Section 9 Introduction to Data Science

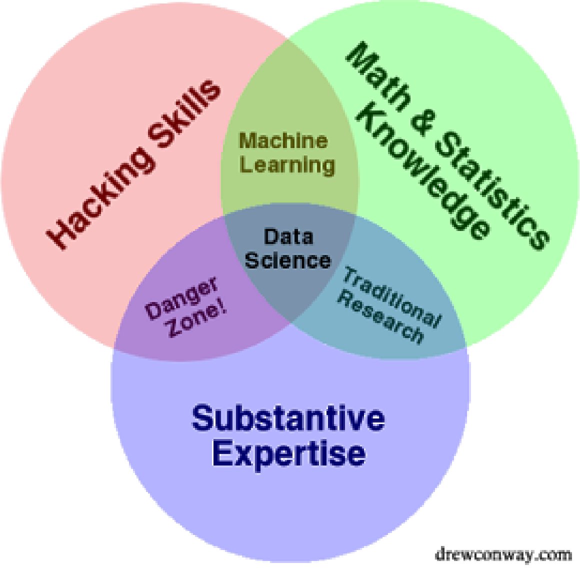
What is Data Science?

Three correlated concepts:

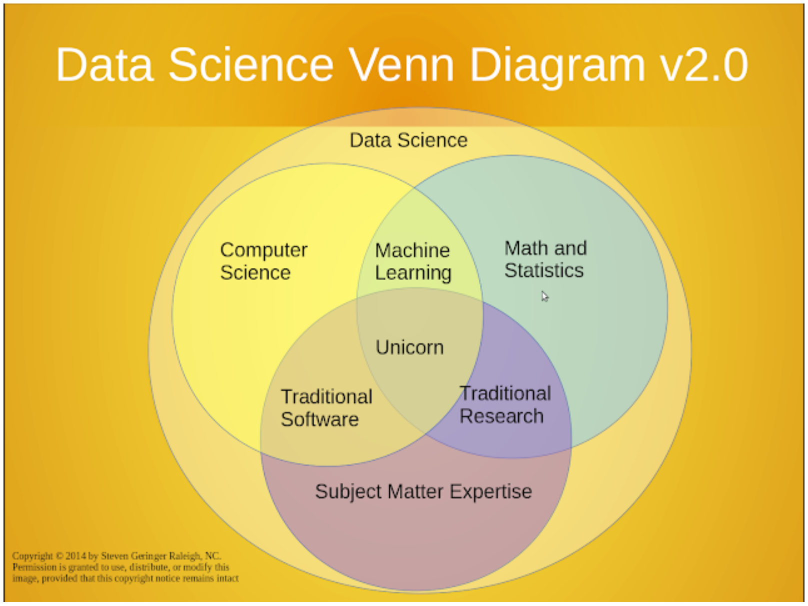
- Data Science
- Artificial Intelligence
- Machine Learning

[Battle of the Data Science Venn Diagrams (https://www.kdnuggets.com/2016/10/battle-data-science-venn-diagrams.html)](https://www.kdnuggets.com/2016/10/battle-data-science-venn-diagrams.html)

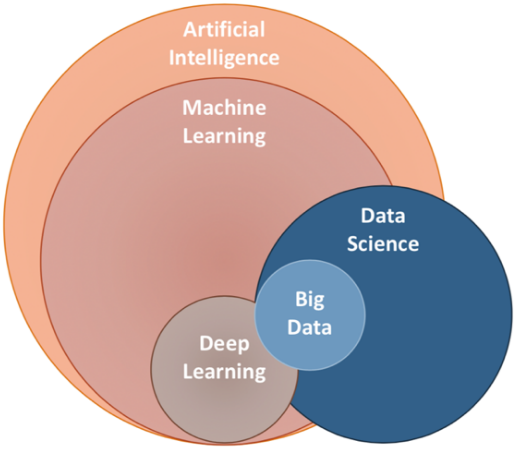
The original Venn diagram from Drew Conway:



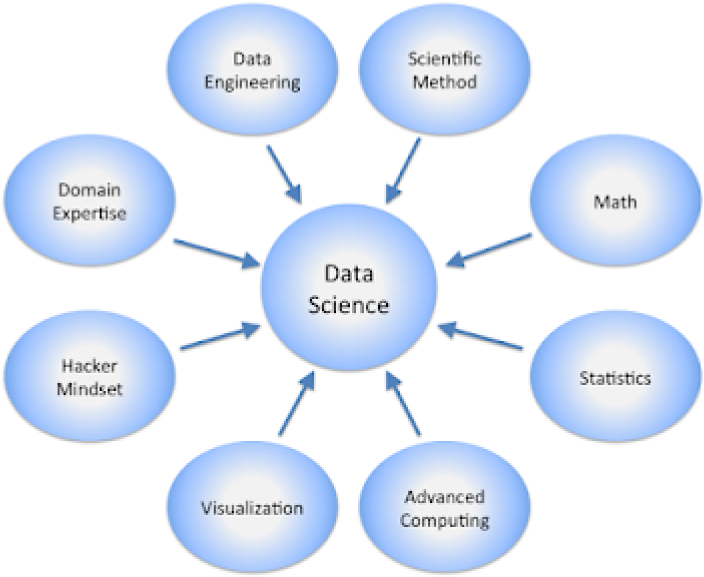
Another diagram from Steven Geringer:

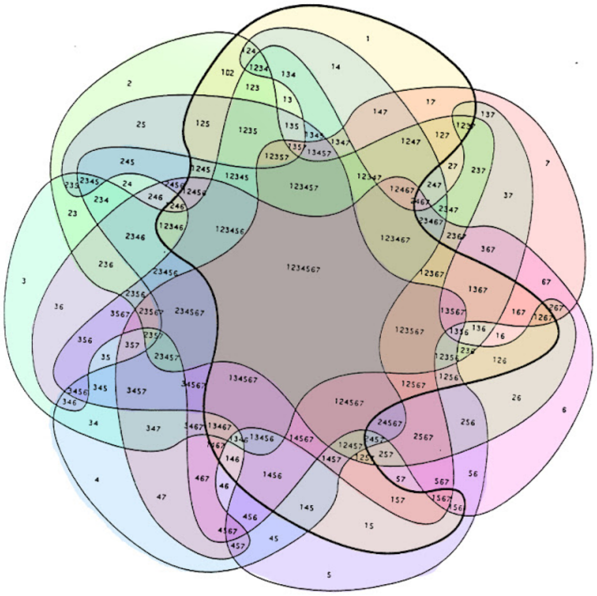

Data Science Venn Diagram v2.0
Data Science
Computer Science
Machine Learning
Math and Statistics
Unicorn
Traditional Software
Traditional Research
Subject Matter Expertise
Copyright © 2014 by Steven Geringer Raleigh, NC.
Permission is granted to use, distribute, or modify this
image, provided that this copyright notice remains intact

Another version:



Perhaps the reality should be:





[David Robinson's Auto-pilot example (http://varianceexplained.org/r/ds-ml-ai/)](http://varianceexplained.org/r/ds-ml-ai/):

- machine learning: **predict** whether there is a stop sign in the camera

- artificial intelligence: design the **action** of applying brakes (either by rules or from data)
- data science: provide the **insights** why the system does not work well after sunrise

Peijie's Definition: Data Science is the science

- *of* the data -- what
- *by* the data -- how
- *for* the data -- why

Mathematics of Data

Representation of Data

What data do we have, and how to relate it with math objects?

Tabular Data

In [6]:

```
import pandas as pd
import numpy as np
df_house = pd.read_csv('./data/kc_house_data.csv')
print(df_house.shape)
df_house.head()
```

```
(21613, 21)
```

Out[6]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0

5 rows × 21 columns

- A structured data table, with n observations and p variables.
- **Mathematical representation**: The data *matrix* $X \in \mathbb{R}^{n \times p}$. For notations we write

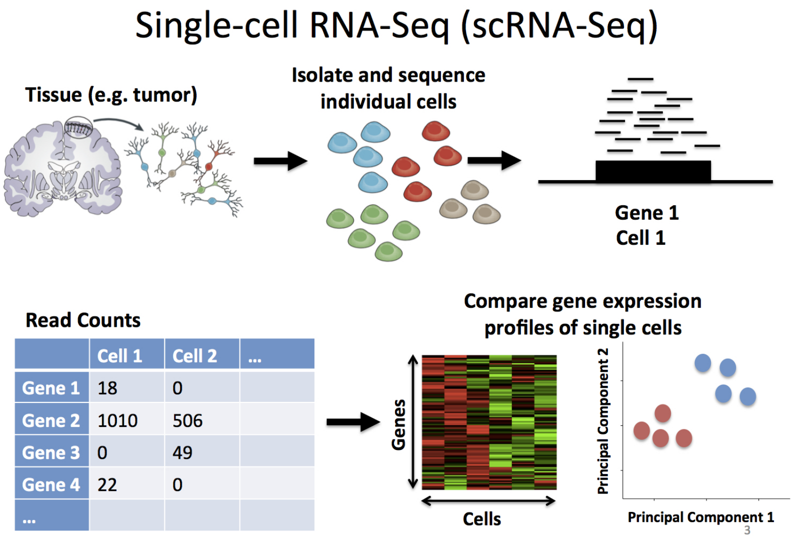
$$X = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \cdots \\ \mathbf{x}^{(n)} \end{pmatrix}, \text{ where the } i\text{-th row vector represents } i\text{-th observation,}$$

$$\mathbf{x}^{(i)} = \left(x_1^{(i)}, \ldots, x_p^{(i)}\right) \in \mathbb{R}^p.$$

To really emphasize that each element is a row, we can also write X as:

$$X = \begin{pmatrix} \mathbf{x}^{(1)} & \longrightarrow \\ \mathbf{x}^{(2)} & \longrightarrow \\ & \dots \\ \mathbf{x}^{(n)} & \longrightarrow \end{pmatrix}$$

- Example: Precision Medicine and Single-cell Sequencing. (https://learn.gencore.bio.nyu.edu/single-cell-rnaseq/)



Single-cell RNA-Seq (scRNA-Seq)

Tissue (e.g. tumor)
Isolate and sequence individual cells
Gene 1
Cell 1
Compare gene expression profiles of single cells
Genes
Cells
Principal Component 2
Principal Component 1
3

Read Counts

	Cell 1	Cell 2	...
Gene 1	18	0	
Gene 2	1010	506	
Gene 3	0	49	
Gene 4	22	0	
...			

- *Roughly speaking*, big data -- large n, high-dimensional data -- large p.

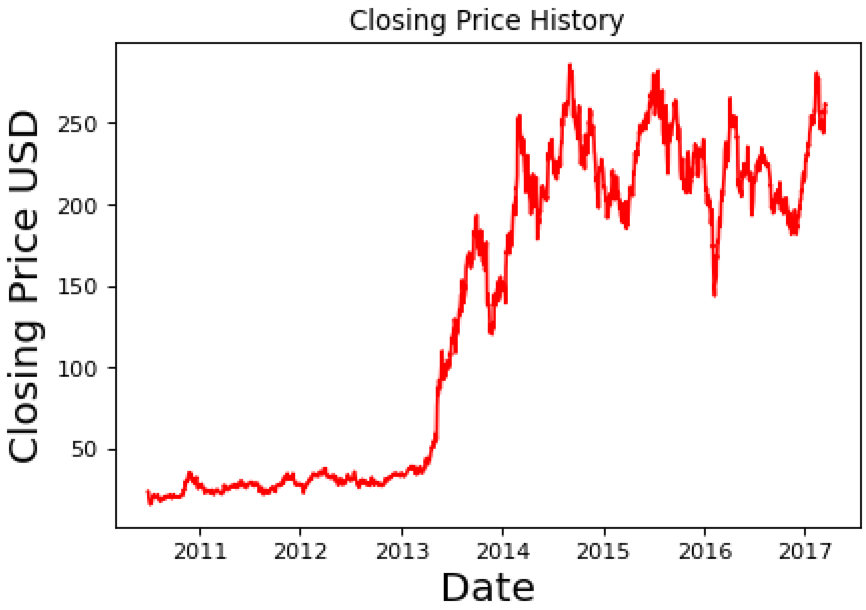
Time-series Data

```
In [7]:  import matplotlib.pyplot as plt
         ts_tesla = pd.read_csv('./data/Tesla.csv')
         print(ts_tesla.head())

         ts_tesla['Date'] = pd.to_datetime(ts_tesla['Date'])
         ts_tesla.set_index('Date',inplace=True)

         # Suppose we only focus on the time-series of close price
         plt.figure(dpi=80)
         plt.title('Closing Price History')
         plt.plot(ts_tesla['Close'], color='red')
         plt.xlabel('Date', fontsize=18)
         plt.ylabel('Closing Price USD', fontsize = 18)
         plt.show()
         # this is only about tesla -- we can also have the time-series of apple,amazo
```

```
        Date       Open   High        Low      Close    Volume  Adj Close
0  6/29/2010  19.000000  25.00  17.540001  23.889999  18766300  23.889999
1  6/30/2010  25.790001  30.42  23.299999  23.830000  17187100  23.830000
2   7/1/2010  25.000000  25.92  20.270000  21.959999   8218800  21.959999
3   7/2/2010  23.000000  23.10  18.709999  19.200001   5139800  19.200001
4   7/6/2010  20.000000  20.00  15.830000  16.110001   6866900  16.110001
```



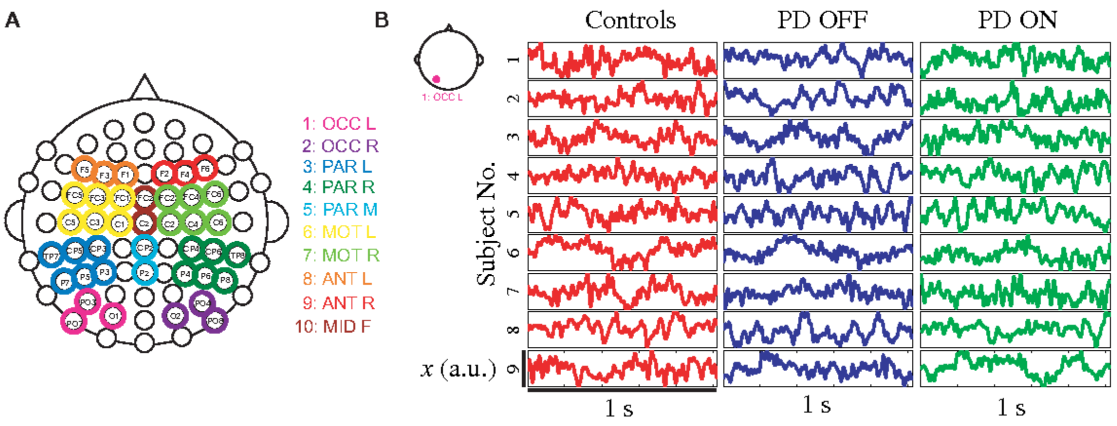
- Simple case: N one-dimensional trajectories with each sampled at T time points.
- **Mathematical representation I**: Still use the data *matrix* $X \in \mathbb{R}^{N \times T}$. For notations we write

$$X = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \cdots \\ \mathbf{x}^{(N)} \end{pmatrix}, \text{ where the } i\text{-th row vector represents } i\text{-th trajectory,}$$

$$\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_T^{(i)}) \in \mathbb{R}^T.$$

- Question: The difference with tabular data?

- **Mathematical representation II**: Each trajectory is a *function* of time t. The whole dataset can be represented as $z = f(\omega, t)$ where ω represents the sample and t represents the time. In probability theory, this is called *stochastic process*.
  - For fixed ω, we have a trajectory, which is the function of time.
  - For fixed t, we obtain an ensemble drawn from random distribution.
- Question: How about N d-dimensional trajectories with each sampled at T time points?
- Example: Electroencephalography (EEG) data and Parkinson's disease (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3858815/).



Images

Example: MNIST handwritten digits data (http://yann.lecun.com/exdb/mnist/):Each image is 28x28 matrix

In [7]:

```
import pandas as pd
mnist = pd.read_csv('./data/train.csv') # stored as data table
#mnist.sample(5)
mnist.head()
```

Out[7]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pix
0	1	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 785 columns

In [2]:

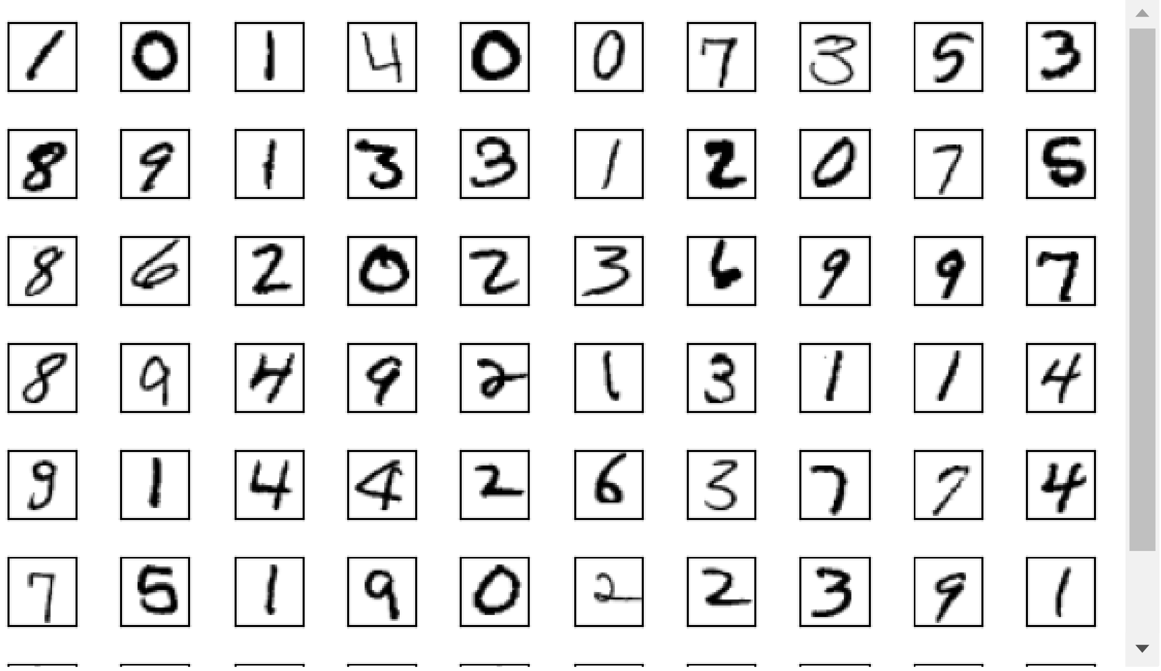
```
mnist.shape
```

Out[2]:

```
(42000, 785)
```

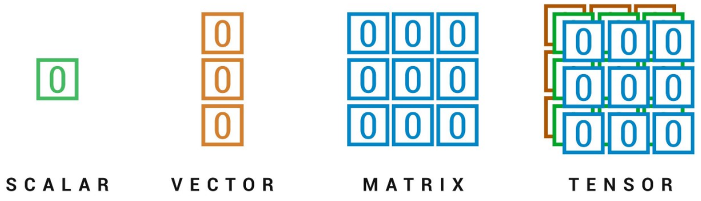
```
In [3]:  target = mnist['label']
          mnist = mnist.drop("label",axis=1)

          import matplotlib.pyplot as plt
          plt.figure(dpi=100)
          for i in range(0,70): #plot the first 70 images
              plt.subplot(7,10,i+1)
              grid_data = mnist.iloc[i,:].to_numpy().reshape(28,28)  # reshape from 1d
              plt.imshow(grid_data,cmap='gray_r', vmin=0, vmax=255)
              plt.xticks([])
              plt.yticks([])
          plt.tight_layout()
```



- Simple case: N grayscale images with $m \times n$ pixels each.
- **Mathematical Representation I**: Each image can be represented by a matrix $I \in \mathbb{R}^{m \times n}$, whose elements denotes the intensities of pixels. The whole datasets have N matrices of m by n, or represented by a $N \times m \times n$ *tensor*.

[Illustrated Introduction to Linear Algebra using NumPy (https://medium.com/@kaaanishk/illustrated-introduction-to-linear-algebra-using-numpy-11d503d244a1)](https://medium.com/@kaaanishk/illustrated-introduction-to-linear-algebra-using-numpy-11d503d244a1)

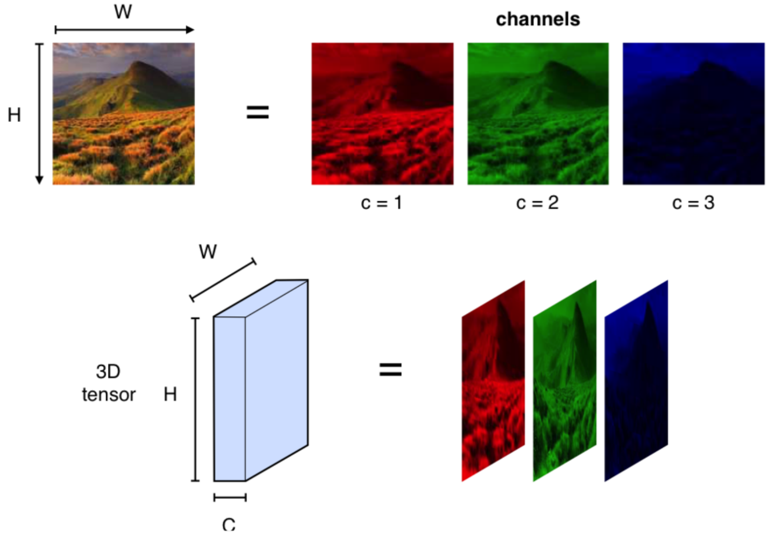


- **Mathematical representation II**: *Random field model* $z = \mathbf{f}(\omega, x, y)$.

- **Color images**: Decompose into RGB (red,green and blue) channels and
  - use three matrices (or three-dimensional tensor) to represent one image, or

    - build the random field model with vector-valued functions $z = \mathbf{f}(\omega, x, y) \in \mathbb{R}^3$

convolutional neural networks (https://www.esantus.com/blog/2019/1/31/convolutional-neural-networks-a-quick-guide-for-newbies)



- Question: Can image datasets also be transformed into tabular data? What are the pros/cons?

In [19]:

```
mnist.head()
```

Out[19]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pi
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 784 columns

Videos

- *Time-series* of images, or *random field* model $z = \mathbf{f}(\omega, x, y, t)$

Texts

In [5]:

```
from sklearn.feature_extraction.text import CountVectorizer

corpus = ['He is a good person',
          'He is bad student',
          'He is hardworking']
df = pd.DataFrame(data=corpus, columns=['sentences'])
print(df)
vectorizer = CountVectorizer(vocabulary=['he', 'is', 'a', 'good', 'person', '
                             stop_words=frozenset(), token_pattern=r"(?u)\b\w
X = vectorizer.fit_transform(df['sentences'].values)
result = pd.DataFrame(data=X.toarray(), columns=vectorizer.get_feature_names(
result.head()
```

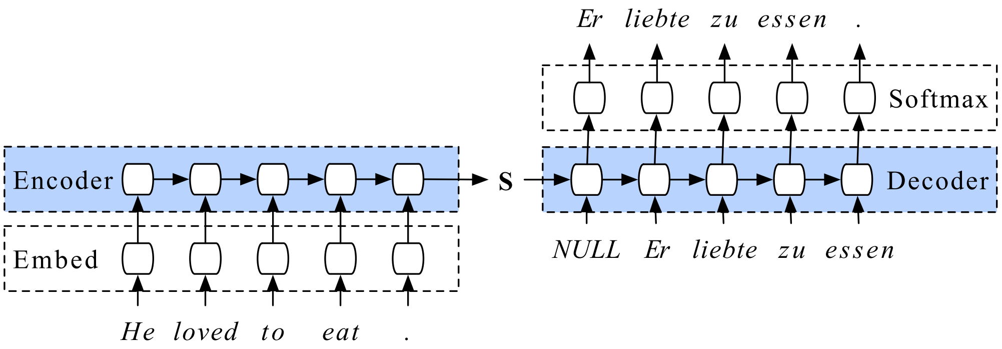
```
             sentences
0  He is a good person
1    He is bad student
2    He is hardworking
```

Out[5]:

	he	is	a	good	person	bad	student	hardworking
0	1	1	1	1	1	0	0	0
1	1	1	0	0	0	1	1	0
2	1	1	0	0	0	0	0	1

- **Proposal I**: Tabular data by extracting key words. "Document-Term Matrix"
  - useful in sentiment analysis, document clustering, topic modelling
  - popular algorithms include tf-idf,Word2Vec,bag of words, etc.
- **Proposal II**: Time-series of individual words.
  - useful in machine translation

[Recurrent neural network model for machine translations (https://smerity.com/articles/2016/google_nmt_arch.html)](https://smerity.com/articles/2016/google_nmt_arch.html)



Networks

- Concepts: node/edge/weight, directed/undirected
- **Mathematical Representation**: adjacency matrix
- Question: what about the whole datasets of networks, and time-evolving networks?

Tasks with Data: Machine Learning

The tasks with data can often be transfromed into *machine learning* problems, which can be generally classified as:

- Supervised Learning -- "learning with training";
- Unsupervised Learning -- "learning without training";
- Reinforcement Learning -- "learning by doing".

Our course will focus on the first two categories.

Supervised Learning

- Given the *training dataset* $\left(x^{(i)}, y^{(i)}\right)$ with $y^{(i)} \in \mathbb{R}^q$ denotes the *labels*, the supervised learning aims to find a mapping $\mathbf{f}: \mathbb{R}^p \rightarrow \mathbb{R}^q$ such that $y^{(i)} \approx \mathbf{f}\left(x^{(i)}\right)$. Then with a new observation $x^{(new)}$, we can predict that $y^{(new)} = \mathbf{f}\left(x^{(new)}\right)$.
  - when $y \in \mathbb{R}$ is continuous, the problem is also called as *regression*. **Example**: Housing price prediction
  - when $y \in \mathbb{R}$ is discrete, the problem is also called as *classification*. **Example**: Handwritten digit recognization

- **Practical Strategy**: Limit the mapping $\mathbf{f}$ to certain space by parametrization $\mathbf{f(x;\theta)}$. Then define the loss function of θ
$$L(\theta) = \sum_{i=1}^{n} \ell\left(y^{(i)}, \mathbf{f}\left(x^{(i)}\right)\right),$$
where ℓ quantifies the "distance" between $y^{(i)}$ and $\mathbf{f}(x^{(i)})$, and a common choice is mean squre error (MSE) for continous data $\ell\left(y^{(i)}, \mathbf{f}\left(x^{(i)}\right)\right) = \left\|y^{(i)} - \mathbf{f}(x^{(i)})\right\|^2$. We then seek to choose the optimal θ that minimizes the loss function
$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L(\theta),$$
which can be tacked numerically by optimzation methods (including the popular stochastic gradient descent).

- Difference choice of $\mathbf{f(x;\theta)}$ leads to various supervised learning models:
  - Linear function : Linear Regression (for regression)/Logistic Regression (for classification)
  - For 1D Linear Regression (finding a line of best fit $y = \omega x + b$), we have $\mathbf{f(x;\theta)} = \mathbf{f(x;\omega, b)} = \omega x + b$
  - Composition of linear + nonlinear functions: Neural Network

- **Important Terms**:
  - **Training Data**: Both X and y are provided. The dataset which we use to fit the function.
  - **Test Data**: In principle, only X is provided (some times y^{test} is also provided as the ground-truth to verify). The dataset which we generate new predictions y^{pred}. -- This is the final judgement of your unsupervised ML model!
  - **Validation Data**: A good-fit model on training data does not guarantee the good performance on test data. To gain more confidence before really applying to test data, we "fake" some test data as the "sample exam". To do this, we further split the original training data into new traning data and validation data, and then learn the mapping on

new training data, and judge on the validation data. We may make some adjustment if the model does not perform well in the "sample exam".

- Intuitive Understanding: Training data is like quizzes -- you want to learn the "mapping" between the question and correct answer. Test data is like your exam. Validation is like you take a sample exam before the real exam and make some "clinics" about your weakpoints.
- See the illustration here (https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7)

Example: The Wisconsin breast cancer dataset (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)) and low-code ML package pycaret (https://pycaret.org/).

Install pycaret -- it's a new package, not included with Anaconda

In [9]:

```
pip install --upgrade pycaret
```

```
Requirement already satisfied: pycaret in e:\programdata\anaconda3\lib\si
te-packages (2.3.2)
Requirement already satisfied: nltk in e:\programdata\anaconda3\lib\site-
packages (from pycaret) (3.6.1)
Requirement already satisfied: imbalanced-learn==0.7.0 in e:\programdata
\anaconda3\lib\site-packages (from pycaret) (0.7.0)
Requirement already satisfied: pandas-profiling>=2.8.0 in e:\programdata
\anaconda3\lib\site-packages (from pycaret) (3.0.0)
Requirement already satisfied: pandas in e:\programdata\anaconda3\lib\sit
e-packages (from pycaret) (1.3.0)
Requirement already satisfied: IPython in e:\programdata\anaconda3\lib\si
te-packages (from pycaret) (7.22.0)
Requirement already satisfied: lightgbm>=2.3.1 in e:\programdata\anaconda
3\lib\site-packages (from pycaret) (3.2.1)
Requirement already satisfied: scikit-plot in e:\programdata\anaconda3\li
b\site-packages (from pycaret) (0.3.7)
Requirement already satisfied: mlxtend>=0.17.0 in e:\programdata\anaconda
3\lib\site-packages (from pycaret) (0.18.0)
Requirement already satisfied: joblib in e:\programdata\anaconda3\lib\sit
```

In [10]:

```
from sklearn.datasets import load_breast_cancer # Load the dataset
X,y = load_breast_cancer(as_frame = True,return_X_y = True)
```

In [11]:
```
X
```

Out[11]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symm
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1

569 rows × 30 columns

In [12]:
```
y
```

Out[12]:
```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target, Length: 569, dtype: int32
```

In this dataset, all labels are known. To mimic a real situation, we manully create train and test datasets.

In [13]:
```
from sklearn.model_selection import train_test_split # manually split into tr
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, rar
```

In [14]:
```
X_train.shape
```

Out[14]:
```
(381, 30)
```

In [15]:
```
y_test.shape
```

Out[15]:
```
(188,)
```

In [16]:
```
import pandas as pd
data_train = pd.concat([X_train,y_train],axis=1) # the whole data table of tr
data_train
```

Out[16]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	n symn
56	19.210	18.57	125.50	1152.0	0.10530	0.12670	0.13230	0.089940	0.
144	10.750	14.97	68.26	355.3	0.07793	0.05139	0.02251	0.007875	0.
60	10.170	14.88	64.55	311.9	0.11340	0.08061	0.01084	0.012900	0.
6	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.074000	0.
8	13.000	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.093530	0.
...	...	...	...	...	...	...	...	...	
277	18.810	19.98	120.90	1102.0	0.08923	0.05884	0.08020	0.058430	0.
9	12.460	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.085430	0.
359	9.436	18.32	59.82	278.6	0.10090	0.05956	0.02710	0.014060	0.
192	9.720	18.22	60.73	288.1	0.06950	0.02344	0.00000	0.000000	0.
559	11.510	23.93	74.52	403.5	0.09261	0.10210	0.11120	0.041050	0.

381 rows × 31 columns

In [17]:

```python
from pycaret.classification import setup
from pycaret.classification import compare_models

bc = setup(data=data_train, target='target') # target is the y column name we
```

	Description	Value
0	session_id	2656
1	Target	target
2	Target Type	Binary
3	Label Encoded	None
4	Original Data	(381, 31)
5	Missing Values	False
6	Numeric Features	30
7	Categorical Features	0
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(266, 29)
12	Transformed Test Set	(115, 29)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	00eb
22	Imputation Type	simple
23	Iterative Imputation Iteration	None
24	Numeric Imputer	mean
25	Iterative Imputation Numeric Model	None
26	Categorical Imputer	constant
27	Iterative Imputation Categorical Model	None
28	Unknown Categoricals Handling	least_frequent
29	Normalize	False
30	Normalize Method	None
31	Transformation	False
32	Transformation Method	None

	Description	Value
33	PCA	False
34	PCA Method	None
35	PCA Components	None
36	Ignore Low Variance	False
37	Combine Rare Levels	False
38	Rare Level Threshold	None
39	Numeric Binning	False
40	Remove Outliers	False
41	Outliers Threshold	None
42	Remove Multicollinearity	False
43	Multicollinearity Threshold	None
44	Remove Perfect Collinearity	True
45	Clustering	False
46	Clustering Iteration	None
47	Polynomial Features	False
48	Polynomial Degree	None
49	Trignometry Features	False
50	Polynomial Threshold	None
51	Group Features	False
52	Feature Selection	False
53	Feature Selection Method	classic
54	Features Selection Threshold	None
55	Feature Interaction	False
56	Feature Ratio	False
57	Interaction Threshold	None
58	Fix Imbalance	False
59	Fix Imbalance Method	SMOTE

```
In [18]:  best = compare_models() # pycaret automatically fits different ML models for
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.9664	0.9814	0.9820	0.9669	0.9737	0.9269	0.9296	0.0890
et	Extra Trees Classifier	0.9664	0.9841	0.9824	0.9666	0.9739	0.9266	0.9287	0.0680
ada	Ada Boost Classifier	0.9625	0.9775	0.9765	0.9660	0.9709	0.9181	0.9192	0.0400
qda	Quadratic Discriminant Analysis	0.9588	0.9842	0.9765	0.9607	0.9681	0.9102	0.9123	0.0070
lda	Linear Discriminant Analysis	0.9587	0.9876	0.9879	0.9512	0.9685	0.9083	0.9123	0.0060
lightgbm	Light Gradient Boosting Machine	0.9587	0.9842	0.9706	0.9657	0.9675	0.9106	0.9125	0.1510
gbc	Gradient Boosting Classifier	0.9585	0.9807	0.9702	0.9657	0.9675	0.9100	0.9115	0.0590
ridge	Ridge Classifier	0.9473	0.0000	0.9875	0.9340	0.9594	0.8840	0.8894	0.0060
lr	Logistic Regression	0.9437	0.9873	0.9643	0.9489	0.9559	0.8779	0.8807	0.6660
nb	Naive Bayes	0.9437	0.9853	0.9640	0.9499	0.9560	0.8776	0.8813	0.0060
dt	Decision Tree Classifier	0.9132	0.9082	0.9287	0.9366	0.9310	0.8141	0.8190	0.0060
knn	K Neighbors Classifier	0.9100	0.9400	0.9467	0.9206	0.9311	0.8011	0.8104	0.0110
svm	SVM - Linear Kernel	0.9026	0.0000	0.9529	0.9071	0.9257	0.7838	0.8000	0.0060

```
In [19]:  best # the best model selected by pycaret
```

```
Out[19]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                criterion='gini', max_depth=None, max_features='aut
         o',
                                max_leaf_nodes=None, max_samples=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=100,
                                n_jobs=-1, oob_score=False, random_state=2656, verbo
         se=0,
                                warm_start=False)
```

In [20]:

```
from pycaret.classification import predict_model
predict_model(best); # predict on the validation data that pycaret have selec
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	0.9478	0.9958	0.9552	0.9552	0.9552	0.8927	0.8927

In [21]:

```
from pycaret.classification import finalize_model
best_final = finalize_model(best) # re-train the dataset with whole input tra
```

In [22]:

```
from pycaret.classification import predict_model
predictions = predict_model(best_final, data = X_test) # make new predictions
predictions
```

Out[22]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symme
512	13.40	20.52	88.64	556.7	0.11060	0.14690	0.14450	0.08172	0.2
457	13.21	25.25	84.10	537.9	0.08791	0.05205	0.02772	0.02068	0.1
439	14.02	15.66	89.59	606.5	0.07966	0.05581	0.02087	0.02652	0.1
298	14.26	18.17	91.22	633.1	0.06576	0.05220	0.02475	0.01374	0.1
37	13.03	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.02923	0.1
...	...	...	...	...	...	...	...	...	
100	13.61	24.98	88.05	582.7	0.09488	0.08511	0.08625	0.04489	0.1
336	12.99	14.23	84.08	514.3	0.09462	0.09965	0.03738	0.02098	0.1
299	10.51	23.09	66.85	334.2	0.10150	0.06797	0.02495	0.01875	0.1
347	14.76	14.74	94.87	668.7	0.08875	0.07780	0.04608	0.03528	0.1
502	12.54	16.32	81.25	476.3	0.11580	0.10850	0.05928	0.03279	0.1

188 rows × 32 columns

In [23]:

```
df_compare = pd.concat([predictions['Label'],y_test],axis = 1) # compare with
df_compare
```

Out[23]:

	Label	target
512	0	0
457	1	1
439	1	1
298	1	1
37	1	1
...	...	...
100	0	0
336	1	1
299	1	1
347	1	1
502	1	1

188 rows × 2 columns

In [24]:

```
import numpy as np
np.mean(predictions['Label'].to_numpy() == y_test.to_numpy()) # calculate the
#mean of the number of matches, using a boolean test on the array.
```

Out[24]:

```
0.9627659574468085
```

In [25]:

```
from pycaret.classification import create_model
lr = create_model('lr') # what if we only want the logistic regression model?
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9630	1.0000	1.0000	0.9444	0.9714	0.9189	0.9220
1	0.9630	1.0000	0.9412	1.0000	0.9697	0.9222	0.9250
2	0.9259	0.9824	0.9412	0.9412	0.9412	0.8412	0.8412
3	0.9259	0.9471	0.9412	0.9412	0.9412	0.8412	0.8412
4	0.9259	1.0000	1.0000	0.8947	0.9444	0.8344	0.8460
5	0.9259	0.9882	0.9412	0.9412	0.9412	0.8412	0.8412
6	0.9231	0.9812	0.9375	0.9375	0.9375	0.8375	0.8375
7	0.9615	0.9869	1.0000	0.9444	0.9714	0.9128	0.9162
8	0.9615	0.9869	0.9412	1.0000	0.9697	0.9172	0.9204
9	0.9615	1.0000	1.0000	0.9444	0.9714	0.9128	0.9162
Mean	0.9437	0.9873	0.9643	0.9489	0.9559	0.8779	0.8807
SD	0.0184	0.0153	0.0291	0.0292	0.0149	0.0390	0.0394

In [26]:

```
predict_model(lr) # validation dataset -- sample exam!
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.9391	0.9925	0.9403	0.9545	0.9474	0.8752	0.8754

Out[26]:

	mean radius	mean texture	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	r symr
0	16.129999	17.879999	807.200012	0.10400	0.15590	0.13540	0.077520	0.
1	19.889999	20.260000	1214.000000	0.10370	0.13100	0.14110	0.094310	0.
2	17.750000	28.030001	981.599976	0.09997	0.13140	0.16980	0.082930	0.
3	13.900000	19.240000	602.900024	0.07991	0.05326	0.02995	0.020700	0.
4	11.680000	16.170000	420.500000	0.11280	0.09263	0.04279	0.031320	0.
...	...	...	...	...	...	...	...	
110	14.290000	16.820000	632.599976	0.06429	0.02675	0.00725	0.006250	0.
111	16.459999	20.110001	832.900024	0.09831	0.15560	0.17930	0.088660	0.
112	9.668000	18.100000	286.299988	0.08311	0.05428	0.01479	0.005769	0.
113	12.400000	17.680000	467.799988	0.10540	0.13160	0.07741	0.027990	0
114	14.420000	19.770000	642.500000	0.09752	0.11410	0.09388	0.058390	0.

115 rows × 32 columns

In [27]:

```
final_lr = finalize_model(lr)
```

In [28]:

```
predictions_lr = predict_model(final_lr, data = X_test)
np.mean(predictions_lr['Label'].to_numpy() == y_test.to_numpy())
```

Out[28]:

```
0.9627659574468085
```

In [29]:

```
from pycaret.classification import tune_model
tuned_lr = tune_model(lr) # fine-tuning the parameters in logistic regression
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9630	0.9941	1.0000	0.9444	0.9714	0.9189	0.9220
1	0.9630	1.0000	0.9412	1.0000	0.9697	0.9222	0.9250
2	0.9259	0.9882	0.9412	0.9412	0.9412	0.8412	0.8412
3	0.9259	0.9588	0.9412	0.9412	0.9412	0.8412	0.8412
4	0.9259	0.9941	1.0000	0.8947	0.9444	0.8344	0.8460
5	0.9630	0.9882	0.9412	1.0000	0.9697	0.9222	0.9250
6	0.9615	0.9812	0.9375	1.0000	0.9677	0.9202	0.9232
7	0.9615	0.9869	1.0000	0.9444	0.9714	0.9128	0.9162
8	0.9615	0.9869	0.9412	1.0000	0.9697	0.9172	0.9204
9	0.9615	1.0000	1.0000	0.9444	0.9714	0.9128	0.9162
Mean	0.9513	0.9879	0.9643	0.9610	0.9618	0.8943	0.8976
SD	0.0166	0.0112	0.0291	0.0348	0.0129	0.0364	0.0360

In [30]:

```
predict_model(tuned_lr) # still doing the sample exam -- validation dataset
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.9565	0.9932	0.9403	0.9844	0.9618	0.9114	0.9127

Out[30]:

	mean radius	mean texture	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	symr
0	16.129999	17.879999	807.200012	0.10400	0.15590	0.13540	0.077520	0.
1	19.889999	20.260000	1214.000000	0.10370	0.13100	0.14110	0.094310	0.
2	17.750000	28.030001	981.599976	0.09997	0.13140	0.16980	0.082930	0.
3	13.900000	19.240000	602.900024	0.07991	0.05326	0.02995	0.020700	0.
4	11.680000	16.170000	420.500000	0.11280	0.09263	0.04279	0.031320	0.
...	...	...	...	...	...	...	...	
110	14.290000	16.820000	632.599976	0.06429	0.02675	0.00725	0.006250	0.
111	16.459999	20.110001	832.900024	0.09831	0.15560	0.17930	0.088660	0.
112	9.668000	18.100000	286.299988	0.08311	0.05428	0.01479	0.005769	0.
113	12.400000	17.680000	467.799988	0.10540	0.13160	0.07741	0.027990	0
114	14.420000	19.770000	642.500000	0.09752	0.11410	0.09388	0.058390	0.

115 rows × 32 columns

In [31]:

```
final_tuned_lr = finalize_model(tuned_lr) #retrain with the whole dataset
```

In [32]:

```
predictions_tuned_lr = predict_model(final_tuned_lr, data = X_test)
np.mean(predictions_tuned_lr['Label'].to_numpy() == y_test.to_numpy())
```

Out[32]:

```
0.9468085106382979
```

Let's recap the workflow above (or about general supervised learning)

- The **minimum requirement** is that we have a training dataset with both X and y (also called labels, targets...). We want to **fit the mapping** between x and y with **training dataset** (the process is indeed called training), and making predictions about the new y given new X in the test dataset.
  - *Remark 1*: The true y in test dataset sometimes can also be known, so that we can know the performance the model immediately. But in general, we won't expect this.
  - *Remark 2*: In our course, just to mimic a real-world situation, sometimes we manually create (split) the train or test data.

- (Optional) We may train multiple models or one model with multiple parameters. How can we compare them and gain more confidence about the final test? Sometimes we further split the training dataset into (real) training dataset and **validation dataset** (imagine it as the sample exam), so that we can get instant feedback because we know the true label in validation dataset.

- (Optional) During training, to be more cautious, sometimes we even make more "quizzes" -- that is called **cross-validation** (will talk about the details in the next lecture)

- (Optional) With 10 "quizzes" (10-fold cross-validation) and "one sample exam" (validation data), for instance, we finally pick up the best candidate model. Before applying to the real test dataset, we don't want to waste any sample. Therefore we **finalize** training by picking up the winner model, while updating it with all the samples (including the validation data) in the training dataset.

- Finally, applying the model to test data -- wait and see!

Of course, as a math course, we are not satisfied with merely calling functions in pycaret. In the rest of lectures this quarter, we are going to dig into details of some algorihms and learn more underlying math -- turn the black box of ML into white (at least gray) one!

Unsupervised Learning

It is still challenging to give a general and rigorous definition for unsupervised learning mathematically. Let's focus on more specific tasks.

- Dimension Reducion

Given $X \in \mathbb{R}^{n\times p}$, finding a mapping function $\mathbf{f}: \mathbb{R}^p \to \mathbb{R}^q (q \ll p)$ such that the low-dimensional coordinates $z^{(i)} = \mathbf{f}(x^{(i)})$ "preserve the information" about $x^{(i)}$.

- Question: Difference with supervised learning?
- Linear mapping: Principle Component Analysis (PCA)
- Nonlinear mapping: Manifold Learning, Autoencoder

In [1]:

```
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y = True) # Note that in the hw this week, it's not
X
```

Out[1]:

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

In [2]:

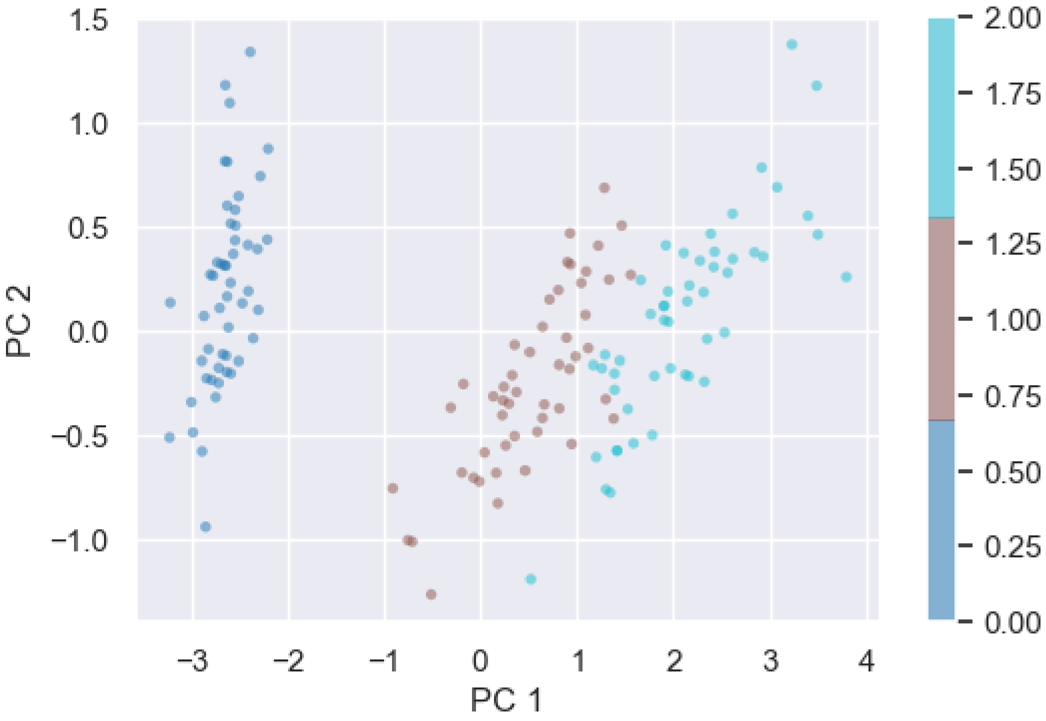
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2) # principle component analysis, reduce 4-dimenional
X_pca = pca.fit_transform(X)
X_pca
```

Out[2]:

```
array([[-2.68412563,  0.31939725],
       [-2.71414169, -0.17700123],
       [-2.88899057, -0.14494943],
       [-2.74534286, -0.31829898],
       [-2.72871654,  0.32675451],
       [-2.28085963,  0.74133045],
       [-2.82053775, -0.08946138],
       [-2.62614497,  0.16338496],
       [-2.88638273, -0.57831175],
       [-2.6727558 , -0.11377425],
       [-2.50694709,  0.6450689 ],
       [-2.61275523,  0.01472994],
       [-2.78610927, -0.235112  ],
       [-3.22380374, -0.51139459],
       [-2.64475039,  1.17876464],
       [-2.38603903,  1.33806233],
       [-2.62352788,  0.81067951],
       [-2.64829671,  0.31184914],
       [-2.19982032,  0.87283904],
```

In [7]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
sns.set() # set the seaborn theme style
figure = plt.figure(dpi=100)
plt.scatter(X_pca[:, 0], X_pca[:, 1],c=y, s=15, edgecolor='none', alpha=0.5,c
#colors determined by y, the true species of each iris flower
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.colorbar();
```



- Clustering

  Given $X \in \mathbb{R}^{n \times p}$, finding a partition of the dataset into K groups such that
  - data within the same group are similiar;
  - data from different groups are dissimiliar.

In [5]:

```python
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0) #call k-means clustering algori
y_km = kmeans.fit_predict(X)
y_km # the groups assigned by algorithm
```

Out[5]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

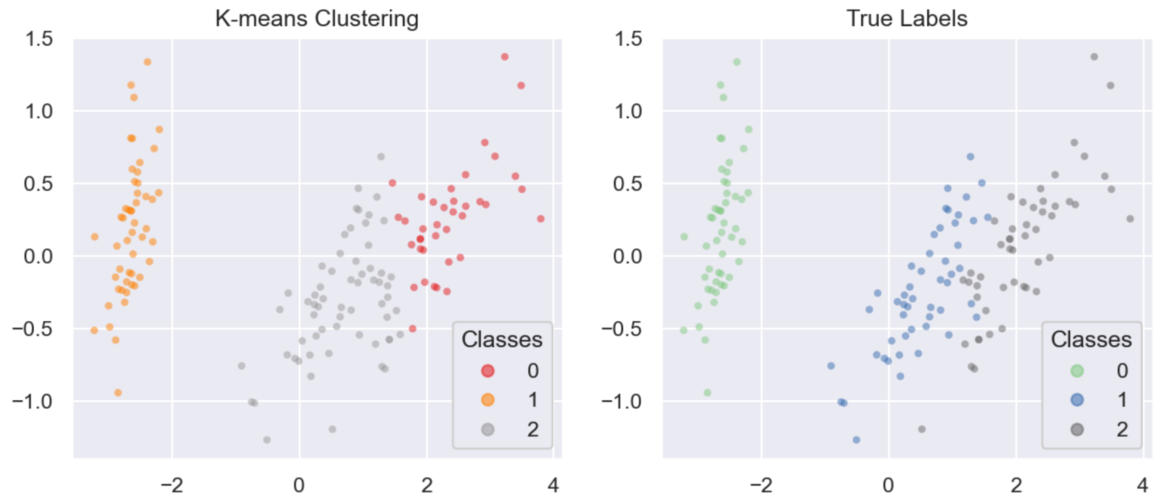
In [6]:

```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
fig, (ax1, ax2) = plt.subplots(1, 2,dpi=150, figsize=(10,4))

fig1 = ax1.scatter(X_pca[:, 0], X_pca[:, 1],c=y_km, s=15, edgecolor='none', a
fig2 = ax2.scatter(X_pca[:, 0], X_pca[:, 1],c=y, s=15, edgecolor='none', alph
ax1.set_title('K-means Clustering')
legend1 = ax1.legend(*fig1.legend_elements(), loc="best", title="Classes")
ax1.add_artist(legend1)
ax2.set_title('True Labels')
legend2 = ax2.legend(*fig2.legend_elements(), loc="best", title="Classes")
ax2.add_artist(legend2)
```

Out[6]:

```
<matplotlib.legend.Legend at 0x2906925b310>
```



Question: What is the difference between clustering and classification? Can you try classification on Iris data with pycaret right now?

In [ ]:

```
# try classification with pycaret for Iris data by yourself!
```