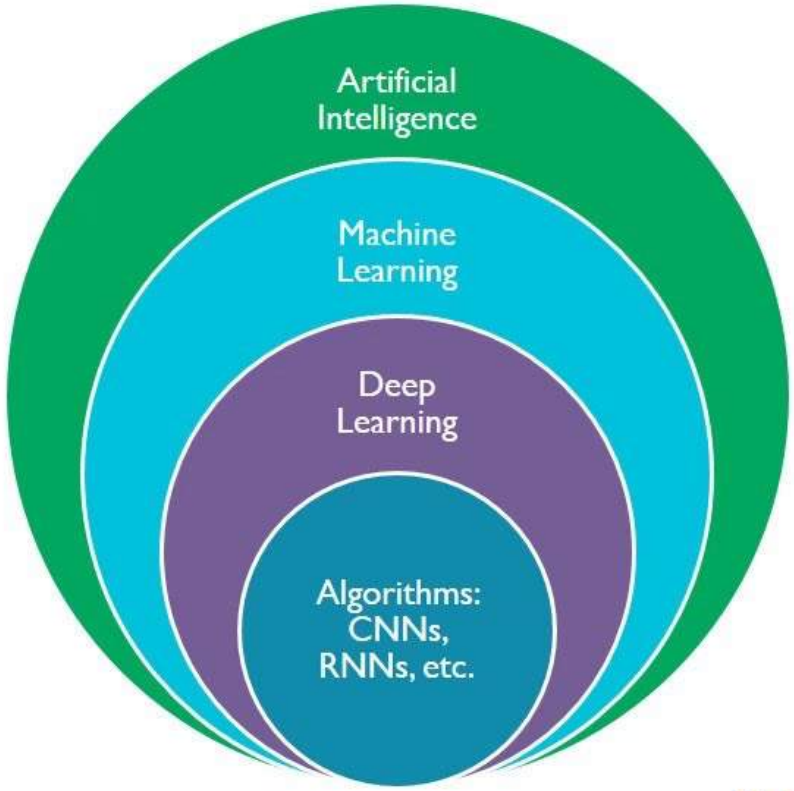




Aurélien Géron (2017), Hands-On Machine Learning with Scikit-Learn and TensorFlow, Publisher: O'Reilly Media, Inc., ISBN: 9781491962299

Artificial Intelligence and Machine Learning

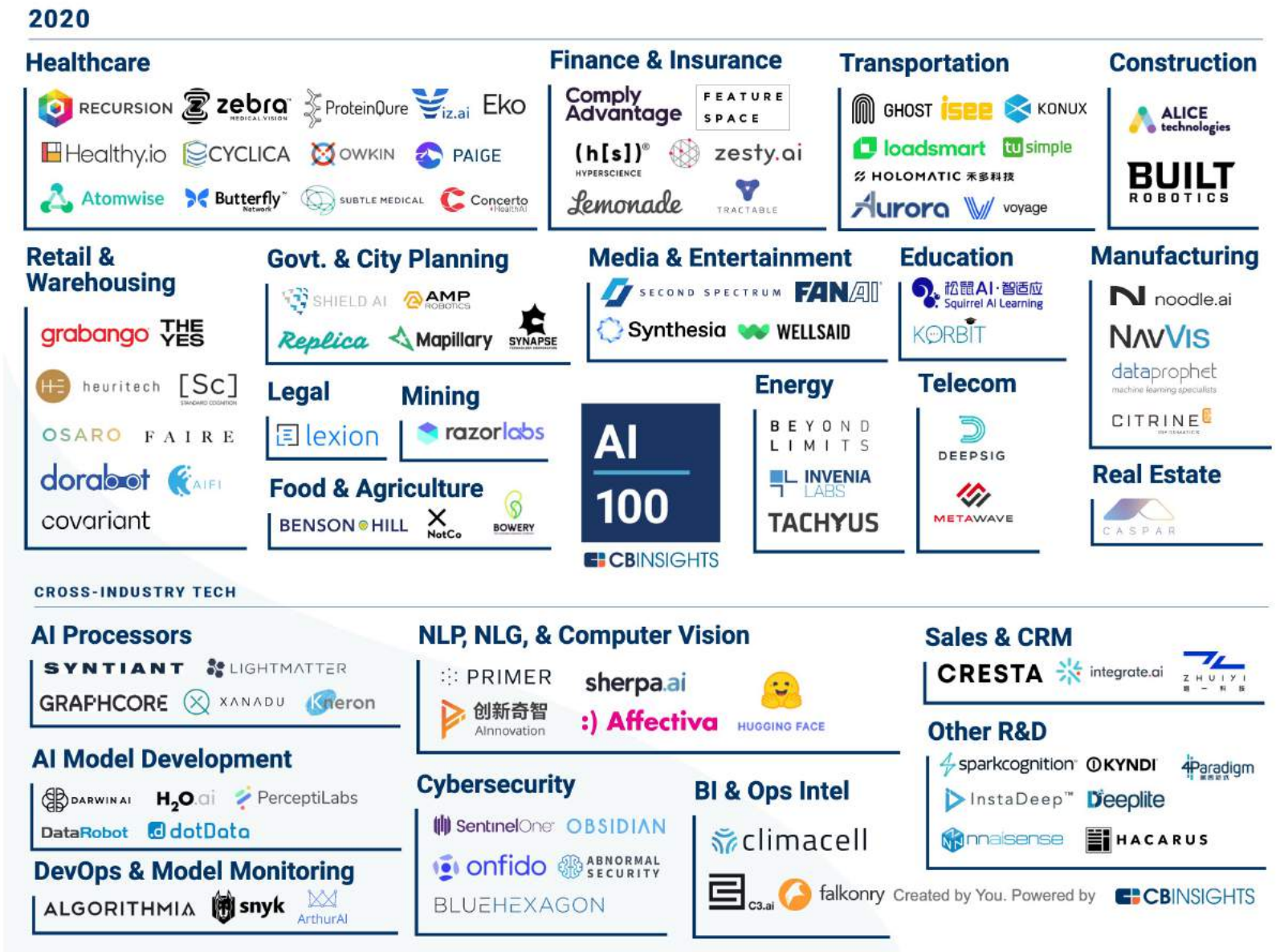




ARM

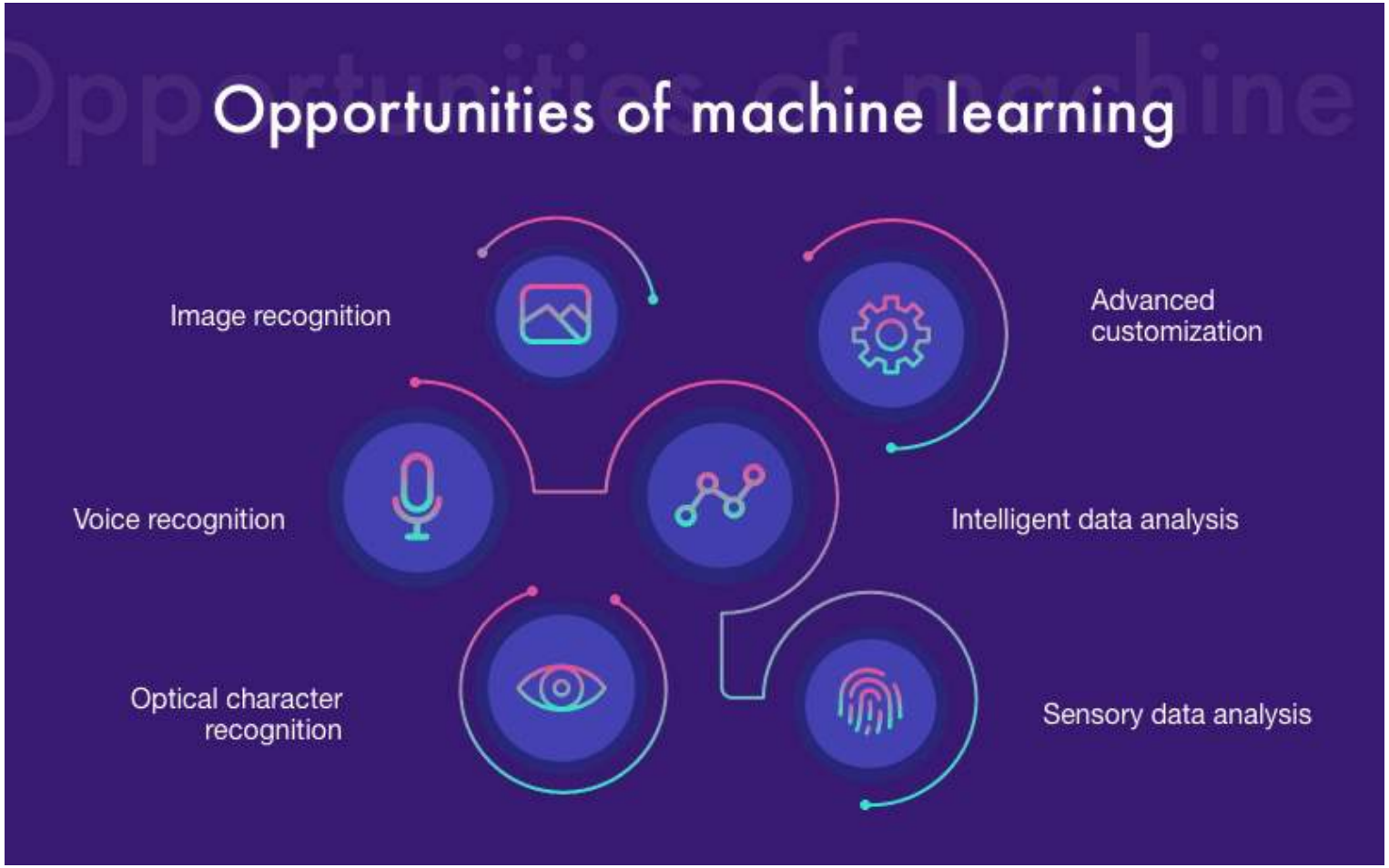
Startups and AI

The AI 100 is CB Insights' annual ranking of the 100 most promising AI startups in the world.



https://www.cbinsights.com/research/artificial-intelligence-top-startups/

Was kann man damit alles machen?



Arten des Machine Learning

1. **Supervised Learning**
2. **Unsupervised Learning**
3. **Reinforcement Learning**

…

sind drei klar unterschiedene Hauptbereiche

Supervised Learning

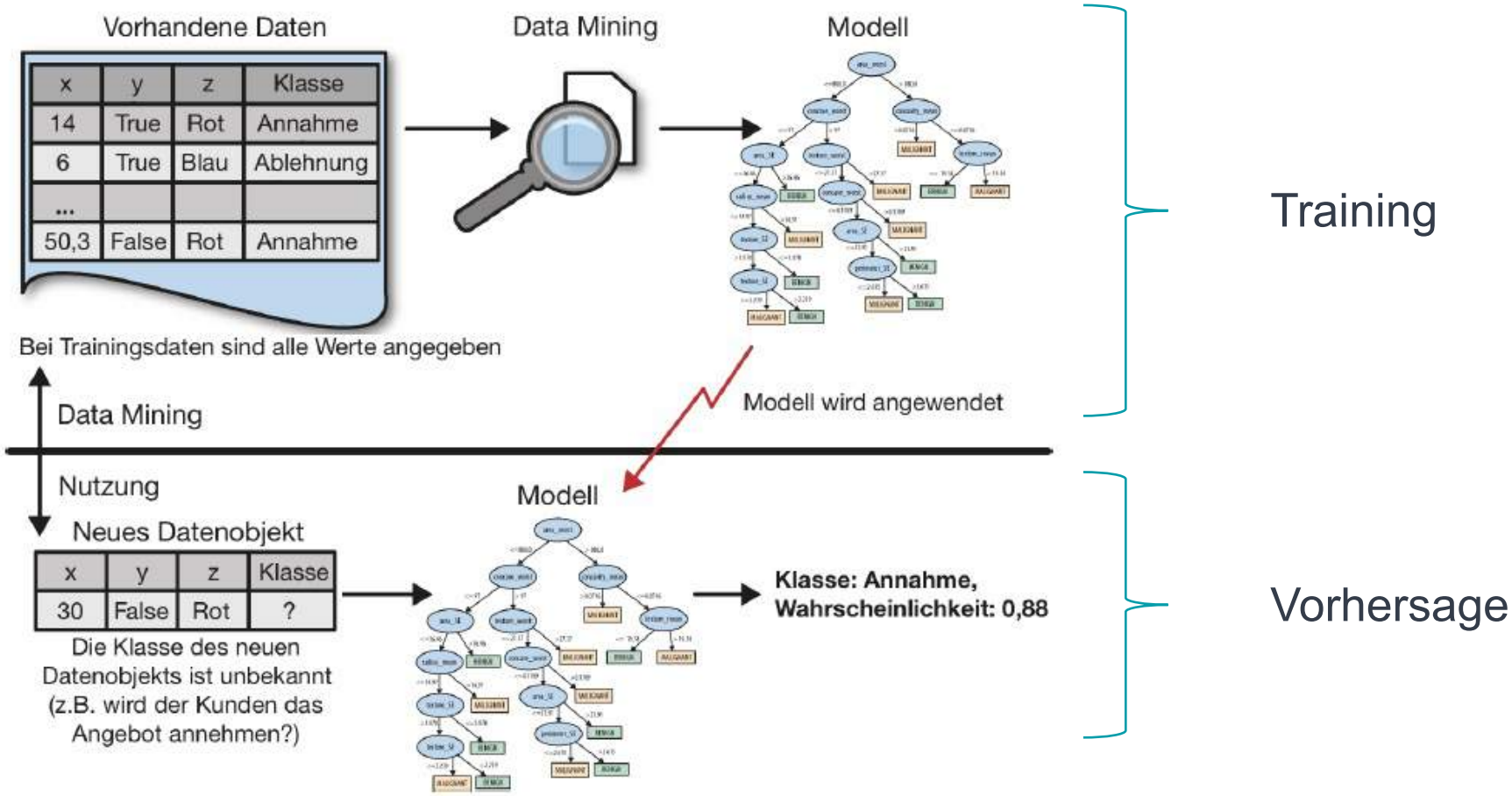
Fragestellung Supervised Learning:

z.B.: Gibt es Kunden, welche eine hohe Wahrscheinlichkeit besitzen ihren Vertrag zu kündigen?

- Training erfolgt auf historischen Daten, welche alle gelabelt sind.
- Ziel ist es durch die vorliegenden Daten (Features), die Zielvariable so genau wie möglich vorhersagen zu können und ggf. eine Wahrscheinlichkeit der Klassenzugehörigkeit zu ermitteln.
- Vorhersage erfolgt mit Daten, bei denen das Target oder die Klasse fehlt.

Live Demo
Teachable Machine

Supervised Learning



Source: Provost, Foster and Tom Fawcett, Data Science für Unternehmen, S. 52

Supervised Learning – Schlüsselkonzept Train-, Test-, Validationsplit

Bevor man ein Machine Learning Modell trainiert, sollte man immer den Datensatz mindestens in Train- und Validationset unterteilen.

Performance auf dem Trainingsdatensatz garantiert nicht die Performance auf ungesehenen Daten!

Die Terminologie ist nicht immer sehr einheitlich. Manchmal spricht man auch von Training, Development und Testdatensätzen. Der dritte Datensatz ist dafür da, dass man nicht durch optimieren der Hyperparameter das Modell an den zweiten Datensatz überanpasst.

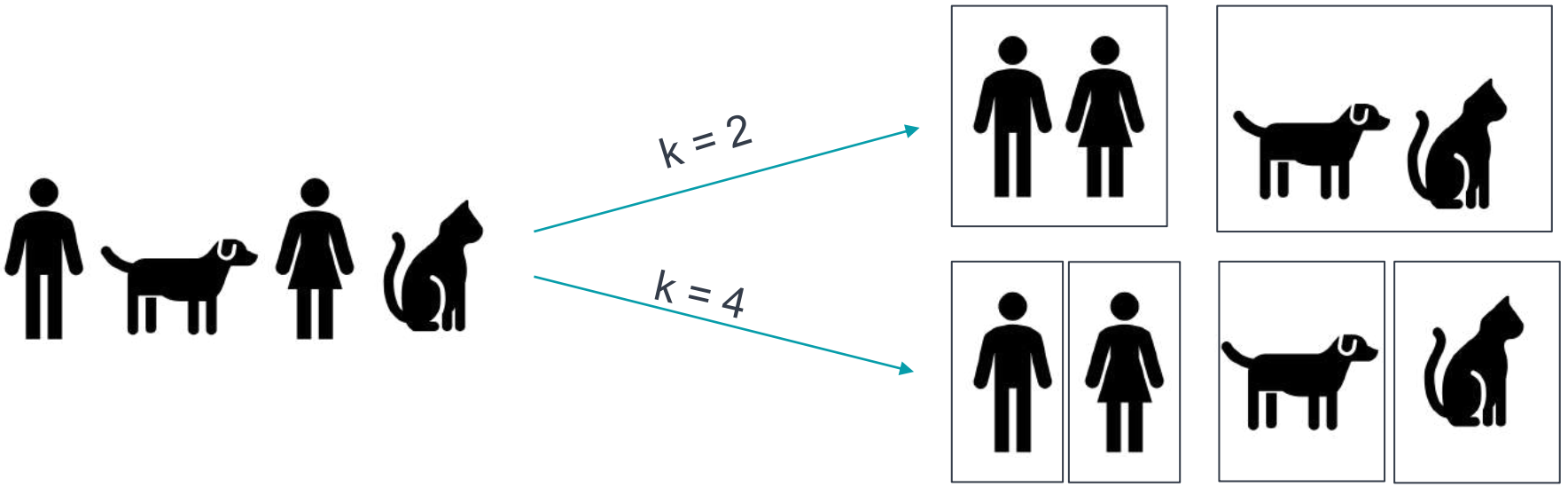
Der dritte Datensatz wird idealerweise nur ein **einziges** Mal ausgewertet!

Unsupervised Learning

Fragestellung Unsupervised Learning:

z.B.: Können Accounts verschiedener Nutzer in bestimmte Gruppen aufgeteilt werden?

- Keine Angaben über ein bestimmtes Ziel in den Daten vorhanden
- Ziel wird allein anhand vorliegender Daten und deren Ähnlichkeit zueinander bestimmt



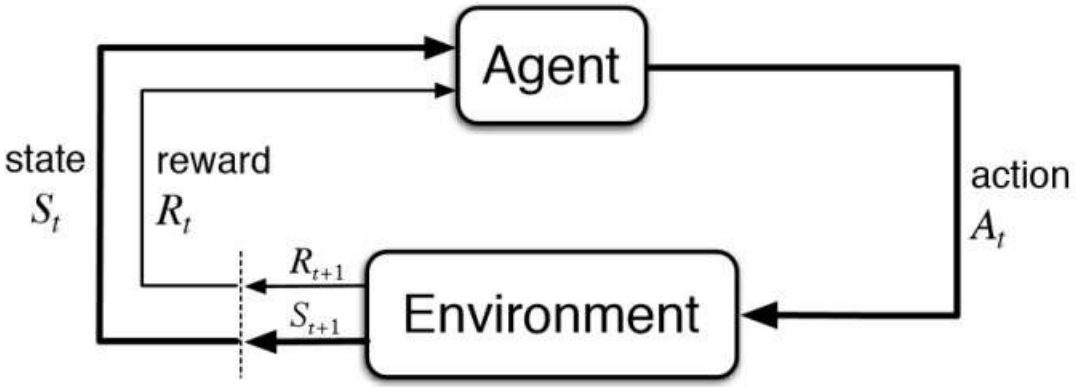
Unsupervised Learning

Arten des Unsupervised Learning:

- Clustering (z.B. kMeans oder DBSCAN)
- Dimensionsreduzierung (z.B. PCA)
- Visualisierung (z.b. t-SNE)
- Einbettungen oder Featurelearning (z.b. über Contrastive Loss)
- Lernen von Wahrscheinlichkeitsverteilungen über Daten (Self-Supervised)

Reinforcement Learning

- Sogenannter „Agent“ wird trainiert
- „Agent“ führt Aktionen aus und bekommt Rückmeldung von Umgebung → z.B.: Belohnung bei Bewegung in die richtige Richtung.
- Verwendung: Robotics, Gaming-Bots, AlphaGo,…



Source: https://www.google.com/search?q=reinforcement+learning&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiw-fG11IrwAhXUgf0HHdJxAwQQ_AUoAnoECAEQBA&biw=1920&bih=937#imgrc=ivmsD3gf5LO7WM

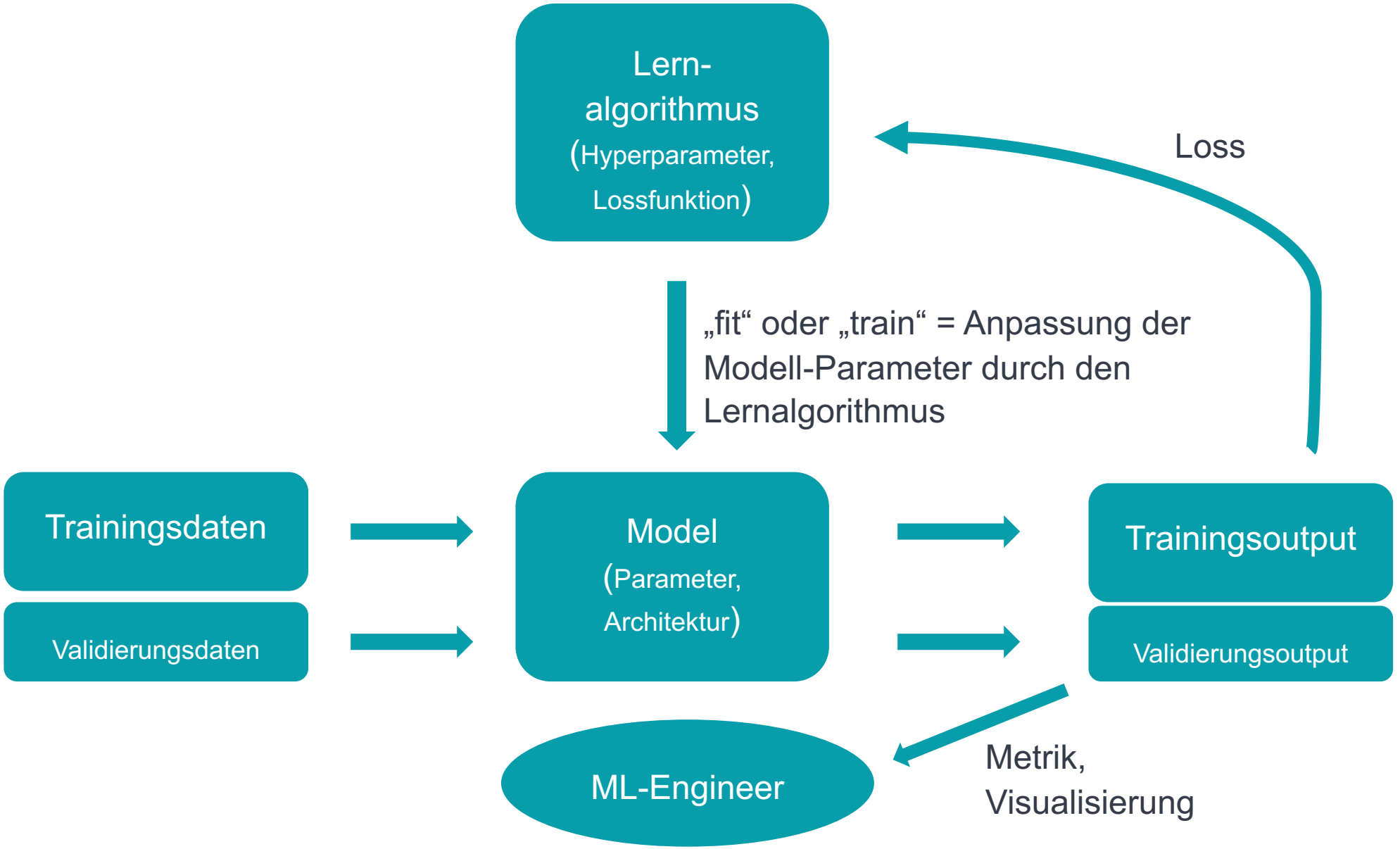
Herausforderungen Machine Learning

- Machine Learning und Predicitive Applications bringen im Vergleich zu traditionellen Software Systemen **neue Herausforderungen** mit sich
  - Unzureichende Trainingsdaten
  - Trainingsdaten sind nicht repräsentativ (sampling bias)
  - Schlechte Datenqualität (Fehlwerte → oder Fehlwerte, die nicht als solche erkennbar sind Bsp.: als „unknown“ gekennzeichnet)
  - Irrelevante oder unzureichende Features
  - Overfitting → grundlegendes Problem ist Bias/Variance Trade-Off
  - Silent failure → Bugs zeigen sich oft nur indirekt durch etwas schlechtere Performance

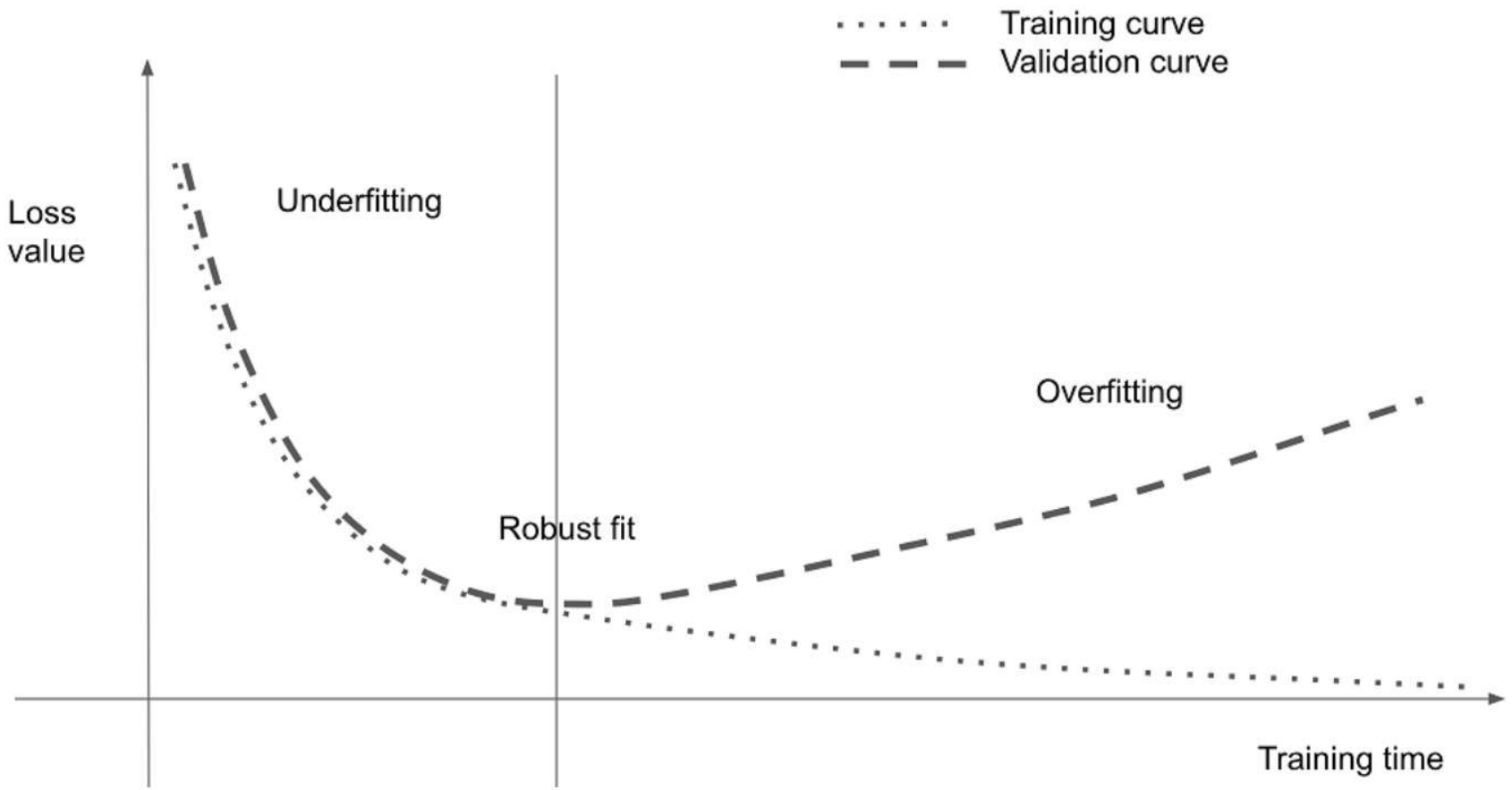
- **„Garbage in, garbage out“ ist zu beachten !**

→ Besonders bei der Automatisierung von Geschäftsentscheidungen

Schlüsselbegriffe Machine Learning

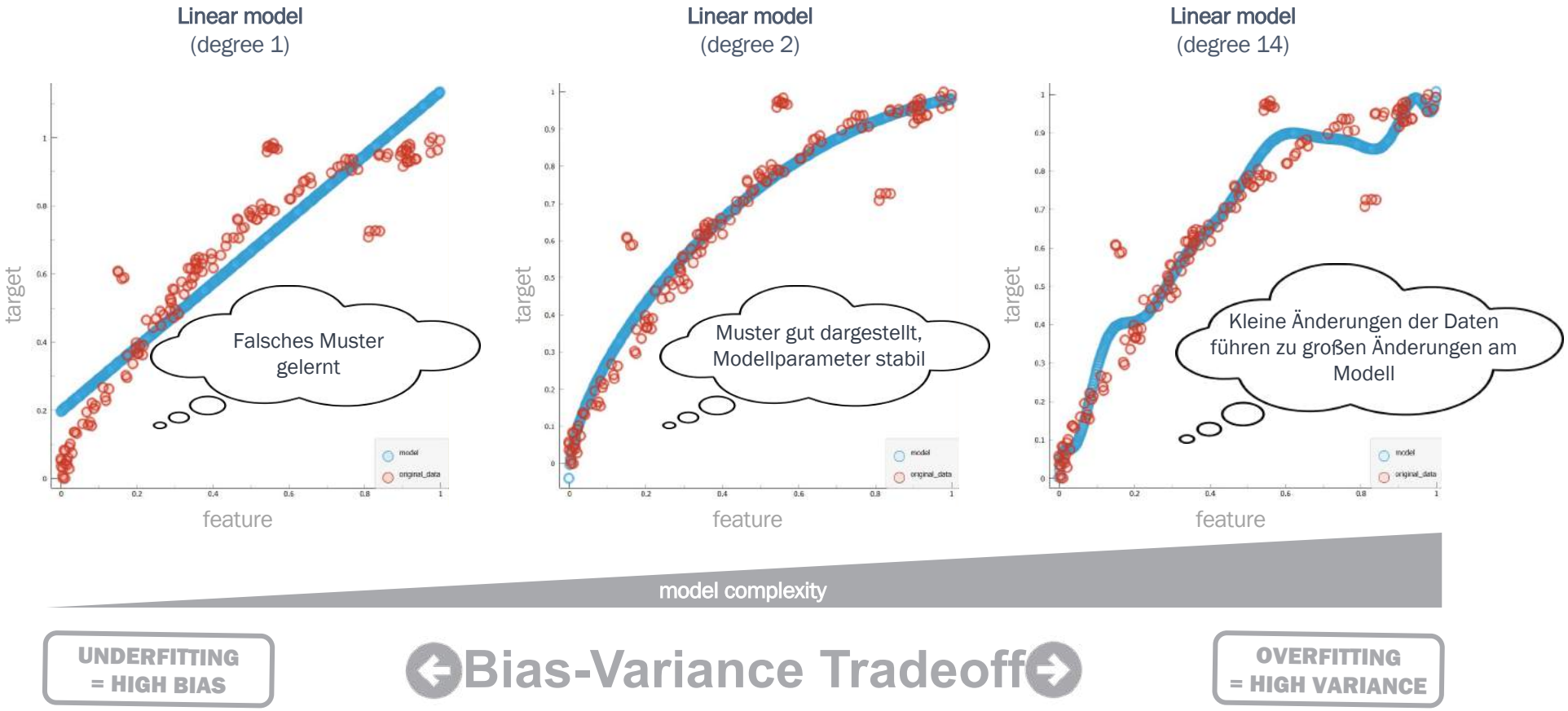


Typische Overfitting Story



Live Demo
Orange – Bias Variance Tradeoff

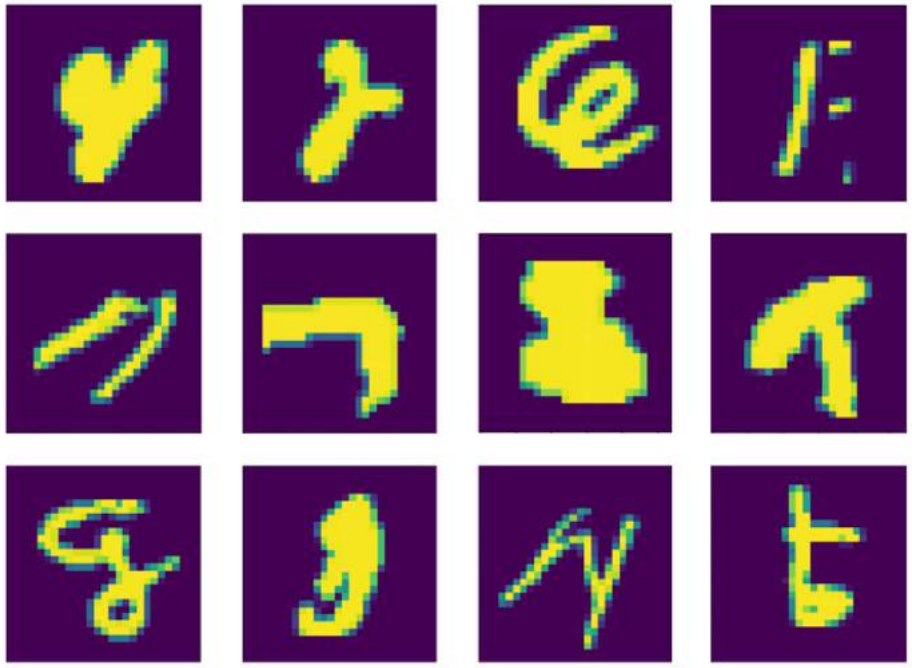
Bias Variance Tradeoff - Fazit

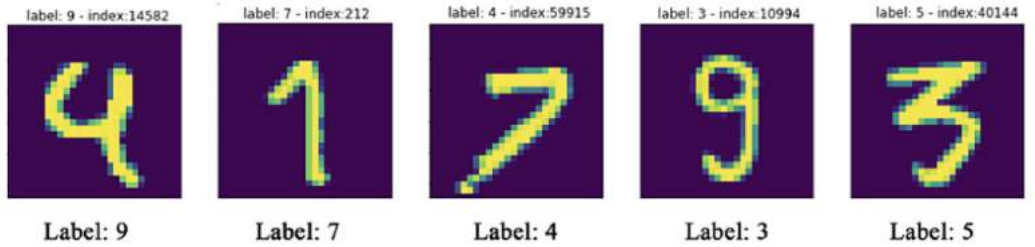


Küppers, P. , MW330 – L2B – Bias Variance Tradeoff, S.15; eigene Übersetzung

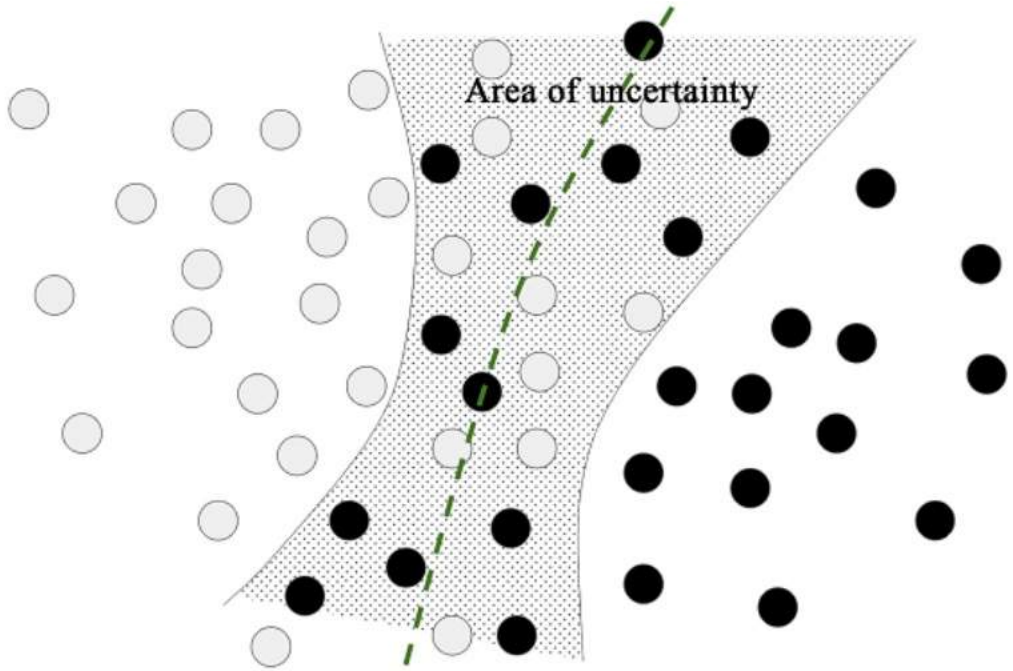
Noise auswendig lernen

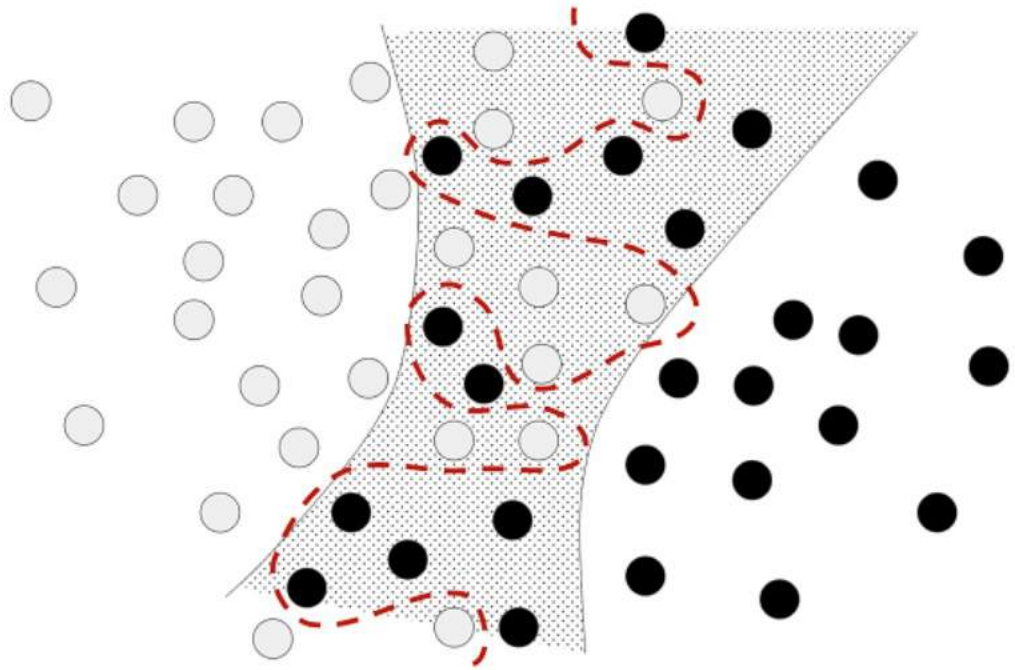
- Falsche Labels
- Seltene Features
- Ambiguität
- Rauschen





Das Ergebnis: Schlechte Generalisierung



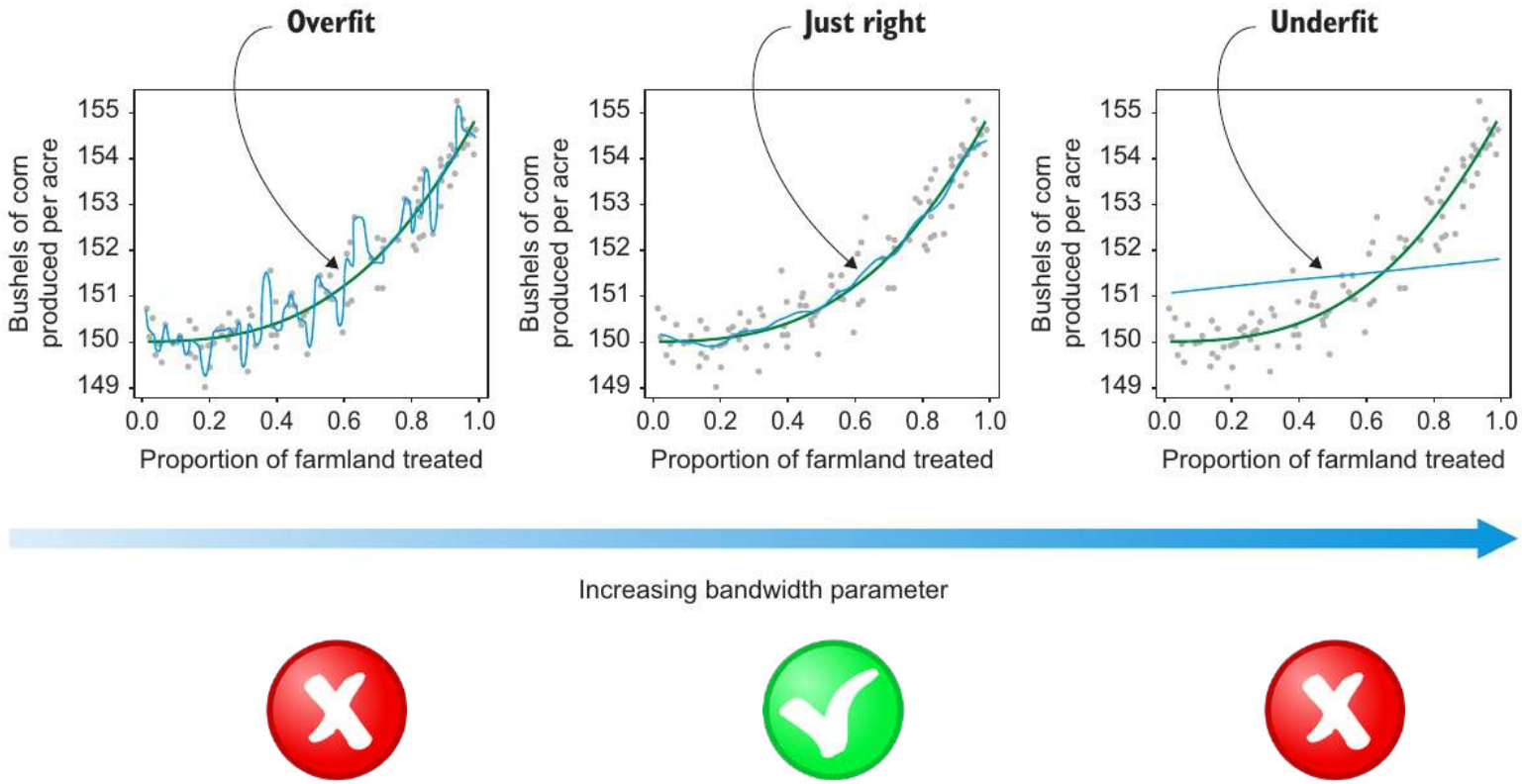


Einflüsse auf Overfitting

- Learning rate → Saturisierung
- Datenmenge
- Datenkomplexität
- Modelgröße
- Regularisierung

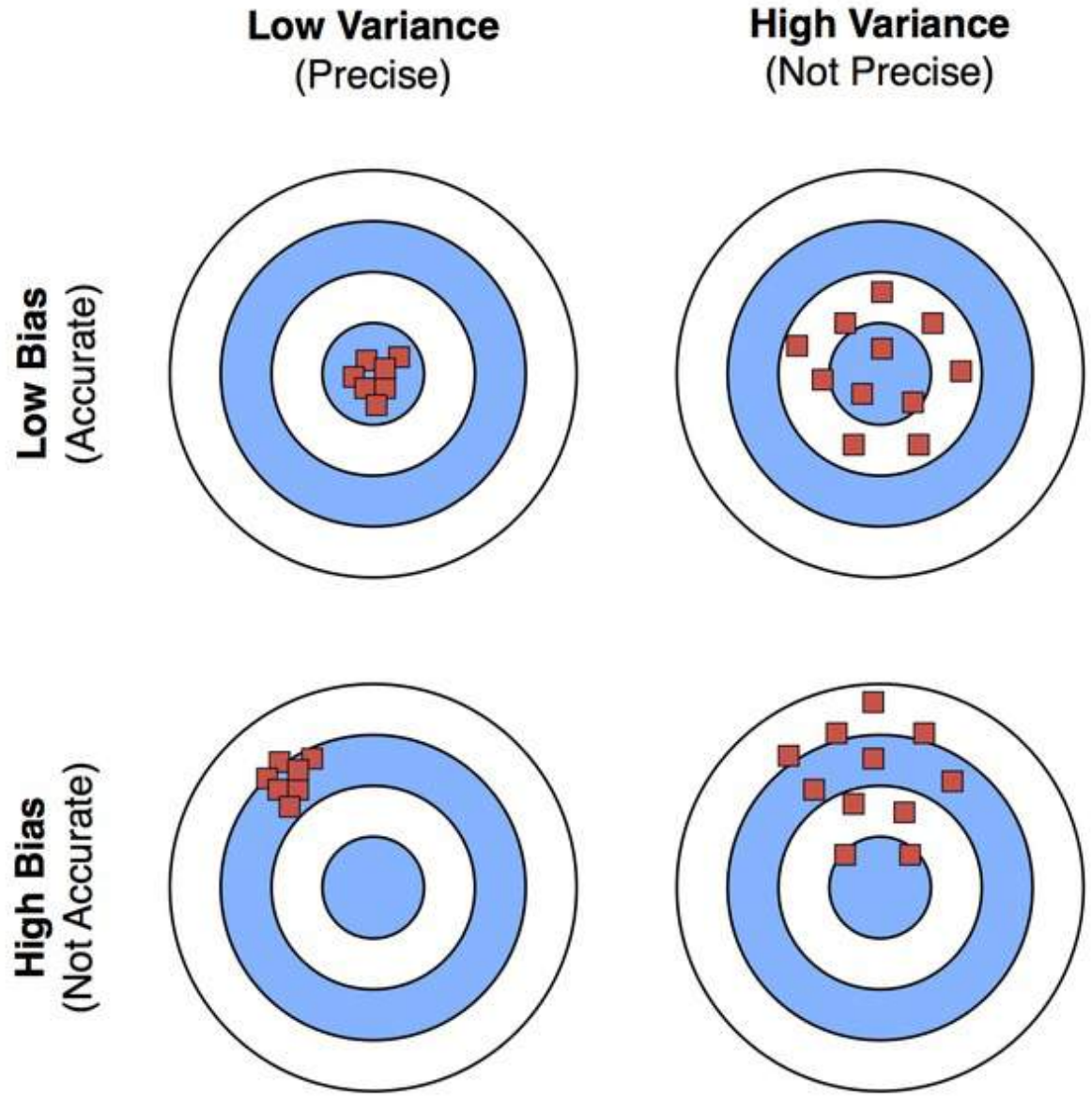
Overfitting und Underfitting

Lernt das Modell auswendig? Verallgemeinert es genau richtig? Oder lernt es zu wenig?



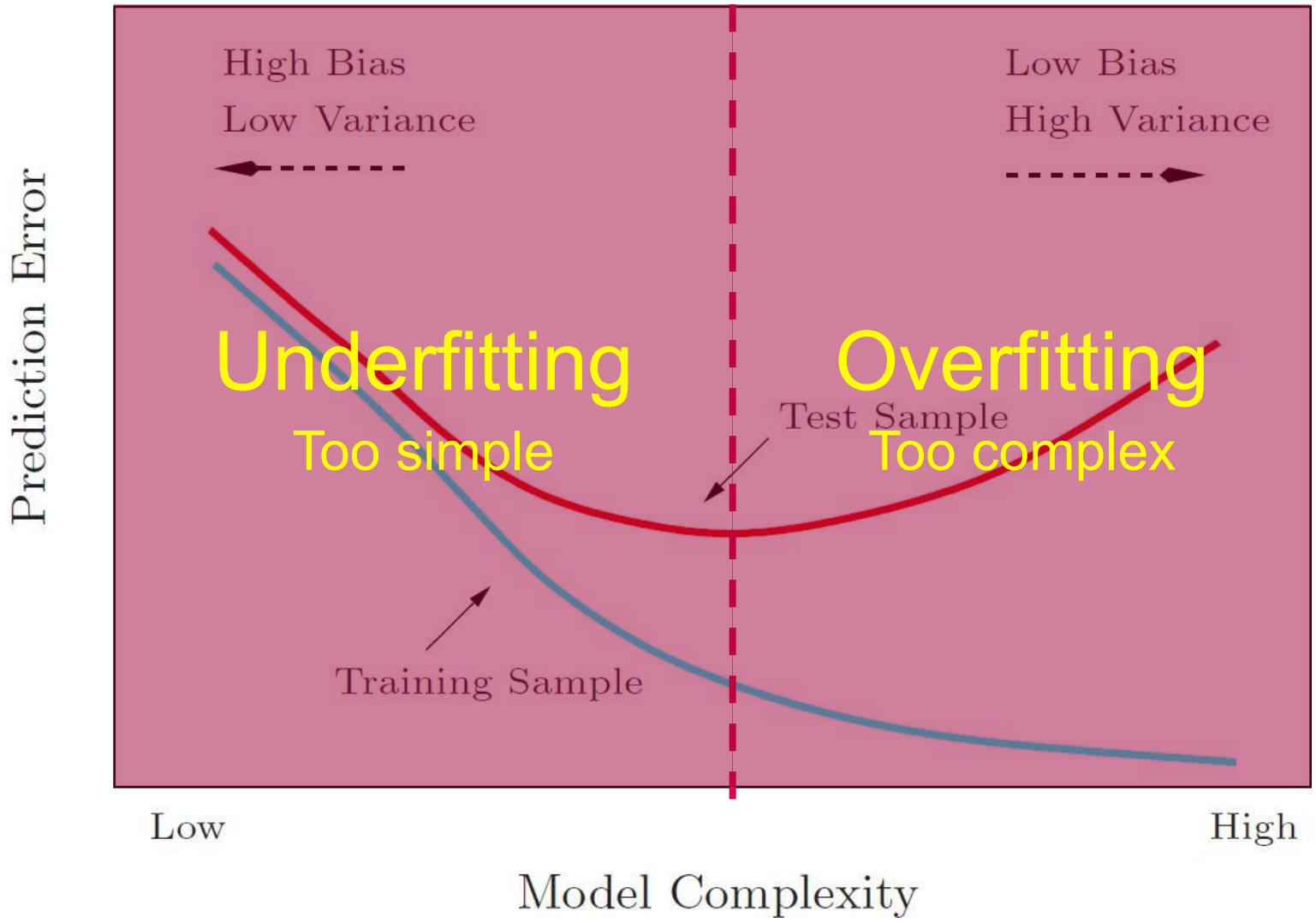
Bias-Variance-Tradeoff

Feste Struktur oder maximale Flexibilität?



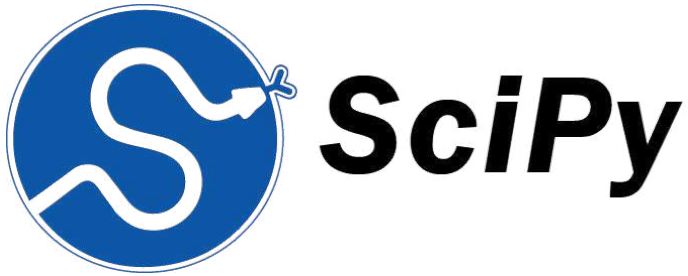
Bias-Variance-Tradeoff

Overfitting und andere Probleme



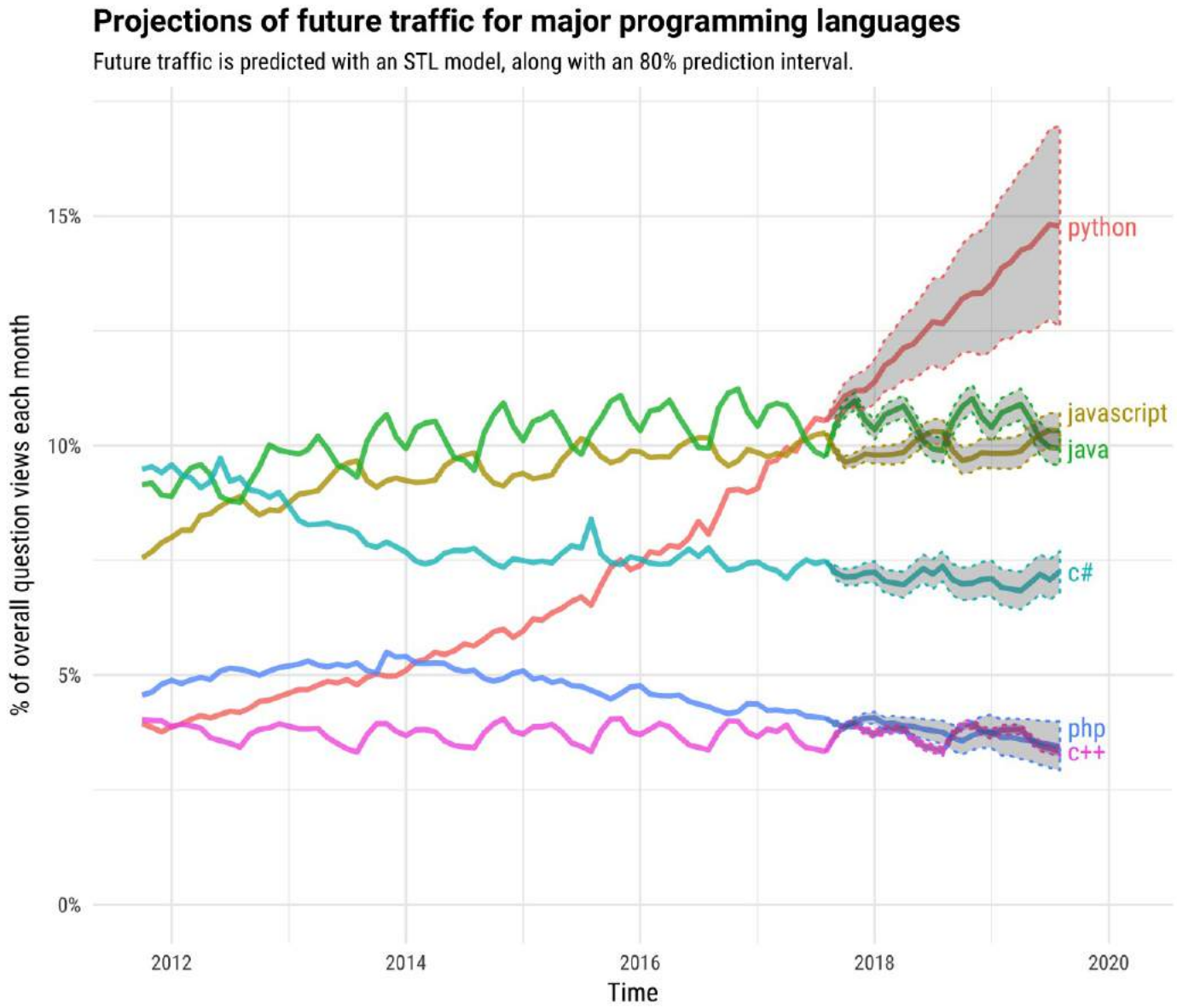
Der Machine Learning Tech Stack

Python und Python-Libraries die aufeinander aufbauen …



Quellen:
WikiCommons, Python.org,
fullstackpython.com,

Warum Python



Source: Stack Overflow
https://medium.com/@UdacityINDIA/why-use-python-for-machine-learning-e4b0b4457a77

Was ist Data Science?

Ein relativ neues Berufsbild – Digitale Datenanalyse auf der Basis von Statistik und Machine Learning

Notebook für interaktive Datenanalyse, Visualisierung und Annotation.

Was ist Data Science?

Ein relativ neues Berufsbild – Digitale Datenanalyse auf der Basis von Statistik und Machine Learning

„Numerical Python“ – optimierte Algorithmen um
mit Vektoren und Matrizen umzugehen.

Was ist Data Science?

Ein relativ neues Berufsbild – Digitale Datenanalyse auf der Basis von Statistik und Machine Learning

Library um tabellarische Daten zu manipulieren. Excel in mächtig.

Was ist Data Science?

Ein relativ neues Berufsbild – Digitale Datenanalyse auf der Basis von Statistik und Machine Learning

Standard Visualisierungslibrary in Python

seaborn

Wenn es gut aussehen soll …

ScikitLearn und SciPy

Die beiden wichtigsten “reinen” Machine Learning Libraries

Machine Learning basiert,
d.h. um das Konzept eines
Modells herum aufgebaut.
Baut auf SciPy auf.

Scientific Computing basiert,
d.h. stärkerer Fokus auf Datenanalysen,
z.b. FFT, PCA, Integration, LinAlg,
Signalverarbeitung, Solver etc.

Scikit-learn Design

https://arxiv.org/abs/1309.0238

Scikit-Learn's API is remarkably well designed. These are the main design principles:

Consistency: All objects share a consistent and simple interface:

- **Estimators** Any object that can estimate some parameters based on a dataset is called an *estimator* (e.g., an imputer is an estimator). The estimation itself is performed by the fit() method, and it takes only a dataset as a parameter (or two for supervised learning algorithms; the second dataset contains the labels). Any other parameter needed to guide the estimation process is considered a hyperparameter (such as an imputer's strategy), and it must be set as an instance variable (generally via a constructor parameter).
- **Transformers** Some estimators (such as an imputer) can also transform a dataset; these are called *transformers*. Once again, the API is simple: the transformation is performed by the transform() method with the dataset to transform as a parameter. It returns the transformed dataset. This transformation generally relies on the learned parameters, as is the case for an imputer. All transformers also have a convenience method called fit_transform() that is equivalent to calling fit() and then transform() (but sometimes fit_transform() is optimized and runs much faster).
- **Predictors** Finally, some estimators, given a dataset, are capable of making predictions; they are called *predictors*. For example, the LinearRegression model in the previous chapter was a predictor: given a country's GDP per capita, it predicted life satisfaction. A predictor has a predict() method that takes a dataset of new instances and returns a dataset of corresponding predictions. It also has a score() method that measures the quality of the predictions, given a test set (and the corresponding labels, in the case of supervised learning algorithms).[18]

Inspection: All the estimator's hyperparameters are accessible directly via public instance variables (e.g., imputer.strategy), and all the estimator's learned parameters are accessible via public instance variables with an underscore suffix (e.g., imputer.statistics_).

Nonproliferation of classes Datasets are represented as NumPy arrays or SciPy sparse matrices, instead of homemade classes. Hyperparameters are just regular Python strings or numbers.

Composition Existing building blocks are reused as much as possible. For example, it is easy to create a Pipeline estimator from an arbitrary sequence of transformers followed by a final estimator, as we will see.

Sensible defaults Scikit-Learn provides reasonable default values for most parameters, making it easy to quickly create a baseline working system.

Entwicklungsumgebung

Python, Jupyter Notebook, Keras, Colab …

Jupyter Notebook – Google Colab

- Notebook vereint Text und Code in einzeln ausführbaren Code-Zellen
- Interaktiv, gut debugbar und gut dokumentierbar

Colab ist ein kostenloses online Notebook von Google

https://colab.research.google.com

- Mit lokaler Laufzeit verbinden → Datei → Neues Notebook
- Laufzeit → Laufzeit ändern → GPU
- !pip install package_name , aber Keras und Tf sind vorinstalliert.

Der Plan

- Jeder öffnet ein Jupyter Notebook

  Startmenü → Anaconda Prompt → >jupyter notebook
- Wir versuchen tensorflow zu importieren
- Außerdem numpy, keras, matplotlib … sollte von der IT vorinstalliert sein

- Eventuell Zugriff auf Colab ausprobieren (Gmail Account vorhanden?)

- Alternativ: Direkt auf Eurem Computer arbeiten. Macht eventuell mehr Sinn.

Jupyter Basics und Markdown

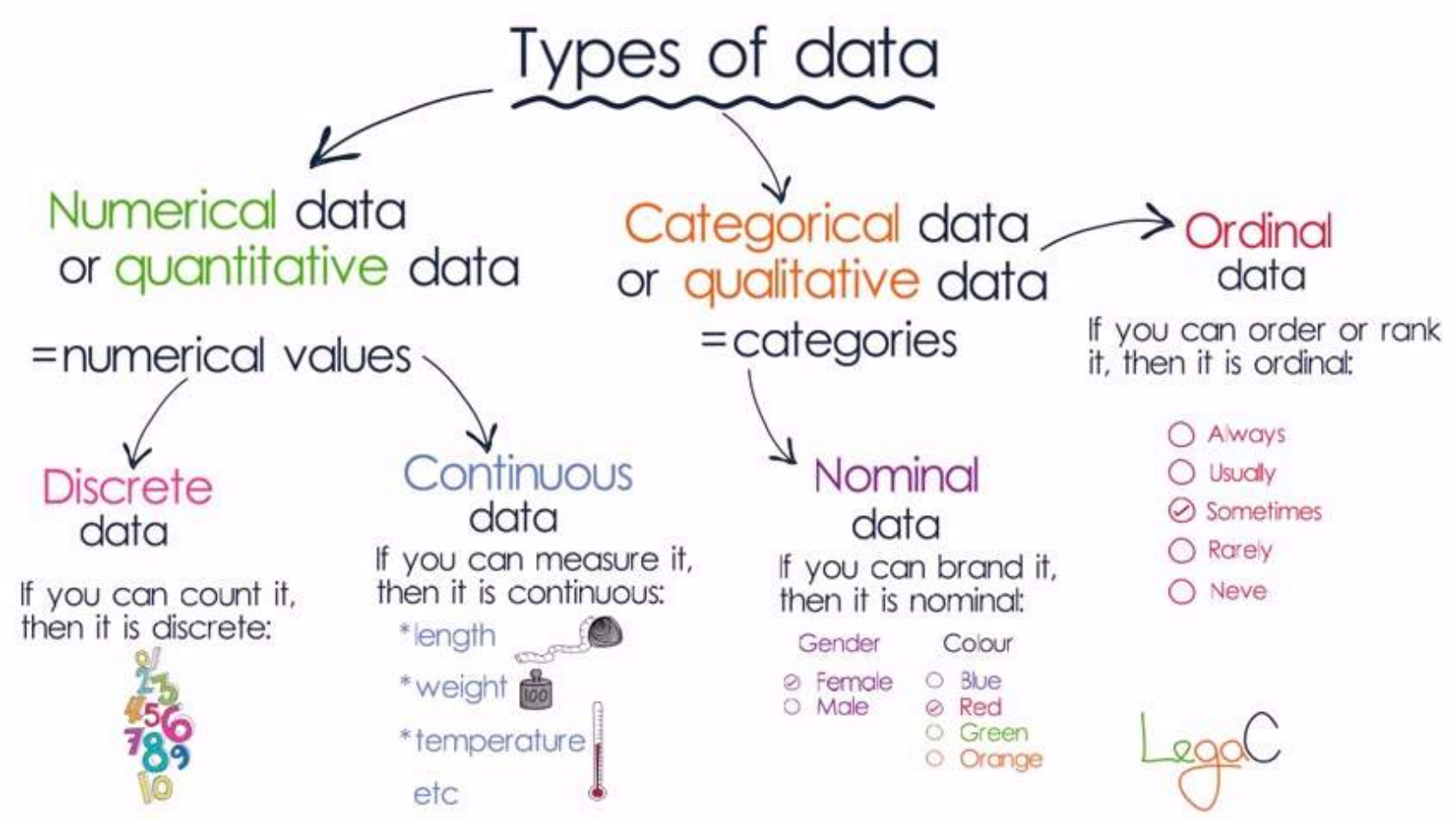
- # Überschrift 1, ## Überschrift 1.1
- Code kann zur Illustration hinzugefügt werden, wird nicht ausgeführt
  - ``` … ``` kann für Code verwendet werden.
- LaTeX-Gleichungen zwischen $ ... $ oder $$ … $$
- *italic* **bold**
- HTML, Videoembeddings, etc.

Pandas Basics

Library um tabellarische Daten zu manipulieren. Excel in mächtig.

Mit Daten umgehen

Wie sehen unsere Daten aus?



Pre-Processing - Überblick

- Daten sind meistens in vielfacher Hinsicht fehlerhaft, z.B. durch Messfehler, Menschliches Versagen oder Computerfehler, Übertragungsfehler, etc.
  - Unvollständig: Attribute fehlen
  - Noisy: verzerrte Daten, Error, Outliers (z.B.: Größe = -100cm)
  - Inkonsistenzen: Diskrepanzen zwischen verschiedenen Einträgen (Age vs. Birthday)
  - Intentional Errors: Vesteckte Fehlwerte

    z.B.: jeder fehlende Eintrag für „birthday“ = „January *1st”*

    → **Ursprung für Fehlwerte kann vielerlei sein.**

Source: Küppers, P. , MW330 Data Pre-Processing S.4; eigene Übersetzung

Warum Pre-Processing?

- Schlechte Datenqualiät → geringe Qualiät der darauf aufbauenden Data Mining und Machine Learning Ergebnisse
- Verbessert die Performance von Predictive Applications → z.B.: Accuracy
- Modellierung setzt gute Datenqualität voraus → Klassifikationsalgorithmen können grundsätzlich nicht mit Fehlwerten umgehen

→ **Datenaufbereitung**, **-säuberung** und **–transformation** beanspruchen den Hauptteil der Arbeit bei Predictive Applications und Data Science Projekten.

Source: Küppers, P. , MW330 Data Pre-Processing S.6; eigene Übersetzung

Hauptaufgaben des Pre-Processing

- **Datenbereinigung**
  - Füllen von Fehlwerten, glätten von „noisy“ data, identifizieren und entfernen von Outliern und „noisy“ data, auflösen von Inkonsistenzen.
- **Datenintegration**
  - Integration von mehreren Datenbanken oder Files (Bsp.: Merging)
- **Datentransformation**
  - Normalisierung und Aggregation
  - Datendiskretisierung
- **Datenreduktion**
  - Reduzierung des Datenvolumens mit Beibehaltung der selben analytischen Ergebnisse

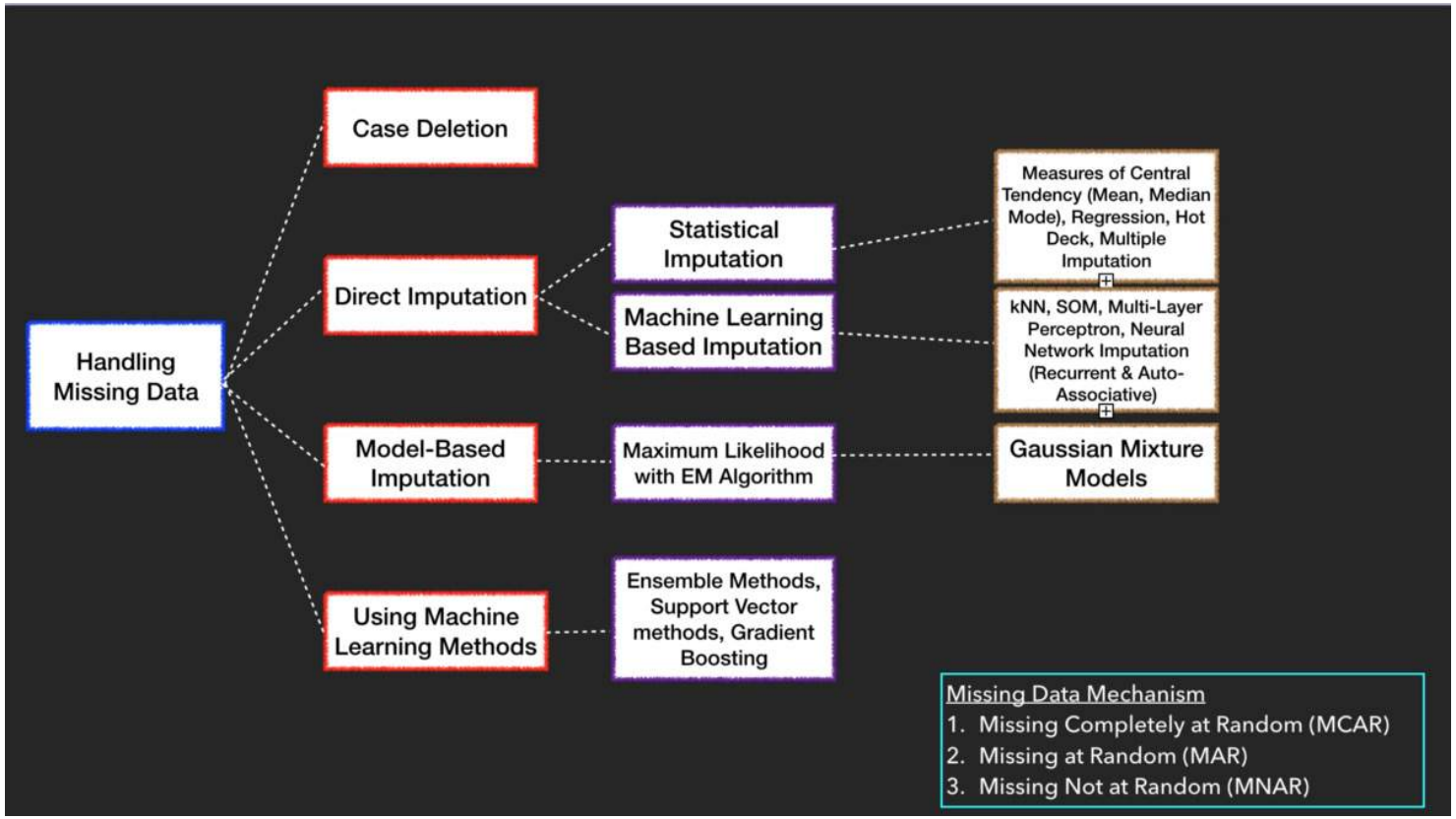
Datenbereinigung - Fehlwerte

- Aus der Statistik werden grundsätzlich 3 verschieden Klassen unterschieden
  - MCAR: Missing Completely at Random
    - Die Wahrscheinlichkeit, dass ein Wert fehlt ist nicht abhängig von den vorliegenden Datenwerten, noch von den fehlenden Datenwerten.
    - Beispiel: Ausfall eines Temperatursensors aufgrund technischer Probleme tritt zufällig auf
  - MAR: Missing at Random
    - Die Wahrscheinlichkeit, dass ein Wert fehlt ist teilweise von den anderen vorliegenden Daten abhängig aber nicht von irgendwelchen anderen Fehlwerten.
    - Beispiel: Ausfall eines Temperatursensors eher bei Nacht, trotzdem noch zufällig.
  - MNAR: Missing Not at Random
    - Die Wahrscheinlichkeit, dass eine Wert fehlt liegt an den Fehlwerten selbst.
    - Beispiel: Ausfall eines Temperatursensors wahrscheinlicher bei extremen Temperaturen.

Source: Küppers, P. , MW330 Data Pre-Processing S.8; eigene Übersetzung

Datenbereinigung – Umgang mit Fehlwerten

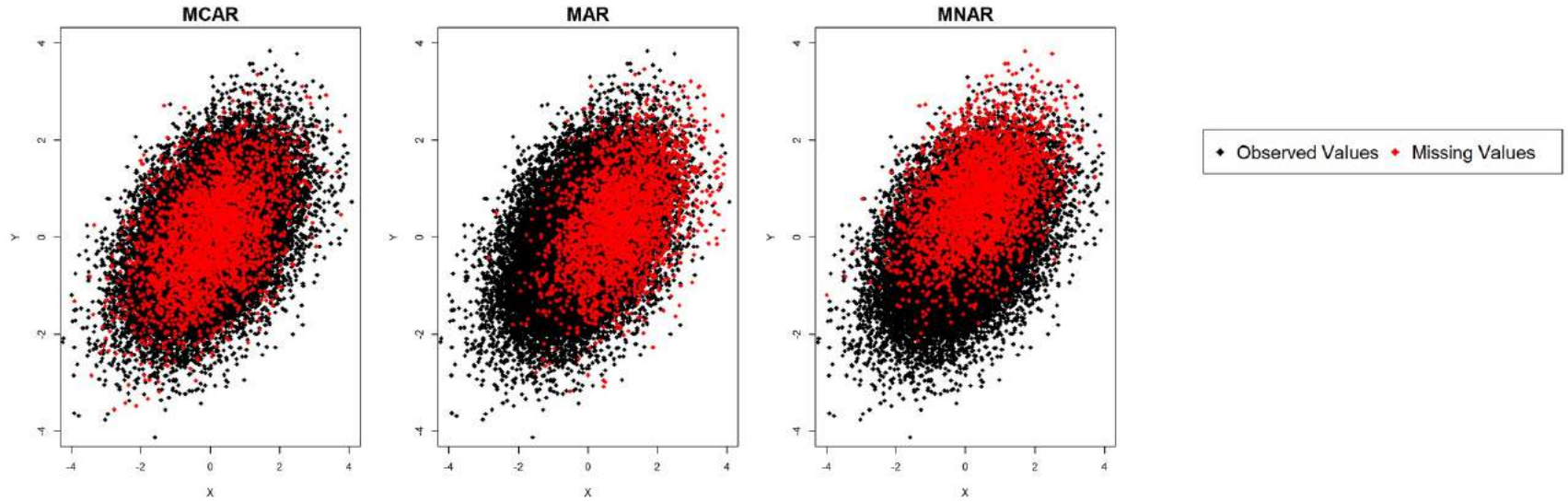
- **Überblick**



Source: https://medium.com/ibm-data-science-experience/missing-data-conundrum-exploration-and-imputation-techniques-9f40abe0fd87

Datenbereinigung – Kategorien von Fehlwerten

Beispiel: Studie/Umfrage



MCAR
- P hängt weder von X noch von Y ab.
- *Beispiel: Manche Antworten wurden ausversehen gelöscht.*

MAR
- P hängt von X ab, aber nicht von Y.
- *Beispiel: Befragte mit höherem Alter haben eine geringere Wahrscheinlichkeit ihre politische Meinung zu teilen*

MNAR
- P hängt von Y (und eventuell von X) ab.
- *Beispiel: Befragte mit höherem Einkommen geben ihr Einkommen seltener an.*

Source: Küppers, P. , MW330 Data Pre-Processing S.9; eigene Übersetzung

Datenbereinigung - MCAR

Arten von „Deletion“ (Löschung/Entfernung)

- **Listwise deletion**

User	Device	OS	Transactions
~~A~~	~~Mobile~~	~~NA~~	~~5~~
B	Mobile	Android	3
~~C~~	~~NA~~	~~iOS~~	~~2~~
D	Tablet	Android	1
E	Mobile	iOS	4

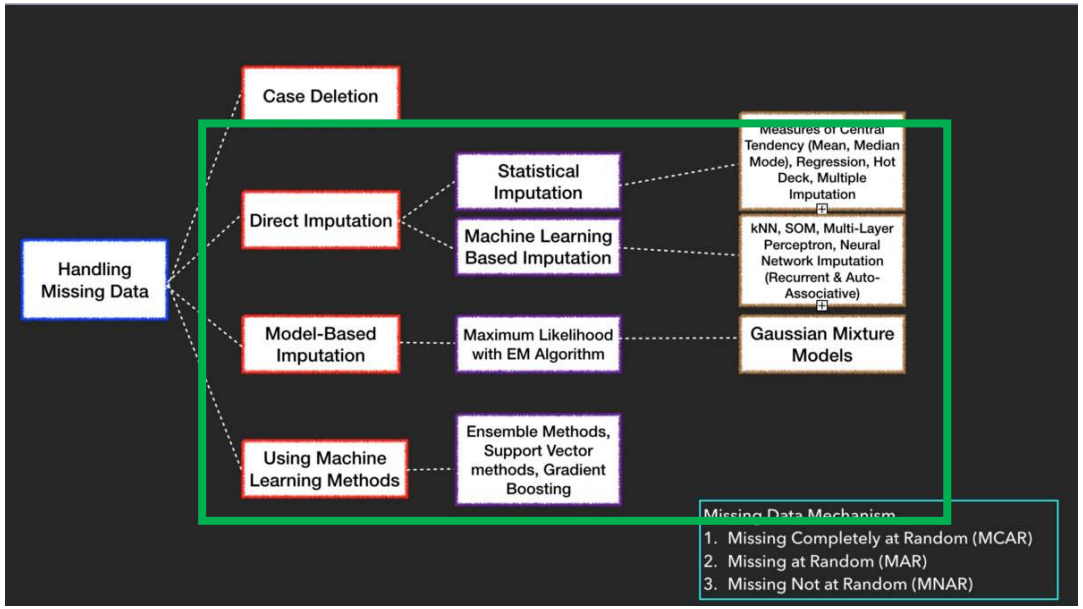
- **Pairwise deletion**

User	Device	OS	Transactions
A	Mobile	~~NA~~	5
B	Mobile	Android	3
C	~~NA~~	iOS	2
D	Tablet	Android	1
E	Mobile	iOS	4

- **Löschen von Spalten (nicht empfohlen)**
  - **Nur falls >60% der Daten fehlen oder das Attribut nicht von Bedeutung ist**

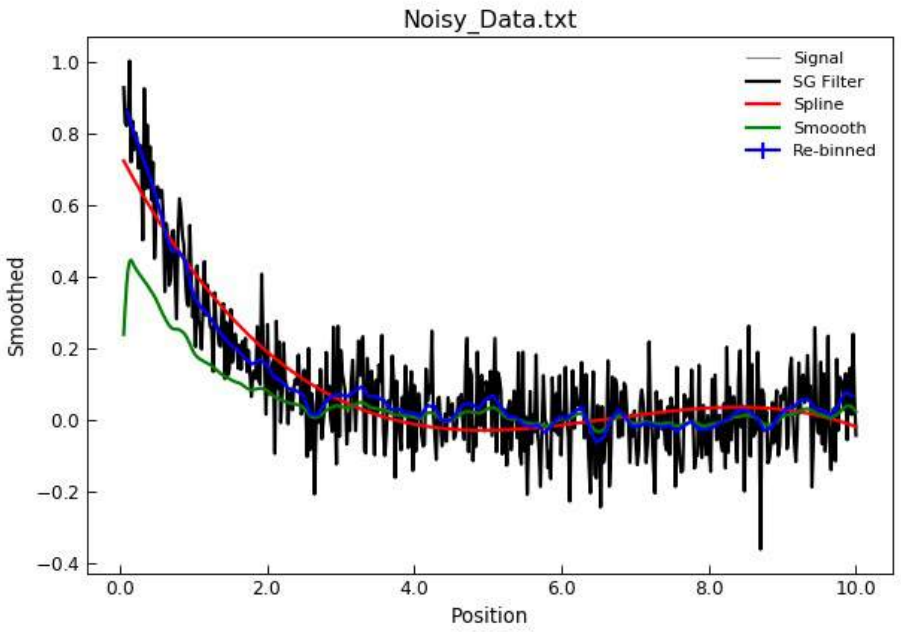
Datenaufbereitung - Imputation

- Imputation ist der Prozess des Ersetzens von Fehlwerten mit Ersatzwerten.
  - Beispiel: Ersetzen von Fehlwerten mit dem Mittelwert einer Spalte.



Noisy Data - Überblick

- Noise: Zufälliger Fehler oder Varianz in einer gemessenen Variable
- Fehlerhafte Daten resultieren beispielsweise aus:
  - Fehlerhafte Datensammlungsgeräte/-tools
  - Dateneingabeproblem (typo)
  - Datenübertragungsprobleme
  - Technologie-Limitierung



Sources: https://stoner-pythoncode.readthedocs.io/en/latest/UserGuide/analysisfile.html

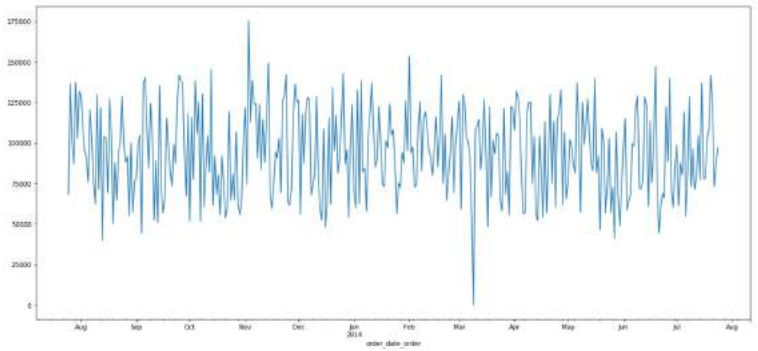
Noisy Data – Umgang mit Noisy Data

- Binning
  - Sortieren und Aufteilen der Daten in (gleichverteilte) bins
  - Dann kann nach dem Mittelwert/Median oder bin boundaries geglättet werden
- Gleitender Durchschnitt (Moving Average) & exponentielle Glättung (exponential smoothing)
- Regression
  - Glätten der Daten durch fitten von Regressionsfunktionen
- Untersuchen der Daten von Computer und Mensch
  - Entdecken von verdächtigen Werten und prüfen von Menschenhand (Bsp.: evtl. Outlier?)
  - EDA
- …viele mehr

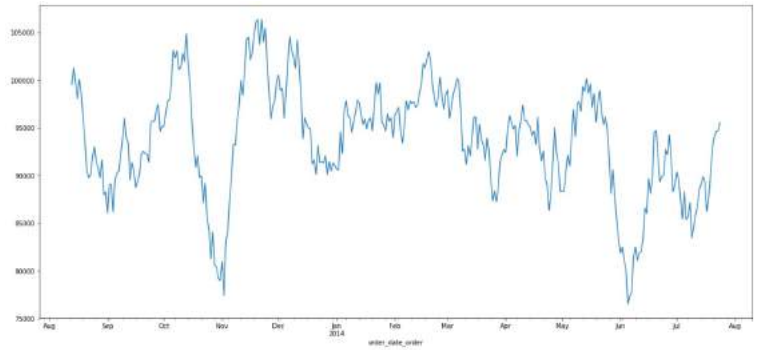
Source: Küppers, P. , MW330 Data Pre-Processing S.24; eigene Übersetzung

Noisy Data – Zeitreihen glätten

Original:



Rolling mean:

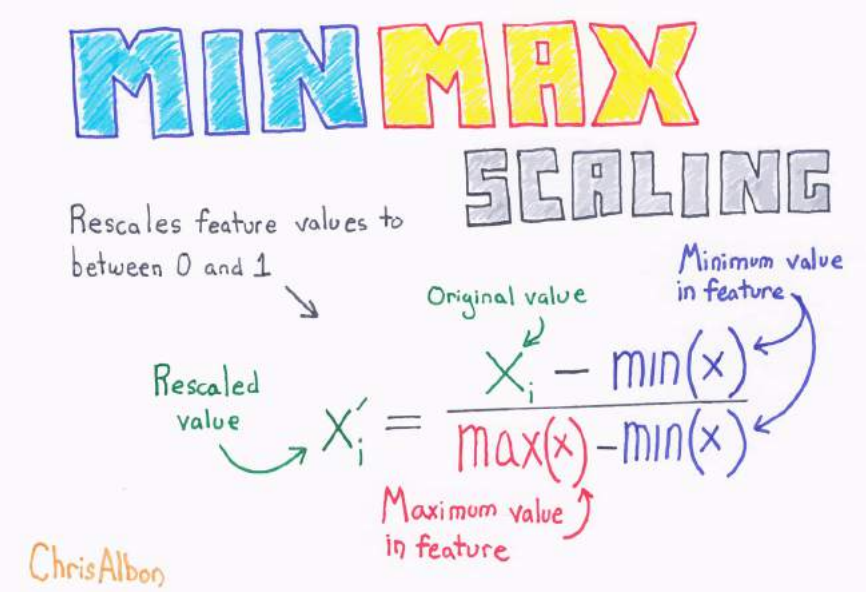


Simple exponential smoothing (alpha=0.05)

Source: Prof. Dr. Küppers, MW330 Data Pre-Processing S.28; eigene Übersetzung

Datentransformation - Normalisierung

- Normalisierung durch min-max Skalierung.
  - Normalisieren aller Werte zu einem Wertebereich von 0 bis 1.

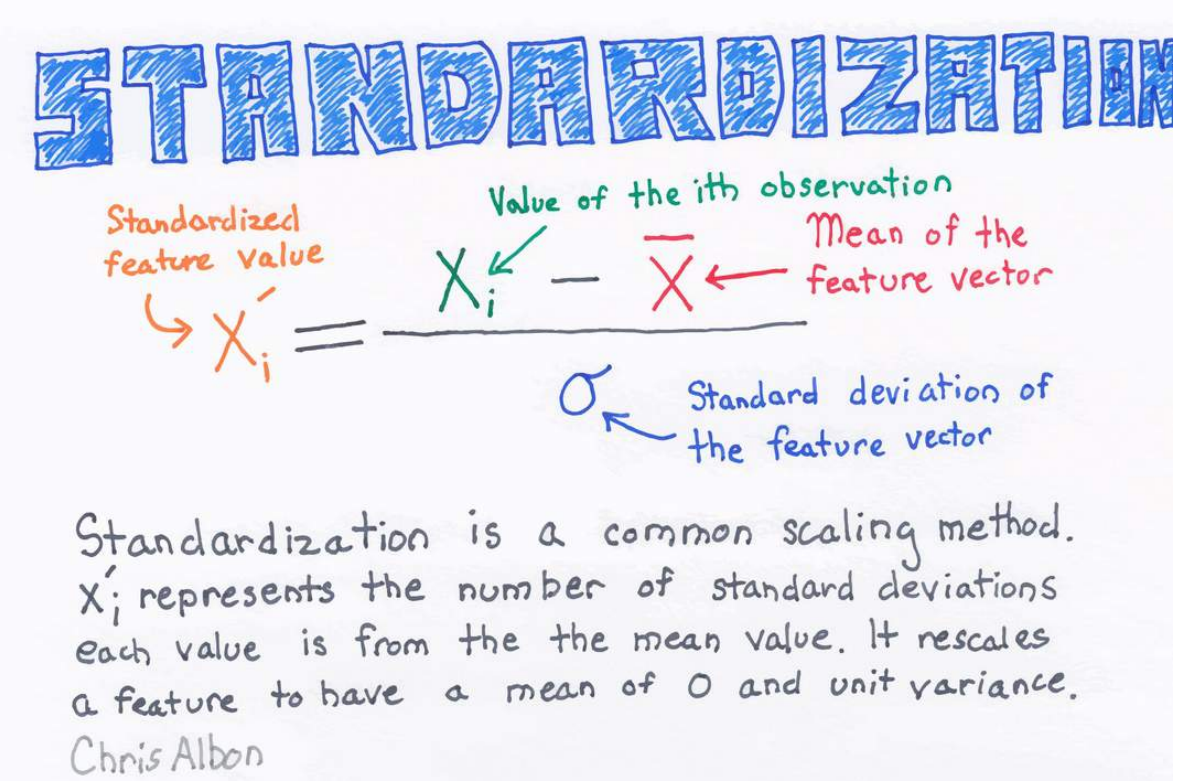


https://machinelearningflashcards.com/

Körpergröße (unskaliert)	Körpergröße (Min-Max skaliert)
190	1
175	0.57
155	0
176	0.6
164	0.257

Datentransformation - Normalisierung

- Normalisierung durch Standard Skalierung (z-Standardisierung)



Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

and standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Unit variance bedeutet, dass die Standardabweichung der Variable gegen 1 geht wenn die sample size gegen unendlich geht.

https://machinelearningflashcards.com/

Datentransformation - Encoding

Sample	Category
1	Human
2	Human
3	Penguin
4	Octopus
5	Alien
6	Octopus
7	Alien

LabelEncoding

Sample	Category	Numerical
1	Human	1
2	Human	1
3	Penguin	2
4	Octopus	3
5	Alien	4
6	Octopus	3
7	Alien	4

Achtung: Wir müssen vorsichtig sein mit Label Encoding! Machine Learning Algorithmen könnten das Attribut als Ordinal skaliert interpretieren.

OneHotEncoding

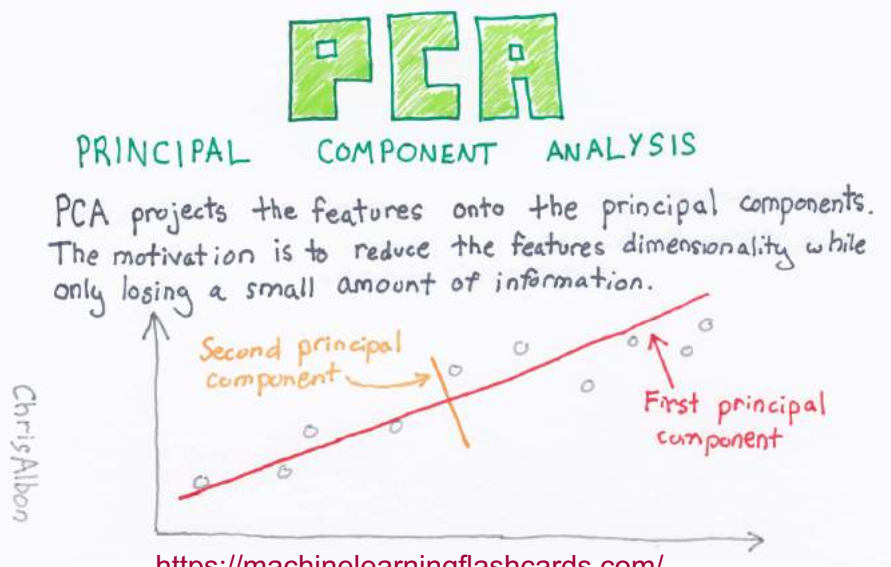
Sample	Human	Penguin	Octopus	Alien
1	1	0	0	0
2	1	0	0	0
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	0	0	1	0
7	0	0	0	1

One-Hot Encoding funktioniert besser mit normal nominalen/kategorischen Attributen für Machine Learning

Source: Küppers, P. , MW330 Data Pre-Processing S.48; eigene Übersetzung

Datenreduktion - PCA

- Ziel: Reduzieren der Anzahl von Features durch Identifikation von Korrelationen zwischen mehreren Features und reduzieren dieser zu einem (oder mehreren) Features, den sogenannten „Principle Components“
  - Reduzierter Rechenaufwand und Feature Dimensionen mit nur geringem Verlust an Informationen



https://machinelearningflashcards.com/

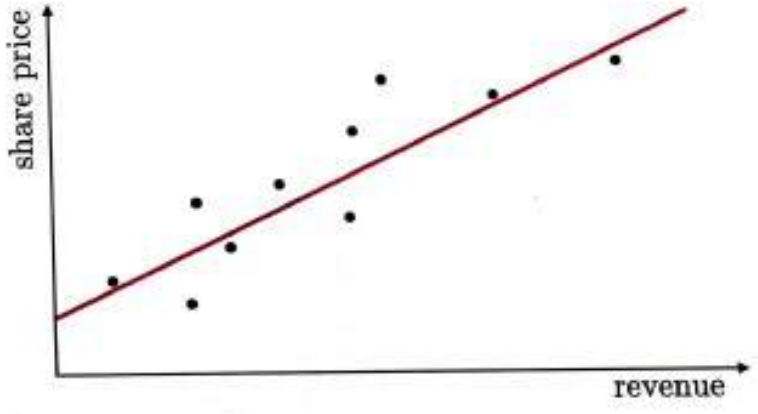
Use Case Pre-Processing & Pre-Processing Pipelines

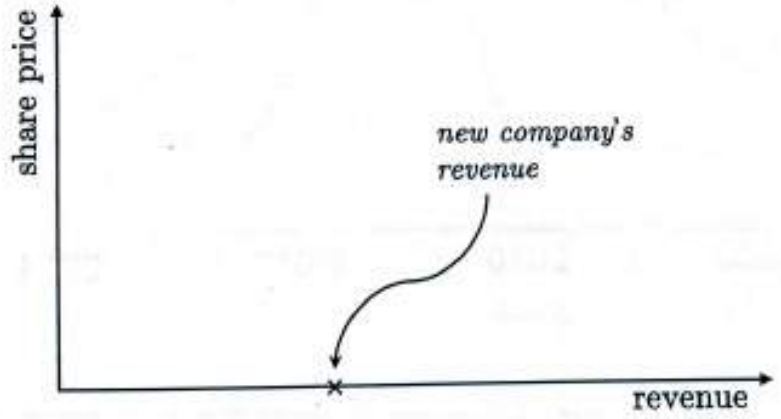
Machine Learning Basics

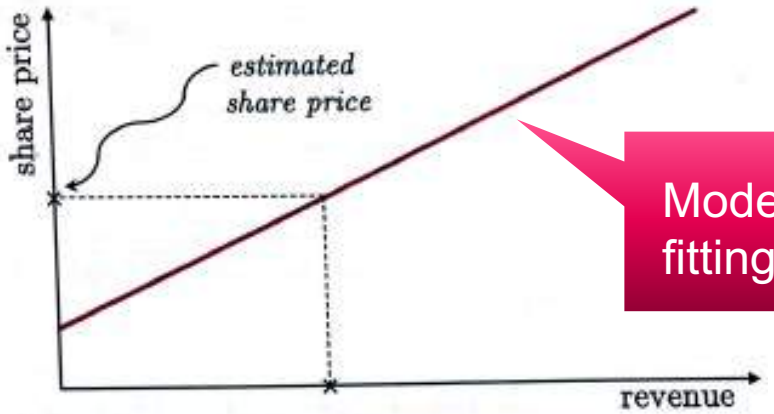
Regression vs Klassifizierung

Klassen oder kontinuierliche Werte

Regression

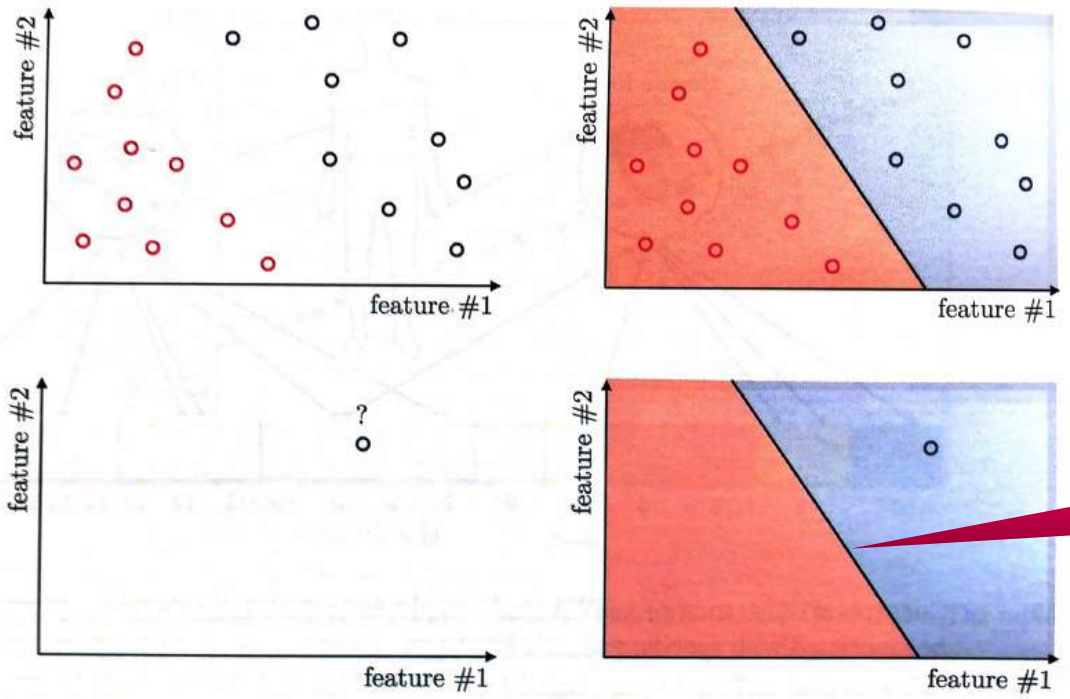






Model data – fitting curve

Klassifizierung

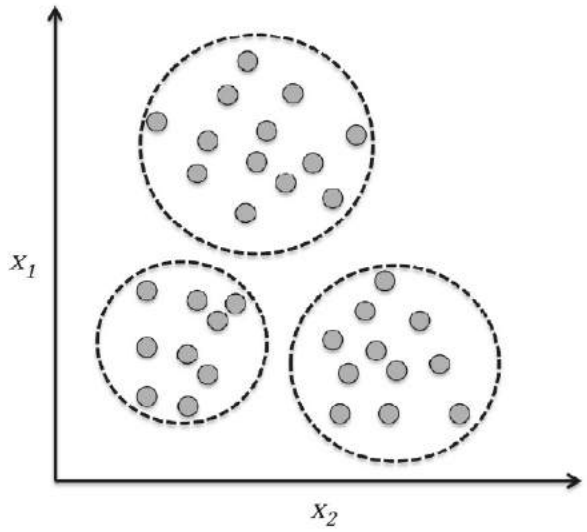


Divide data –
Decision plane

Unsupervised Learning - Clustering

Clustering:

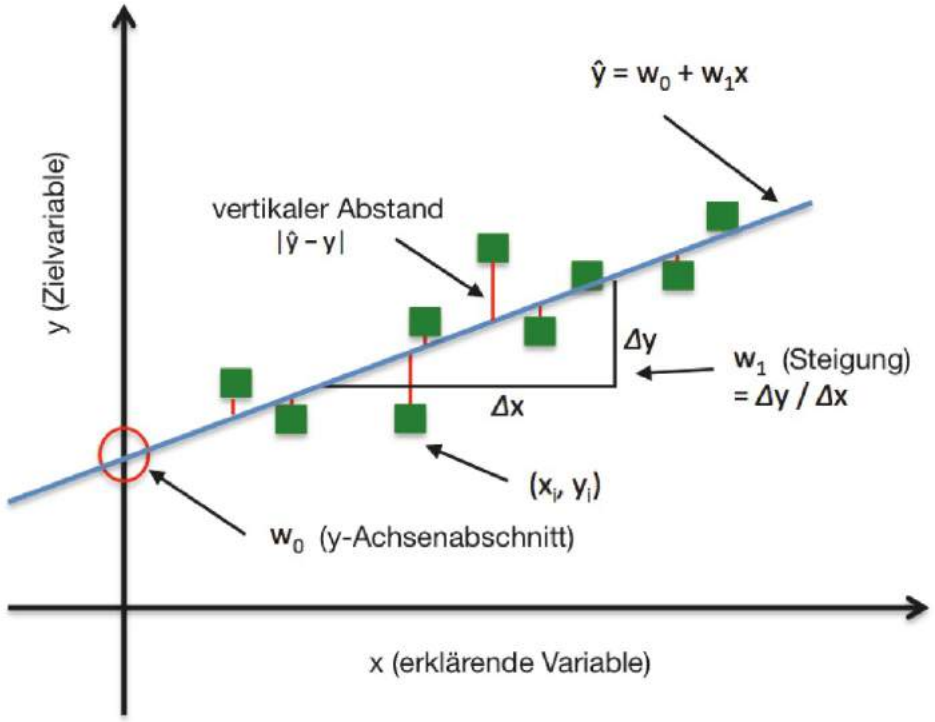
- Aufteilung von nicht gelabelten Daten in ähnliche Gruppen



Raschka, S., Machine Learning mit Python: Das Praxis Handbuch für Data Science, Predictive Analytics und Deep Learning, 2016, S.26 ff

Lineare Regression – Einfache Lineare Regression

- Dient zur Vorhersage eines Zielwerts y anhand eines Merkmals/Features x



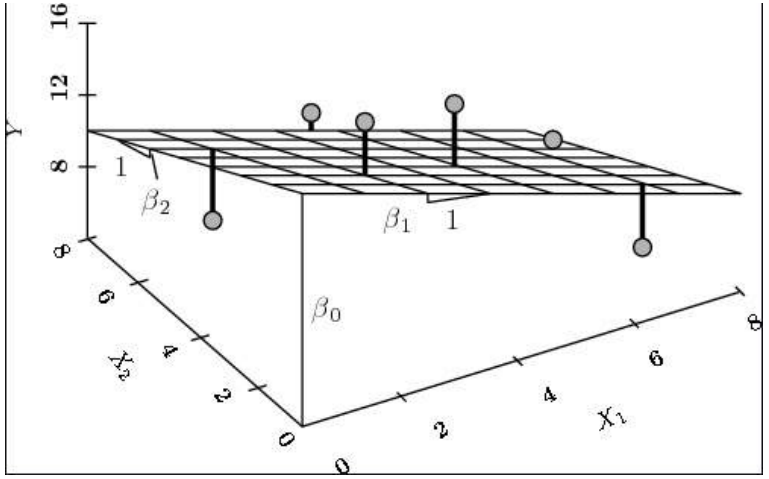
Raschka, S., Machine Learning mit Python: Das Praxis Handbuch für Data Science, Predictive Analytics und Deep Learning, 2016, S.279 ff.

Lineare Regression – Multiple Lineare Regression

Es können mehrere Features zur Vorhersage der Zielvariable hinzugezogen werden

$$y = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{i=0}^{m} w_i x_i = w^T x$$

- Jedes Merkmal x hat seinen eigenen Koeffizienten und somit auch einen individuellen Einfluss auf die Zielvariable
- Aus Gerade wird eine Ebene:

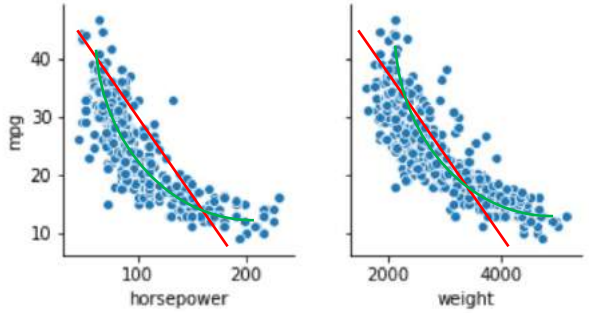


Source:https://www.google.com/search?q=Regressionsebene&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiBn4GPv43wAhU4hf0HHVw6Av0Q_AUoA3oECAEQBQ&biw=1920&bih=937#imgrc=ICJ4cfQrU2z-FM

Raschka, S., Machine Learning mit Python: Das Praxis Handbuch für Data Science, Predictive Analytics und Deep Learning, 2016, S.279 ff.

Polynomiale Regression

- Was wenn die Beziehung zwischen Target „y“ und unseren Features „komplexer“ ist?



- Polynomiale Regression erlaubt es die optimalen Koeffizienten für ein polynomial n-ten Grades zu ermitteln:

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \ldots + w_n x^n$$

Source: Küppers, P. , MW330 – L6 – Linear Regression, S.14 f.; eigene Übersetzung

Polynomiale Regression

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \ldots + w_n x^n$$

- Um die Koeffizienten w zu ermitteln, können wir die Werte für x^i berechnen und diese als normale numerische Features behandeln.
  - Es müssen keine weiteren Änderungen vorgenommen werden !

	horsepower	weight	horsepower^2	weight^2
0	130.0	3504	16900.0	12278016
1	165.0	3693	27225.0	13638249
2	150.0	3436	22500.0	11806096
3	150.0	3433	22500.0	11785489
4	140.0	3449	19600.0	11895601

Bei der Anwendung einer „normalen“ Linearen Regression werden diese Features dann verwendet und es werden bessere Resultate erzielt (falls eine nicht-lineare Beziehung besteht)

Source: Küppers, P. , MW330 – L6 – Linear Regression, S.14 f.; eigene Übersetzung

Use Case
Lineare Regression

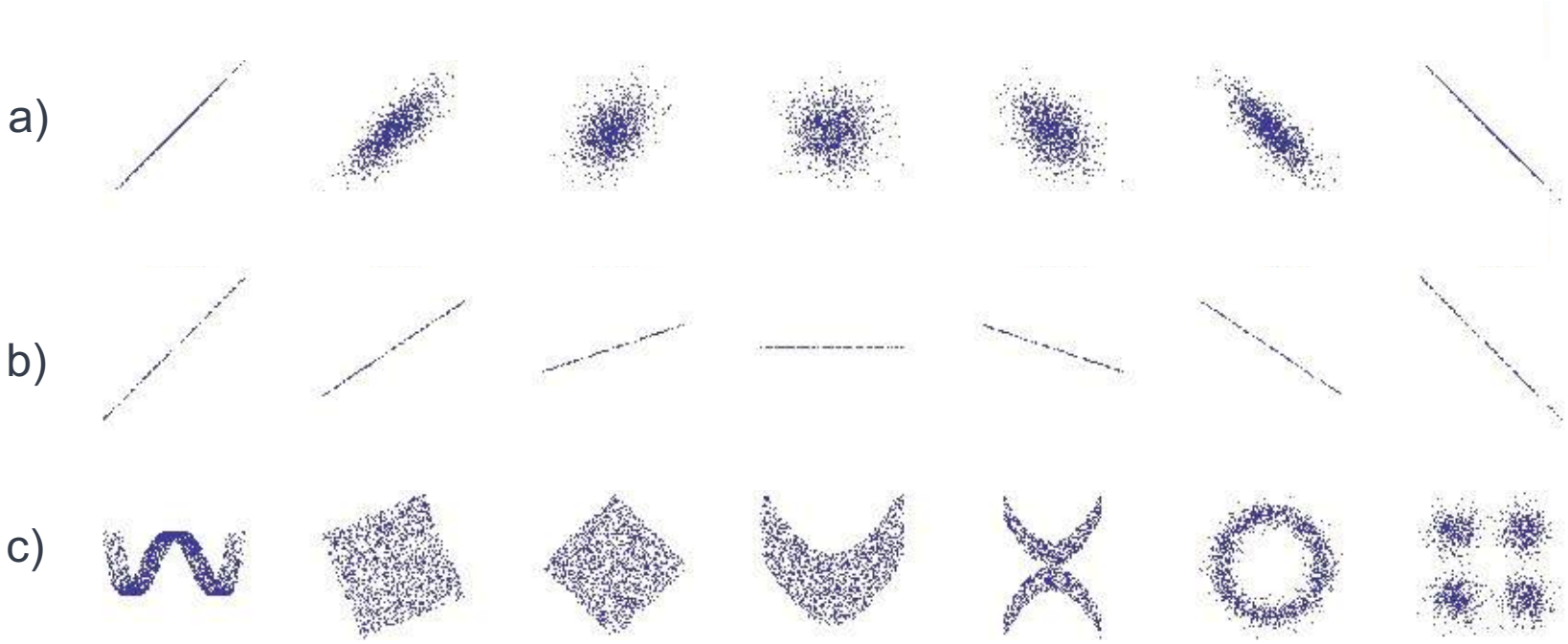
Korrelation – Pearson Korrelationskoeffizient

- Mithilfe der Korrelation lassen sich Beziehungen zwischen den vorliegenden Variablen finden
- Der Wert der Korrelation (Korrelationskoeffizient) zeigt wie stark eine Variable x, eine andere Variable y beeinflusst.
- Misst den **linearen Zusammenhang** zweier Variablen
- Korrelationskoeffizient variiert von -1 bis +1
  - +1 = Wenn x um 1 Einheit steigt, steigt y ebenfalls um 1 Einheit
  - 0 = keine Korrelation
  - -1 = Wenn x um 1 Einheit steigt, sinkt y um 1 Einheit

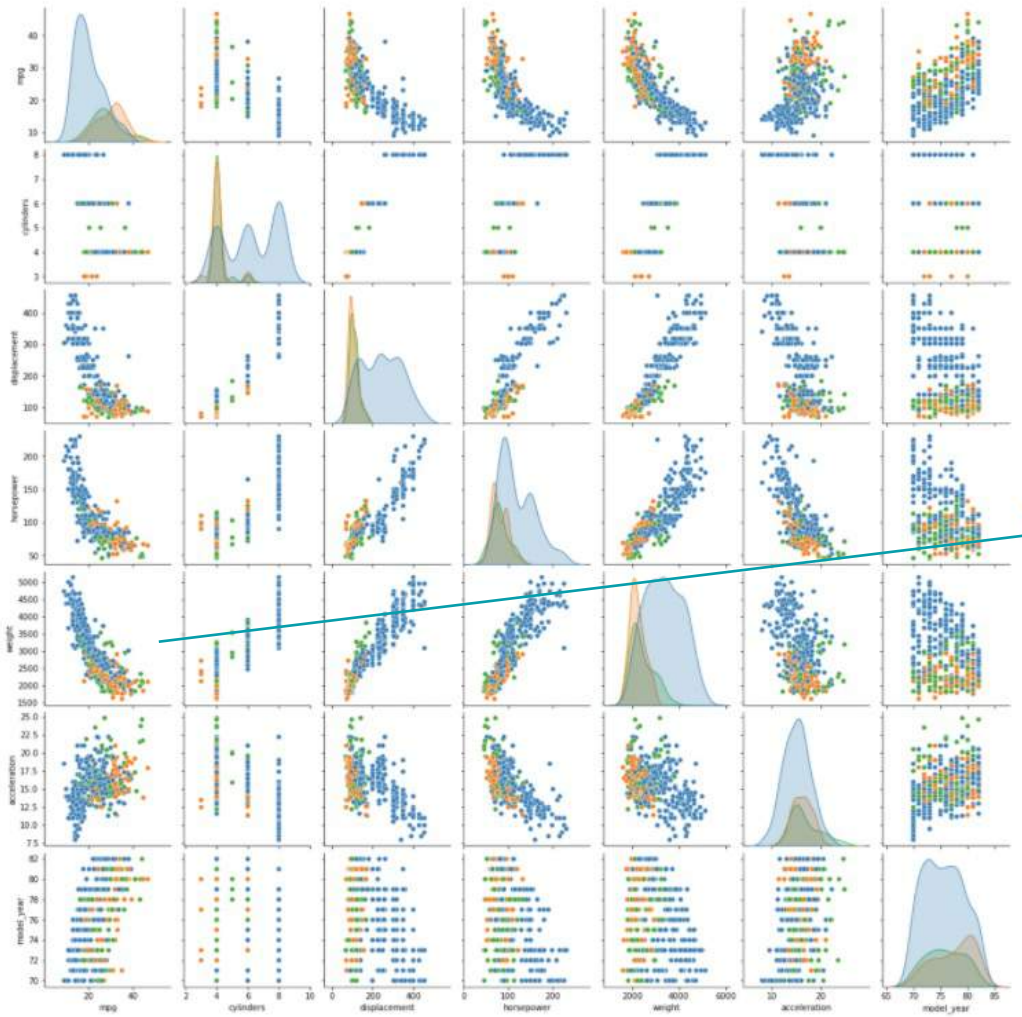
Berechnung beispielsweise durch Pearson Korrelationskoeffizienten:

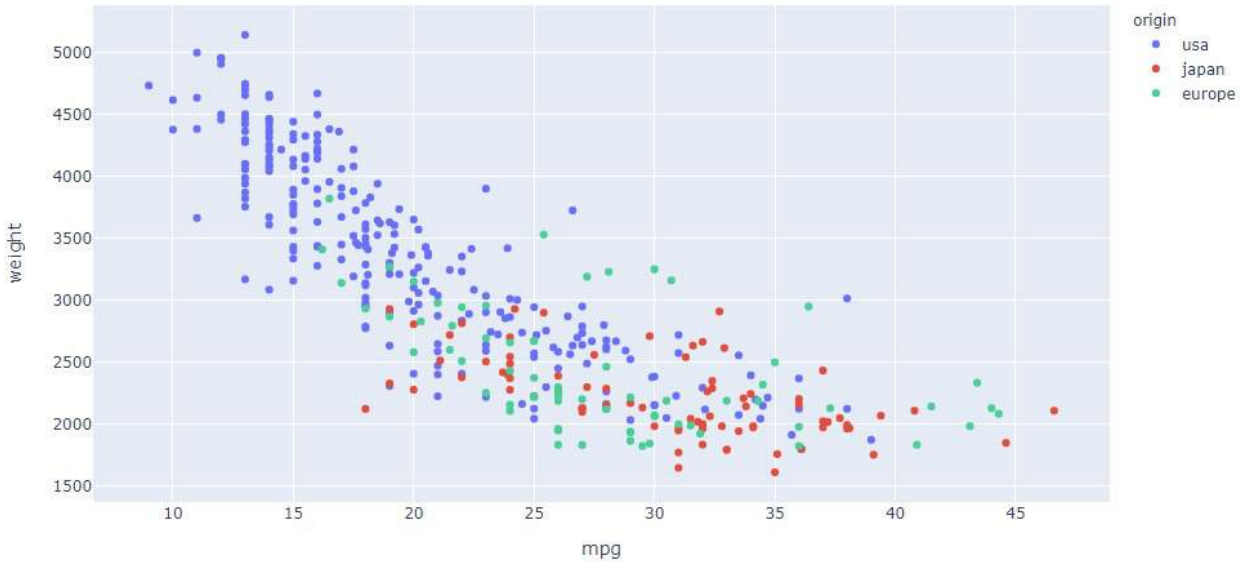
$$\mathrm{cor}_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})}} = \frac{\mathrm{cov}_{x,y}}{\sigma_x \sigma_y}$$

Korrelation erkennen

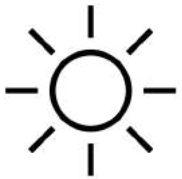


Korrelation – Pairplot zur Überprüfung von linearen Zusammenhängen





Korrelation ≠ Kausalität
Beispiel Hawaii



Gutes Wetter

Eisverkäufe steigen

Kein Kausalzusammenhang !

Korrelation

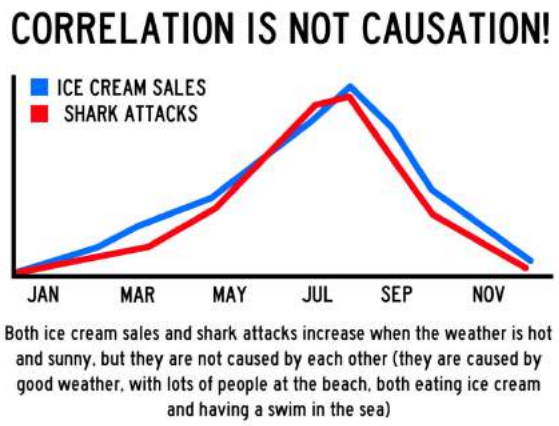
Mehr Leute am Strand

Mehr Haiattacken

Korrelation ≠ Kausalität

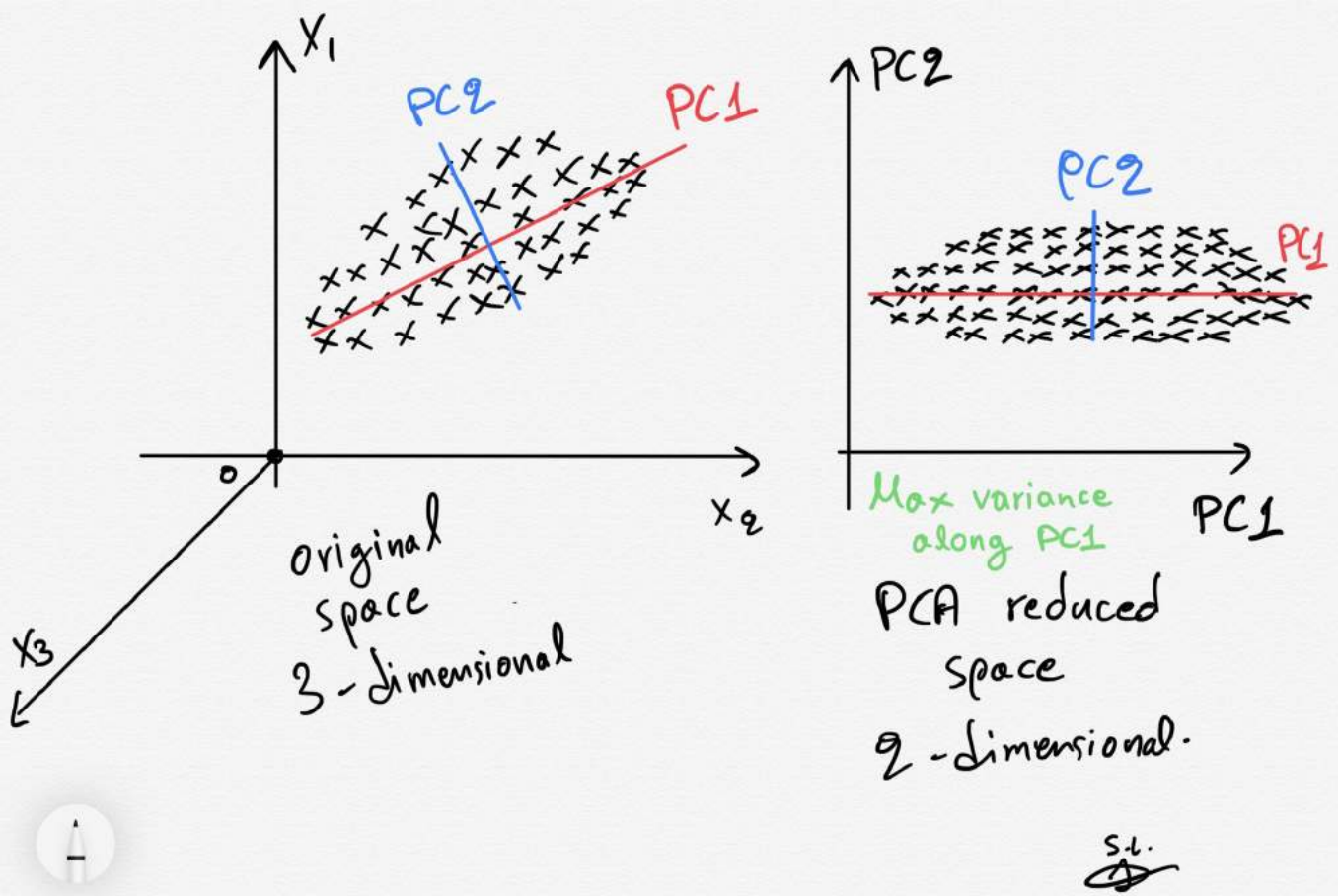
- Wenn ein Feature A und B korrelieren, kann man nicht zwangsläufig davon ausgehen, dass A der Auslöser/Treiber für B ist.

- Bsp.: Bei der Untersuchung einer demografischen Datenbank, wird eventuell ein Zusammenhang zwischen der „Anzahl von Krankenhäusern“ und der „Anzahl von Autodiebstählen in der Region“ gefunden, welche korrelieren.
  - Dies bedeutet nicht, dass das eine die Ursache für das andere ist.
  - Beide sind offensichtlich verbunden, jedoch durch ein drittes Attribut, nämlich „Bevölkerung“



Source: Küppers, P. , MW330 EDA, S 32.; eigene Übersetzung

Lineare Regression, Pearson Korrelation und PCA sind drei mächtige Werkzeuge um linearen Zusammenhängen auf den Grund zu gehen

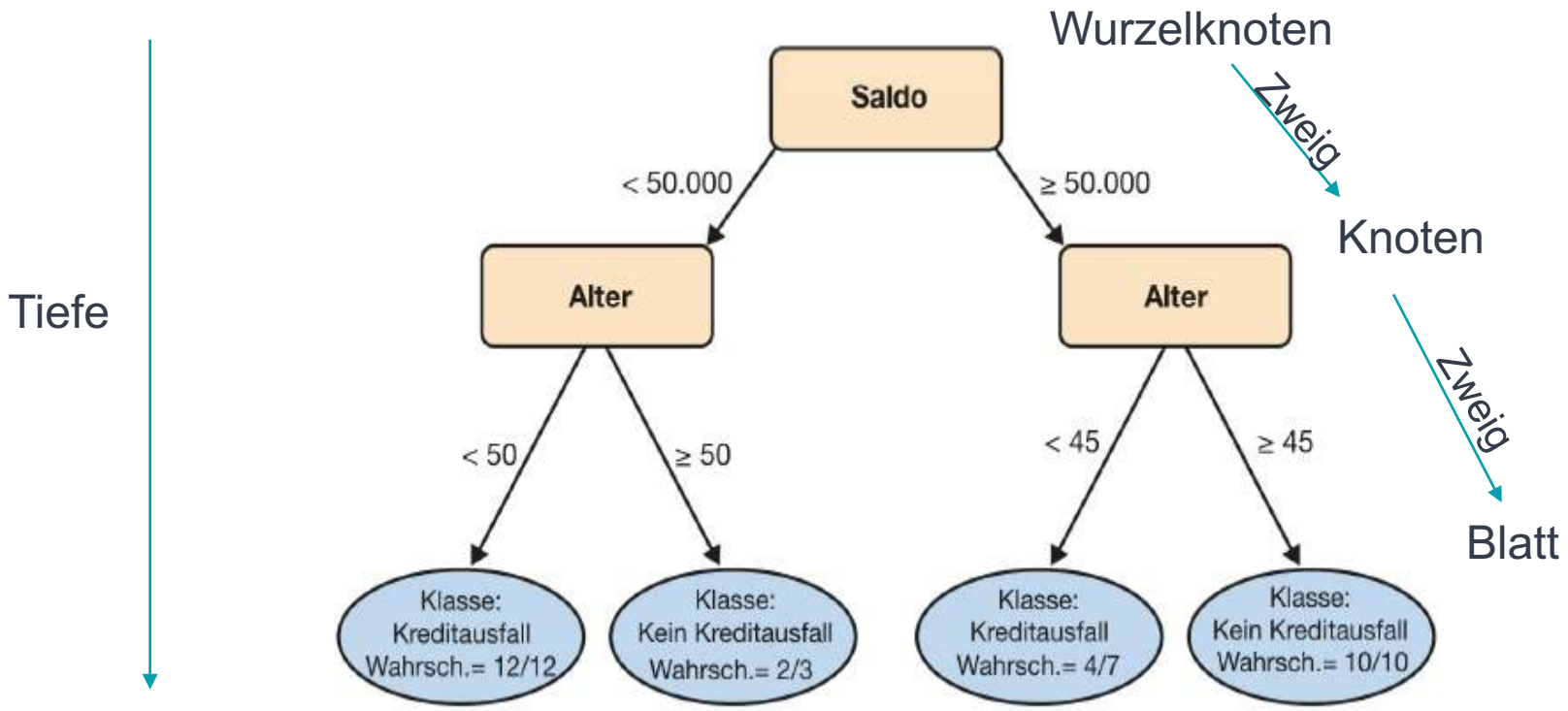


Source: https://towardsdatascience.com/pca-clearly-explained-how-when-why-to-use-it-and-feature-importance-a-guide-in-python-7c274582c37e

Use Case
Korrelation (+PCA)

Klassifikation - Entscheidungsbaum

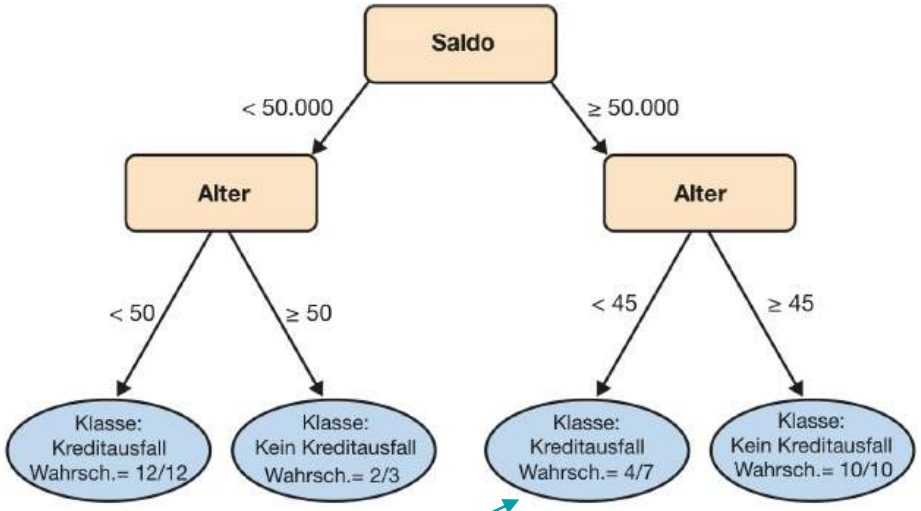
- Entscheidungsbaum zur Beurteilung der Kreditwürdigkeit von Kunden anhand von Merkmal Saldo und Alter



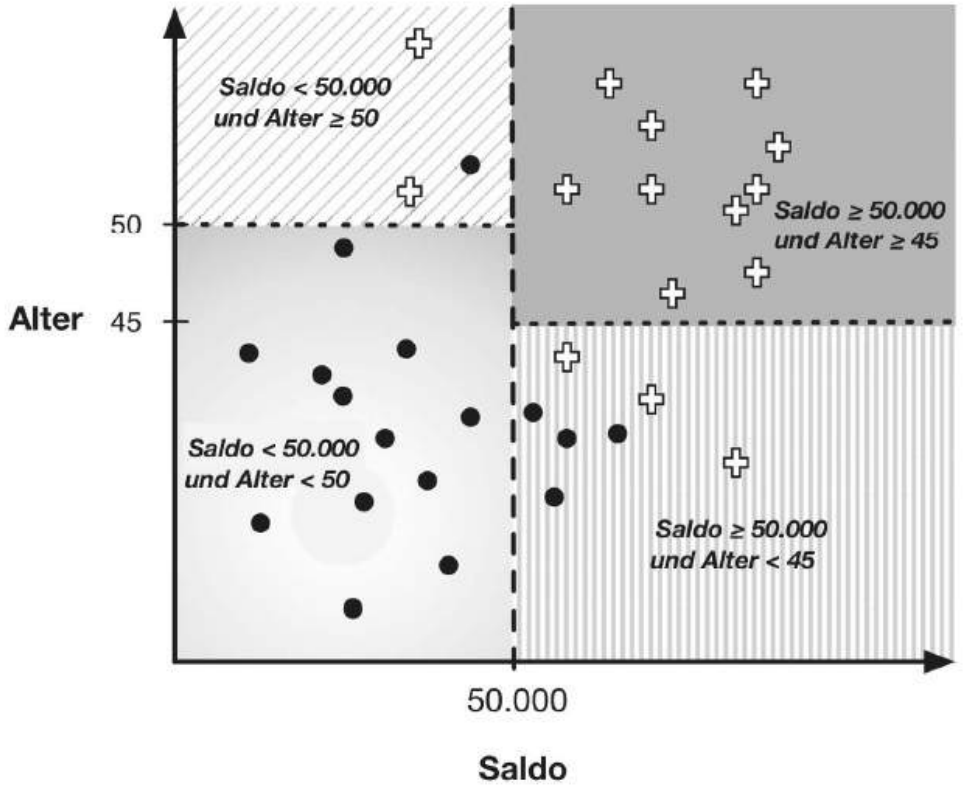
Foster P. , Fawcett, T., Data Science für Unternehmen, 2017, S.99

Klassifikation - Entscheidungsbaum

- Entscheidungsgrenzen sind keine Geraden oder Ebenen sondern Stufen/Bereiche



$$p(Kreditausfall) = \frac{samples\ Kreditausfall}{samples\ Gesamt\ in\ Bereich}$$

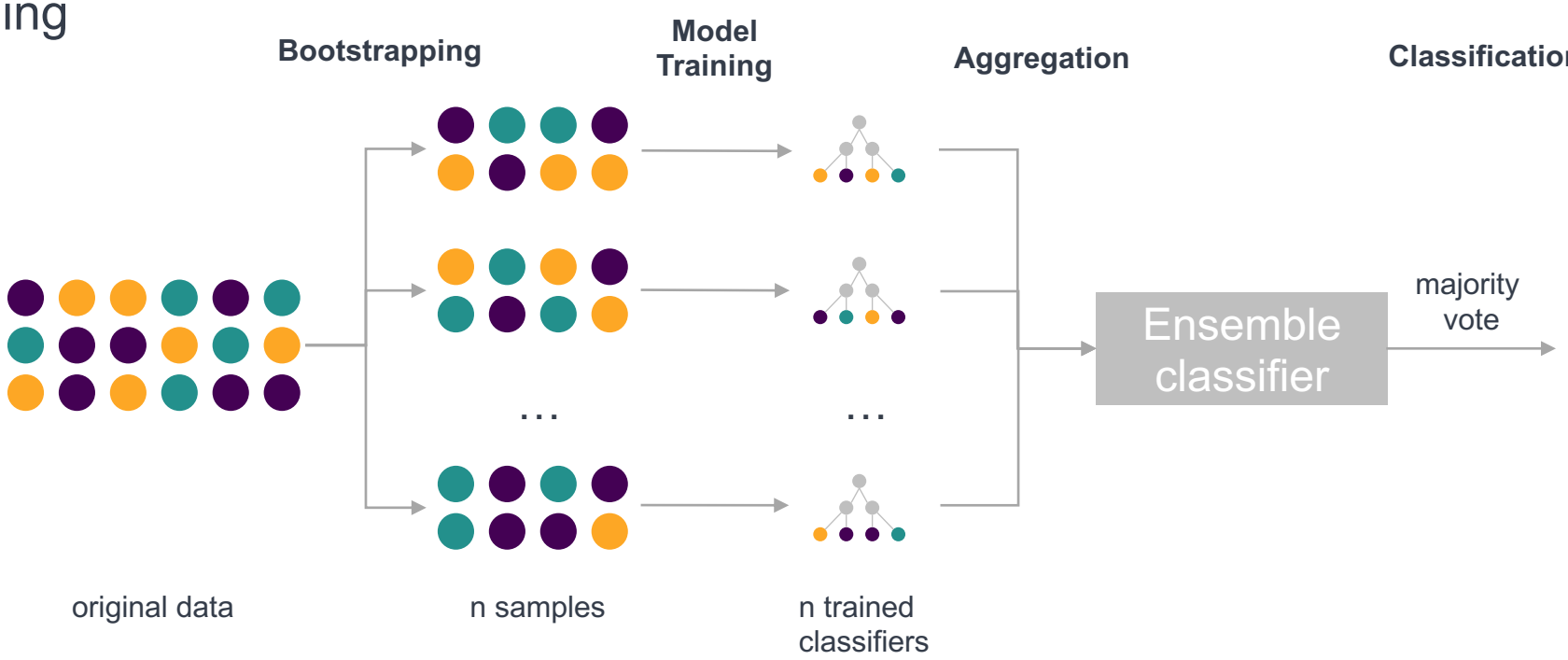


Foster P. , Fawcett, T., Data Science für Unternehmen, 2017, S.99

Ensemble Methoden - Bagging

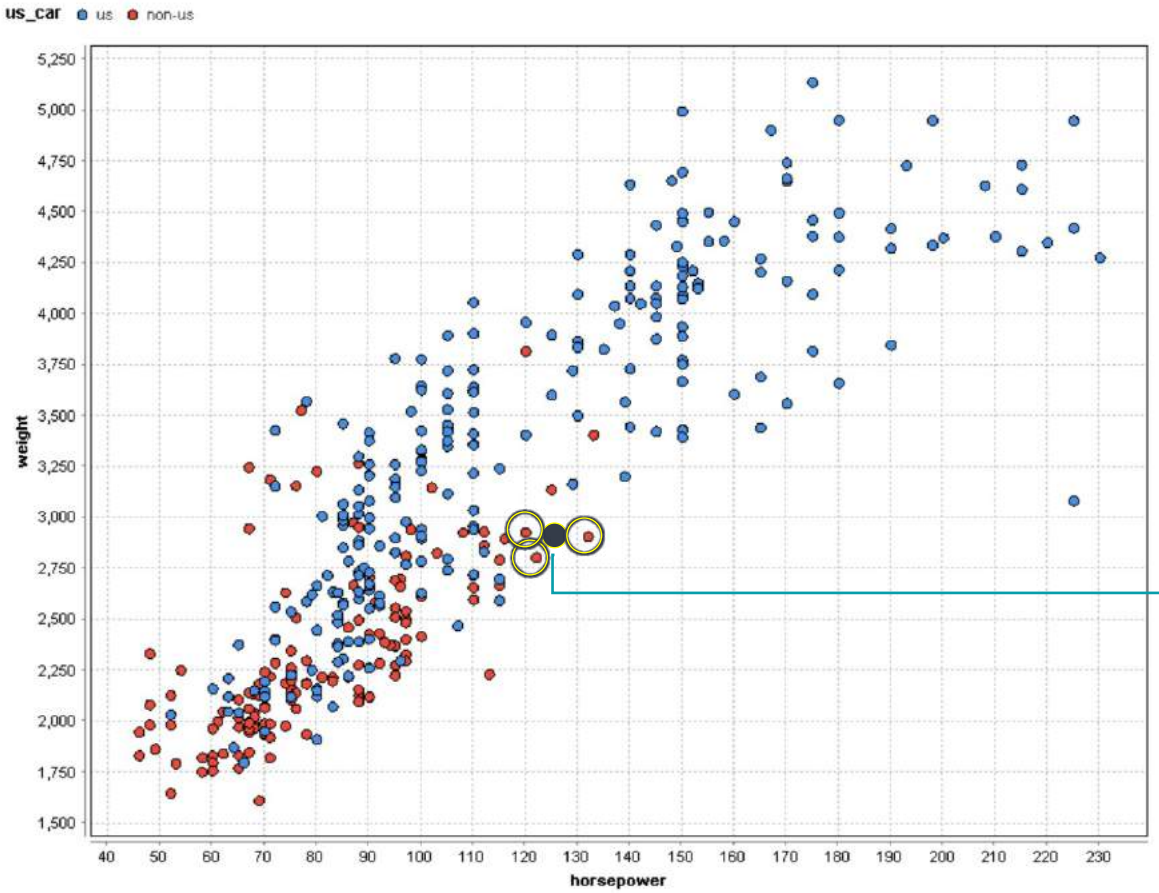
- Um der Gefahr des Overfittings vorzubeugen können sogenannte Ensemble Methoden verwendet werden

Beispiel: Bagging



Küppers, P. , MW330 – L2B – Bias Variance Tradeoff, S.20; eigene Übersetzung

Klassifikation - K-Nearest-Neighbor



Für k=3 würde die Vorhersage “non-us” lauten

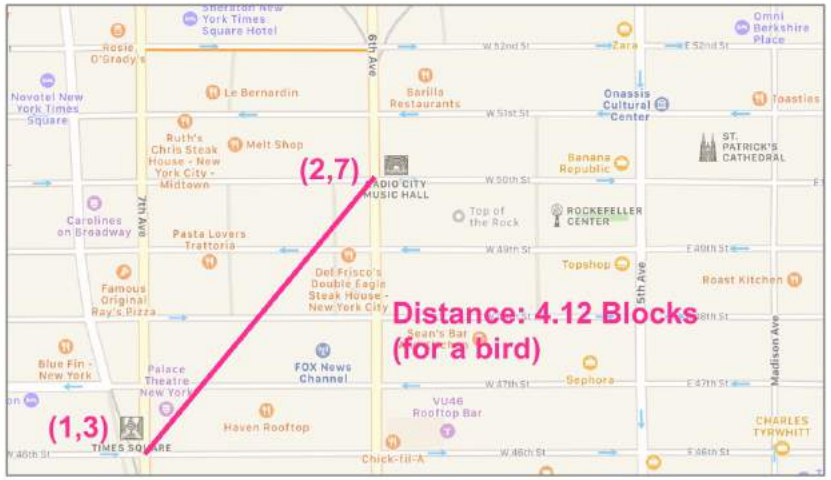
Küppers, P. , MW330, L4 – Classification, p.6 ; eigene Übersetzung

Klassifikation - K-Nearest-Neighbor

Distanzmaße:

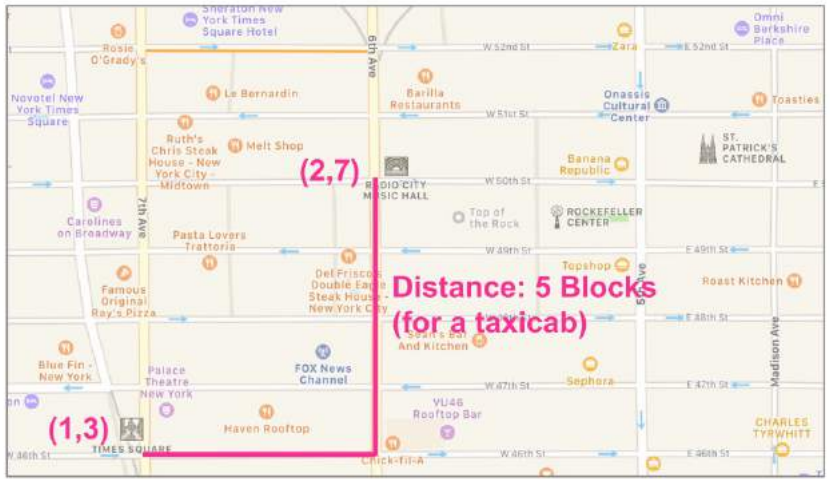
Euclidian distance

$$d(x, x') = \sqrt{\sum_{i=1}^{m}(x_i - x'_i)^2}$$



Manhattan distance

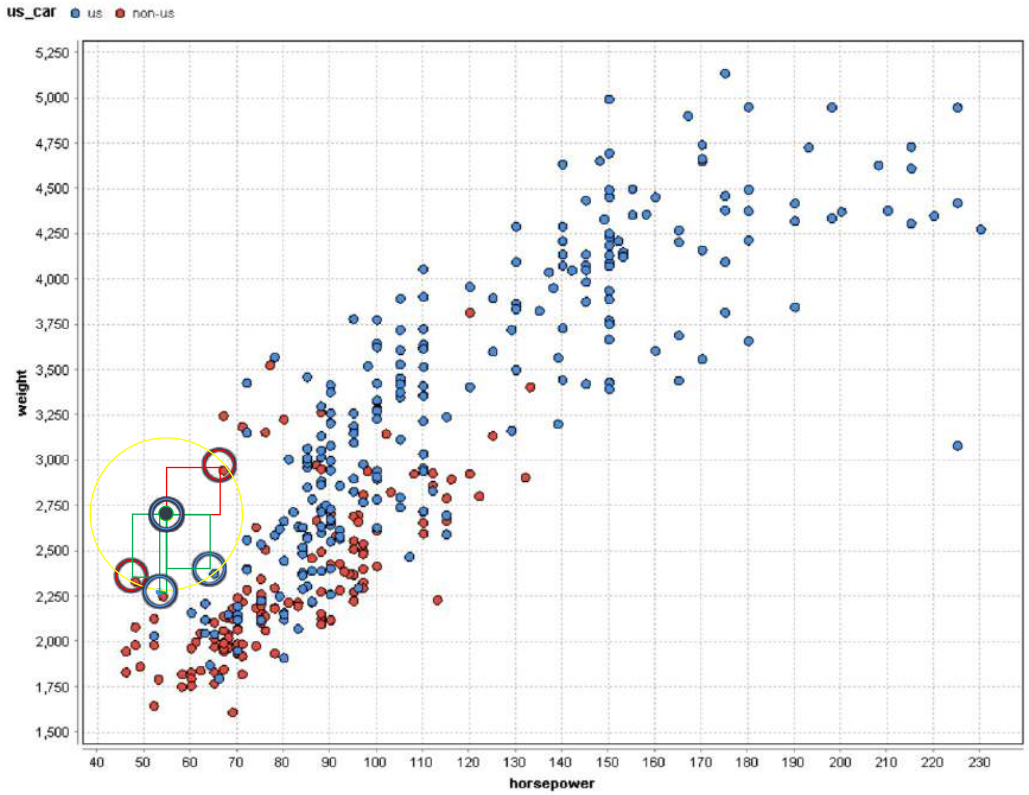
$$d(x, x') = \sum_{i=1}^{m}|x_i - x'_i|$$



Küppers, P. , MW330, L4 – Classification, p.7 ; eigene Übersetzung

Klassifikation - K-Nearest-Neighbor

- Beispiel, k=3
  - Euclidian distance
  - Manhattan distance
- Die Wahl des richtigen Distanzmaßes hängt von den Daten und den getroffenen Annahmen ab
  - Euclidian distance zieht eine „Homogenität“ des zu klassifizierenden Datenpunkts über alle Features vor (hier horsepower und weight)
  - Manhattan distance wird mit steigender Dimensionierung der Daten eher genutzt



Küppers, P. , MW330, L4 – Classification, p.8 ; eigene Übersetzung

Use Case
Klassifikation

Clustering - KMeans

Algorithmus:

1. Auswahl aus Objekten k Zentroide als anfängliche Clusterzentren.
2. Alle Objekte dem nächsten Zentroiden zuweisen.
3. Neuberechnung des Zentroiden mit den aus Schritt 2 zugewiesenen Objekten.
4. Wiederholung von 2+3, bis sich die Zuordnung nicht mehr ändert (entweder Schwellenwert oder maximale Iterationen werden vorgegeben).

Aber was ist unser Ähnlichkeitsmaß ?

Raschka, S., Machine Learning mit Python, 2016, S.314 ff.

Clustering - KMeans

Euklidische Distanz:

- Datenobjekte die sich ähnlich sind nahe beieinander platziert

Formel: $d\,(x\,,y\,)^2 = \sum_{j=1}^{m}(x_j - y_j\,)^2 = \|x - y\|_2^2$

Durch euklidische Distanz kann k-Means Algorithmus als Optimierungsaufgabe formuliert werden

- Summe der quadrierten Abweichungen innerhalb eines Clusters soll minimiert werden

Formel: SSE = $\sum_{i=1}^{n}\sum_{j=1}^{k} w^{(i,j)} \left\|x^{(i)} - \mu^{(j)}\right\|_2^2$

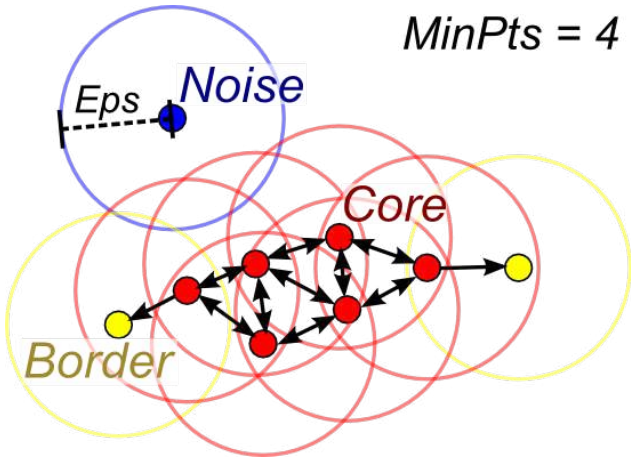
Raschka, S., Machine Learning mit Python, 2016, S.314 ff.

Clustering - DBSCAN

(Density-Based Spatial Clustering of Applications with Noise)

1. Berechnen der Density für jeden Datenpunkt abhängig von epsilon → Density > MinPts = High-Density Area
2. Wenn Datenpunkt = High-Density Area → Core Point

   Falls Datenpunkt ≠ High-Density Area, aber Core Point innerhalb Radius (epsilon) → Border Point
3. Falls Datenpunkt ≠ High-Density Area und kein Core Point innerhalb Radius (epsilon) → Noise



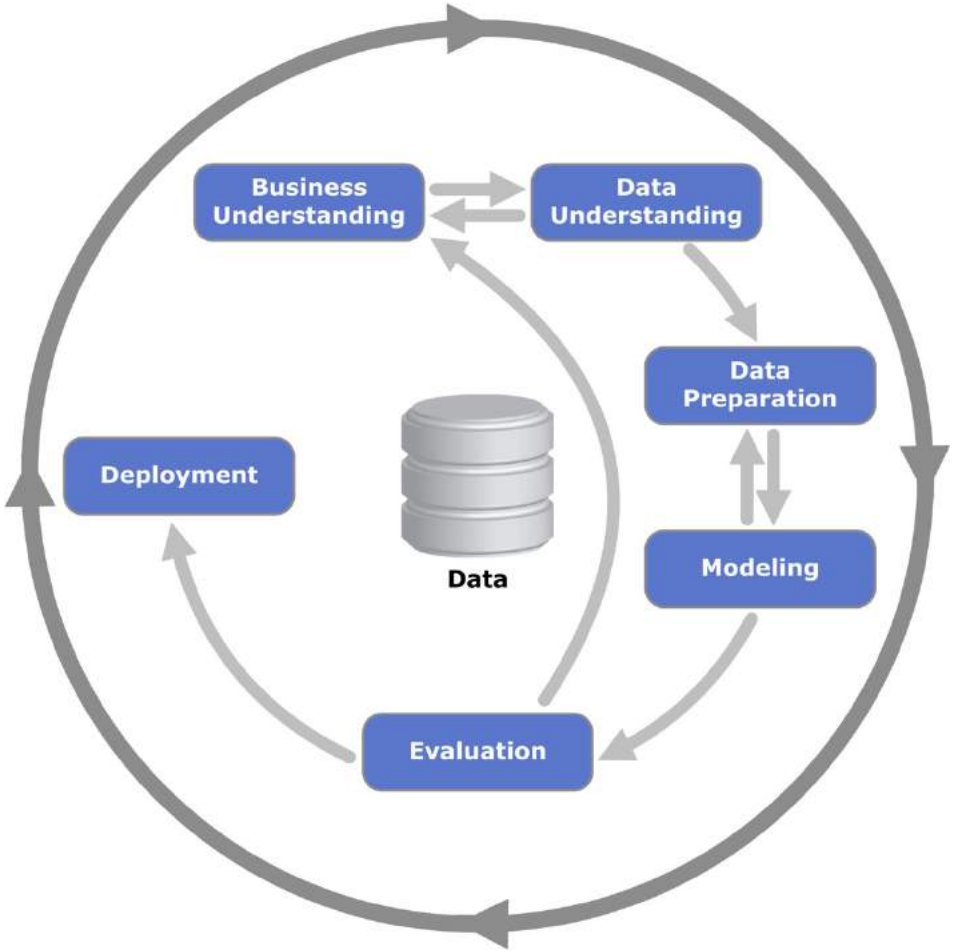
Source: https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818

Live Demo
Kmeans & DBSCAN

Use Case
Clustering

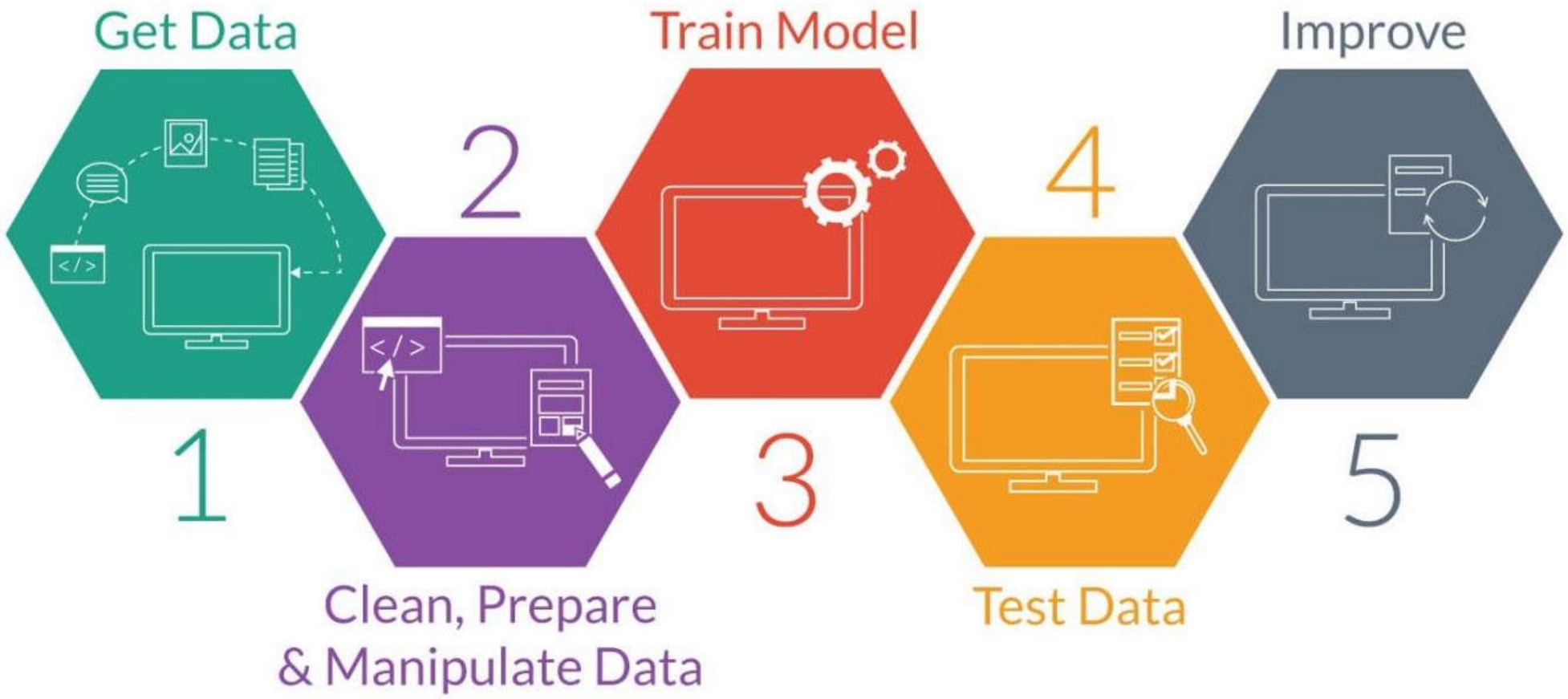
Machine Learning Workflow

The bigger picture – CRISP-DM



https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining

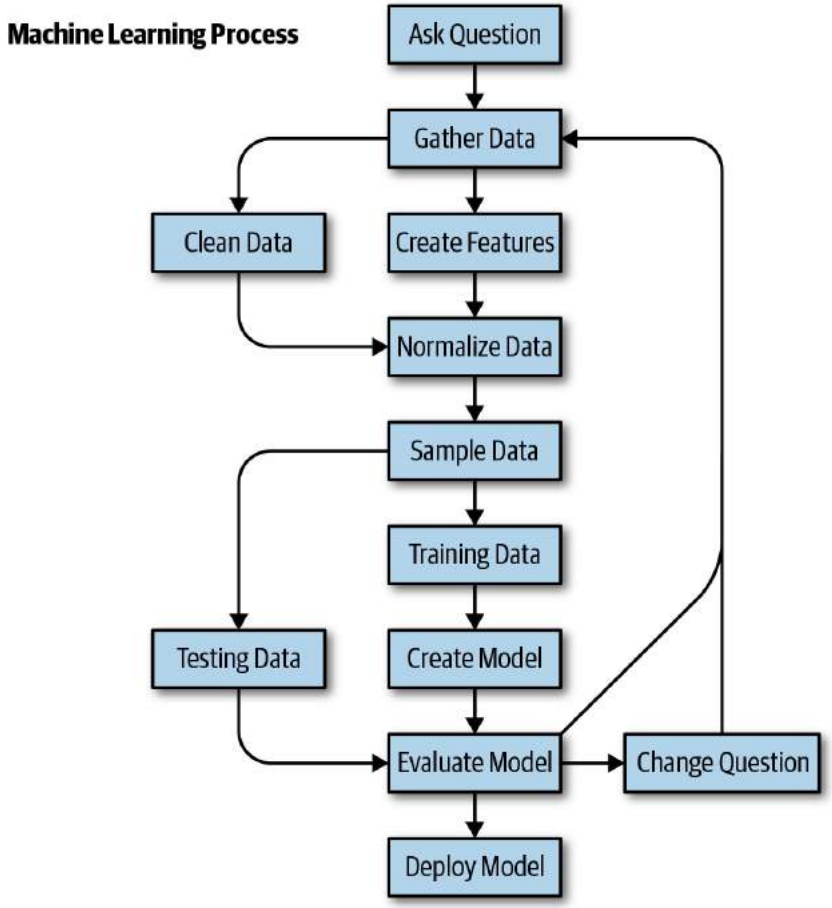
The Machine Learning workflow - simplified



https://www.freecodecamp.org/news/every-single-machine-learning-course-on-the-internet-ranked-by-your-reviews-3c4a7b8026c0/

A more detailed workflow

Machine Learning Pocket Reference by Matt Harrison



THE END

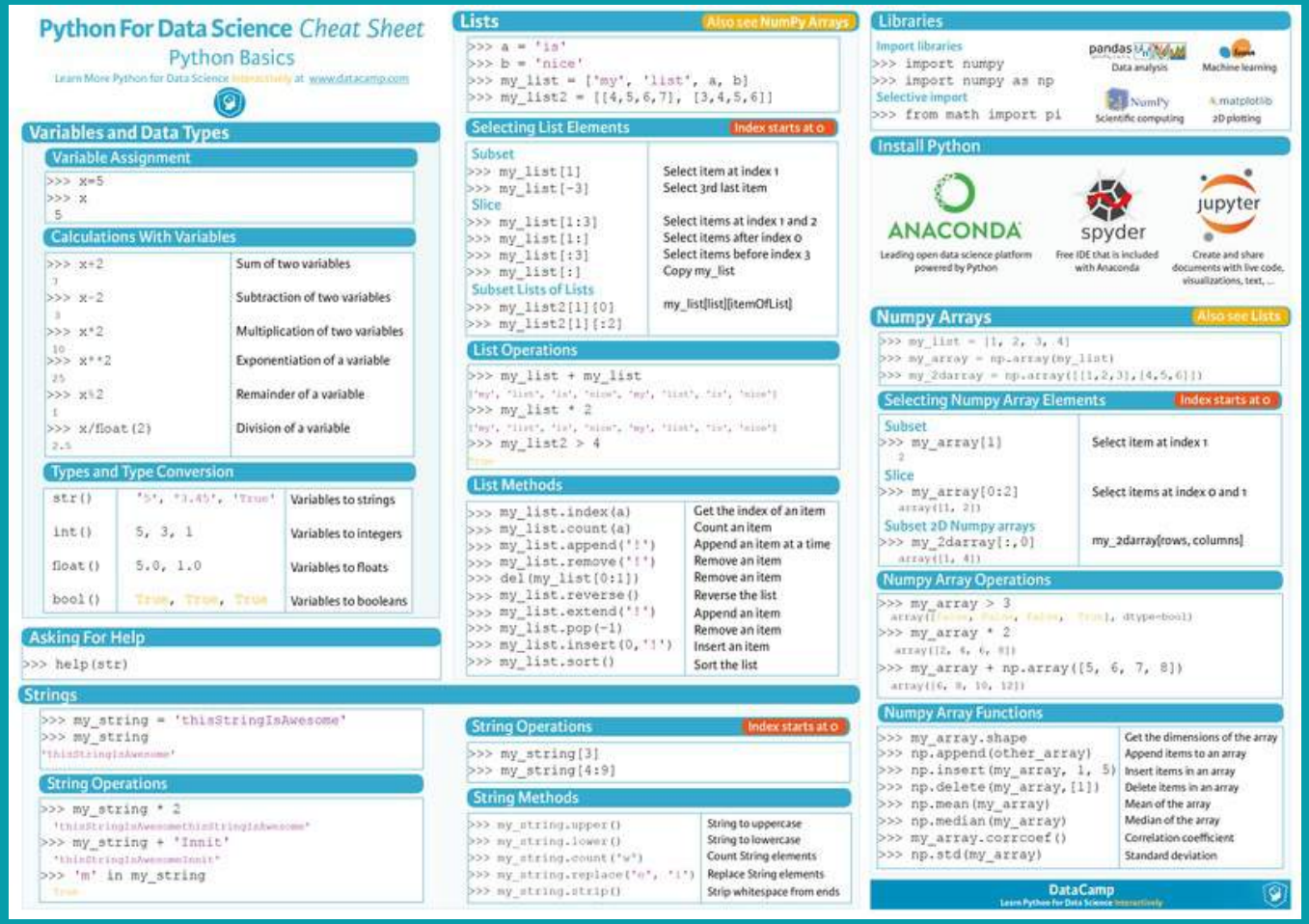
CONNECTING WORLDS

Kontakt

Data Scientist

philipp.bongartz@EXXETA.com

Cheat Sheets



Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

`>>> x+2` → `7`	Sum of two variables
`>>> x-2` → `3`	Subtraction of two variables
`>>> x*2` → `10`	Multiplication of two variables
`>>> x**2` → `25`	Exponentiation of a variable
`>>> x%2` → `1`	Remainder of a variable
`>>> x/float(2)` → `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5'`, `'3.45'`, `'True'`	Variables to strings
`int()`	`5`, `3`, `1`	Variables to integers
`float()`	`5.0`, `1.0`	Variables to floats
`bool()`	`True`, `True`, `True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements — Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

String Operations — Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespace from ends

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis; Machine learning; NumPy — Scientific computing; matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements — Index starts at 0

Subset	
`>>> my_array[1]` → `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` → `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` → `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

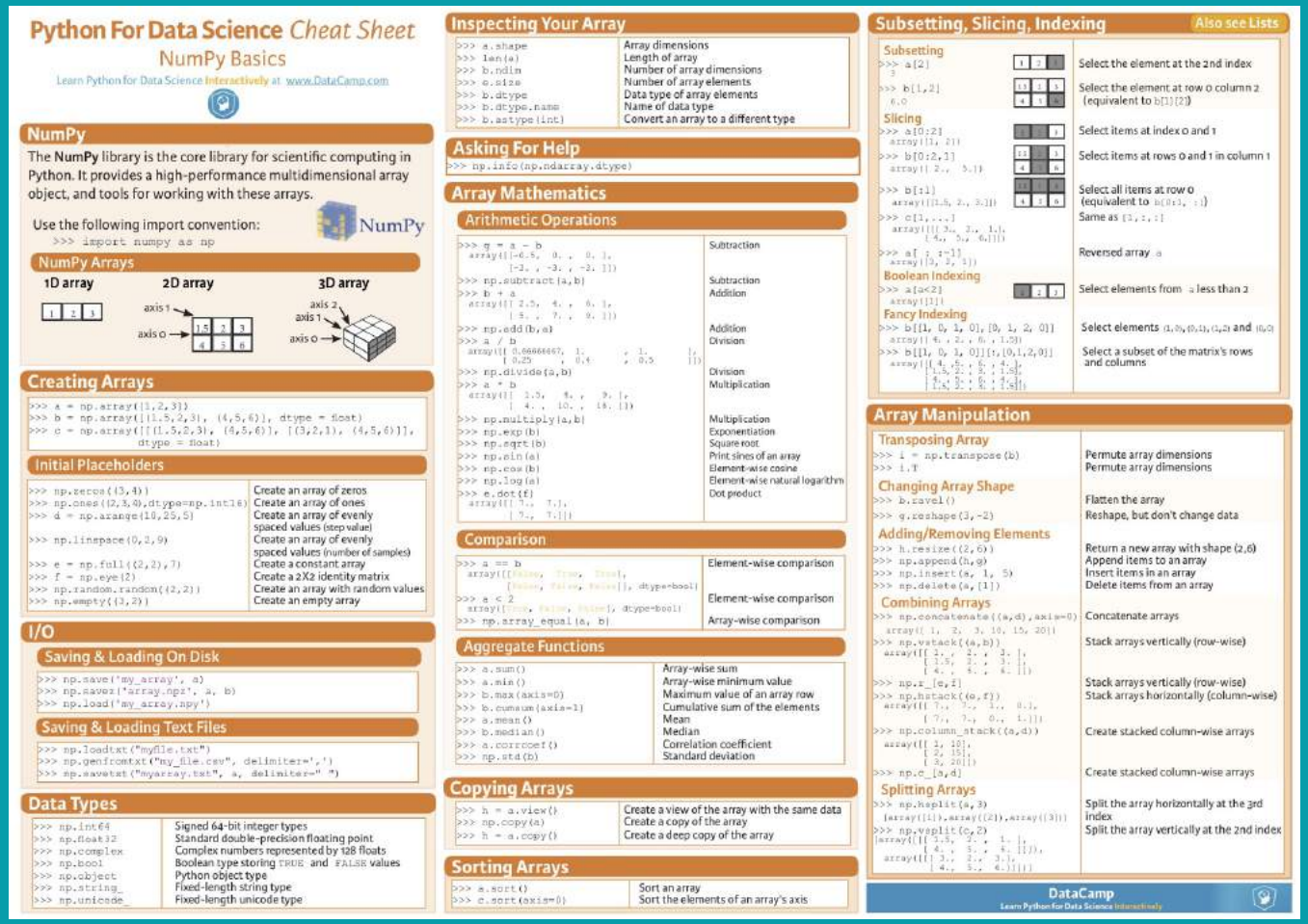
```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

DataCamp — Learn Python for Data Science Interactively

Numpy Cheat Sheet



Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays

1D array

2D array

3D array

Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))
>>> np.ones((2,3,4),dtype=np.int16)
>>> d = np.arange(10,25,5)

>>> np.linspace(0,2,9)

>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of evenly spaced values (number of samples)
Create a constant array
Create a 2X2 identity matrix
Create an array with random values
Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64
>>> np.float32
>>> np.complex
>>> np.bool
>>> np.object
>>> np.string_
>>> np.unicode_
```

Signed 64-bit integer types
Standard double-precision floating point
Complex numbers represented by 128 floats
Boolean type storing TRUE and FALSE values
Python object type
Fixed-length string type
Fixed-length unicode type

Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> e.size
>>> b.dtype
>>> b.dtype.name
>>> b.astype(int)
```

Array dimensions
Length of array
Number of array dimensions
Number of array elements
Data type of array elements
Name of data type
Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
>>> np.subtract(a,b)
>>> b + a
>>> np.add(b,a)
>>> a / b
>>> np.divide(a,b)
>>> a * b
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
```

Subtraction
Subtraction
Addition
Addition
Division
Division
Multiplication
Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Comparison

```
>>> a == b
>>> a < 2
>>> np.array_equal(a, b)
```

Element-wise comparison
Element-wise comparison
Array-wise comparison

Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.corrcoef()
>>> np.std(b)
```

Array-wise sum
Array-wise minimum value
Maximum value of an array row
Cumulative sum of the elements
Mean
Median
Correlation coefficient
Standard deviation

Copying Arrays

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

Create a view of the array with the same data
Create a copy of the array
Create a deep copy of the array

Sorting Arrays

```
>>> a.sort()
>>> c.sort(axis=0)
```

Sort an array
Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]
>>> b[1,2]
```

Select the element at the 2nd index
Select the element at row 0 column 2 (equivalent to b[1][2])

Slicing

```
>>> a[0:2]
>>> b[0:2,1]
>>> b[:1]
>>> c[1,...]
>>> a[ : :-1]
```

Select items at index 0 and 1
Select items at rows 0 and 1 in column 1
Select all items at row 0 (equivalent to b[0:1, :])
Same as [1,:,:]
Reversed array a

Boolean Indexing

```
>>> a[a<2]
```

Select elements from a less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
```

Select elements (1,0),(0,1),(1,2) and (0,0)
Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```

Permute array dimensions
Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
>>> g.reshape(3,-2)
```

Flatten the array
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a,[1])
```

Return a new array with shape (2,6)
Append items to an array
Insert items in an array
Delete items from an array

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
>>> np.vstack((a,b))
>>> np.r_[e,f]
>>> np.hstack((e,f))
>>> np.column_stack((a,d))
>>> np.c_[a,d]
```

Concatenate arrays
Stack arrays vertically (row-wise)
Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)
Create stacked column-wise arrays
Create stacked column-wise arrays

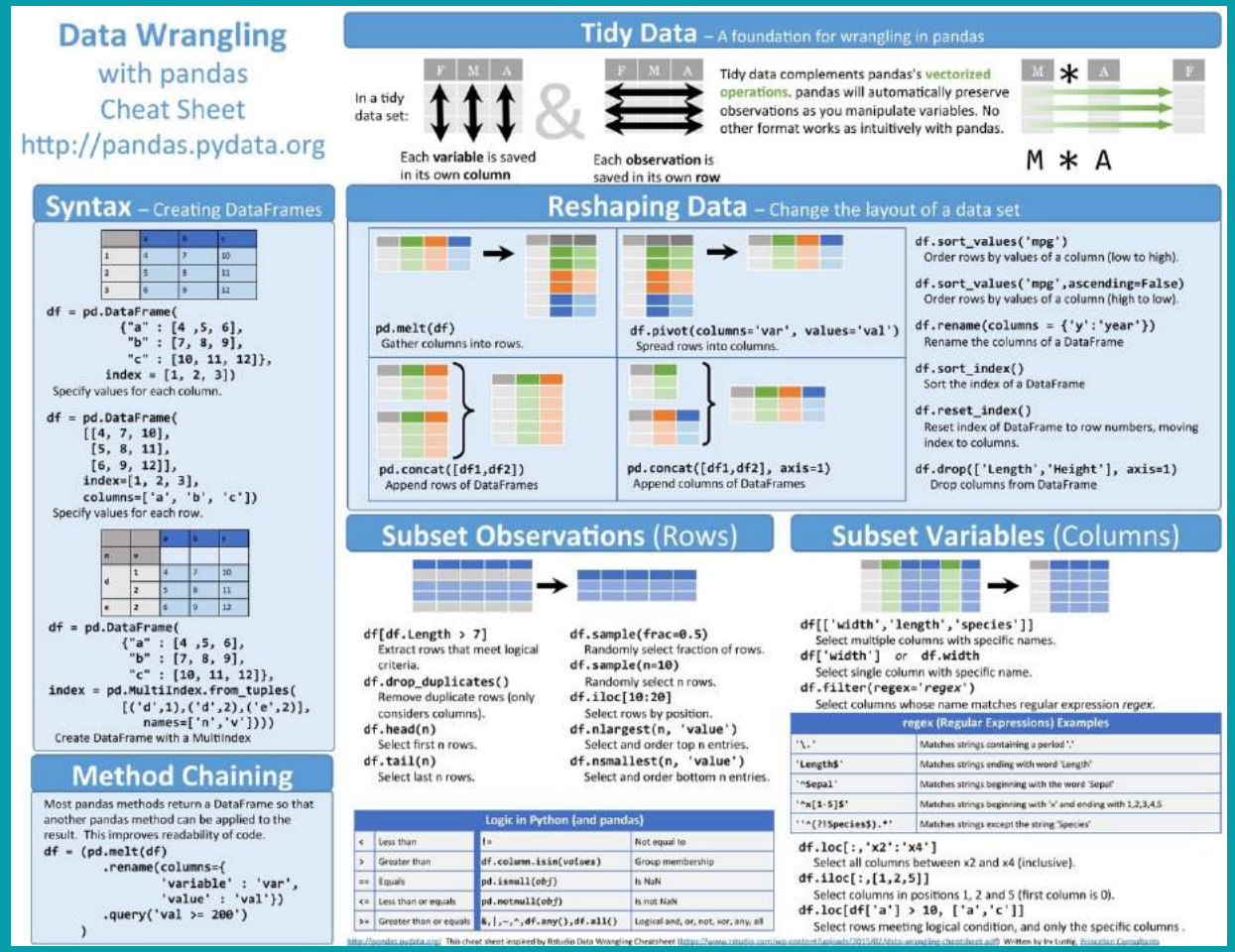
Splitting Arrays

```
>>> np.hsplit(a,3)
>>> np.vsplit(c,2)
```

Split the array horizontally at the 3rd index
Split the array vertically at the 2nd index

DataCamp
Learn Python for Data Science Interactively

Pandas Cheat Sheet



Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
        index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
         names=['n','v'])))
```

Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set

`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(['Length','Height'], axis=1)`
Drop columns from DataFrame

Subset Observations (Rows)

`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	&,|,~,^,`df.any()`,`df.all()`	Logical and, or, not, xor, any, all

Subset Variables (Columns)

`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

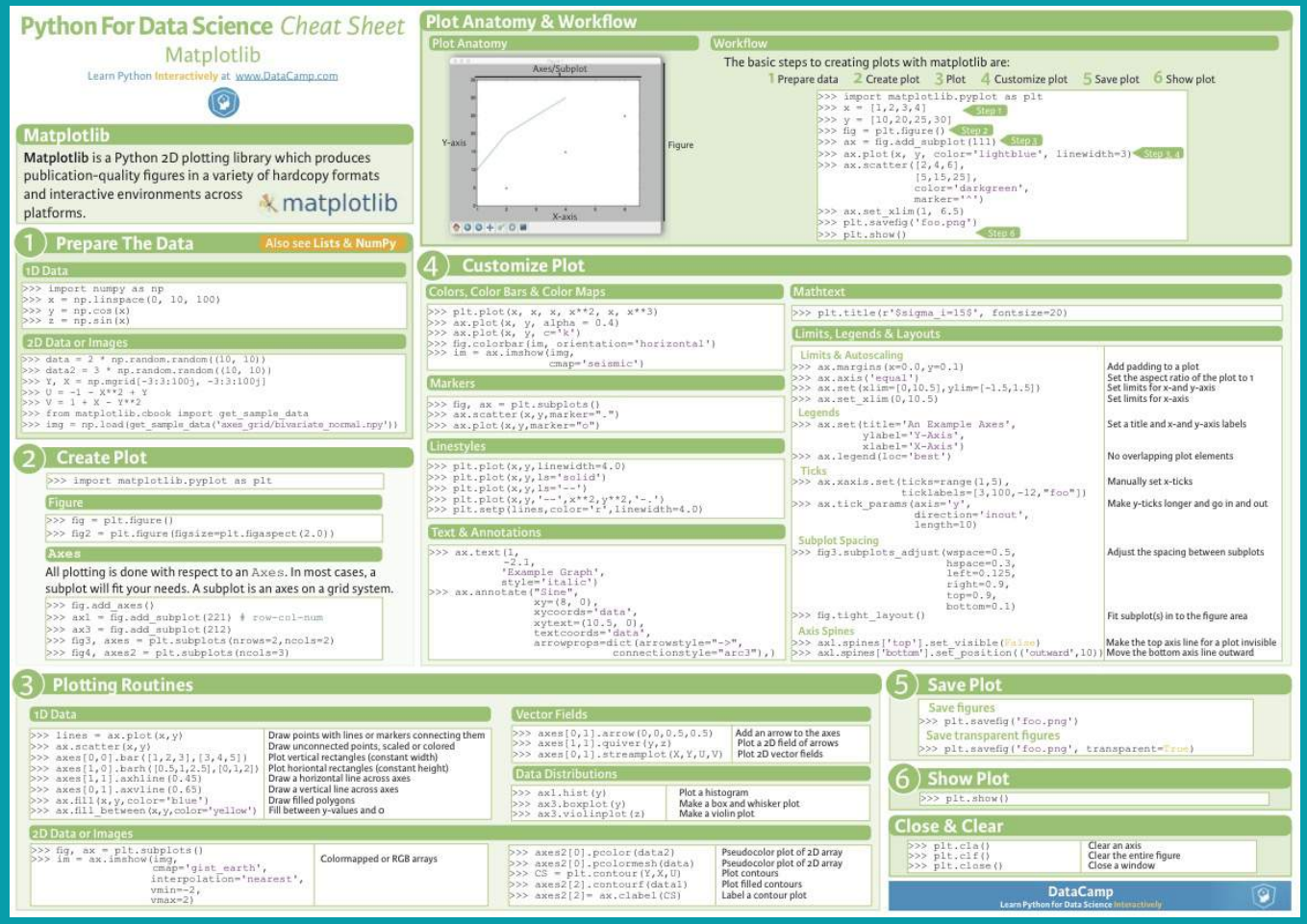
`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`'^(?!Species$).*'`	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

Matplotlib Cheat Sheet



Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

Plot Anatomy & Workflow

Plot Anatomy

Axes/Subplot, Y-axis, X-axis, Figure

Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

3 Plotting Routines

1D Data

```
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them
Draw unconnected points, scaled or colored
Plot vertical rectangles (constant width)
Plot horiontal rectangles (constant height)
Draw a horizontal line across axes
Draw a vertical line across axes
Draw filled polygons
Fill between y-values and 0

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)
>>> axes[1,1].quiver(y,z)
>>> axes[0,1].streamplot(X,Y,U,V)
```

Add an arrow to the axes
Plot a 2D field of arrows
Plot 2D vector fields

Data Distributions

```
>>> ax1.hist(y)
>>> ax3.boxplot(y)
>>> ax3.violinplot(z)
```

Plot a histogram
Make a box and whisker plot
Make a violin plot

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

Colormapped or RGB arrays

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2]= ax.clabel(CS)
```

Pseudocolor plot of 2D array
Pseudocolor plot of 2D array
Plot contours
Plot filled contours
Label a contour plot

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

Add padding to a plot
Set the aspect ratio of the plot to 1
Set limits for x-and y-axis
Set limits for x-axis

Legends

```
>>> ax.set(title='An Example Axes',
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')
```

Set a title and x-and y-axis labels
No overlapping plot elements

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',
                   direction='inout',
                   length=10)
```

Manually set x-ticks
Make y-ticks longer and go in and out

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()
```

Adjust the spacing between subplots
Fit subplot(s) in to the figure area

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Make the top axis line for a plot invisible
Move the bottom axis line outward

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

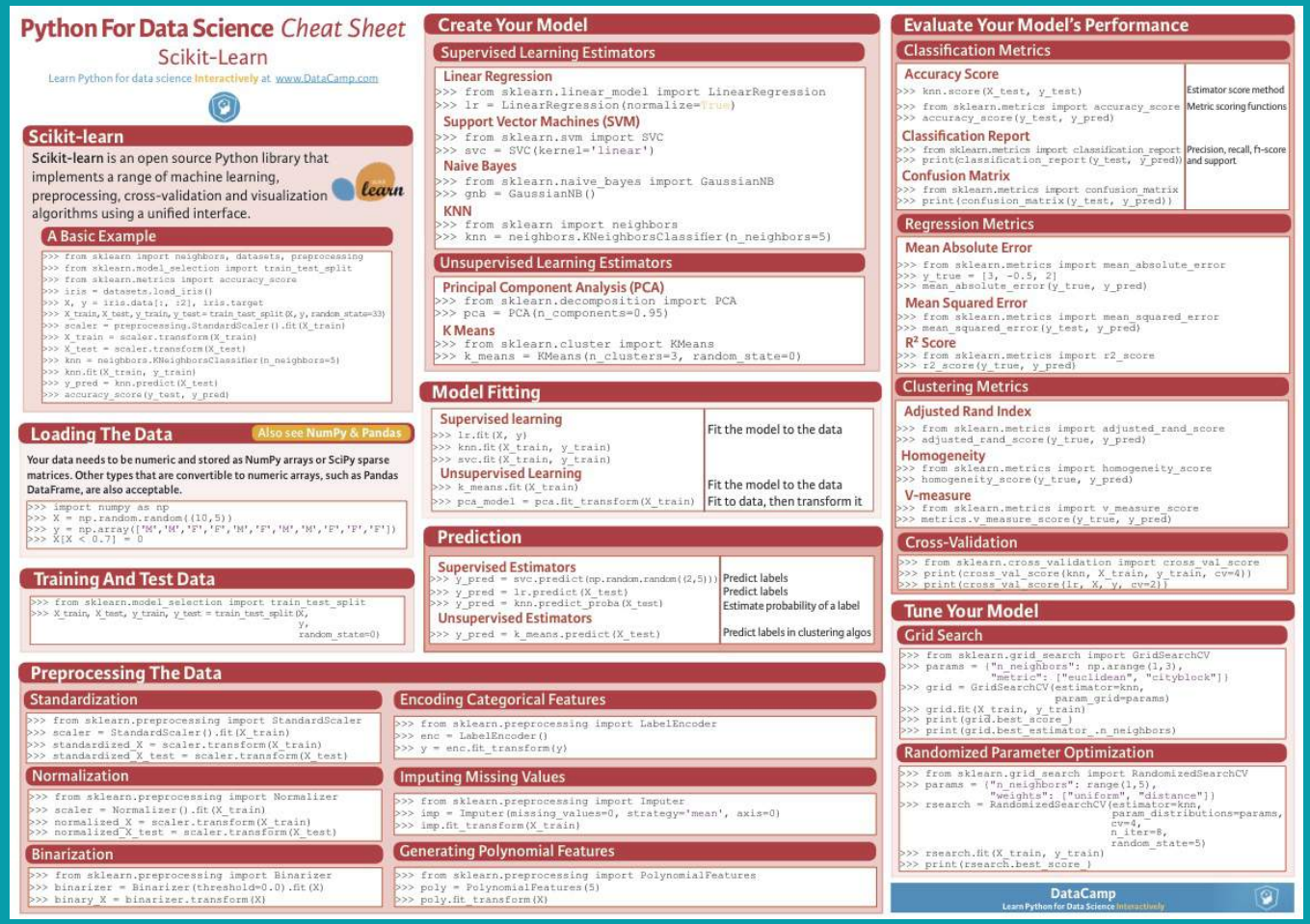
Close & Clear

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

Clear an axis
Clear the entire figure
Close a window

DataCamp
Learn Python for Data Science Interactively

Sscikit-Learn Cheat Sheet



Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp
Learn Python for Data Science Interactively

Scikit-learn Design

https://arxiv.org/abs/1309.0238

Scikit-Learn's API is remarkably well designed. These are the main design principles:

Consistency: All objects share a consistent and simple interface:

- **Estimators** Any object that can estimate some parameters based on a dataset is called an *estimator* (e.g., an imputer is an estimator). The estimation itself is performed by the fit() method, and it takes only a dataset as a parameter (or two for supervised learning algorithms; the second dataset contains the labels). Any other parameter needed to guide the estimation process is considered a hyperparameter (such as an imputer's strategy), and it must be set as an instance variable (generally via a constructor parameter).
- **Transformers** Some estimators (such as an imputer) can also transform a dataset; these are called *transformers*. Once again, the API is simple: the transformation is performed by the transform() method with the dataset to transform as a parameter. It returns the transformed dataset. This transformation generally relies on the learned parameters, as is the case for an imputer. All transformers also have a convenience method called fit_transform() that is equivalent to calling fit() and then transform() (but sometimes fit_transform() is optimized and runs much faster).
- **Predictors** Finally, some estimators, given a dataset, are capable of making predictions; they are called *predictors*. For example, the LinearRegression model in the previous chapter was a predictor: given a country's GDP per capita, it predicted life satisfaction. A predictor has a predict() method that takes a dataset of new instances and returns a dataset of corresponding predictions. It also has a score() method that measures the quality of the predictions, given a test set (and the corresponding labels, in the case of supervised learning algorithms).[18]

Inspection: All the estimator's hyperparameters are accessible directly via public instance variables (e.g., imputer.strategy), and all the estimator's learned parameters are accessible via public instance variables with an underscore suffix (e.g., imputer.statistics_).

Nonproliferation of classes Datasets are represented as NumPy arrays or SciPy sparse matrices, instead of homemade classes. Hyperparameters are just regular Python strings or numbers.

Composition Existing building blocks are reused as much as possible. For example, it is easy to create a Pipeline estimator from an arbitrary sequence of transformers followed by a final estimator, as we will see.

Sensible defaults Scikit-Learn provides reasonable default values for most parameters, making it easy to quickly create a baseline working system.

Where to go from here

1. Learn the language
2. Learn the tools
3. Work on projects!

Resources

- **Machine Learning Mastery**
- **Hands-On Machine Learning with Scikit-Learn and TensorFlow**
- **The Complete Python Course for Machine Learning Engineers**
- **Cheat Sheet of Machine Learning and Python (and Math) Cheat Sheets**

Webinar’s Agenda

1. Machine Learning Basics

- Definition
- Relation to other fields
- Little terminology

2. The ML Workflow

- Get data
- Prepare data
- Model selection and training
- Model evaluation
- Model improvement

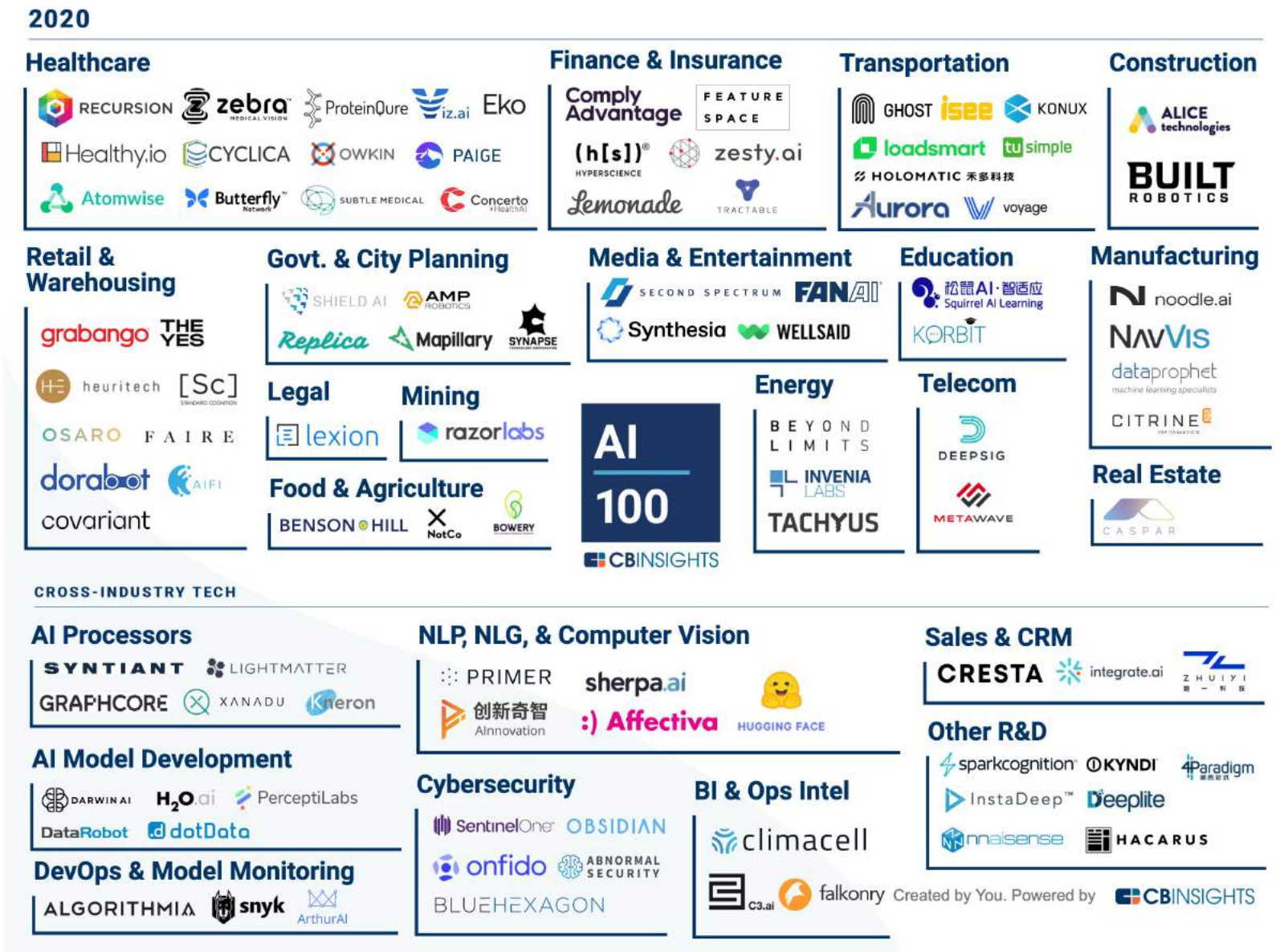
3. Resources

Machine Learning
What comes to your mind?

“Every serious technology company now has an Artificial Intelligence team in place. These companies are investing millions into intelligent systems for situation assessment, prediction analysis, learning-based recognition systems, conversational interfaces, and recommendation engines. Companies such as Google, Facebook, and Amazon aren’t just employing AI, but have made it a central part of their core intellectual property.” - Kristian J. Hammond

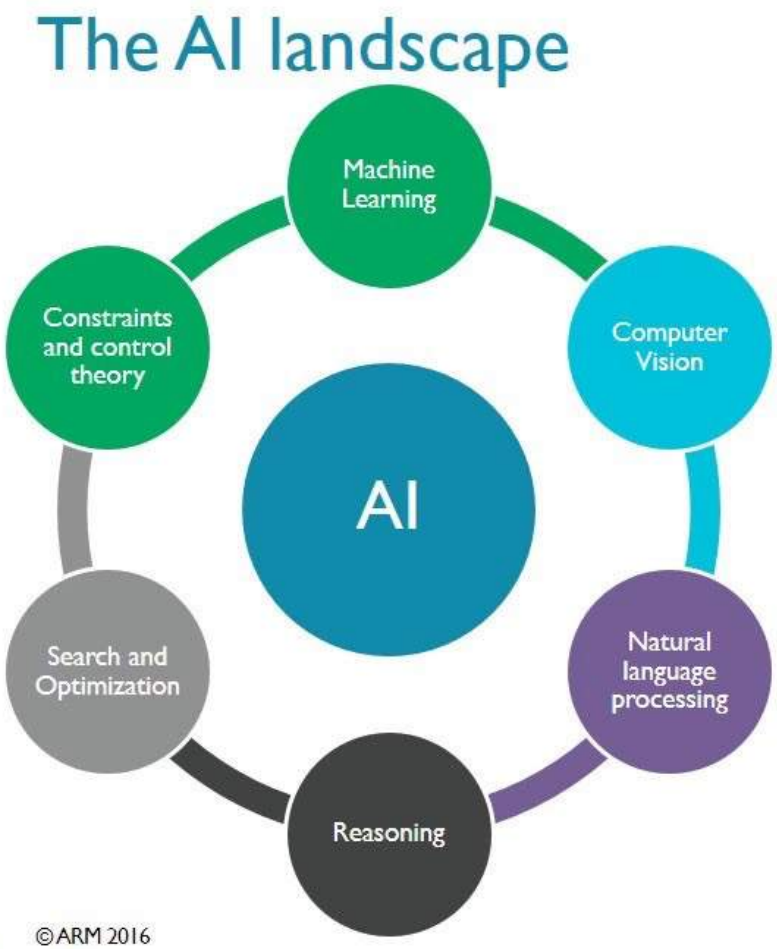
Startups and AI

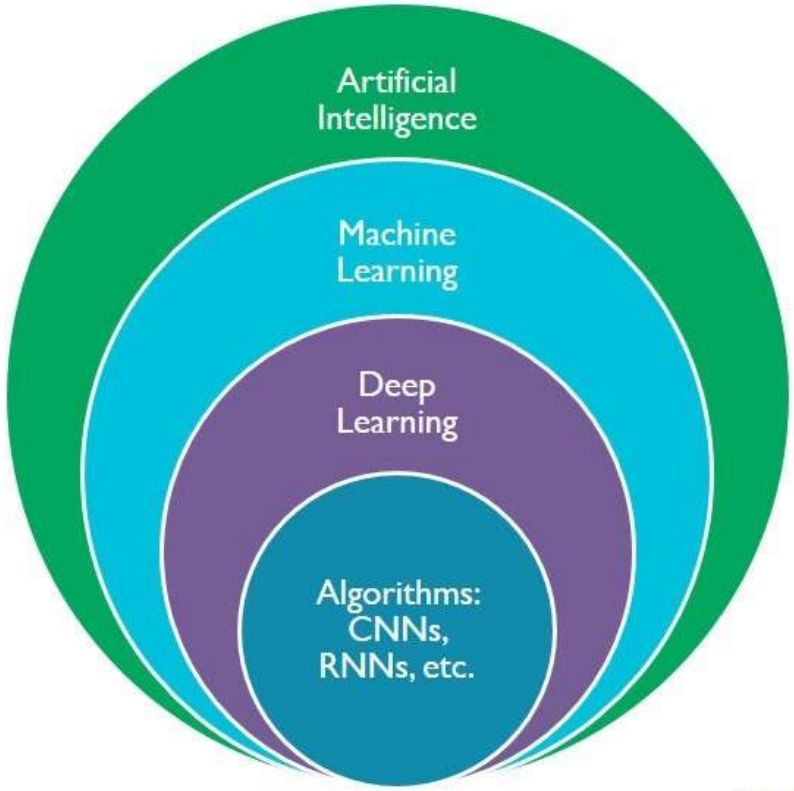
The AI 100 is CB Insights' annual ranking of the 100 most promising AI startups in the world.



https://www.cbinsights.com/research/artificial-intelligence-top-startups/

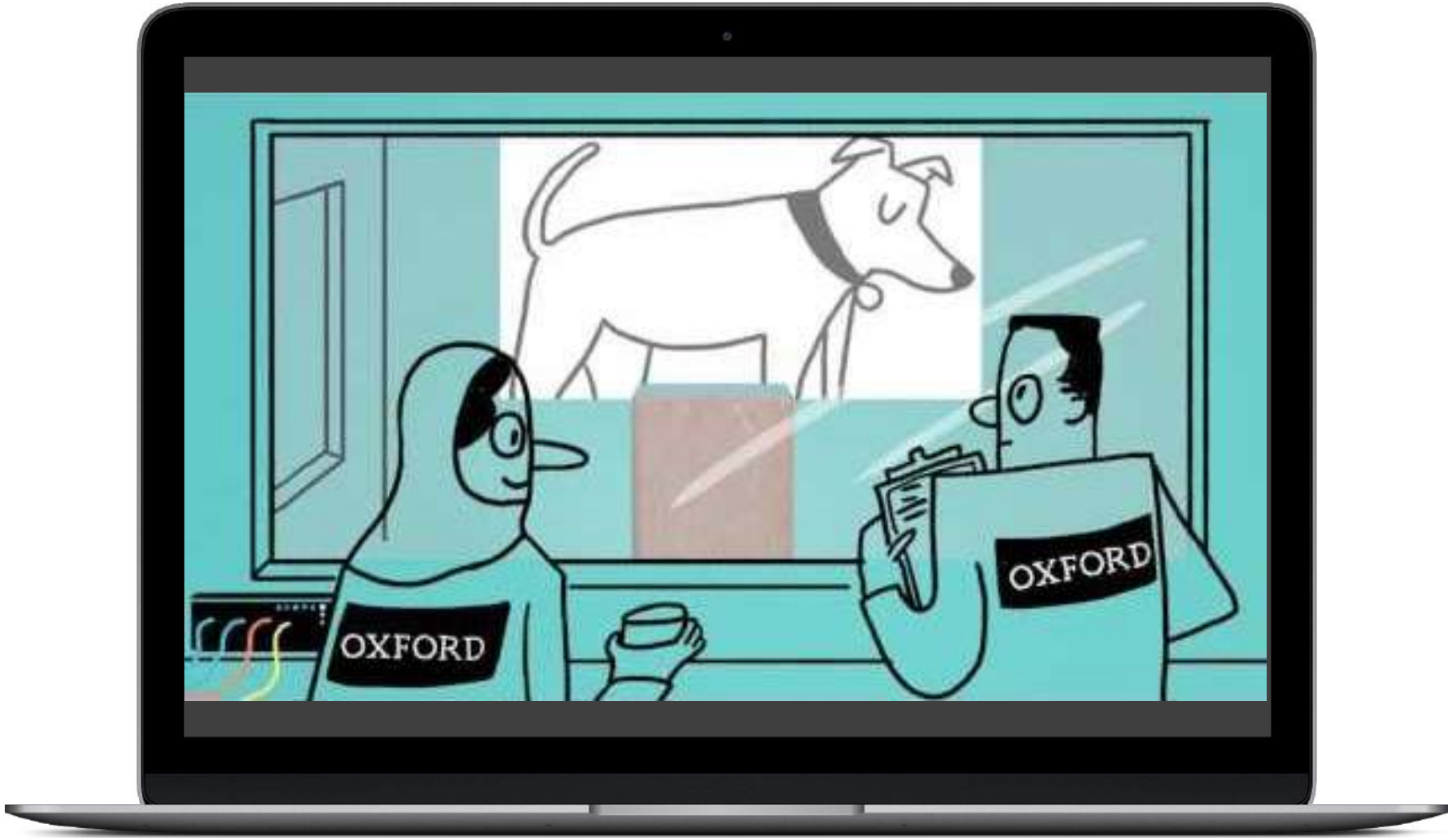
Artificial Intelligence and Machine Learning





ARM

What does “learning” mean?



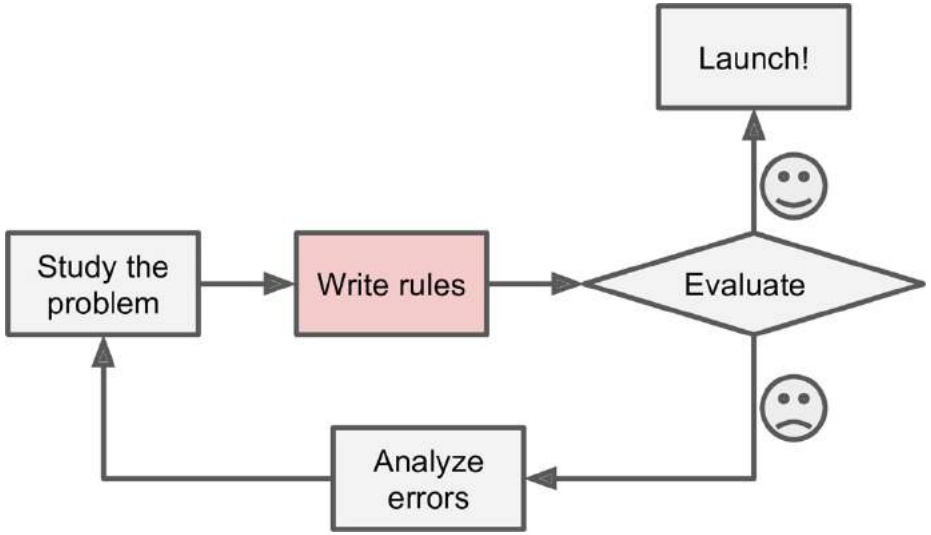
https://youtu.be/f_uwKZIAeM0

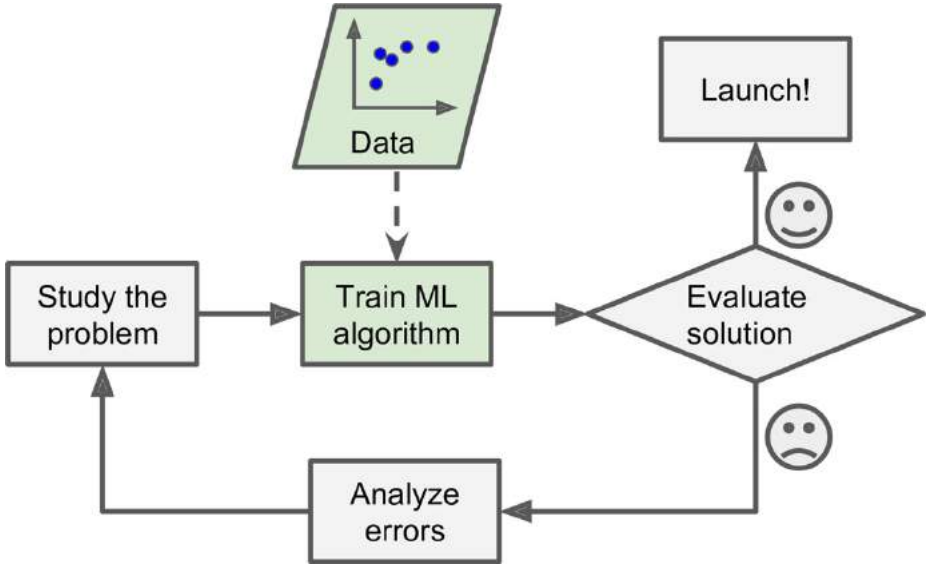
What does “learning” mean?

1. “Machine learning is the science of getting computers to act without being explicitly programmed.” – Andrew Ng, Stanford University (based on Arthur Samuel, *1959)*
2. “Machine learning algorithms can figure out how to perform important tasks by generalizing from examples.” – University of Washington
3. “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” – Tom Mitchell

Why Use Machine Learning?

How would you implement a spam filter?

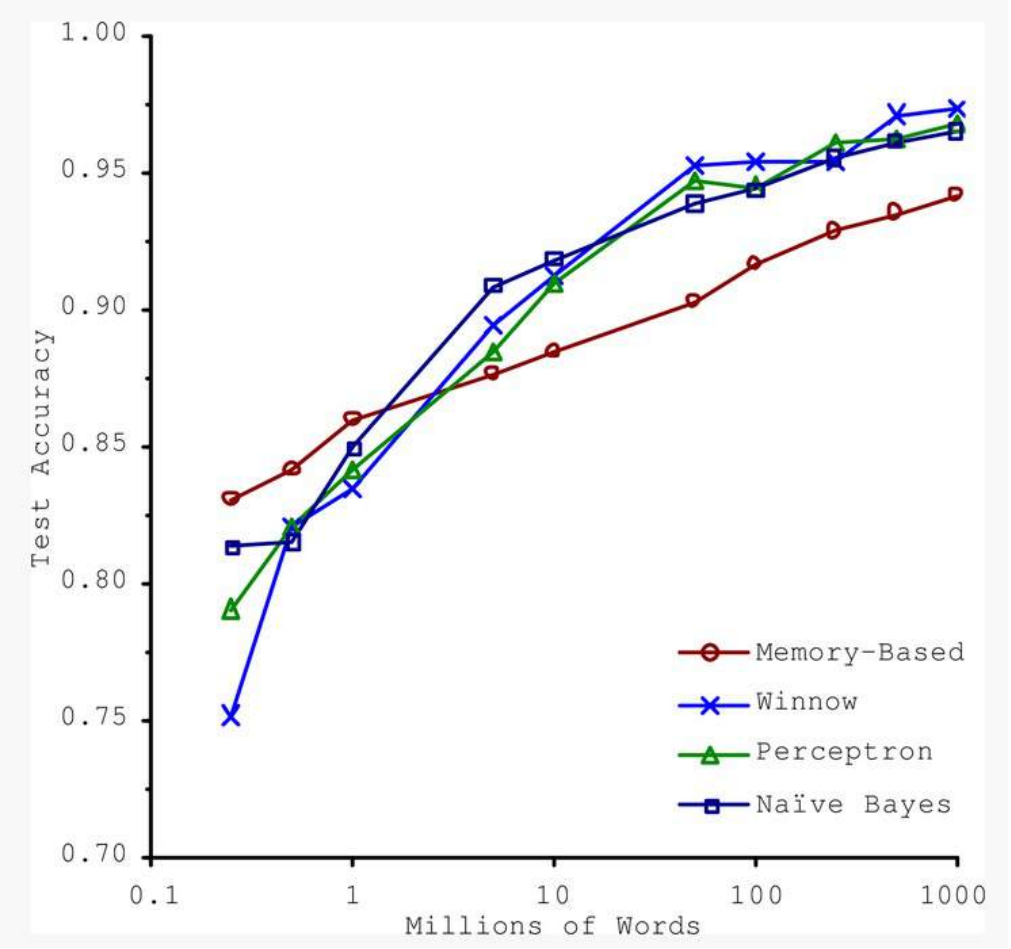




Aurélien Géron (2017), Hands-On Machine Learning with Scikit-Learn and TensorFlow, Publisher: O'Reilly Media, Inc., ISBN: 9781491962299

Main challenges of Machine Learning

- **Insufficient Quantity of Training Data**
- **Nonrepresentative Training Data**
- **Poor-Quality Data**
- **Irrelevant Features**
- **Overfitting the Training Data**
- **Underfitting the Training Data**

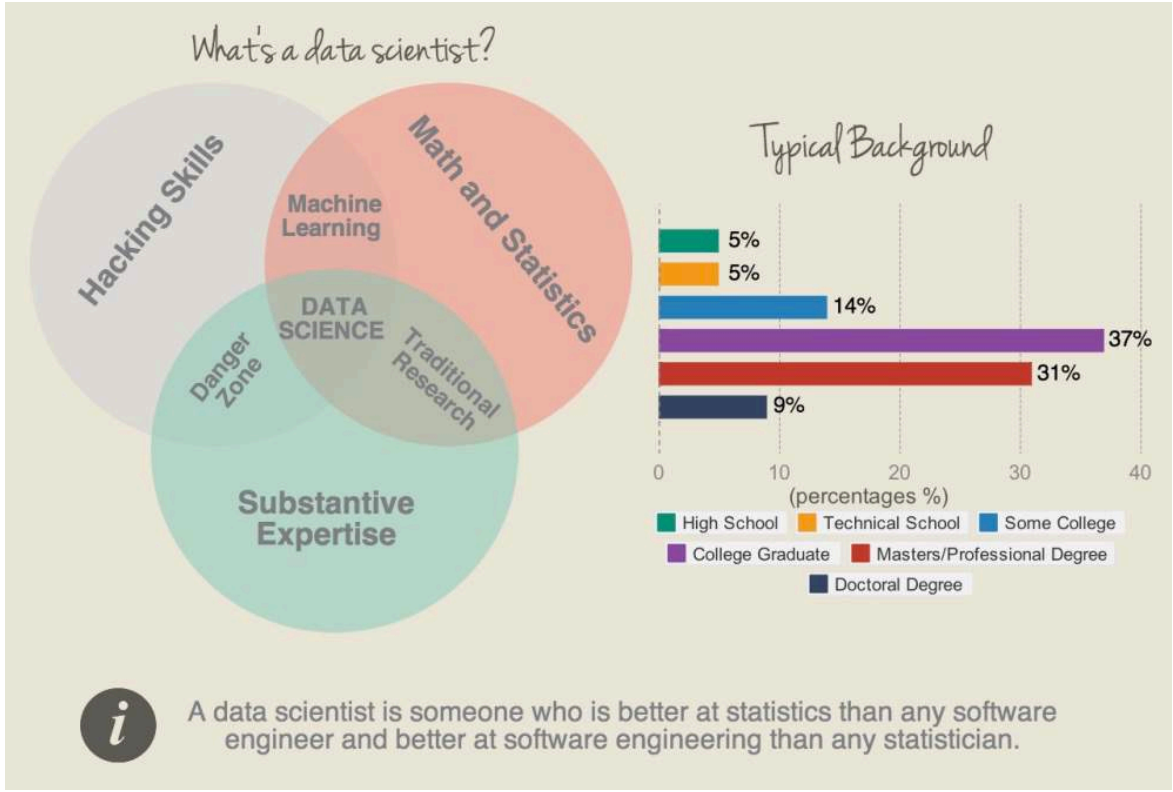


Michele Banko and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics* (*ACL '01*). Association for Computational Linguistics, USA, 26–33. DOI:https://doi.org/10.3115/1073012.1073017

Machine Learning Workflow

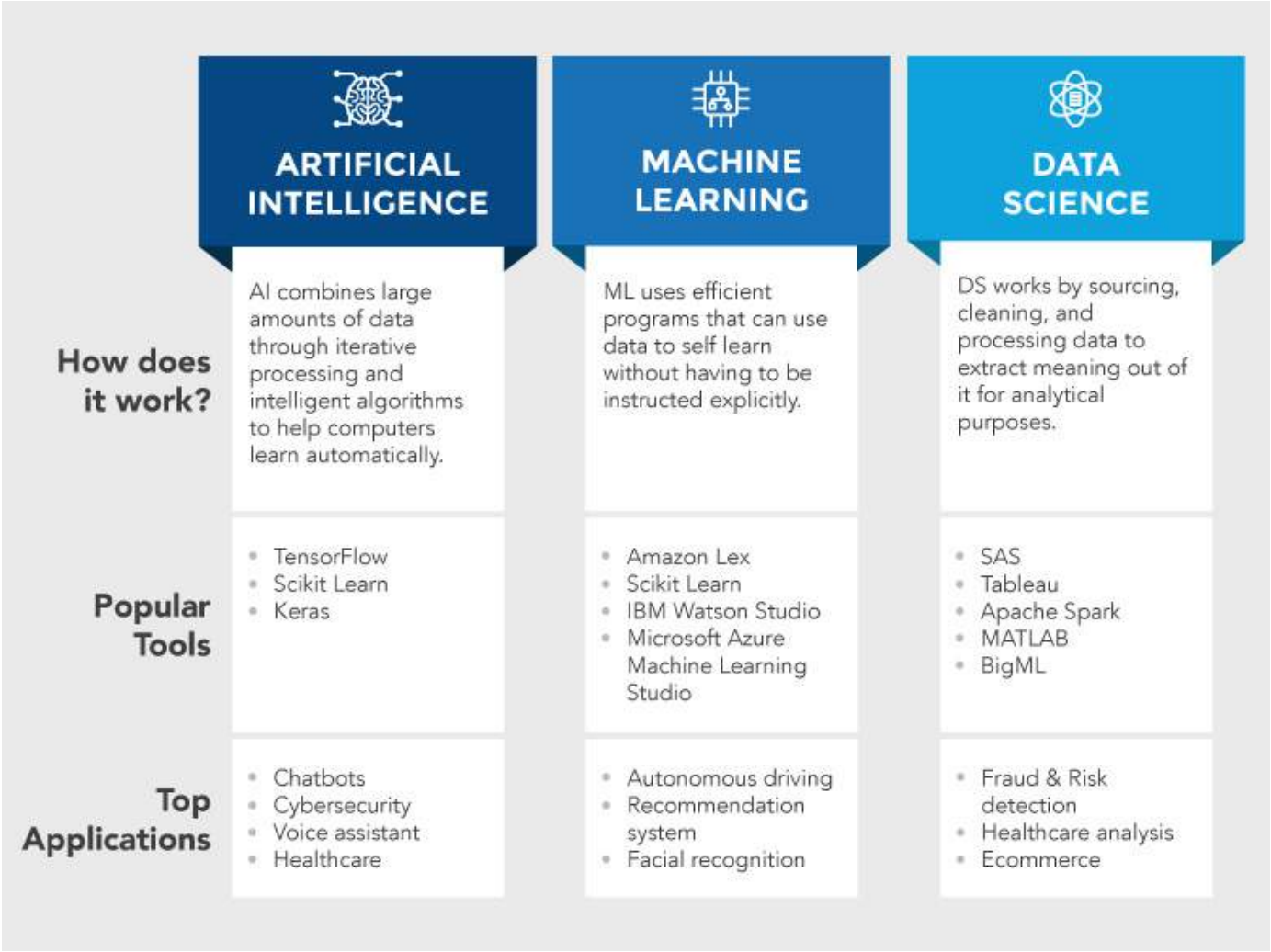
The bigger picture – skillset of a data scientist

https://www.simplilearn.com/data-science-vs-data-analytics-vs-machine-learning-article



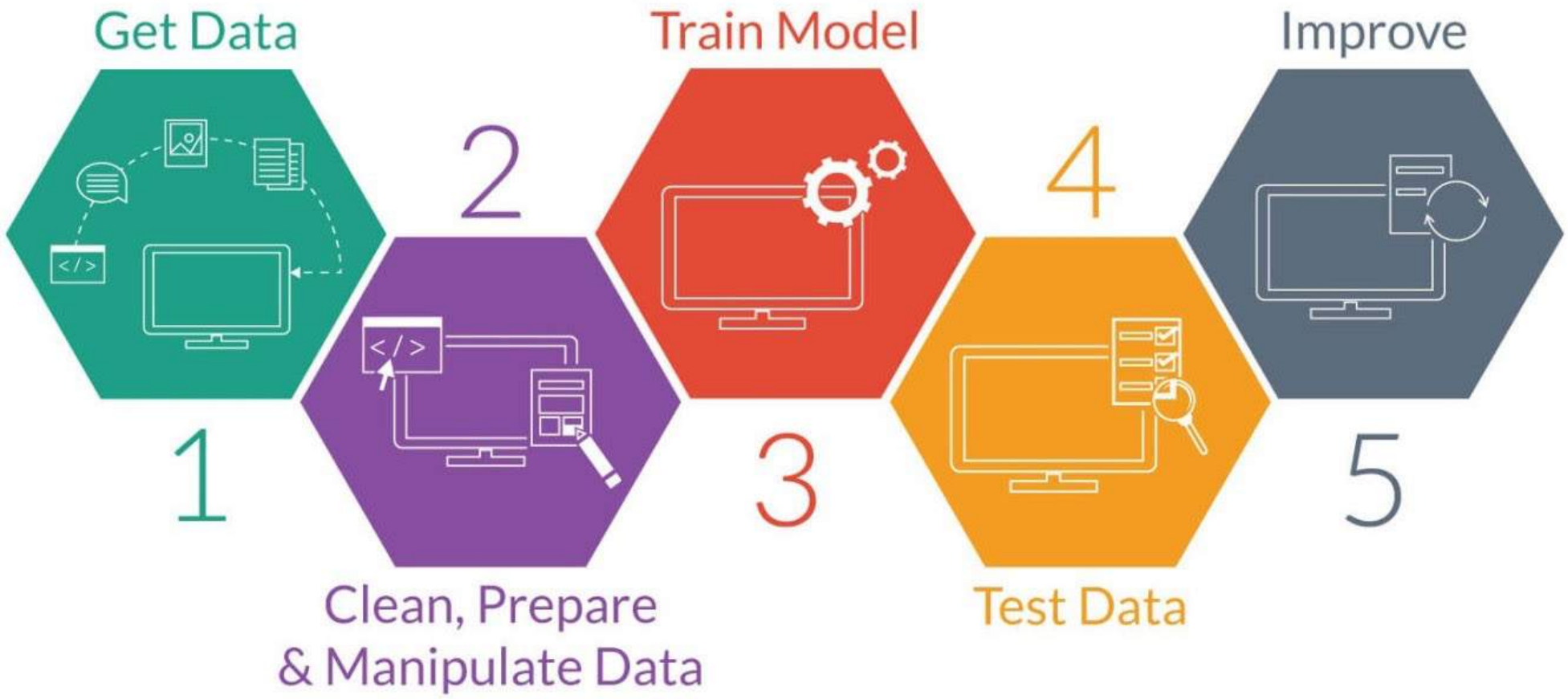
https://machinelearningmastery.com/become-data-scientist/

The bigger picture – AI vs ML vs DS



https://www.mygreatlearning.com/blog/difference-data-science-machine-learning-ai/

The Machine Learning workflow - simplified



The Machine Learning workflow - example

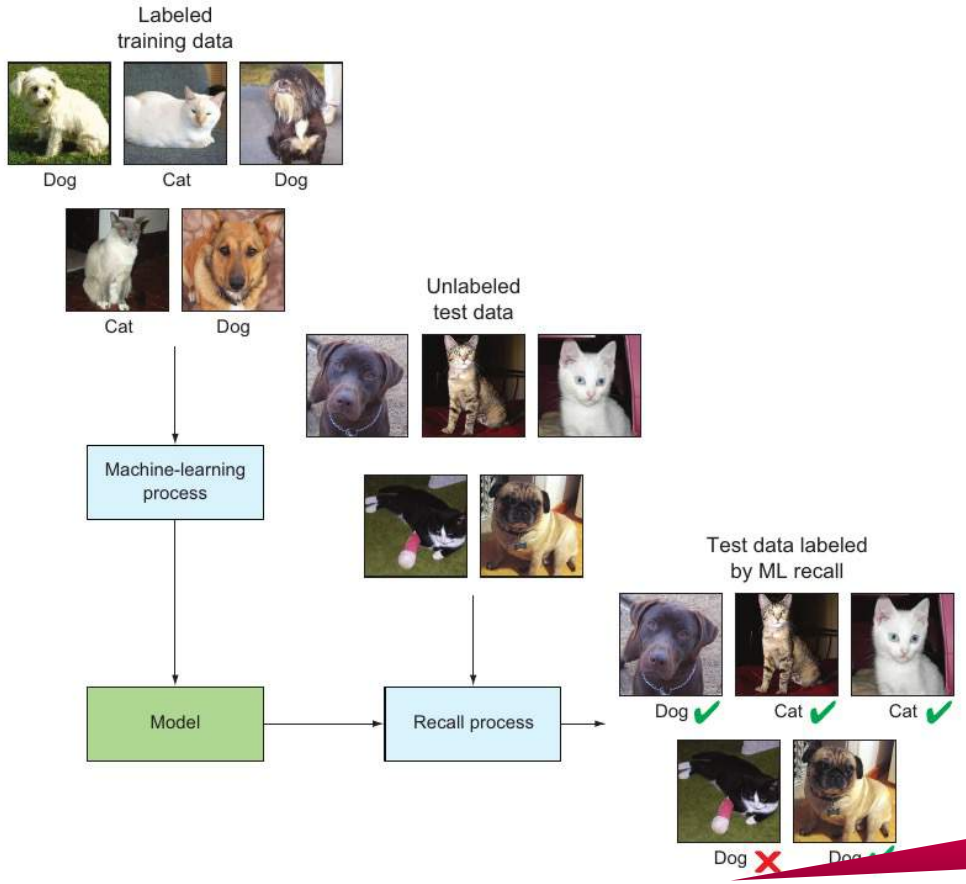


Figure 1.1 Machine-learning process for the cats and dogs competition

Any error is a valuable information by itself!

Henrik Brink, Joseph W. Richards, and Mark Fetherolf , Real-World Machine Learning, Manning, September 2016, ISBN 9781617291920

Why you should care about your data set

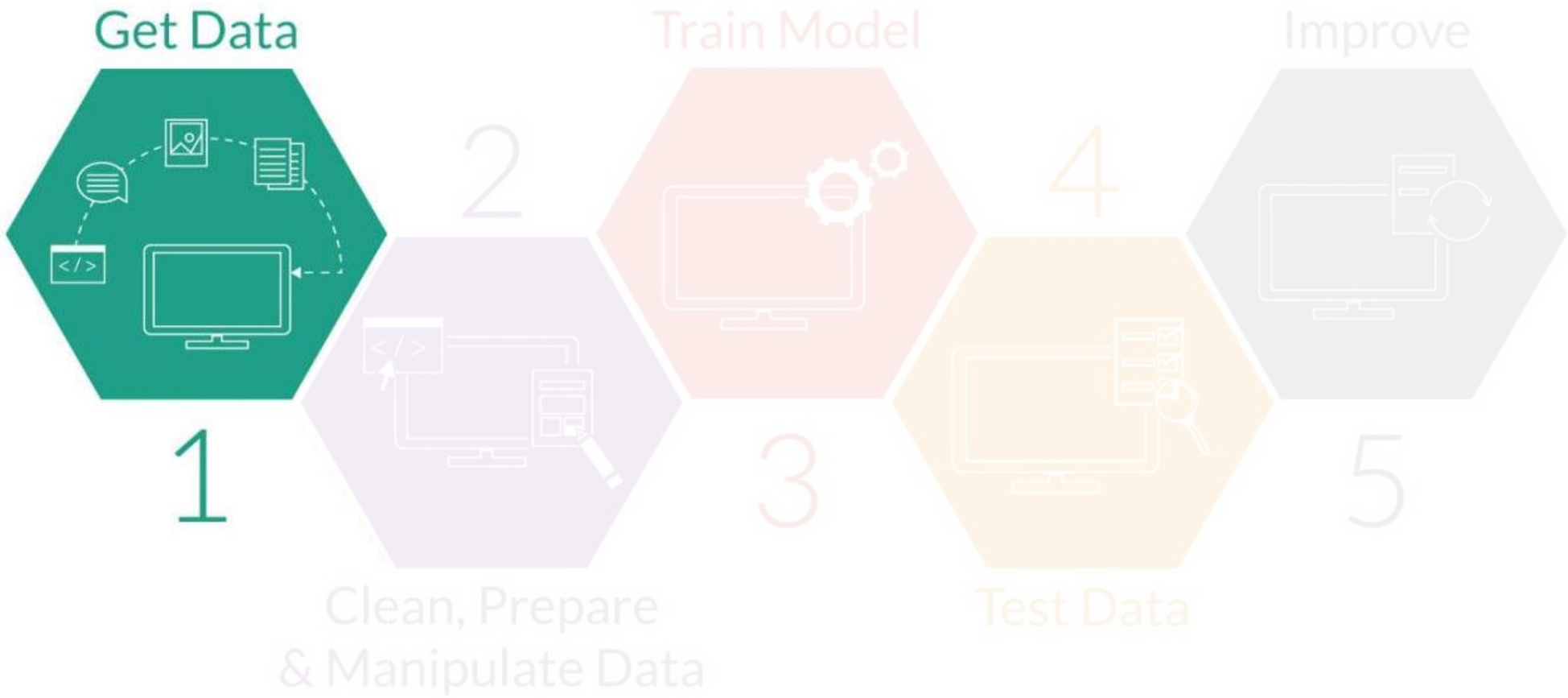
Detecting tanks

US Army wanted to use Neural Networks to detect tanks hidden behind trees

However...something went wrong

https://neil.fraser.name/writing/tank/

Get Data


Get Data
1
Clean, Prepare
& Manipulate Data
2
Train Model
3
Test Data
4
Improve
5

Get Data

Machine Learning Terminology

Header

Rows/ Examples/ Instances

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1	0	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
5	1	1	Anderson, Mr. Harry	male	48.0000	0	0	19952	26.5500	E12	S	3	NaN	New York, NY
6	1	1	Andrews, Miss. Kornelia Theodosia	female	63.0000	1	0	13502	77.9583	D7	S	10	NaN	Hudson, NY
7	1	0	Andrews, Mr. Thomas Jr	male	39.0000	0	0	112050	0.0000	A36	S	NaN	NaN	Belfast, NI
8	1	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.0000	2	0	11769	51.4792	C101	S	D	NaN	Bayside, Queens, NY
9	1	0	Artagaveytia, Mr. Ramon	male	71.0000	0	0	PC 17609	49.5042	NaN	C	NaN	22.0	Montevideo, Uruguay

Index

Features/Attributes/Dimensions

How to make sense of your data

You can visualize your data

- mosaic plot
- ...

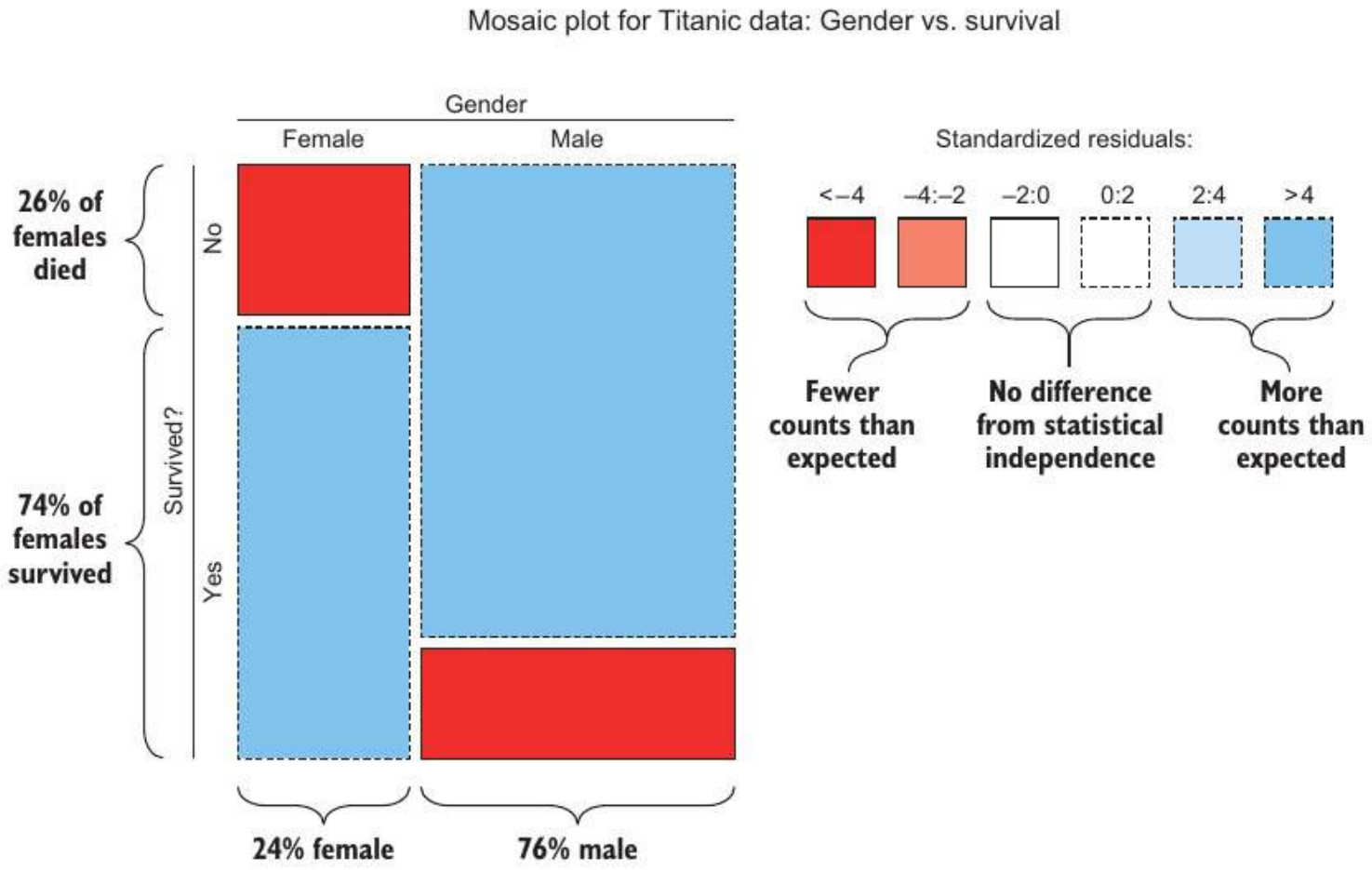


Figure 2.12 Mosaic plot showing the relationship between gender and survival on the Titanic. The visualization shows that a much higher proportion of females (and much smaller proportion of males) survived than would have been expected if survival were independent of gender. "Women and children first."

How to make sense of your data

You can visualize your data

- Mosaic plot
- Box plot
- …

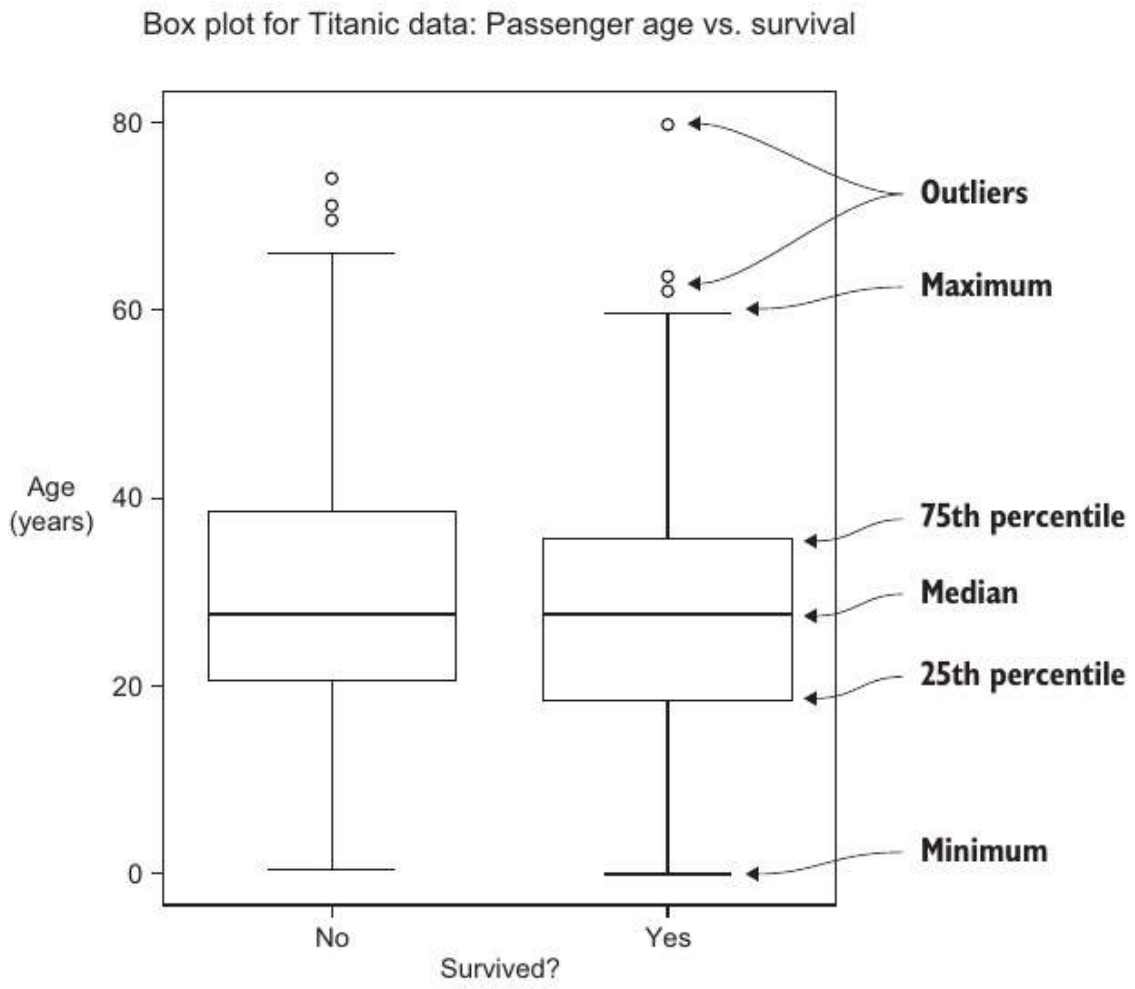


Figure 2.14 Box plot showing the relationship between passenger age and survival on the Titanic. No noticeable differences exist between the age distributions for survivors versus fatalities. (This alone *shouldn't* be a reason to exclude age from the ML model, as it may still be a predictive factor.)

How to make sense of your data

You can visualize your data

- Mosaic plot
- Box plot
- Density plot
- Scatter plot (matrix)
- …

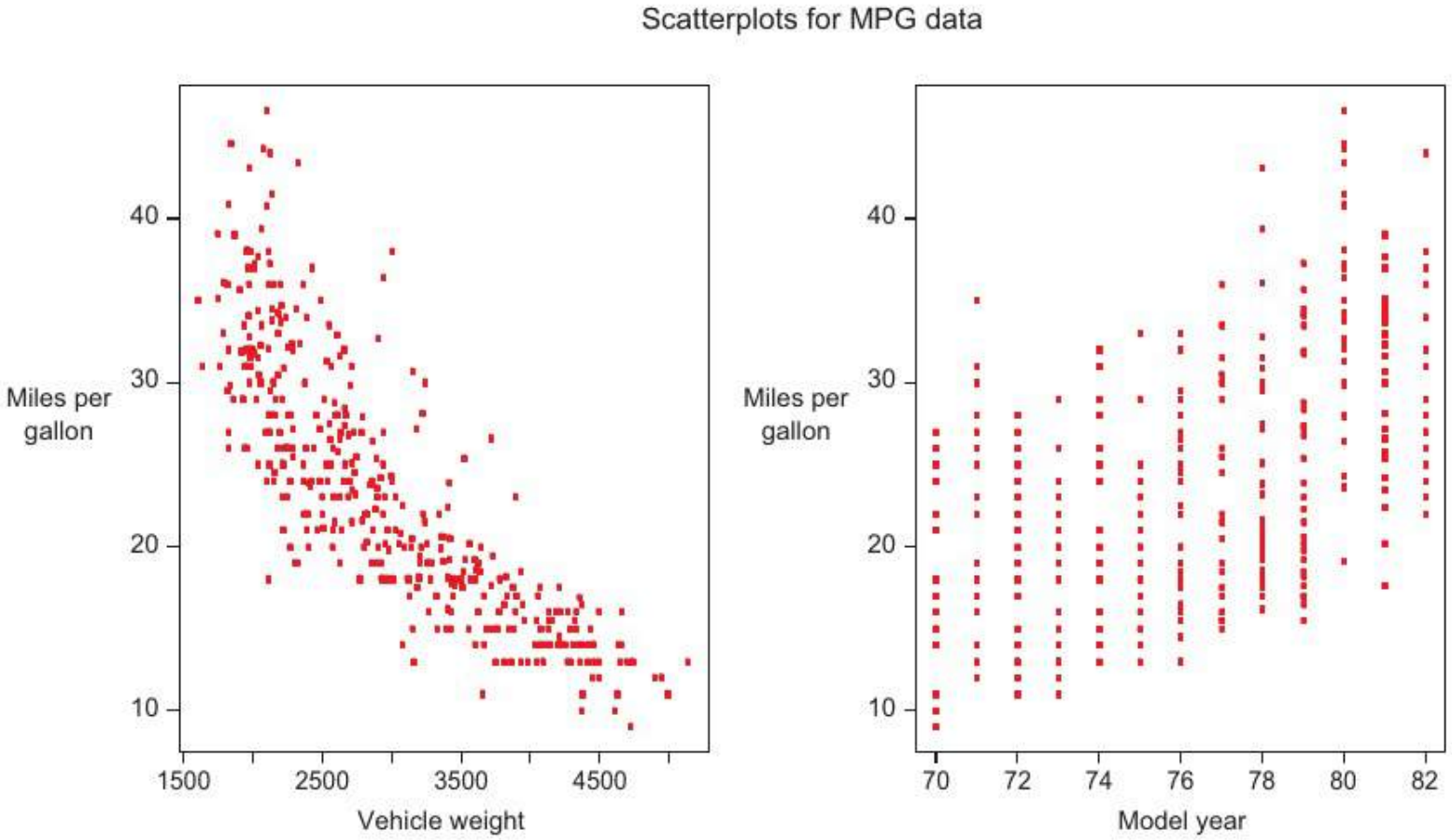
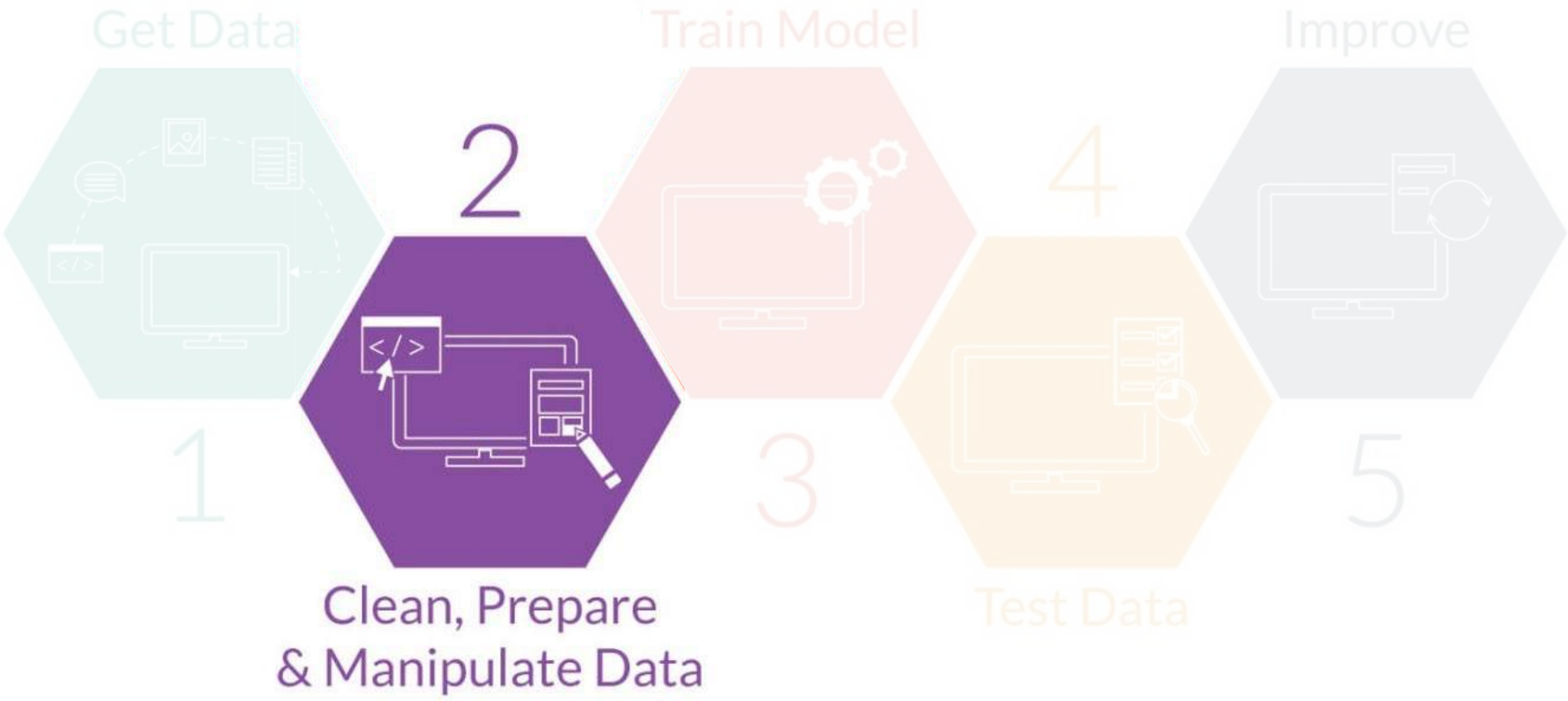


Figure 2.17 Scatter plots for the relationship of vehicle miles per gallon versus vehicle weight (left) and vehicle model year (right)

Clean and Prepare Data



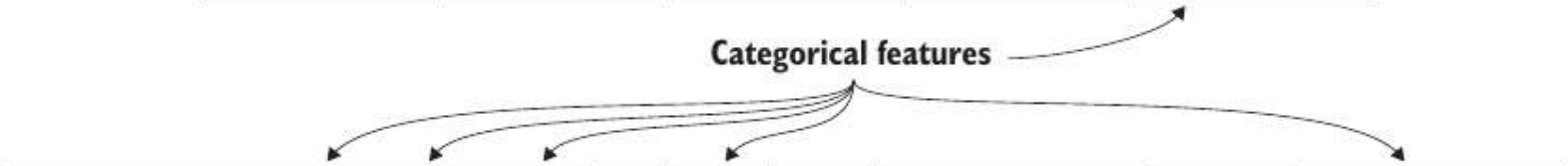
Clean and Prepare Data

PassengerId	Survived	Pclass	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Male	22	1	0	A/5 21171	7.25		S
2	1	1	Female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Female	26	0	0	STON/02. 3101282	7.925		S
4	1	1	Female	35	1	0	113803	53.1	C123	S
5	0	3	Male	35	0	0	373450	8.05		S
6	0	3	Male		0	0	330877	8.4583		Q

Figure 2.7 The Titanic Passengers dataset has missing values in the Age and Cabin columns. The passenger information has been extracted from various historical sources, so in this case the missing values stem from information that couldn't be found in the sources.

Clean and Prepare Data

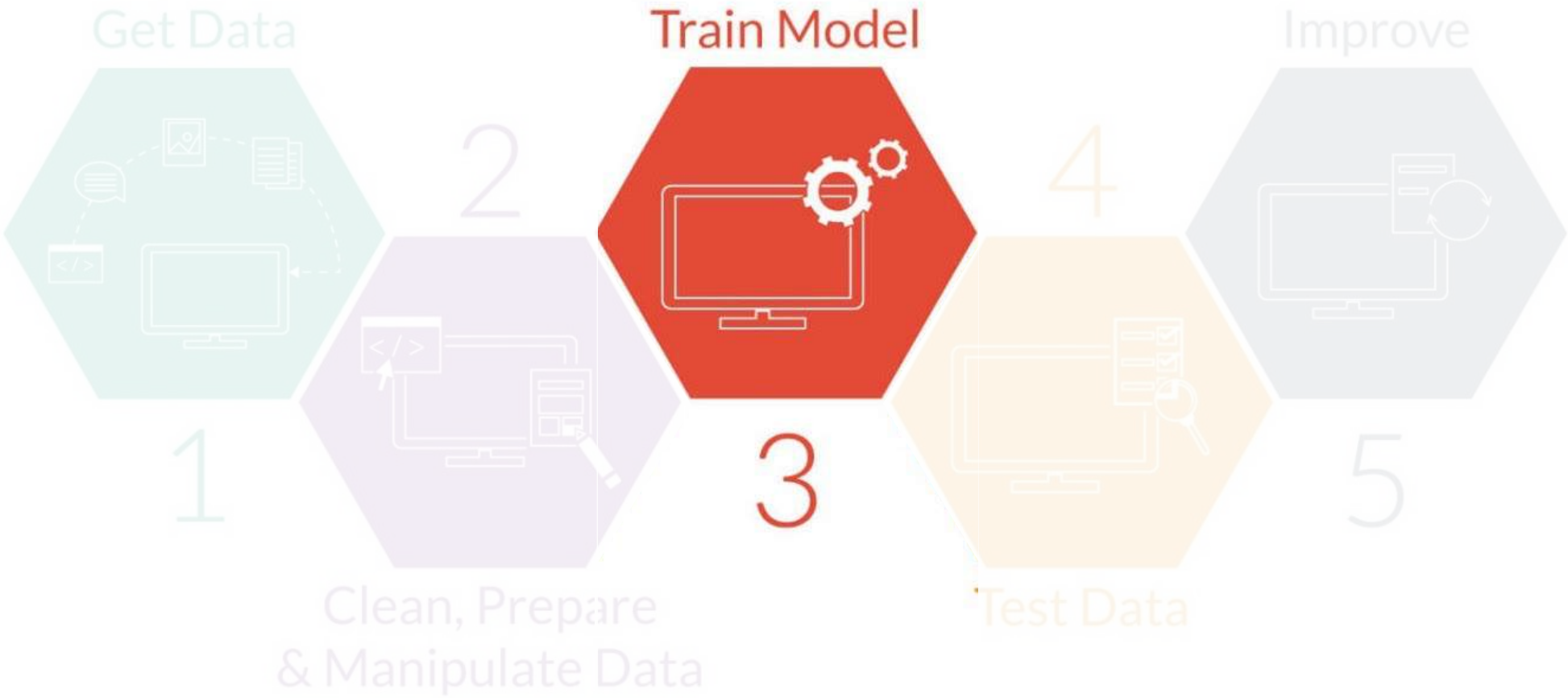
Person	Name	Age	Income	Marital status
1	Jane Doe	24	81,200	Single
2	John Smith	41	121,000	Married



PassengerId	Survived	Pclass	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Male	22	1	0	A/5 21171	7.25		S
2	1	1	Female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Female	26	0	0	STON/02. 3101282	7.925		S
4	1	1	Female	35	1	0	113803	53.1	C123	S
5	0	3	Male	35	0	0	373450	8.05		S
6	0	3	Male		0	0	330877	8.4583		Q

Figure 2.4 Identifying categorical features. At the top is the simple Person dataset, which has a Marital Status categorical feature. At the bottom is a dataset with information about Titanic passengers. The features identified as categorical here are Survived (whether the passenger survived or not), Pclass (what class the passenger was traveling on), Gender (male or female), and Embarked (from which city the passenger embarked).

Clean and Prepare Data



Train Model

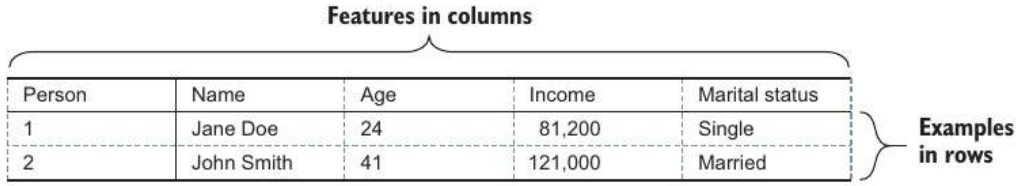


Figure 1.8 **In a tabular dataset, rows are called *instances* and columns represent *features*.**

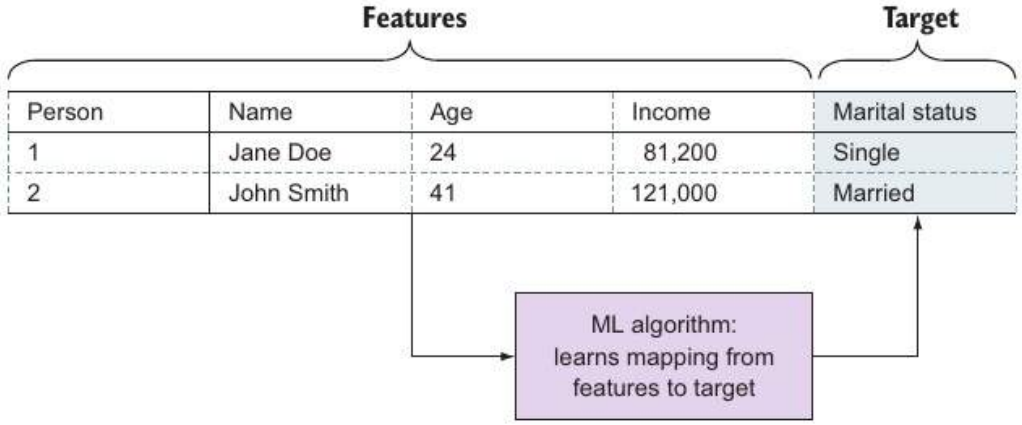
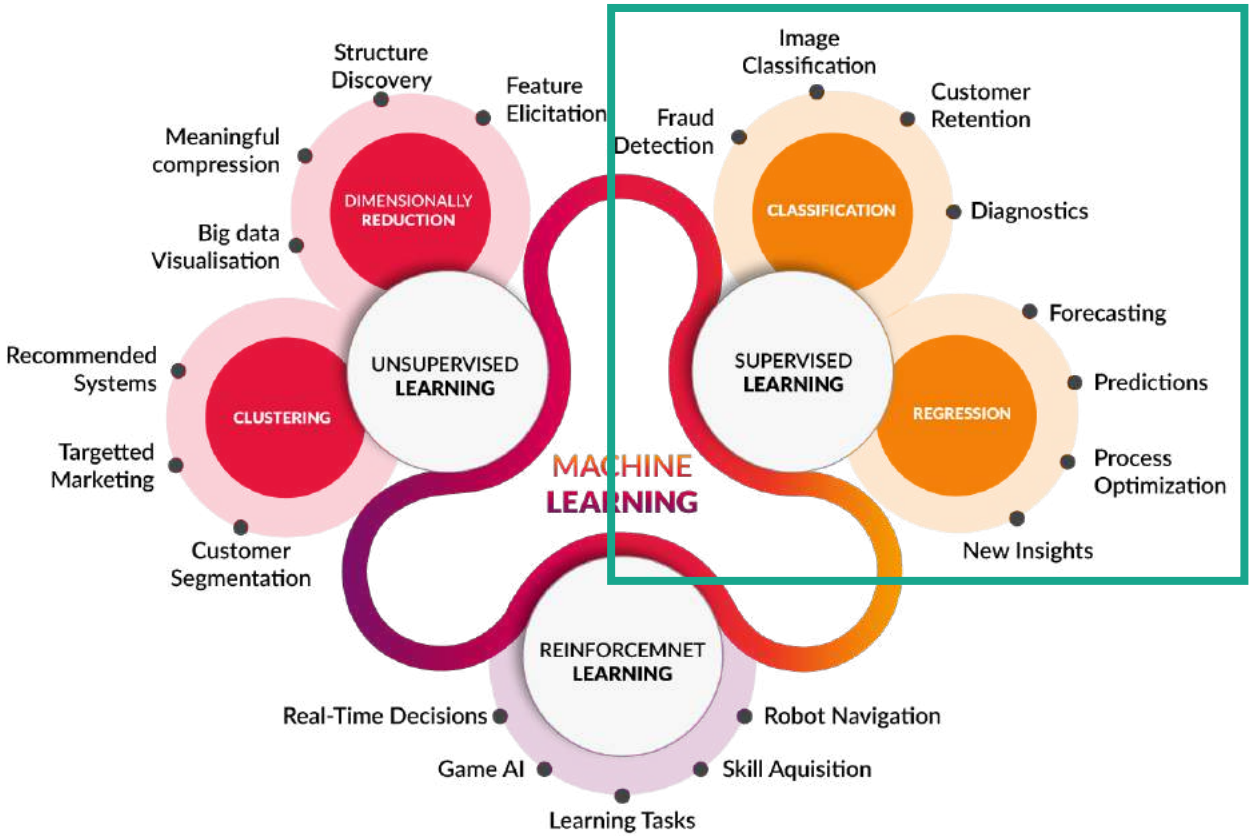
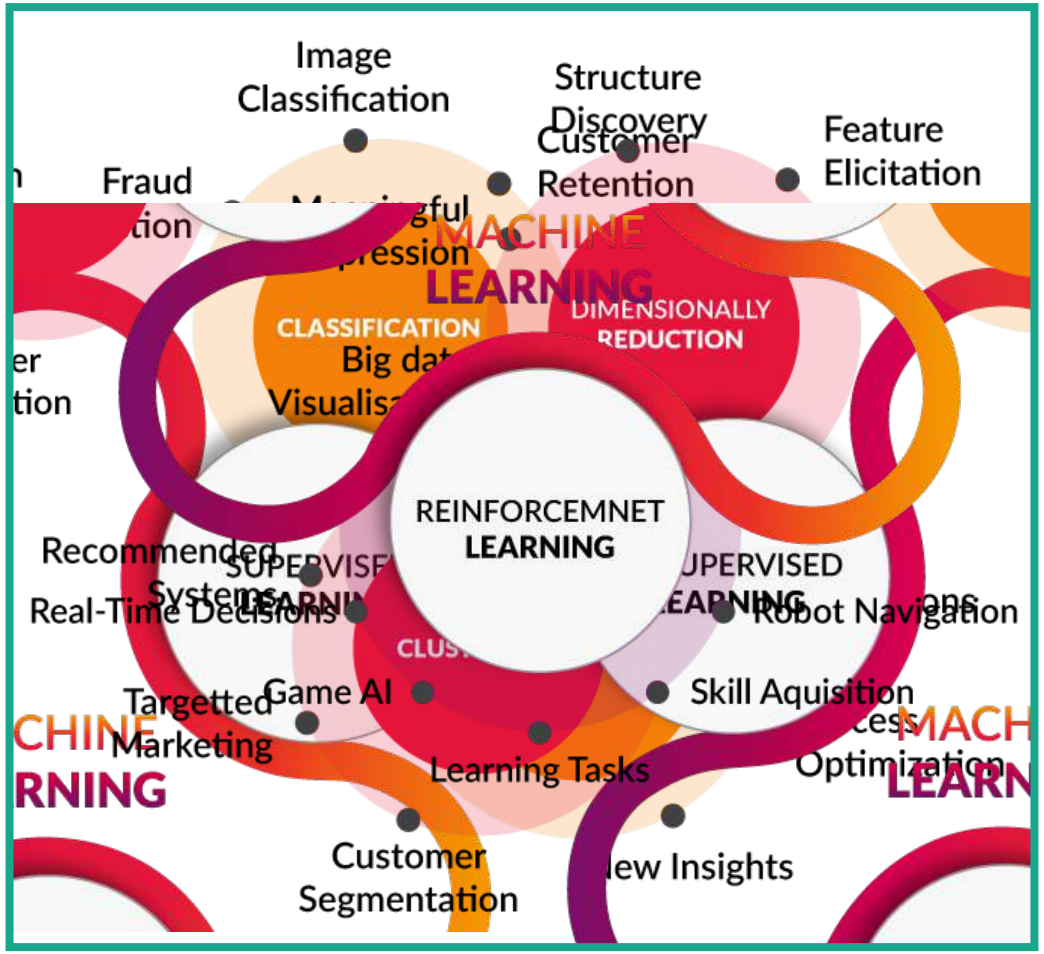


Figure 1.9 **The machine-learning modeling process**

But what algorithm?

The landscape of Machine Learning

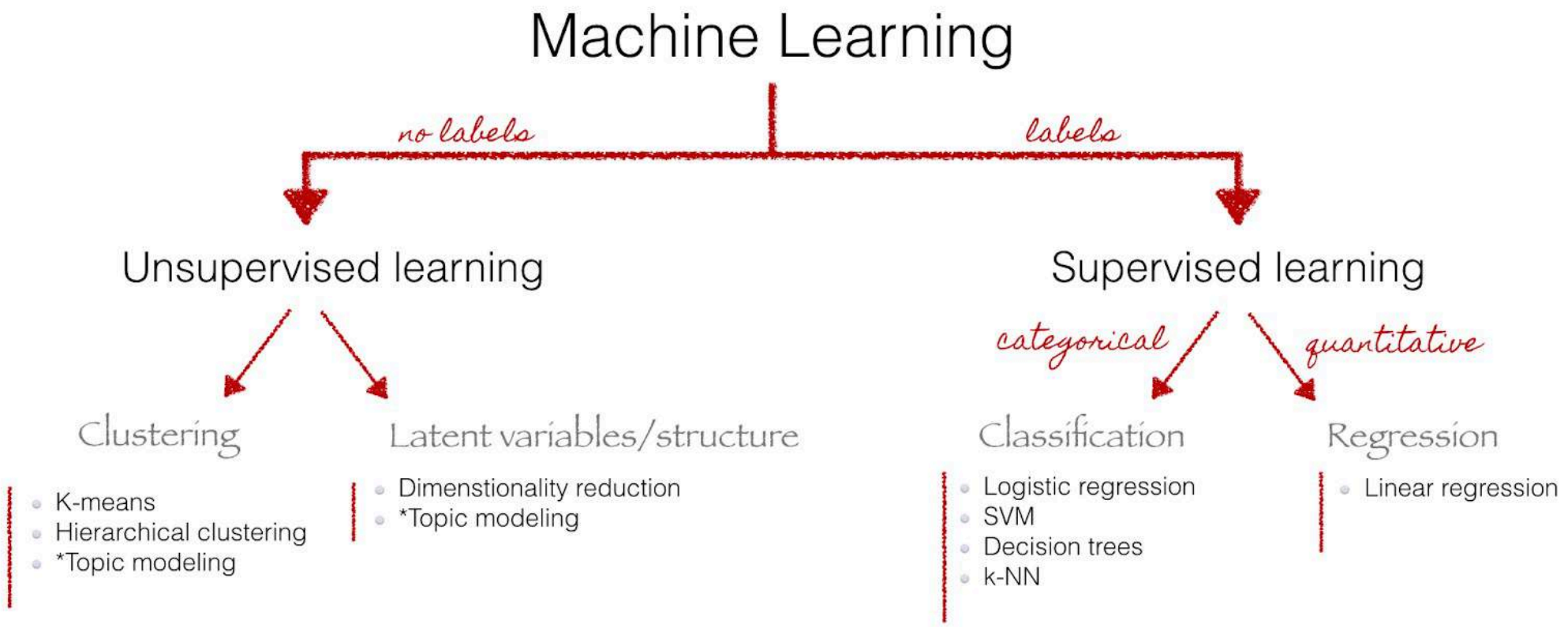


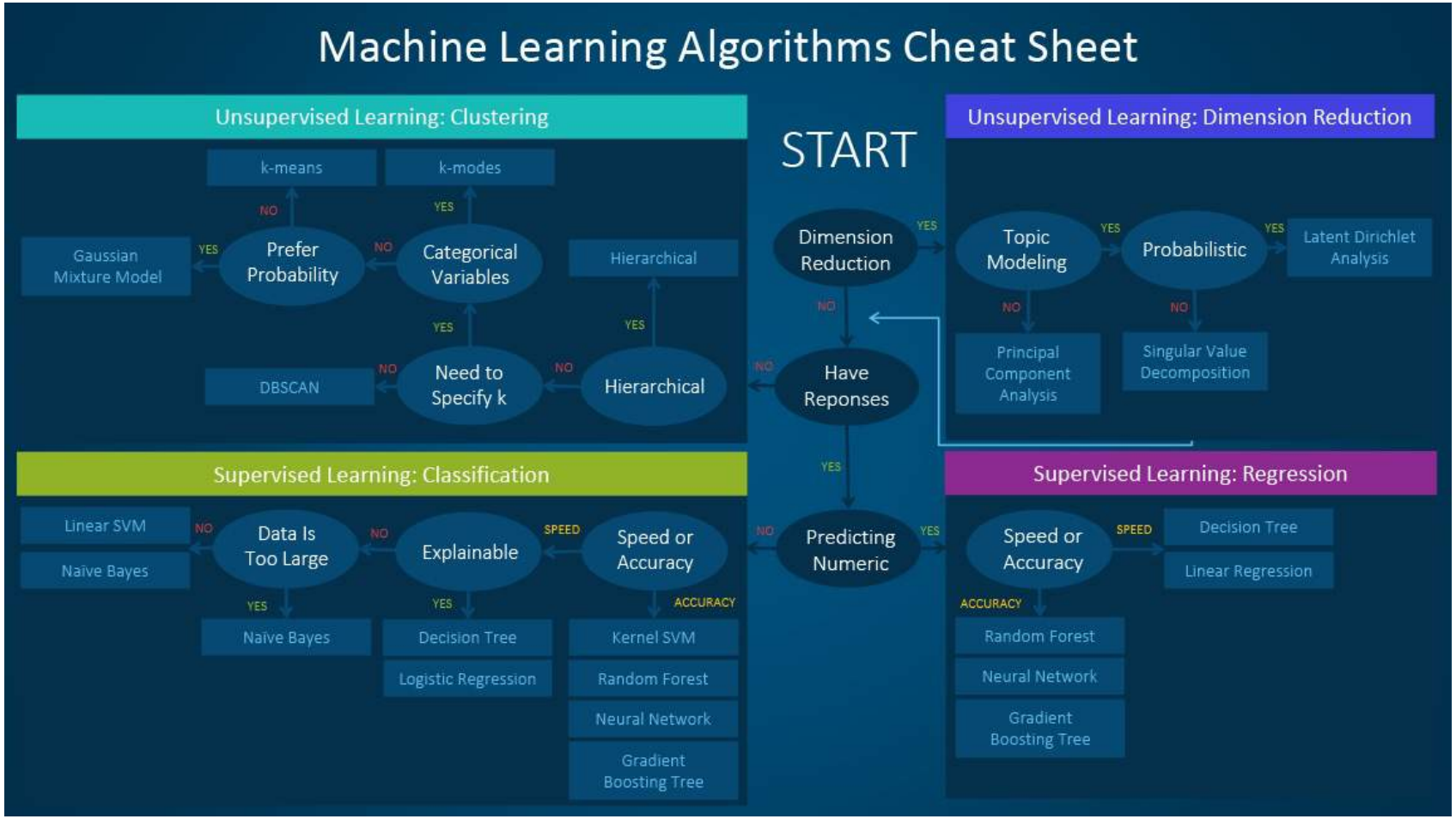


Source: Towards Data Science
Read further: https://litslink.com/blog/an-introduction-to-machine-learning-algorithms

The landscape of Machine Learning

Common algorithms

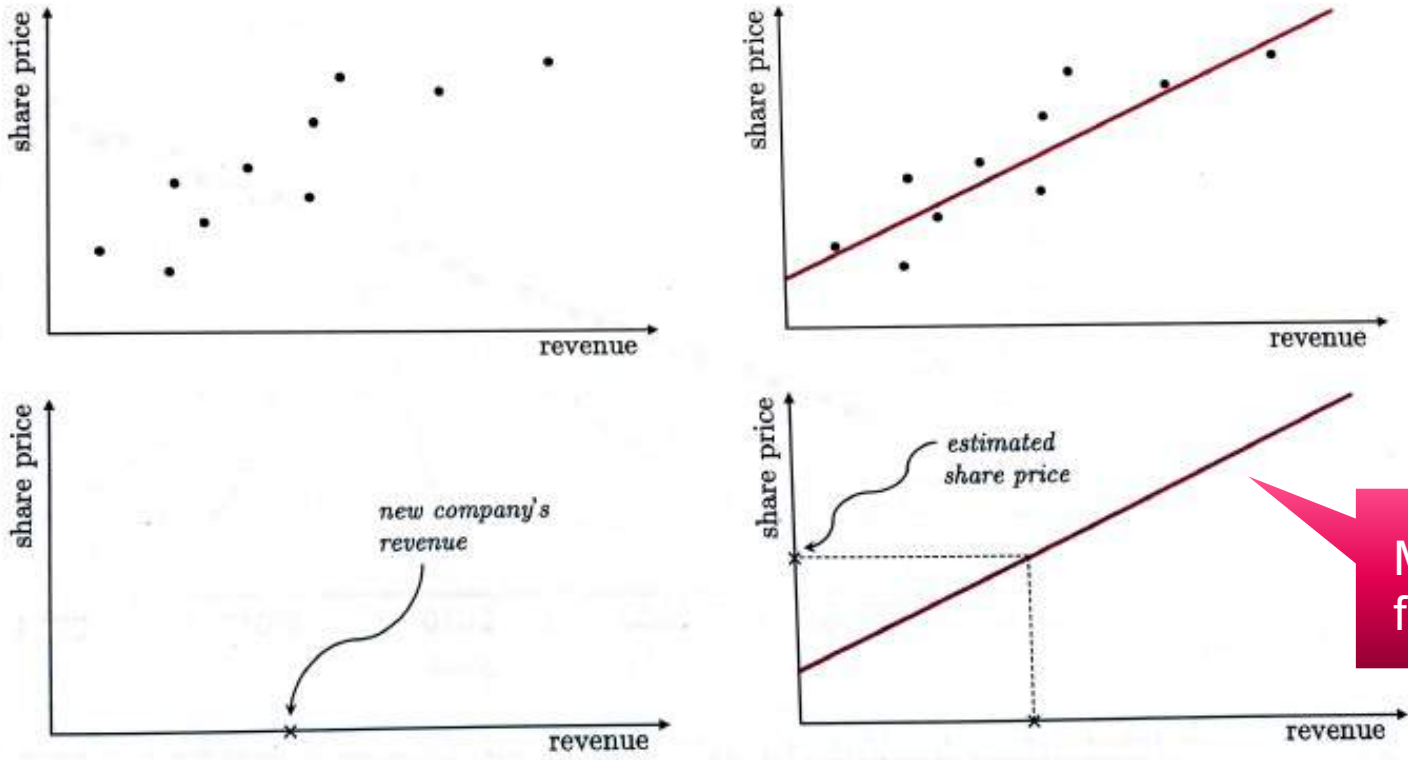




https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/

Classification vs Regression

Regression

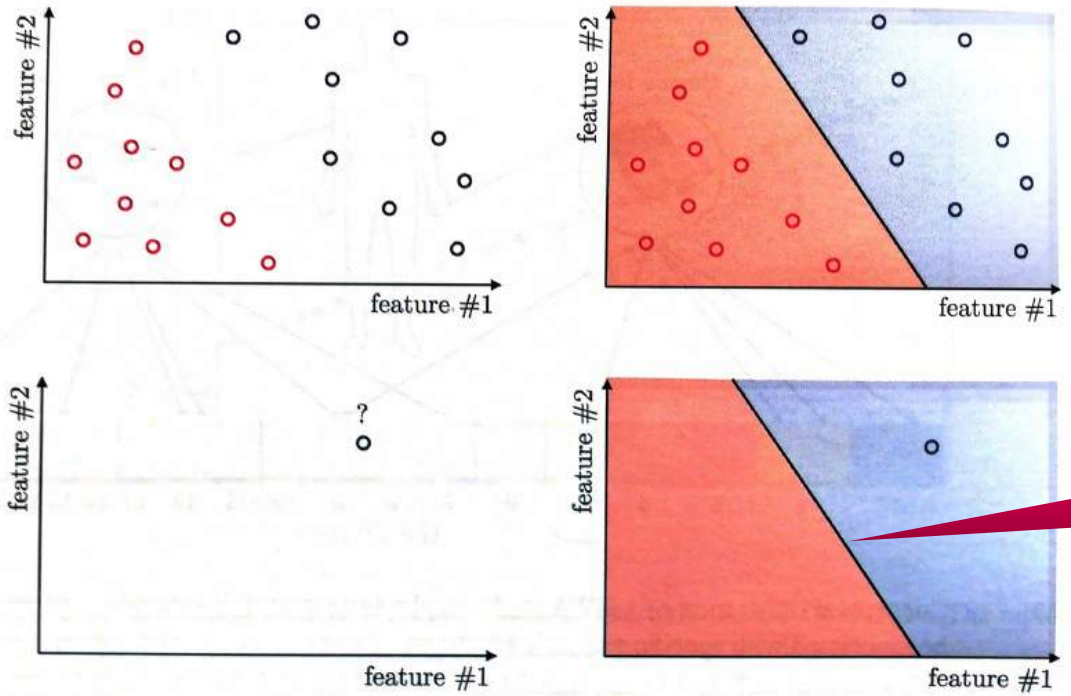


Model data – fitting curve

Fig. 1.7 (top left panel) A toy training dataset of ten corporations with their associated share price and revenue values. (top right panel) A linear model is fit to the data. This trend line models the overall trajectory of the points and can be used for prediction in the future as shown in the bottom left and bottom right panels.

Classification vs Regression

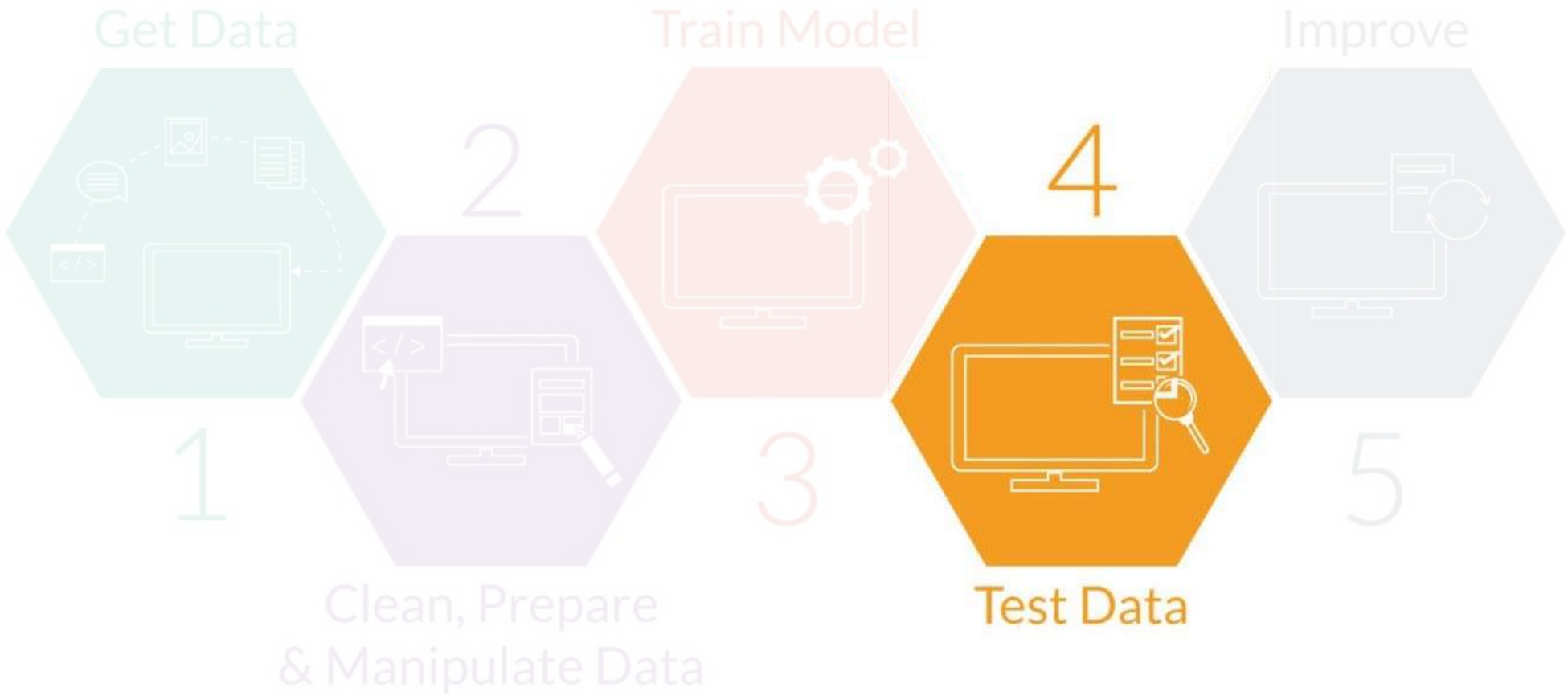
Classification



Divide data – Decision plane

Fig. 1.10 (top left panel) A toy 2-dimensional training set consisting of two distinct classes, red and blue. (top right panel) A linear model is trained to separate the two classes. (bottom left panel) A test point whose class is unknown. (bottom right panel) The test point is classified as blue since it lies on the blue side of the trained linear classifier.

Test Data


Get Data
Train Model
Improve
2
4
1
3
5
Clean, Prepare
& Manipulate Data
Test Data

Test Data

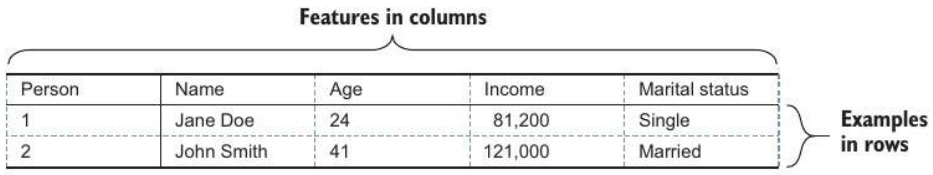
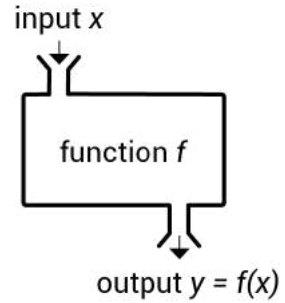


Figure 1.8 In a tabular dataset, rows are called *instances* and columns represent *features*.



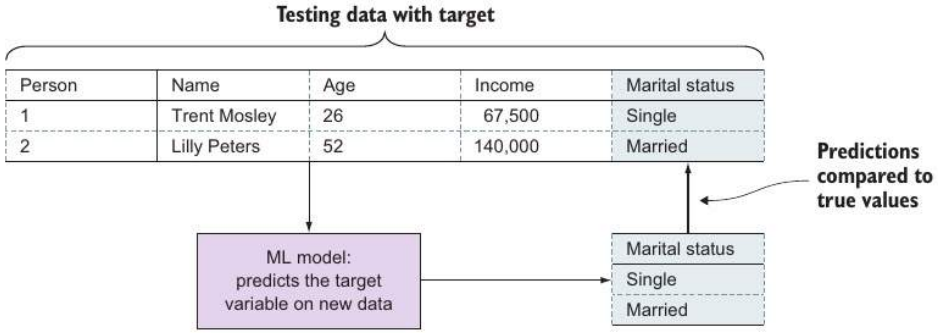


Figure 1.11 When using a testing set to evaluate model performance, you "pretend" that the target variable is unknown and compare the predictions with the true values.

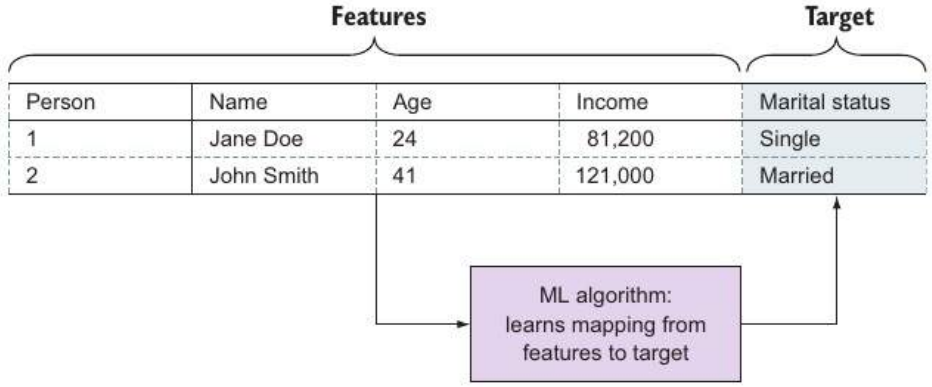


Figure 1.9 The machine-learning modeling process

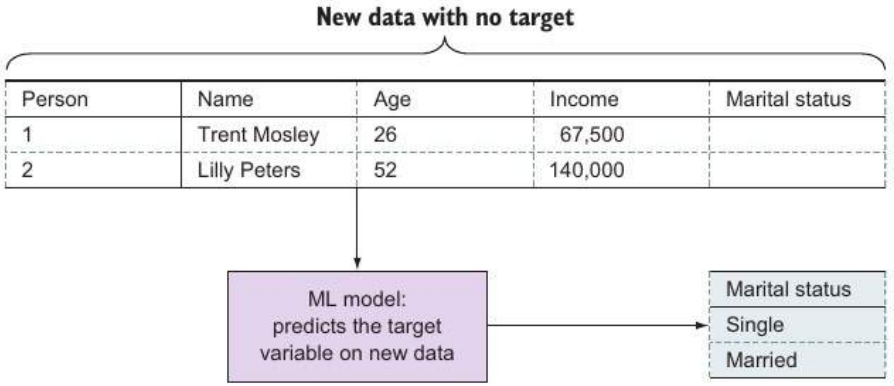


Figure 1.10 Using the model for prediction on new data

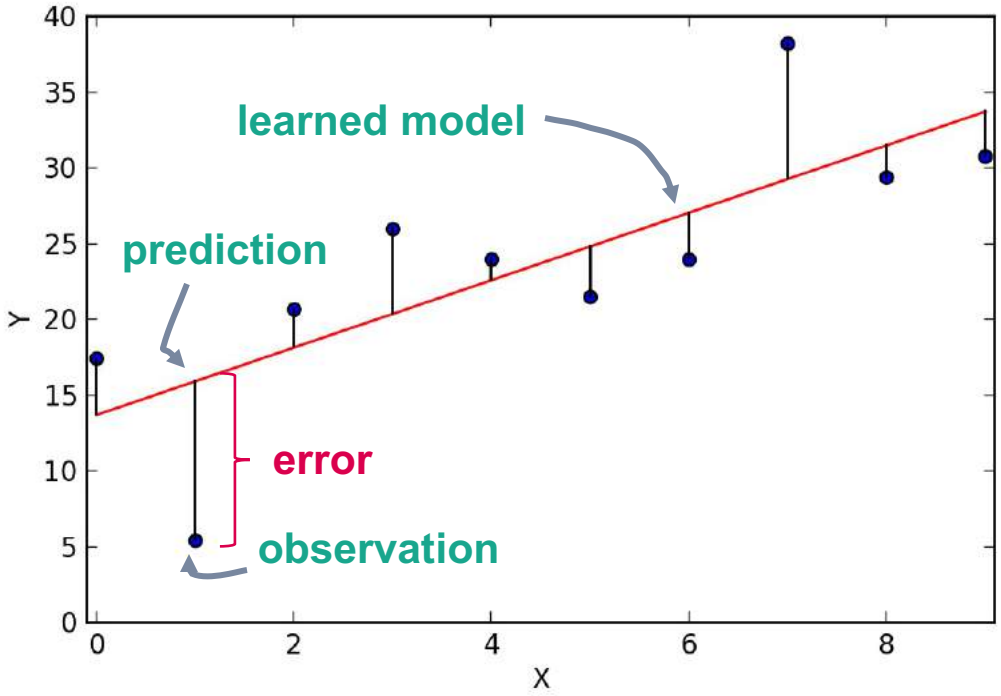
How do we measure our model's performance?

Regression

- (root) mean square error

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Mean deviation



How do we measure our model's performance?

Classification

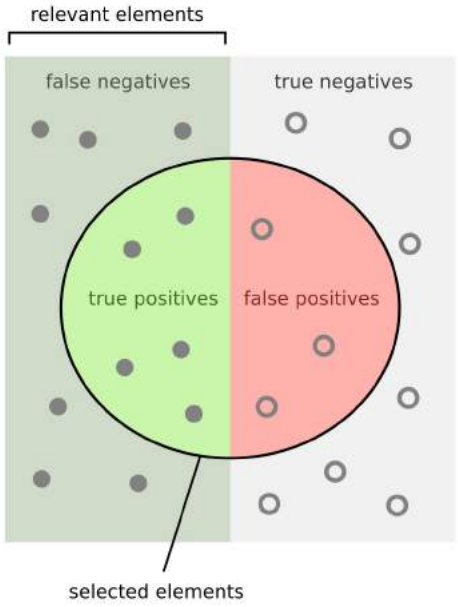
- (root) mean square error
- confusion matrix

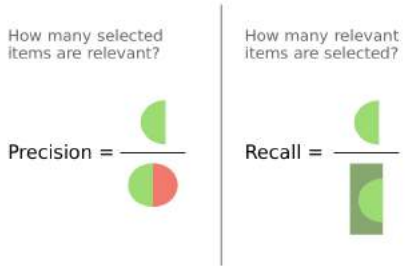
$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{All Samples}}$$

Percentage of correct predictions

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

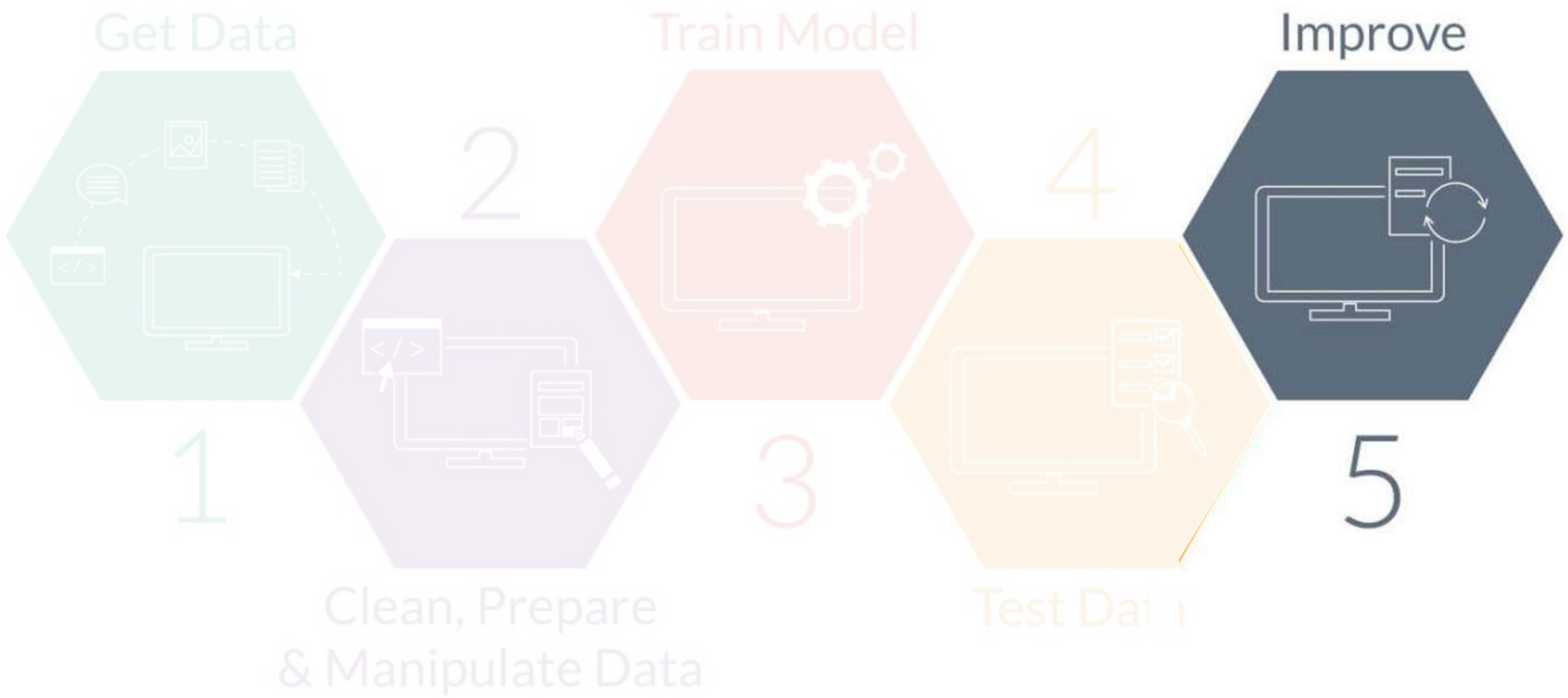
Beyond accuracy





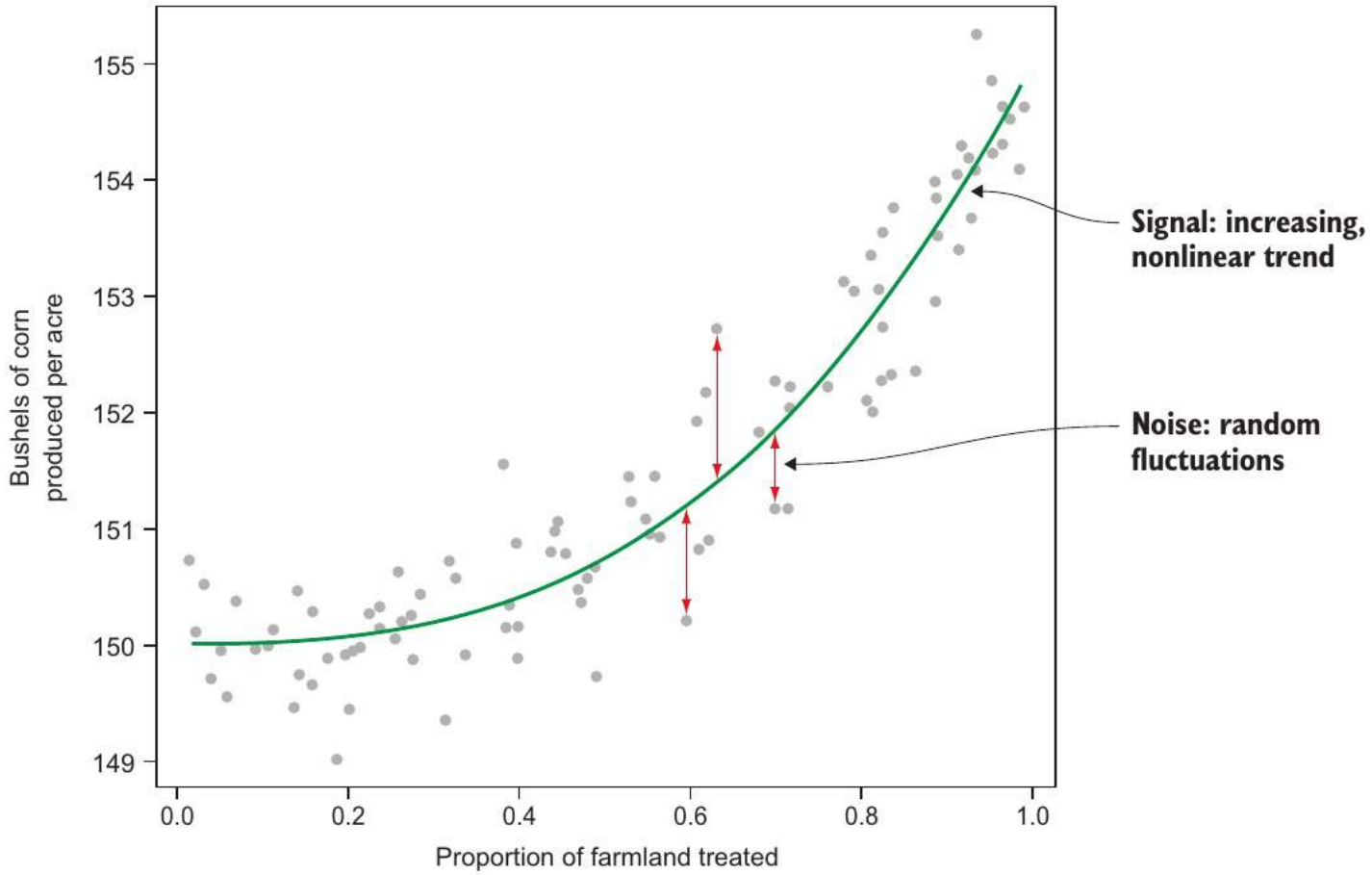
		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

Improve Model



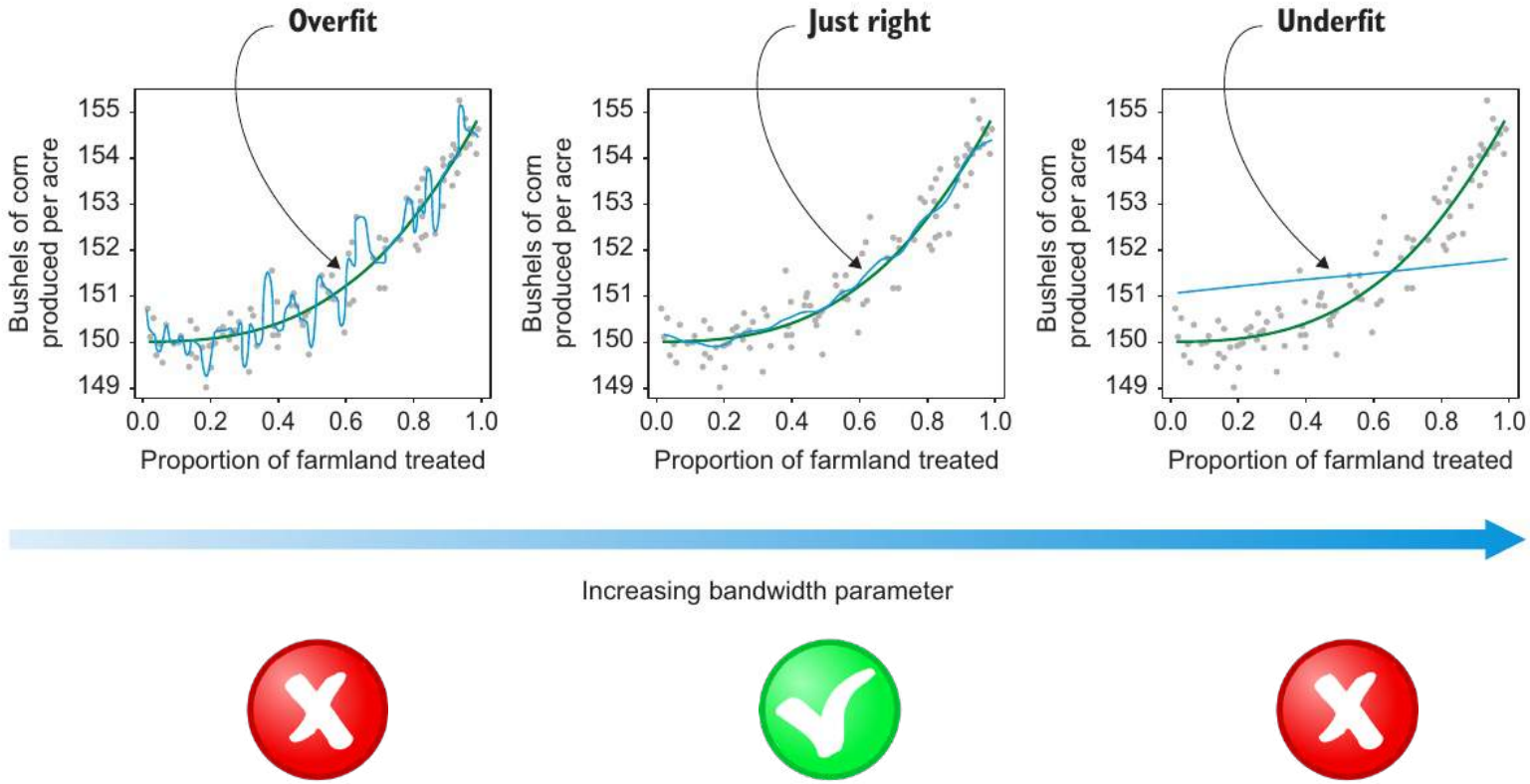
Real world data

signal-to-noise ratio



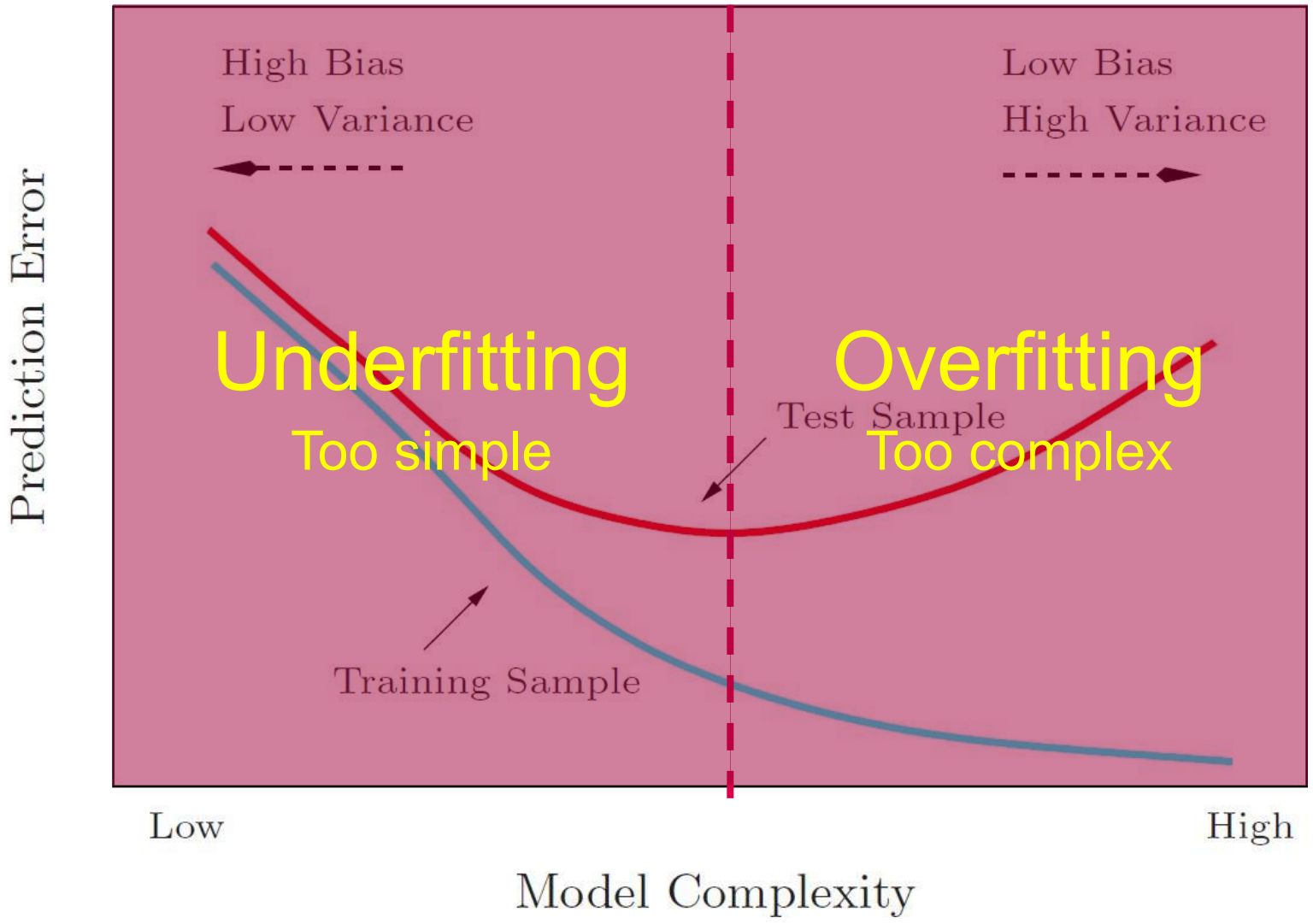
What makes a good model?

It may happen that the models we build are underfitted or overfitted



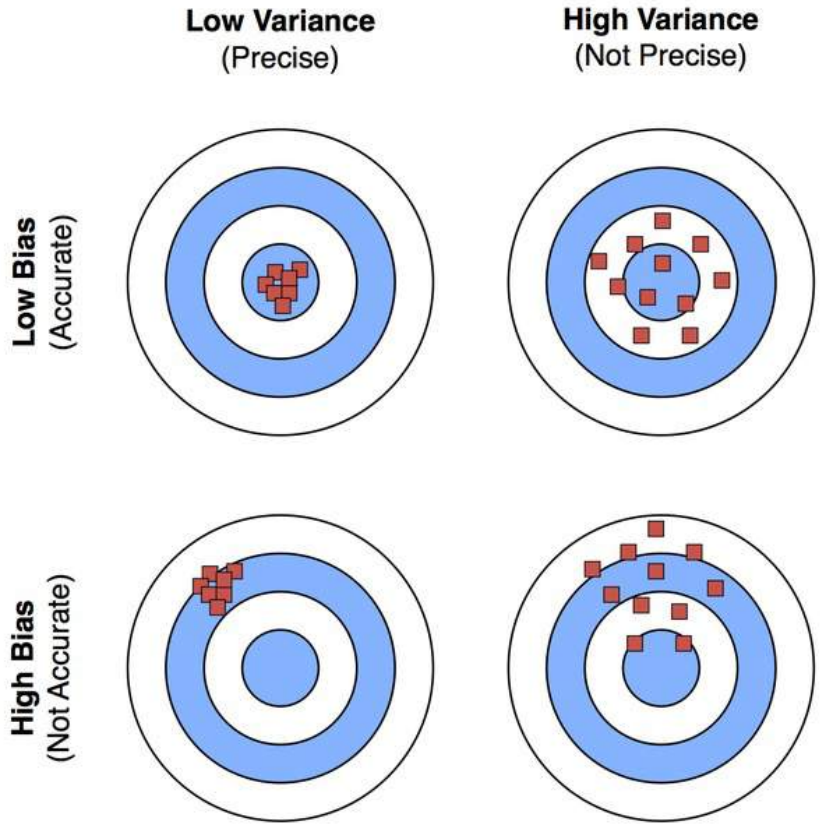
How do we measure our model's performance?

When we have a performance measure, we can compare different models and their complexity



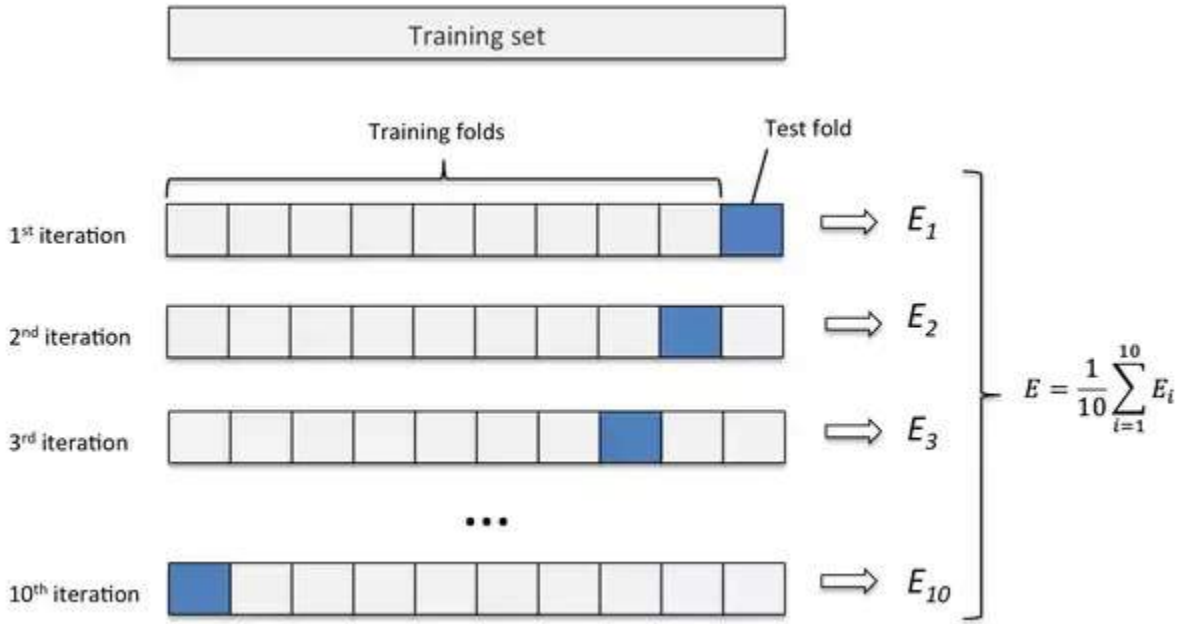
How do we measure our model's performance?

Variance-Bias Trade-off



What about small data sets?

Cross-validation



What about large data sets?

Curse of dimensionality

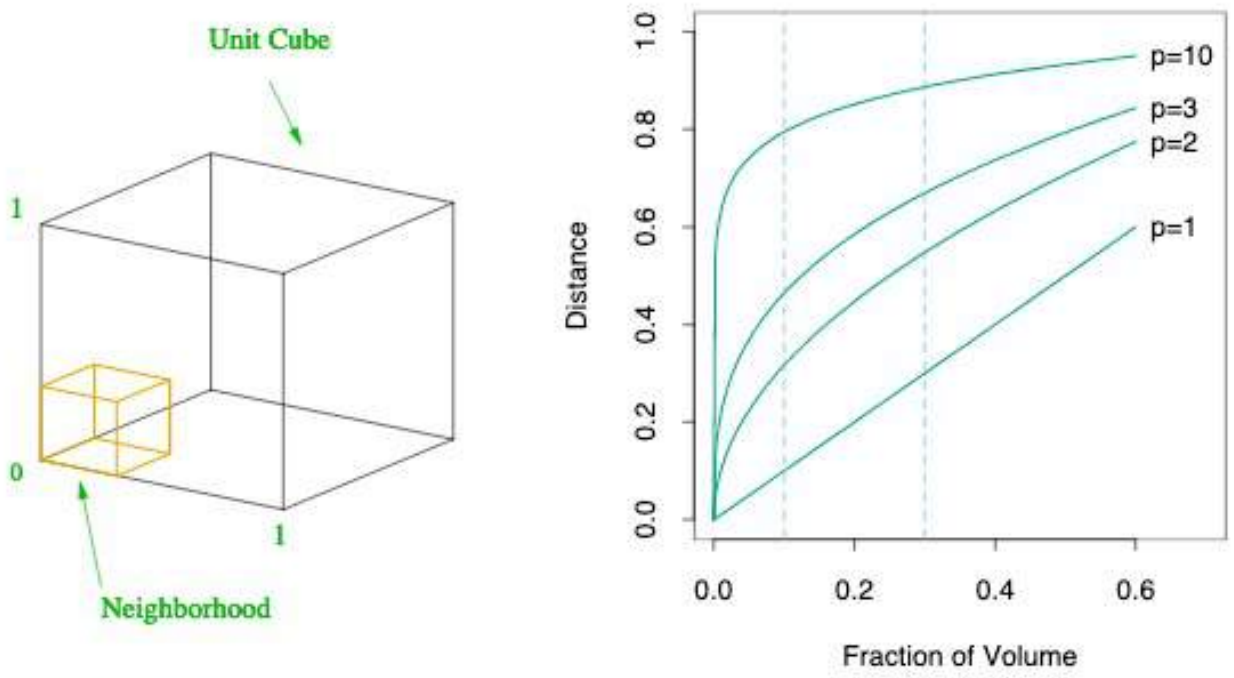
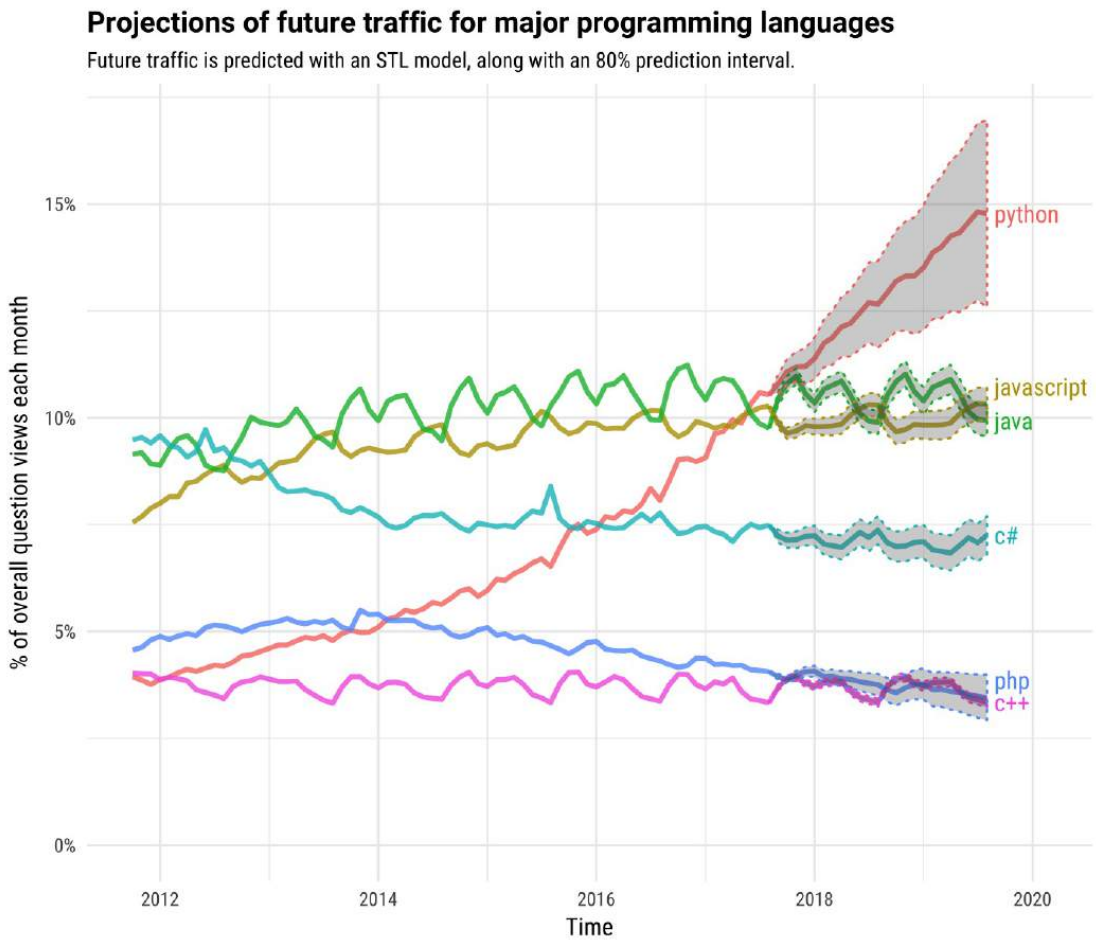


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.*

Practice

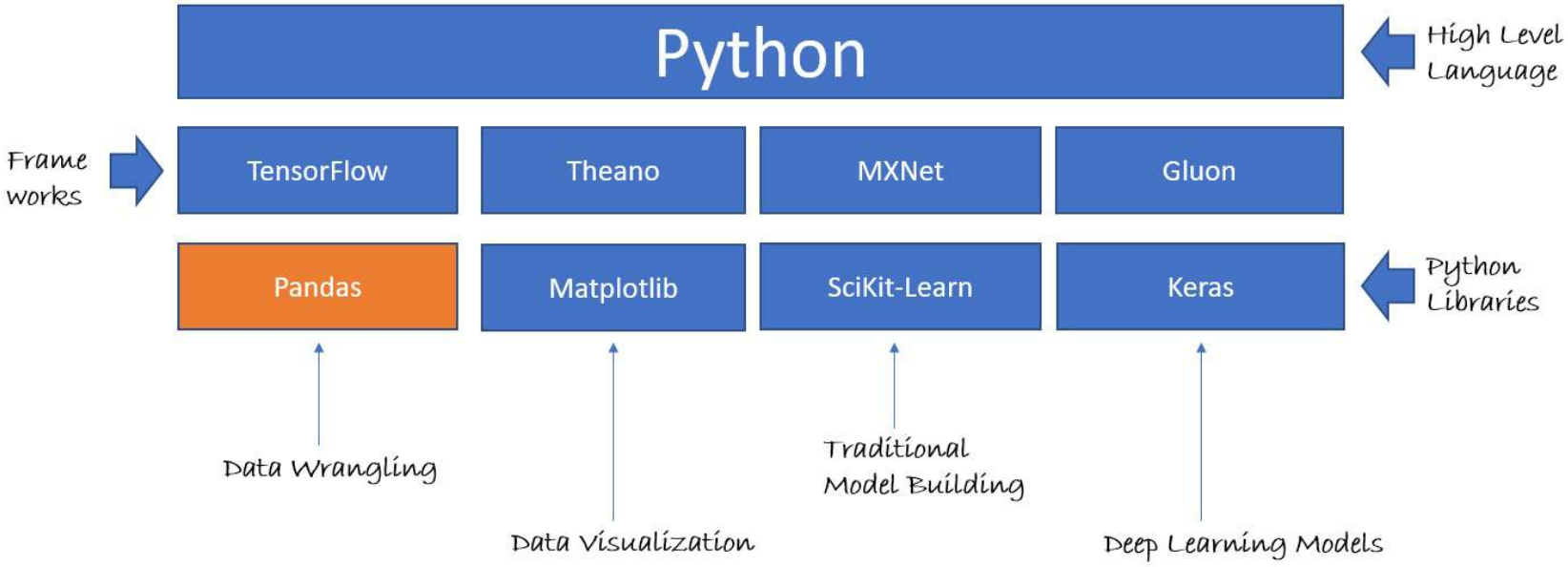
Why Python



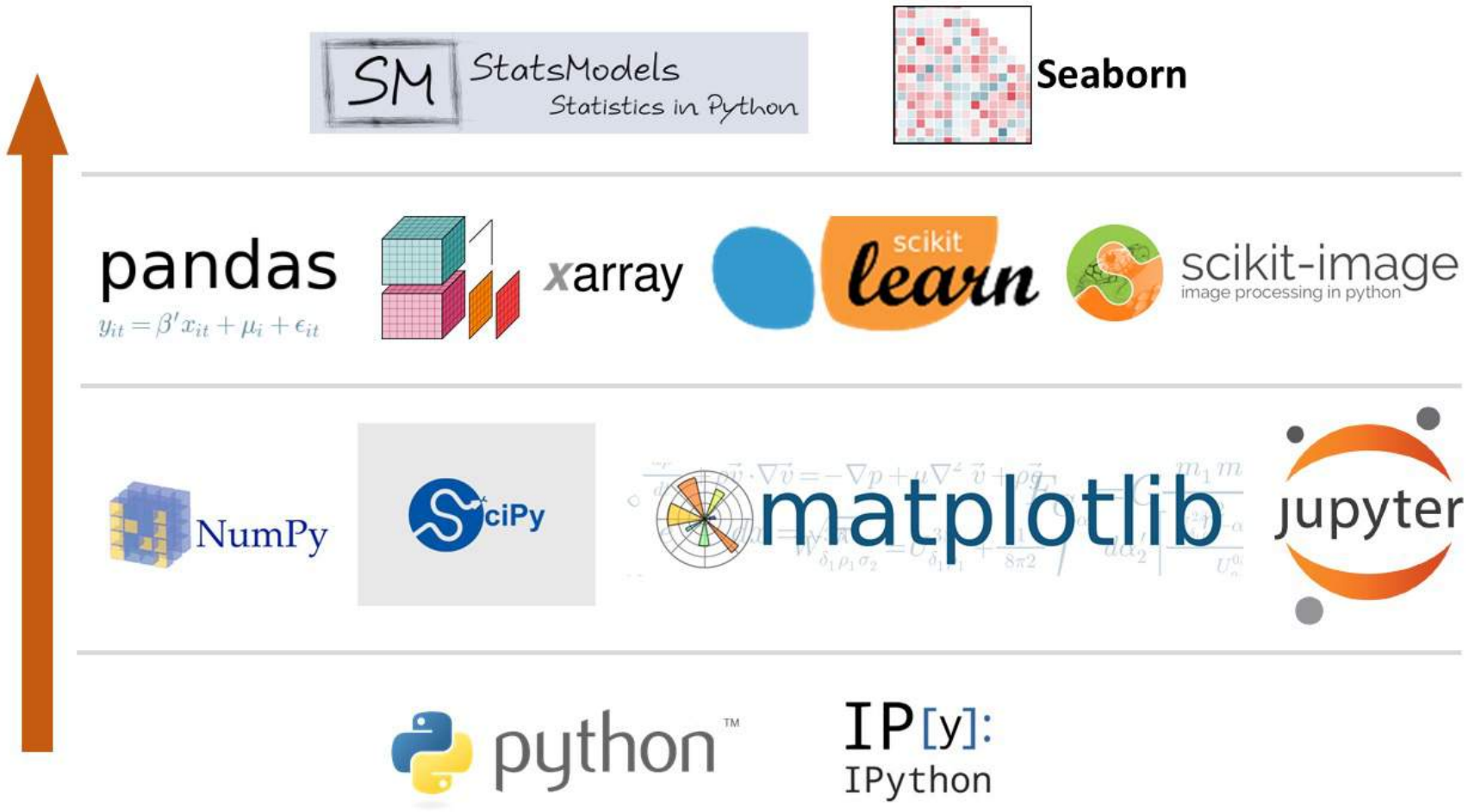
Source: Stack Overflow
https://medium.com/@UdacityINDIA/why-use-python-for-machine-learning-e4b0b4457a77

Python ecosystem

Machine learning is a **method of data analysis** that automates **analytical model building**. In the applied space most these models are built in a language called **Python.**



Python learning path

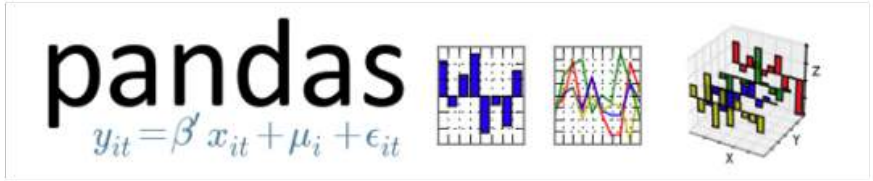


Packages, Packages everywhere !

- Want to work with images — numpy, opencv, scikit
- Want to work in text — nltk, numpy, scikit
- Want to work in audio — librosa
- Want to solve **machine learning problem** — pandas, scikit
- Want to see the data clearly — matplotlib, seaborn, scikit
- Want to use **deep learning** — tensorflow, pytorch
- Want to do scientific computing — scipy
- Want to integrate web applications — Django
- Want to take a shower …. Well

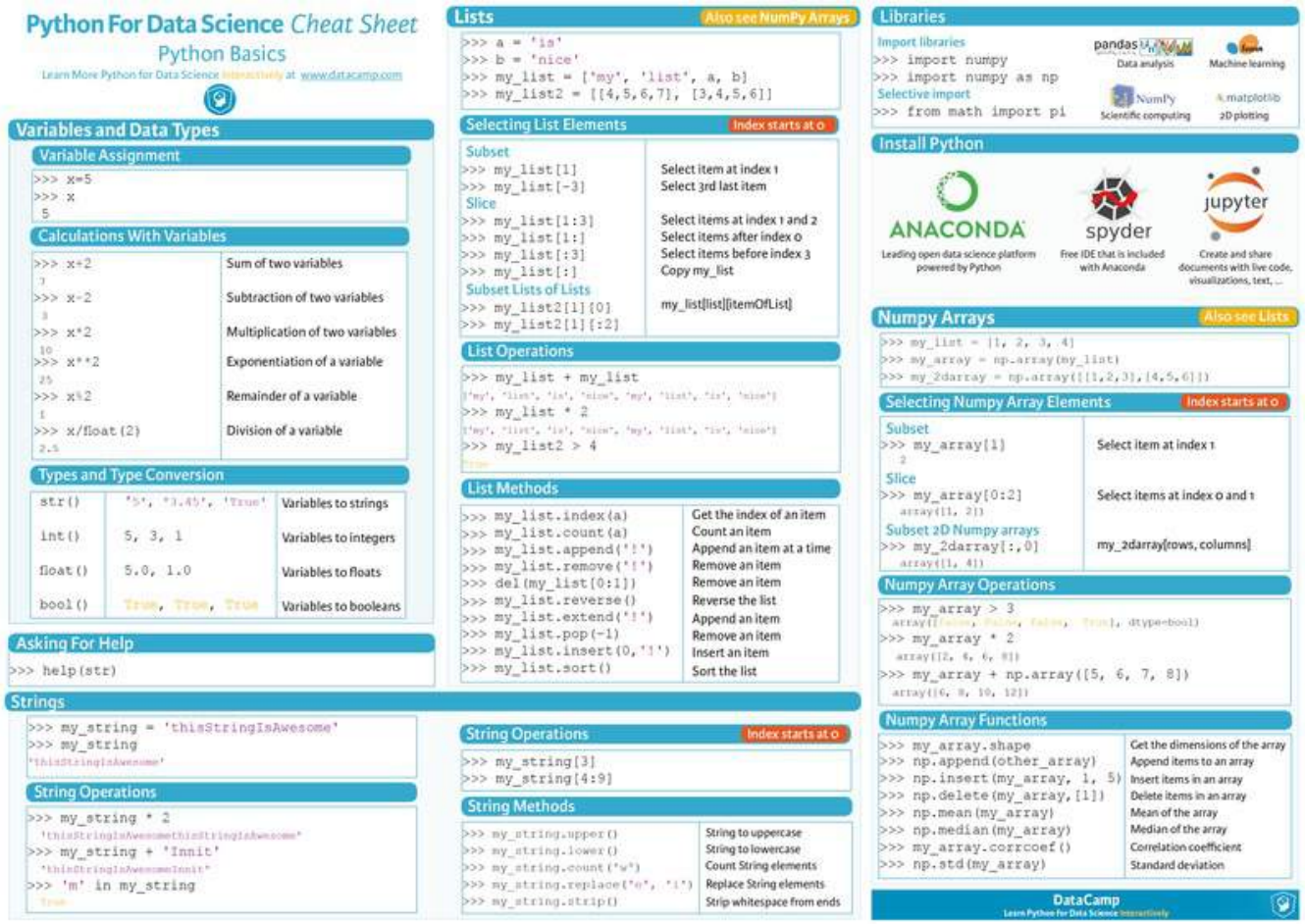
Machine Learning Landscape

Python ecosystem



https://hackernoon.com/top-8-python-libraries-for-machine-learning-and-artificial-intelligence-y08id3031

Cheat Sheets



Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

`>>> x+2` → `7`	Sum of two variables
`>>> x-2` → `3`	Subtraction of two variables
`>>> x*2` → `10`	Multiplication of two variables
`>>> x**2` → `25`	Exponentiation of a variable
`>>> x%2` → `1`	Remainder of a variable
`>>> x/float(2)` → `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements — Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Operations — Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespace from ends

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis; Machine learning; NumPy — Scientific computing; matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements — Index starts at 0

Subset	
`>>> my_array[1]` → `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` → `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` → `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

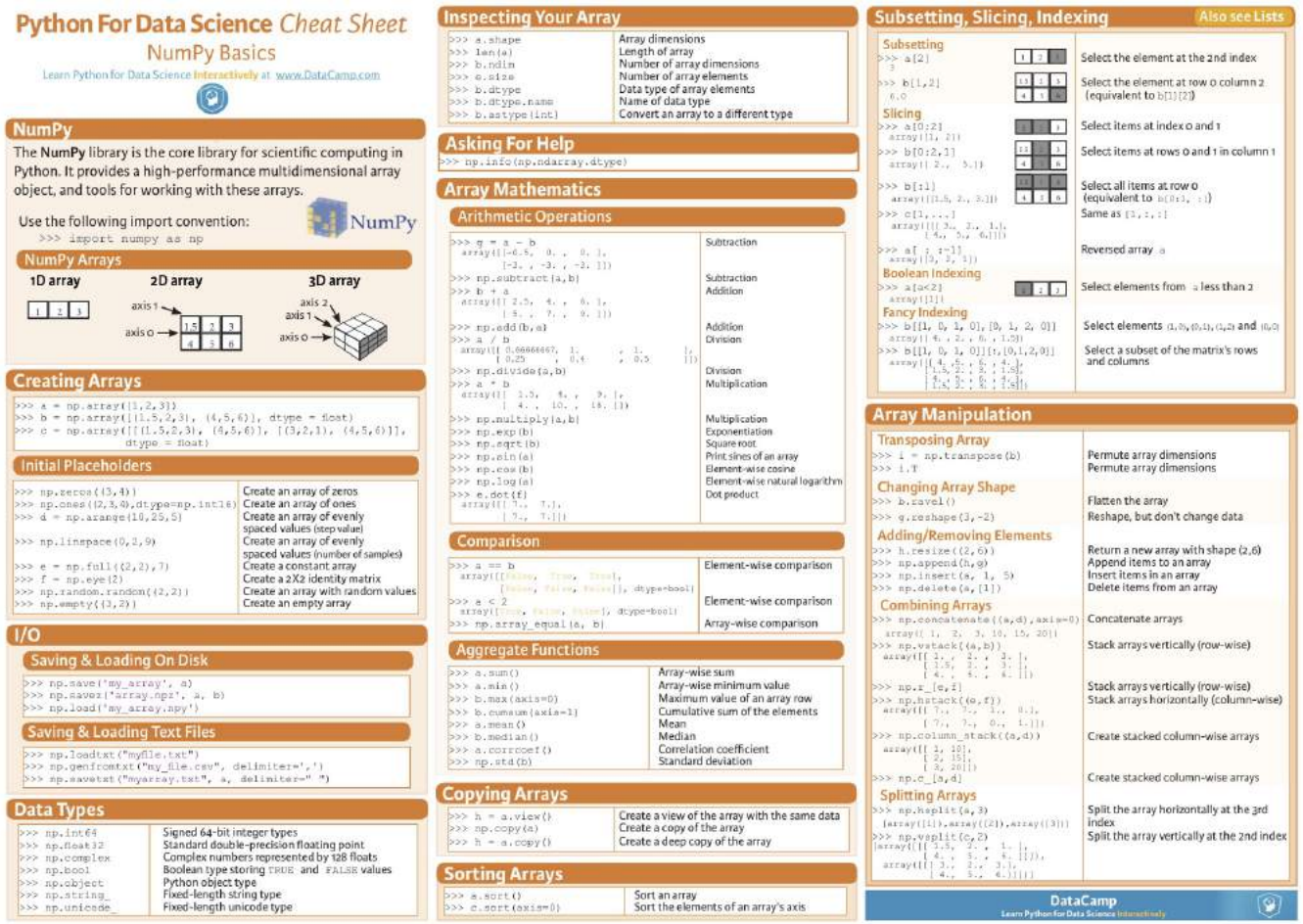
```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

DataCamp — Learn Python for Data Science Interactively

Numpy Cheat Sheet



Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays

1D array

2D array

axis 1

axis 0

3D array

axis 2

axis 1

axis 0

Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([1.5,2,3], [4,5,6]], dtype = float)
>>> c = np.array([[[1.5,2,3], [4,5,6]], [[3,2,1], [4,5,6]]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))
>>> np.ones((2,3,4),dtype=np.int16)
>>> d = np.arange(10,25,5)

>>> np.linspace(0,2,9)

>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of evenly spaced values (number of samples)
Create a constant array
Create a 2X2 identity matrix
Create an array with random values
Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64
>>> np.float32
>>> np.complex
>>> np.bool
>>> np.object
>>> np.string_
>>> np.unicode_
```

Signed 64-bit integer types
Standard double-precision floating point
Complex numbers represented by 128 floats
Boolean type storing TRUE and FALSE values
Python object type
Fixed-length string type
Fixed-length unicode type

Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> e.size
>>> b.dtype
>>> b.dtype.name
>>> b.astype(int)
```

Array dimensions
Length of array
Number of array dimensions
Number of array elements
Data type of array elements
Name of data type
Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
 array([[-0.5,  0. ,  0. ],
        [-2. , -3. , -2. ]])
>>> np.subtract(a,b)
>>> b + a
 array([[ 2.5,  4. ,  6. ],
        [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)
>>> a / b
 array([[ 0.66666667,  1.        ,  1.        ],
        [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)
>>> a * b
 array([[  1.5,   4. ,   9. ],
        [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
 array([[ 7.,  7.],
        [ 7.,  7.]])
```

Subtraction
Subtraction
Addition
Addition
Division
Division
Multiplication
Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Comparison

```
>>> a == b
 array([[False,  True,  True],
        [False, False, False]], dtype=bool)
>>> a < 2
 array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison
Element-wise comparison
Array-wise comparison

Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.corrcoef()
>>> np.std(b)
```

Array-wise sum
Array-wise minimum value
Maximum value of an array row
Cumulative sum of the elements
Mean
Median
Correlation coefficient
Standard deviation

Copying Arrays

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

Create a view of the array with the same data
Create a copy of the array
Create a deep copy of the array

Sorting Arrays

```
>>> a.sort()
>>> c.sort(axis=0)
```

Sort an array
Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]
 3
>>> b[1,2]
 6.0
```

Select the element at the 2nd index
Select the element at row 0 column 2 (equivalent to b[1][2])

Slicing

```
>>> a[0:2]
 array([1, 2])
>>> b[0:2,1]
 array([ 2.,  5.])
>>> b[:1]
 array([[1.5, 2., 3.]])
>>> c[1,...]
 array([[[ 3.,  2.,  1.],
         [ 4.,  5.,  6.]]])
>>> a[ : :-1]
 array([3, 2, 1])
```

Select items at index 0 and 1
Select items at rows 0 and 1 in column 1
Select all items at row 0 (equivalent to b[0:1, :])
Same as [1,:,:]
Reversed array a

Boolean Indexing

```
>>> a[a<2]
 array([1])
```

Select elements from a less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
 array([ 4. ,  2. ,  6. ,  1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
 array([[ 4. ,  5. ,  6. ,  4. ],
        [ 1.5,  2. ,  3. ,  1.5],
        [ 4. ,  5. ,  6. ,  4. ],
        [ 1.5,  2. ,  3. ,  1.5]])
```

Select elements (1,0),(0,1),(1,2) and (0,0)
Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```

Permute array dimensions
Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
>>> g.reshape(3,-2)
```

Flatten the array
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a,[1])
```

Return a new array with shape (2,6)
Append items to an array
Insert items in an array
Delete items from an array

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
 array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))
 array([[ 1. ,  2. ,  3. ],
        [ 1.5,  2. ,  3. ],
        [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]
>>> np.hstack((e,f))
 array([[ 7.,  7.,  1.,  0.],
        [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
>>> np.c_[a,d]
```

Concatenate arrays
Stack arrays vertically (row-wise)
Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)
Create stacked column-wise arrays
Create stacked column-wise arrays

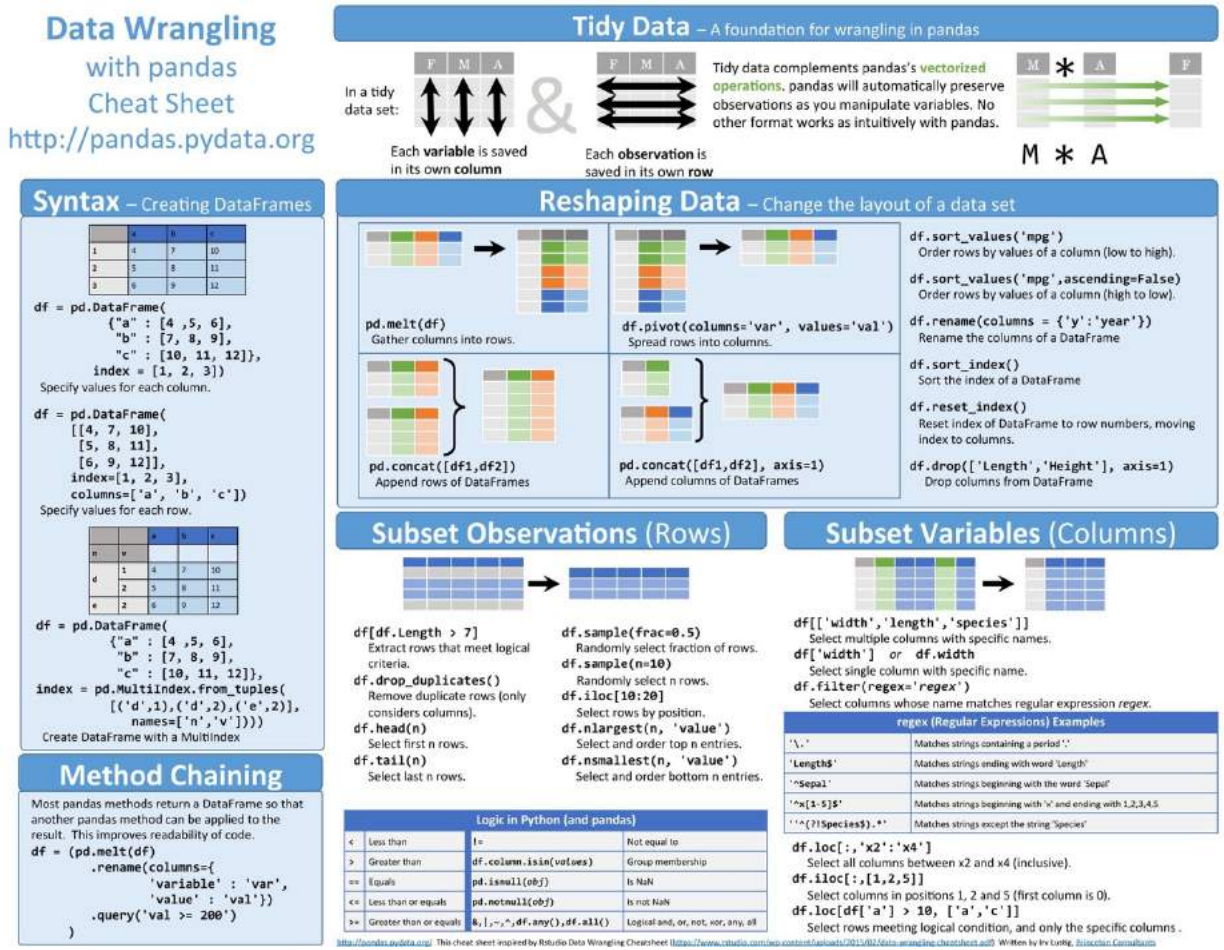
Splitting Arrays

```
>>> np.hsplit(a,3)
 [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)
 [array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
  array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

Split the array horizontally at the 3rd index
Split the array vertically at the 2nd index

DataCamp
Learn Python for Data Science Interactively

Pandas Cheat Sheet



Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
        index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
         names=['n','v'])))
```

Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set

`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(['Length','Height'], axis=1)`
Drop columns from DataFrame

Subset Observations (Rows)

`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Subset Variables (Columns)

`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`'^(?!Species$).*'`	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

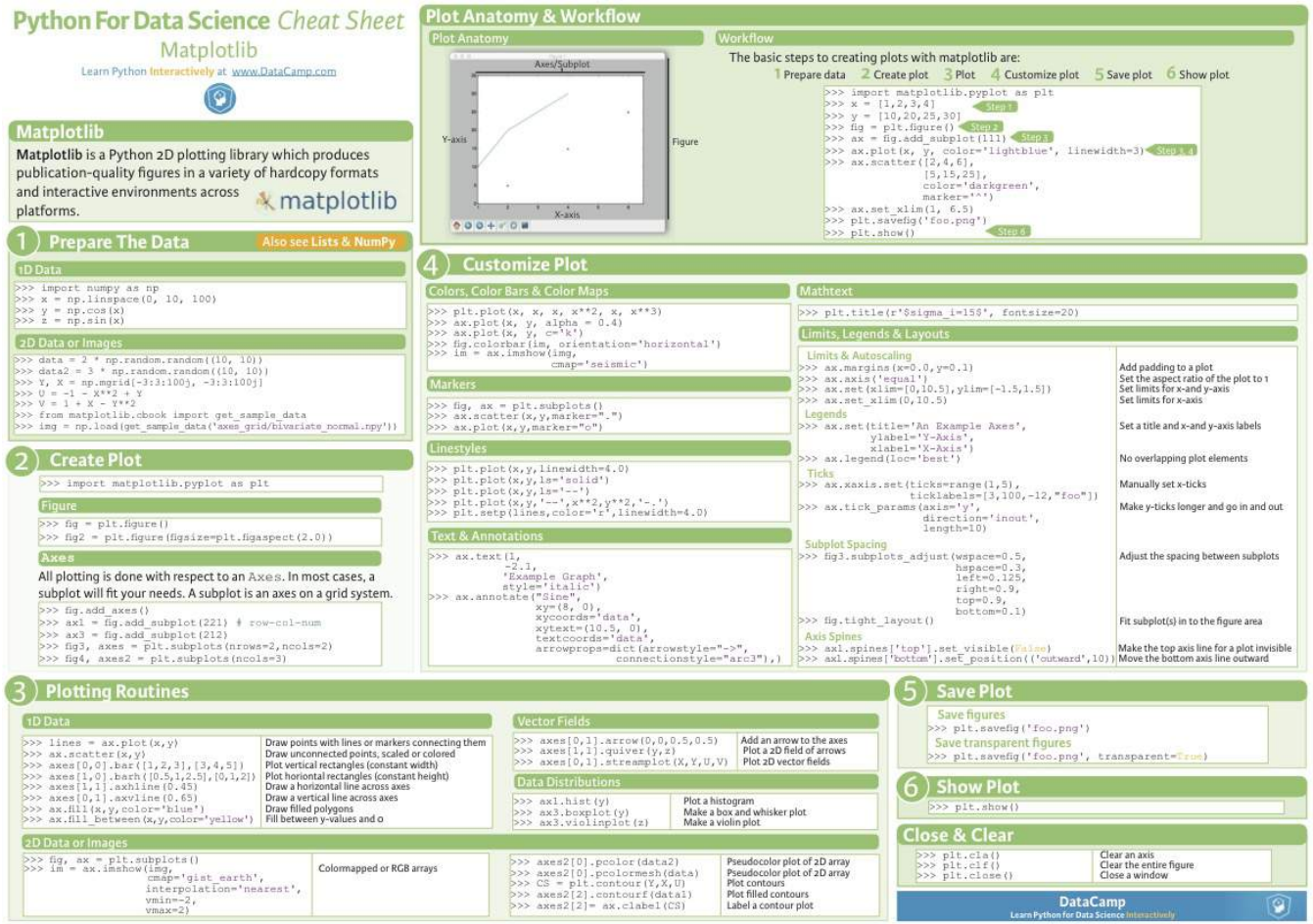
`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

Logic in Python (and pandas)

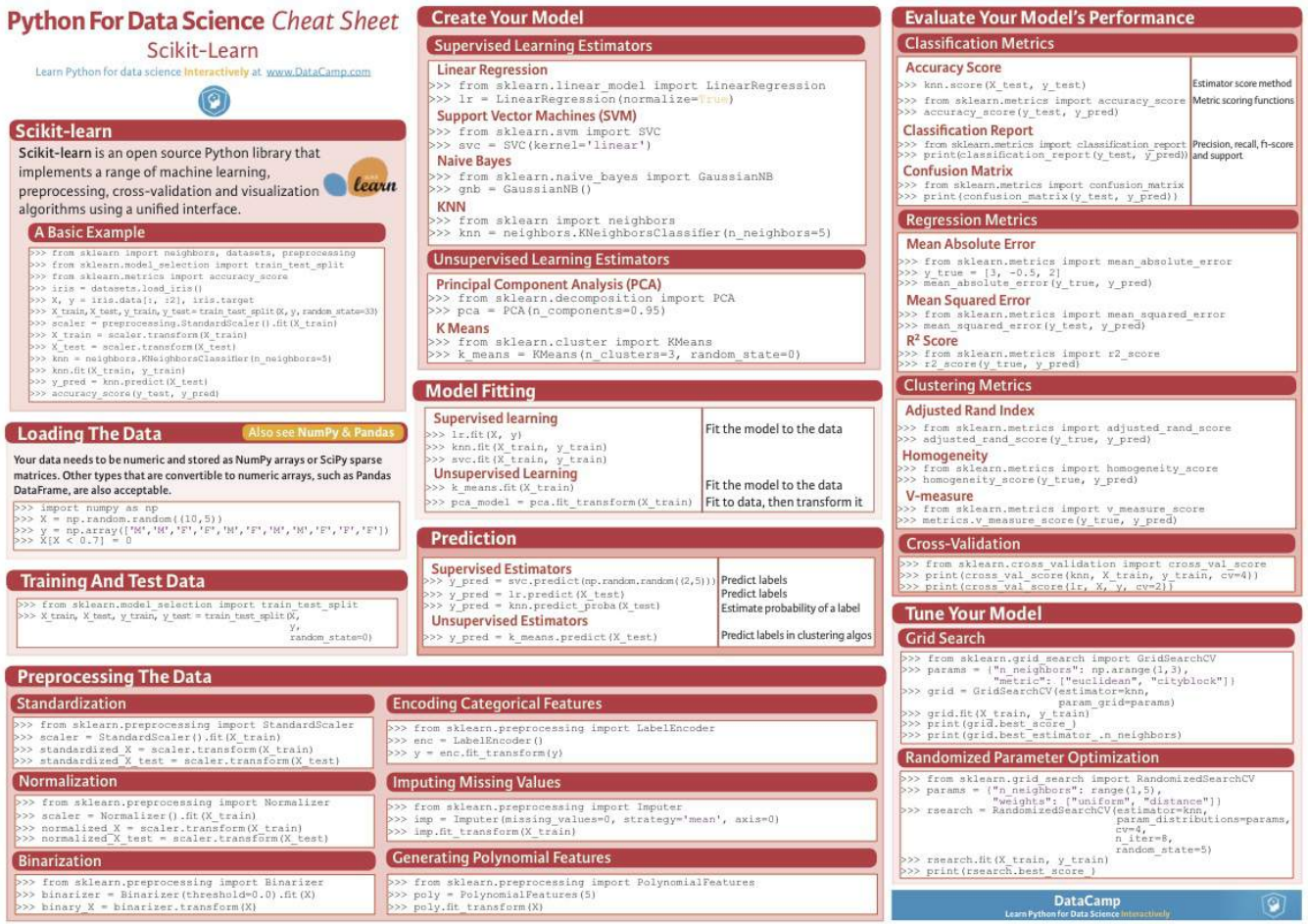
<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	&,|,~,^,`df.any()`,`df.all()`	Logical and, or, not, xor, any, all

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Matplotlib Cheat Sheet



Sscikit-Learn Cheat Sheet



Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Estimator score method

Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp

Learn Python for Data Science Interactively

Scikit-learn Design

https://arxiv.org/abs/1309.0238

Scikit-Learn's API is remarkably well designed. These are the main design principles:

Consistency: All objects share a consistent and simple interface:

- **Estimators** Any object that can estimate some parameters based on a dataset is called an *estimator* (e.g., an imputer is an estimator). The estimation itself is performed by the fit() method, and it takes only a dataset as a parameter (or two for supervised learning algorithms; the second dataset contains the labels). Any other parameter needed to guide the estimation process is considered a hyperparameter (such as an imputer's strategy), and it must be set as an instance variable (generally via a constructor parameter).
- **Transformers** Some estimators (such as an imputer) can also transform a dataset; these are called *transformers*. Once again, the API is simple: the transformation is performed by the transform() method with the dataset to transform as a parameter. It returns the transformed dataset. This transformation generally relies on the learned parameters, as is the case for an imputer. All transformers also have a convenience method called fit_transform() that is equivalent to calling fit() and then transform() (but sometimes fit_transform() is optimized and runs much faster).
- **Predictors** Finally, some estimators, given a dataset, are capable of making predictions; they are called *predictors*. For example, the LinearRegression model in the previous chapter was a predictor: given a country's GDP per capita, it predicted life satisfaction. A predictor has a predict() method that takes a dataset of new instances and returns a dataset of corresponding predictions. It also has a score() method that measures the quality of the predictions, given a test set (and the corresponding labels, in the case of supervised learning algorithms).[18]

Inspection: All the estimator's hyperparameters are accessible directly via public instance variables (e.g., imputer.strategy), and all the estimator's learned parameters are accessible via public instance variables with an underscore suffix (e.g., imputer.statistics_).

Nonproliferation of classes Datasets are represented as NumPy arrays or SciPy sparse matrices, instead of homemade classes. Hyperparameters are just regular Python strings or numbers.

Composition Existing building blocks are reused as much as possible. For example, it is easy to create a Pipeline estimator from an arbitrary sequence of transformers followed by a final estimator, as we will see.

Sensible defaults Scikit-Learn provides reasonable default values for most parameters, making it easy to quickly create a baseline working system.

Where to go from here

1. Learn the language
2. Learn the tools
3. Work on projects!

Resources

- **Machine Learning Mastery**
- **Hands-On Machine Learning with Scikit-Learn and TensorFlow**
- **The Complete Python Course for Machine Learning Engineers**
- **Cheat Sheet of Machine Learning and Python (and Math) Cheat Sheets**

Questions

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition by Aurélien Géron, Publisher: O'Reilly Media, Inc. Release Date: September 201

1. How would you define Machine Learning?
2. Can you name four types of problems where it shines?
3. What is a labeled training set?
4. What are the two most common supervised tasks?
5. Can you name four common unsupervised tasks?
6. What type of Machine Learning algorithm would you use to allow a robot to walk in various unknown terrains?
7. What type of algorithm would you use to segment your customers into multiple groups?
8. Would you frame the problem of spam detection as a supervised learning problem or an unsupervised learning problem?
9. What is an online learning system?
10. What is out-of-core learning?
11. What type of learning algorithm relies on a similarity measure to make predictions?
12. What is the difference between a model parameter and a learning algorithm’s hyperparameter?
13. What do model-based learning algorithms search for? What is the most common strategy they use to succeed? How do they make predictions?
14. Can you name four of the main challenges in Machine Learning?
15. If your model performs great on the training data but generalizes poorly to new instances, what is happening? Can you name three possible solutions?
16. What is a test set, and why would you want to use it?
17. What is the purpose of a validation set?
18. What is the train-dev set, when do you need it, and how do you use it?
19. What can go wrong if you tune hyperparameters using the test set?

Answers

1. Machine Learning is about building systems that can learn from data. Learning means getting better at some task, given some performance measure.
2. Machine Learning is great for complex problems for which we have no algorithmic solution, to replace long lists of hand-tuned rules, to build systems that adapt to fluctuating environments, and finally to help humans learn (e.g., data mining).
3. A labeled training set is a training set that contains the desired solution (a.k.a. a label) for each instance.
4. The two most common supervised tasks are regression and classification.
5. Common unsupervised tasks include clustering, visualization, dimensionality reduction, and association rule learning.
6. Reinforcement Learning is likely to perform best if we want a robot to learn to walk in various unknown terrains, since this is typically the type of problem that Reinforcement Learning tackles. It might be possible to express the problem as a supervised or semisupervised learning problem, but it would be less natural.
7. If you don't know how to define the groups, then you can use a clustering algorithm (unsupervised learning) to segment your customers into clusters of similar customers. However, if you know what groups you would like to have, then you can feed many examples of each group to a classification algorithm (supervised learning), and it will classify all your customers into these groups.
8. Spam detection is a typical supervised learning problem: the algorithm is fed many emails along with their labels (spam or not spam).
9. An online learning system can learn incrementally, as opposed to a batch learning system. This makes it capable of adapting rapidly to both changing data and autonomous systems, and of training on very large quantities of data.
10. Out-of-core algorithms can handle vast quantities of data that cannot fit in a computer's main memory. An out-of-core learning algorithm chops the data into mini-batches and uses online learning techniques to learn from these mini-batches.
11. An instance-based learning system learns the training data by heart; then, when given a new instance, it uses a similarity measure to find the most similar learned instances and uses them to make predictions.
12. A model has one or more model parameters that determine what it will predict given a new instance (e.g., the slope of a linear model). A learning algorithm tries to find optimal values for these parameters such that the model generalizes well to new instances. A hyperparameter is a parameter of the learning algorithm itself, not of the model (e.g., the amount of regularization to apply).
13. Model-based learning algorithms search for an optimal value for the model parameters such that the model will generalize well to new instances. We usually train such systems by minimizing a cost function that measures how bad the system is at making predictions on the training data, plus a penalty for model complexity if the model is regularized. To make predictions, we feed the new instance's features into the model's prediction function, using the parameter values found by the learning algorithm.
14. Some of the main challenges in Machine Learning are the lack of data, poor data quality, nonrepresentative data, uninformative features, excessively simple models that underfit the training data, and excessively complex models that overfit the data.
15. If a model performs great on the training data but generalizes poorly to new instances, the model is likely overfitting the training data (or we got extremely lucky on the training data). Possible solutions to overfitting are getting more data, simplifying the model (selecting a simpler algorithm, reducing the number of parameters or features used, or regularizing the model), or reducing the noise in the training data.
16. A test set is used to estimate the generalization error that a model will make on new instances, before the model is launched in production.
17. A validation set is used to compare models. It makes it possible to select the best model and tune the hyperparameters.
18. The train-dev set is used when there is a risk of mismatch between the training data and the data used in the validation and test datasets (which should always be as close as possible to the data used once the model is in production). The train-dev set is a part of the training set that's held out (the model is not trained on it). The model is trained on the rest of the training set, and evaluated on both the train-dev set and the validation set. If the model performs well on the training set but not on the train-dev set, then the model is likely overfitting the training set. If it performs well on both the training set and the train-dev set, but not on the validation set, then there is probably a significant data mismatch between the training data and the validation + test data, and you should try to improve the training data to make it look more like the validation + test data.
19. If you tune hyperparameters using the test set, you risk overfitting the test set, and the generalization error you measure will be optimistic (you may launch a model that performs worse than you expect).

CONNECTING WORLDS

Kontakt

Michael Aydinbas

Data Scientist

@ michael.aydinbas@EXXETA.com

m +49 174 99 50 913