Lecture 1 Introduction to Data Science

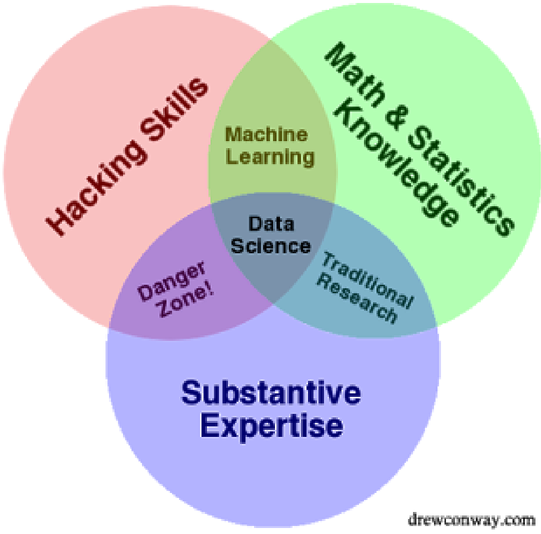
What is Data Science?

Three correlated concepts:

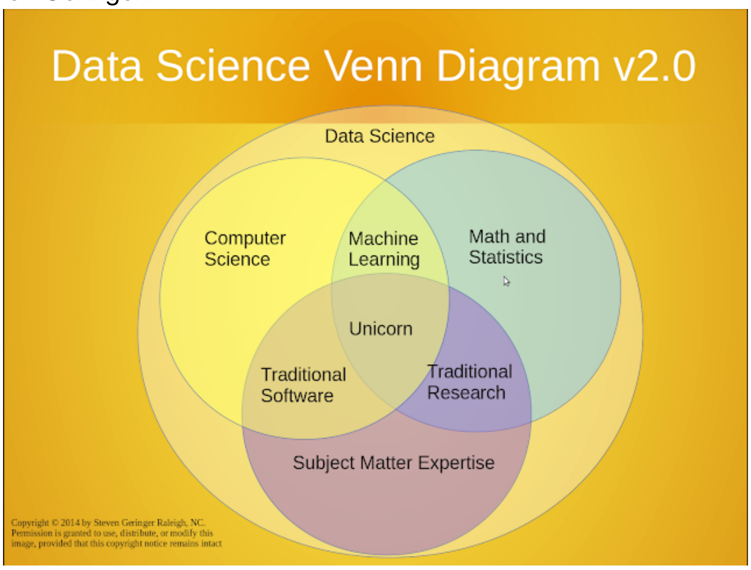
- Data Science
- Artificial Intelligence
- Machine Learning

[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)

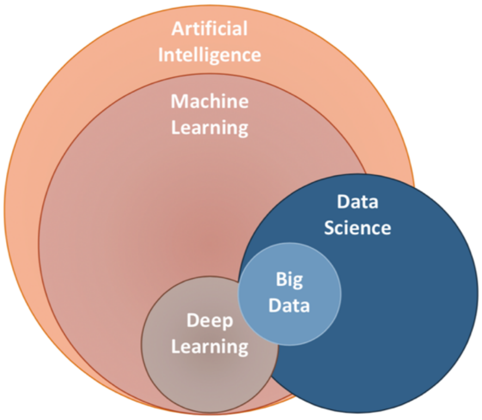
The original Venn diagram from Drew Conway:



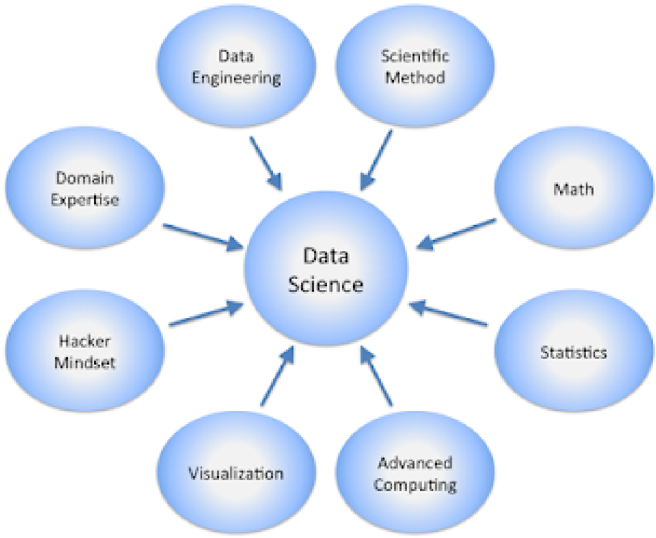
Another diagram from Steven Geringer:

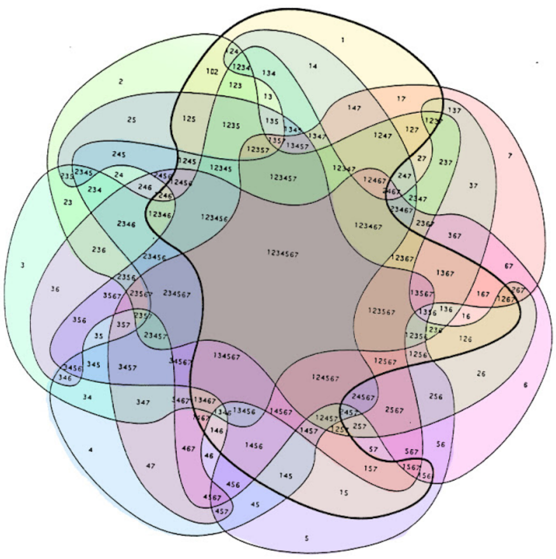


Another version:



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?

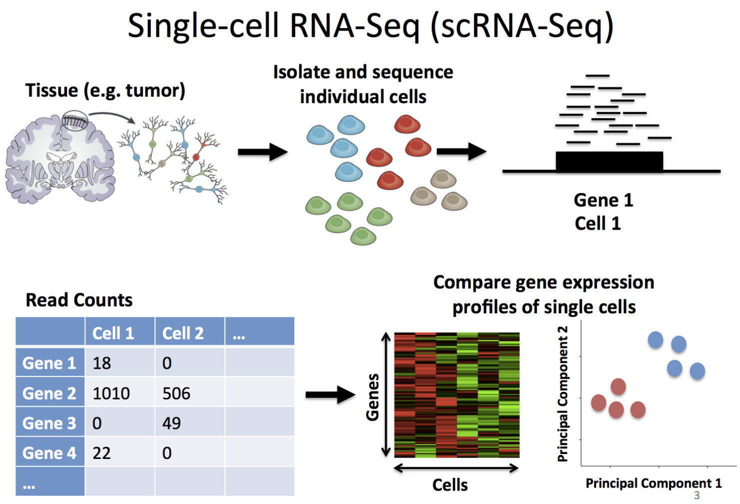
Tabular Data

In [ ]:
```python
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()
```

- 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

$$X = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \cdots \\ \mathbf{x}^{(n)} \end{pmatrix}, \text{ where the } i\text{-th row vector represents } i\text{-th observation, } \mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_p^{(i)}) \in \mathbb{R}^p.$$

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



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

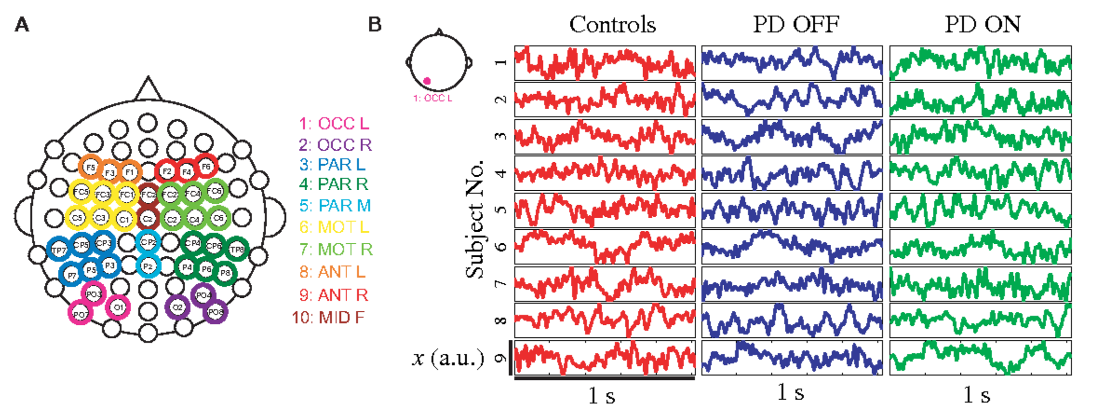
Time-series Data

In [ ]:
```python
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('Close Price History')
plt.plot(ts_tesla['Close'], color='red')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD', fontsize = 18)
plt.show()
# this is only about tesla -- we can also have the time-series of apple,amazon,facebook.
```

- 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

$$X = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \cdots \\ \mathbf{x}^{(N)} \end{pmatrix}$$, where the i-th row vector represents i-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/)](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3858815/).



Images

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

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

Out[5]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixe
10926	2	0	0	0	0	0	0	0	0	0	0	0	0	0	
26965	5	0	0	0	0	0	0	0	0	0	0	0	0	0	
38920	8	0	0	0	0	0	0	0	0	0	0	0	0	0	
10225	3	0	0	0	0	0	0	0	0	0	0	0	0	0	
14465	1	0	0	0	0	0	0	0	0	0	0	0	0	0	

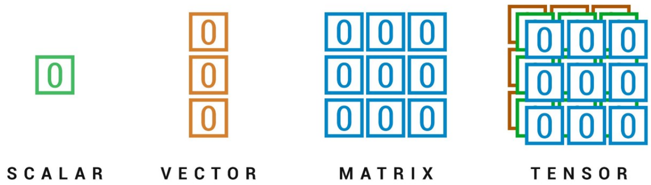
5 rows × 785 columns

In [ ]:
```
mnist.shape
```

In [ ]:
```
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 to 2d pixel
    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*.

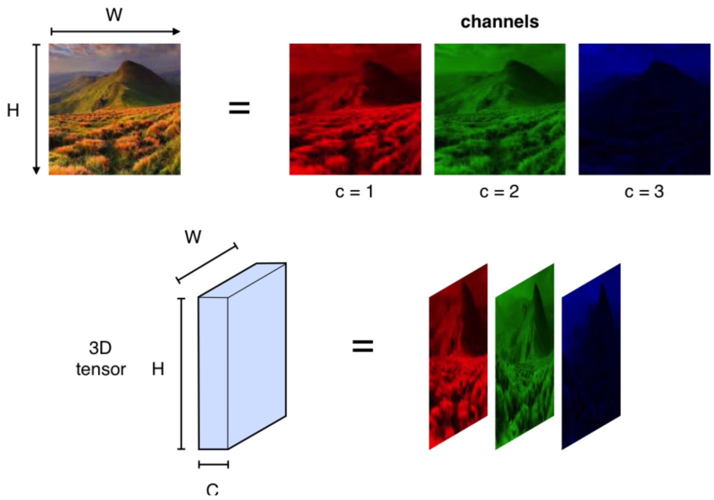
[Illustrated Introduction to Linear Algebra using NumPy (https://medium.com/@kaaanishk/illustrated-introduction-to-linear-algebra-using-numpy-11d503d244a1)](https://medium.com/@kaaanishk/illustrated-introduction-to-linear-algebra-using-numpy-11d503d244a1)



- **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)](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 [ ]:
```
mnist.head()
```

Videos

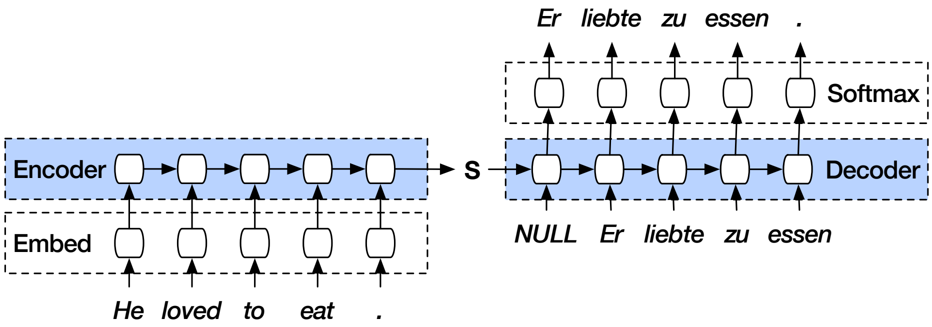
- *Time-series* of images, or *random field* model $z = \mathbf{f}(\omega, x, y, t)$

Texts

In [ ]:
```
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', 'bad', 'stud
                             stop_words=frozenset(), token_pattern=r"(?u)\b\w+\b")
X = vectorizer.fit_transform(df['sentences'].values)
result = pd.DataFrame(data=X.toarray(), columns=vectorizer.get_feature_names())
result.head()
```

- **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";
- Reinforment Learning -- "learning by doing".

Our course will focus on the first two categories.

Supervised Learning

- Given the *training dataset* $(x^{(i)}, y^{(i)})$ with $y^{(i)} \in \mathbb{R}^q$ denotes the *labels*, the supervised learning aims to find a mapping $\mathbf{f} : \mathbb{R}^p \to \mathbb{R}^q$ such that $y^{(i)} \approx \mathbf{f}(x^{(i)})$. Then with a new observation $x^{(new)}$, we can predict that $y^{(new)} = \mathbf{f}(x^{(new)})$.
  - 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 $\mathbf{f}(\mathbf{x}; \theta)$. Then define the loss function of θ

$$L(\theta) = \sum_{i=1}^{n} \ell(y^{(i)}, \mathbf{f}(x^{(i)})),$$

  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 $\ell(y^{(i)}, \mathbf{f}(x^{(i)})) = ||y^{(i)} - \mathbf{f}(x^{(i)})||^2$. We then seek to choose the optimal θ that minimizes the loss function

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ L(\theta),$$

  which can be tacked numeracally by optimzation methods (including the popular stochastic gradient descent).

- Difference choice of $\mathbf{f}(\mathbf{x}; \theta)$ leads to various supervised learning models:
  - Linear function : Linear Regression (for regression)/Logistic Regression (for classification)
  - 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)](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))](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)) and low-code ML package [pycaret (https://pycaret.org/)](https://pycaret.org/).

In [ ]:
```
pip install pycaret #install pycaret -- it's a new package, not coming with Anaconda
```

In [2]:
```
from sklearn.datasets import load_breast_cancer # load the dataset
X,y = load_breast_cancer(as_frame = True,return_X_y = True)
```

In [ ]:
```
X
```

In [ ]:
```
y
```

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

In [3]:
```
from sklearn.model_selection import train_test_split # manually split into train and tes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0
```

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

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

In [6]:
```
data_train = pd.concat([X_train,y_train],axis=1) # the whole data table of training
data_train
```

128	15.100	16.39	99.58	674.5	0.11500	0.18070	0.113800	0.085340	0.2001	0.06467	0.4309
28	15.300	25.27	102.40	732.4	0.10820	0.16970	0.168300	0.087510	0.1926	0.06540	0.4390
183	11.410	14.92	73.53	402.0	0.09059	0.08155	0.061810	0.023610	0.1167	0.06217	0.3344
459	9.755	28.20	61.68	290.9	0.07984	0.04626	0.015410	0.010430	0.1621	0.05952	0.1781
510	11.740	14.69	76.31	426.0	0.08099	0.09661	0.067260	0.026390	0.1499	0.06758	0.1924
151	8.219	20.70	53.27	203.9	0.09405	0.13050	0.132100	0.021680	0.2222	0.08261	0.1935
244	19.400	23.50	129.10	1155.0	0.10270	0.15580	0.204900	0.088860	0.1978	0.06000	0.5243
543	13.210	28.06	84.88	538.4	0.08671	0.06877	0.029870	0.032750	0.1628	0.05781	0.2351
544	13.870	20.70	89.77	584.8	0.09578	0.10180	0.036880	0.023690	0.1620	0.06688	0.2720
265	20.730	31.12	135.70	1419.0	0.09469	0.11430	0.136700	0.086460	0.1769	0.05674	1.1720

In [7]:
```
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 want to pr
```

	Description	Value
0	session_id	5279
1	Target	target
2	Target Type	Binary
3	Label Encoded	0: 0, 1: 1
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, 28)
12	Transformed Test Set	(115, 28)
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	bfab
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
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

	Description	Value
39	Numeric Binning	False
40	Remove Outliers	False
41	Outliers Threshold	None
42	Remove Multicollinearity	False
43	Multicollinearity Threshold	None
44	Clustering	False
45	Clustering Iteration	None
46	Polynomial Features	False
47	Polynomial Degree	None
48	Trignometry Features	False
49	Polynomial Threshold	None
50	Group Features	False
51	Feature Selection	False
52	Features Selection Threshold	None
53	Feature Interaction	False
54	Feature Ratio	False
55	Interaction Threshold	None
56	Fix Imbalance	False
57	Fix Imbalance Method	SMOTE

In [8]:
```
ML models for you, and compare their performance on the training dataset with cross-valid
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lda	Linear Discriminant Analysis	0.9625	0.9890	0.9941	0.9513	0.9714	0.9173	0.9225	0.0100
ada	Ada Boost Classifier	0.9624	0.9892	0.9824	0.9598	0.9704	0.9189	0.9213	0.0380
rf	Random Forest Classifier	0.9551	0.9952	0.9647	0.9654	0.9640	0.9044	0.9076	0.1950
xgboost	Extreme Gradient Boosting	0.9551	0.9851	0.9647	0.9650	0.9636	0.9051	0.9089	0.1100
ridge	Ridge Classifier	0.9550	0.0000	0.9882	0.9437	0.9648	0.9023	0.9065	0.0090
qda	Quadratic Discriminant Analysis	0.9550	0.9885	0.9706	0.9590	0.9638	0.9043	0.9075	0.0090
et	Extra Trees Classifier	0.9514	0.9944	0.9643	0.9598	0.9607	0.8968	0.9006	0.1750
catboost	CatBoost Classifier	0.9514	0.9916	0.9647	0.9592	0.9611	0.8964	0.8989	2.7280
lightgbm	Light Gradient Boosting Machine	0.9476	0.9886	0.9585	0.9584	0.9577	0.8887	0.8910	0.1560
lr	Logistic Regression	0.9399	0.9872	0.9699	0.9399	0.9530	0.8694	0.8763	0.4130
nb	Naive Bayes	0.9399	0.9929	0.9643	0.9414	0.9522	0.8712	0.8736	0.0080
gbc	Gradient Boosting Classifier	0.9289	0.9800	0.9467	0.9410	0.9421	0.8499	0.8545	0.0530
dt	Decision Tree Classifier	0.9177	0.9105	0.9401	0.9312	0.9328	0.8258	0.8342	0.0080
knn	K Neighbors Classifier	0.9057	0.9557	0.9397	0.9156	0.9261	0.7958	0.8006	0.0470
svm	SVM - Linear Kernel	0.7419	0.0000	0.6658	0.8253	0.7032	0.5149	0.5602	0.0080

In [9]:
```
best # the best model selected by pycaret
```

Out[9]:
```
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                           solver='svd', store_covariance=False, tol=0.0001)
```

In [20]:
```
redict_model(best); # predict on the validation data that pycaret have selected -- sample
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Linear Discriminant Analysis	0.9391	0.9754	1.0000	0.9103	0.9530	0.8671	0.8749

In [12]:
```
from pycaret.classification import finalize_model
best_final = finalize_model(best) # re-train the dataset with whole input training data
```

In [13]:
```
from pycaret.classification import predict_model
predictions = predict_model(best_final, data = X_test) # make new predictions on new-com
predictions
```

Out[13]:

mean ymmetry	mean fractal dimension	...	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	Labe
0.2116	0.07325	...	113.30	844.4	0.15740	0.38560	0.51060	0.20510	0.3585	0.11090	(
0.1619	0.05584	...	91.29	632.9	0.12890	0.10630	0.13900	0.06005	0.2444	0.06788	1
0.1589	0.05586	...	96.53	688.9	0.10340	0.10170	0.06260	0.08216	0.2136	0.06710	1
0.1635	0.05586	...	105.80	819.7	0.09445	0.21670	0.15650	0.07530	0.2636	0.07676	1
0.1467	0.05863	...	84.46	545.9	0.09701	0.04619	0.04833	0.05013	0.1987	0.06169	1
...	...	...	...	...	...	...	...	...	...	...	..
0.1609	0.05871	...	108.60	906.5	0.12650	0.19430	0.31690	0.11840	0.2651	0.07397	(
0.1652	0.07238	...	87.38	576.0	0.11420	0.19750	0.14500	0.05850	0.2432	0.10090	1
0.1695	0.06556	...	70.10	362.7	0.11430	0.08614	0.04158	0.03125	0.2227	0.06777	1
0.1521	0.05912	...	114.20	880.8	0.12200	0.20090	0.21510	0.12510	0.3109	0.08187	1
0.1943	0.06612	...	86.67	552.0	0.15800	0.17510	0.18890	0.08411	0.3155	0.07538	1

In [14]:
```
df_compare = pd.concat([predictions['Label'],y_test],axis = 1) # compare with the ground
df_compare
```

Out[14]:

	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 [15]:
```
import numpy as np
np.mean(predictions['Label'].to_numpy() == y_test.to_numpy()) # calculate the percentage
```

Out[15]: 0.973404255319149

In [17]:
```python
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	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	0.9259	1.0000	1.0000	0.8947	0.9444	0.8344	0.8460
2	0.9259	0.9941	0.9412	0.9412	0.9412	0.8412	0.8412
3	0.8889	0.9588	0.8824	0.9375	0.9091	0.7666	0.7689
4	0.9259	0.9882	1.0000	0.8947	0.9444	0.8344	0.8460
5	0.9630	1.0000	0.9375	1.0000	0.9677	0.9244	0.9270
6	0.8846	0.9438	1.0000	0.8421	0.9143	0.7417	0.7678
7	0.9231	0.9938	1.0000	0.8889	0.9412	0.8312	0.8433
8	0.9615	0.9938	0.9375	1.0000	0.9677	0.9202	0.9232
9	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Mean	0.9399	0.9872	0.9699	0.9399	0.9530	0.8694	0.8763
SD	0.0384	0.0187	0.0400	0.0554	0.0295	0.0844	0.0793

In [19]:
```python
predict_model(lr) # validation dataset -- sample exam!
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.9478	0.9936	0.9437	0.9710	0.9571	0.8905	0.8911

Out[19]:

mean fractal limension	radius error	...	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target	Labe
0.06110	0.2273	...	527.200012	0.11440	0.08906	0.09203	0.06296	0.2785	0.07408	1	1
0.05223	0.5858	...	1210.000000	0.11110	0.14860	0.19320	0.10960	0.3275	0.06469	0	(
0.06612	0.2560	...	507.200012	0.09457	0.33990	0.32180	0.08750	0.2305	0.09952	1	1
0.05504	1.2140	...	698.799988	0.09387	0.05131	0.02398	0.02899	0.1565	0.05504	0	(
0.06284	0.4768	...	1025.000000	0.15510	0.42030	0.52030	0.21150	0.2834	0.08234	0	(
...	...	...	...	...	...	...	...	...	...	...	..
0.05660	0.3242	...	683.400024	0.12780	0.12910	0.15330	0.09222	0.2530	0.06510	1	1
0.06673	0.9806	...	1610.000000	0.14780	0.56340	0.37860	0.21020	0.3751	0.11080	0	(
0.06582	0.2315	...	512.500000	0.14310	0.18510	0.19220	0.08449	0.2772	0.08756	1	1
0.05636	0.4204	...	808.900024	0.13060	0.19760	0.33490	0.12250	0.3020	0.06846	0	(
0.06697	0.7923	...	1600.000000	0.14120	0.30890	0.35330	0.16630	0.2510	0.09445	0	(

In [21]:
```python
final_lr = finalize_model(lr)
```

In [22]:
```python
predictions_lr = predict_model(final_lr, data = X_test)
np.mean(predictions_lr['Label'].to_numpy() == y_test.to_numpy())
```

Out[22]: 0.9627659574468085

In [24]:
```python
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	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	0.9630	1.0000	1.0000	0.9444	0.9714	0.9189	0.9220
2	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
3	0.8889	0.9824	0.8824	0.9375	0.9091	0.7666	0.7689
4	0.9259	0.9882	1.0000	0.8947	0.9444	0.8344	0.8460
5	0.9630	1.0000	0.9375	1.0000	0.9677	0.9244	0.9270
6	0.9615	0.9312	1.0000	0.9412	0.9697	0.9172	0.9204
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	0.9231	0.9812	0.9375	0.9375	0.9375	0.8375	0.8375
9	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Mean	0.9625	0.9883	0.9757	0.9655	0.9700	0.9199	0.9222
SD	0.0373	0.0204	0.0397	0.0368	0.0300	0.0796	0.0779

In [25]:
```python
predict_model(tuned_lr) # still doing the sample exam -- validation dataset
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.9478	0.9923	0.9437	0.9710	0.9571	0.8905	0.8911

Out[25]:

	mean radius	mean texture	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radiu erro
0	11.940000	18.240000	437.600006	0.08261	0.04751	0.01972	0.01349	0.1868	0.06110	0.227
1	17.010000	20.260000	904.299988	0.08772	0.07304	0.06950	0.05390	0.2026	0.05223	0.585
2	11.870000	21.540001	432.000000	0.06613	0.10640	0.08777	0.02386	0.1349	0.06612	0.256
3	14.990000	25.200001	698.799988	0.09387	0.05131	0.02398	0.02899	0.1565	0.05504	1.214
4	15.060000	19.830000	705.599976	0.10390	0.15530	0.17000	0.08815	0.1855	0.06284	0.476
...	...	...	...	...	...	...	...	...	...	.
110	13.640000	15.600000	575.299988	0.09423	0.06630	0.04705	0.03731	0.1717	0.05660	0.324
111	18.049999	16.150000	1006.000000	0.10650	0.21460	0.16840	0.10800	0.2152	0.06673	0.980
112	11.600000	12.840000	412.600006	0.08983	0.07525	0.04196	0.03350	0.1620	0.06582	0.231
113	14.480000	21.459999	648.200012	0.09444	0.09947	0.12040	0.04938	0.2075	0.05636	0.420
114	18.490000	17.520000	1068.000000	0.10120	0.13170	0.14910	0.09183	0.1832	0.06697	0.792

115 rows × 31 columns

In [26]:
```python
final_tuned_lr = finalize_model(tuned_lr) #retrain with the whole dataset
```

In [27]:
```python
predictions_tuned_lr = predict_model(final_tuned_lr, data = X_test)
np.mean(predictions_tuned_lr['Label'].to_numpy() == y_test.to_numpy())
```

Out[27]: `0.9627659574468085`

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

In [ ]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2) # principle component analysis, ruduce 4-dimenional data to 2-
X_pca = pca.fit_transform(X)
X_pca
```

In [ ]:
```
rt matplotlib.pyplot as plt
rt seaborn as sns
set() # set the seaborn theme style
re = plt.figure(dpi=100)
scatter(X_pca[:, 0], X_pca[:, 1],c=y, s=15, edgecolor='none', alpha=0.5,cmap=plt.cm.get_c
xlabel('PC 1')
ylabel('PC 2')
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 [ ]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0) #call k-means clustering algorithm
y_km = kmeans.fit_predict(X)
y_km # the groups assigned by algorithm
```

In [ ]:
```
ib.pyplot as plt
as sns; sns.set()
 = plt.subplots(1, 2,dpi=150, figsize=(10,4))

ter(X_pca[:, 0], X_pca[:, 1],c=y_km, s=15, edgecolor='none', alpha=0.5,cmap=plt.cm.get_cm
ter(X_pca[:, 0], X_pca[:, 1],c=y, s=15, edgecolor='none', alpha=0.5,cmap=plt.cm.get_cmap(
K-means Clustering')
egend(*fig1.legend_elements(), loc="best", title="Classes")
legend1)
True Labels')
egend(*fig2.legend_elements(), loc="best", title="Classes")
legend2)
```

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!
```