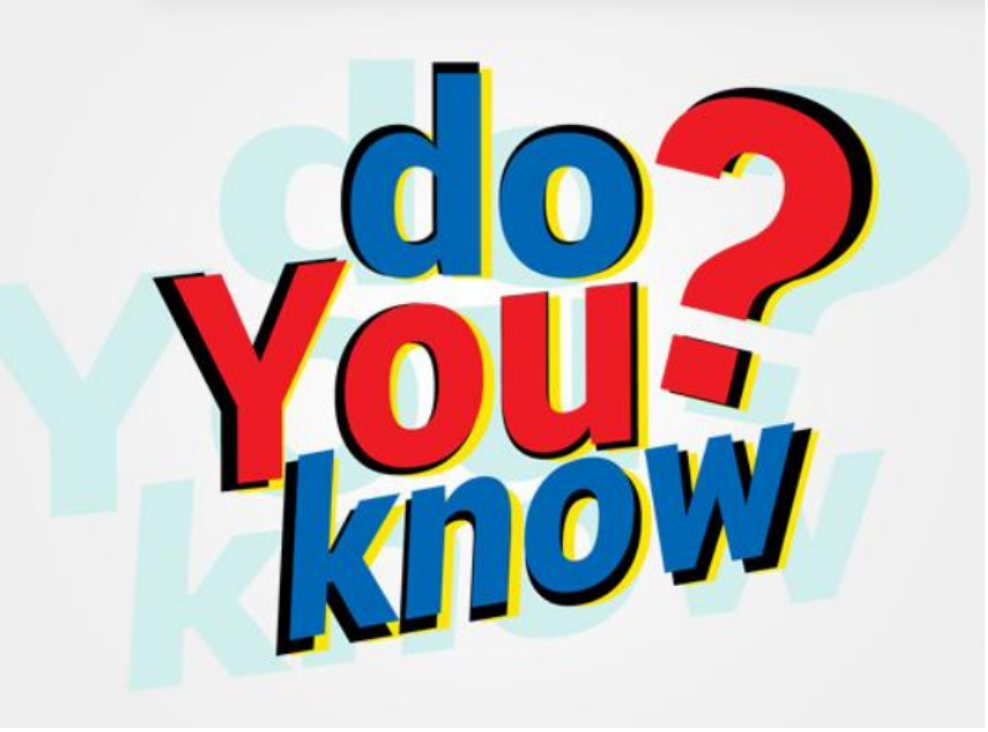
# Machine Learning Python Programming Training

## Kevin Lee, Genpact

## Agenda

- Introduction of Machine Learning
- Machine Learning current implementation and implementation
- Machine Learning Concepts – Hypothesis function, Cost function, Gradient Descent, Learning Rate
- Machine Learning Process / Workflow
- Machine Learning Types – Supervised and Unsupervised
- Machine Learning Algorithms: SVM, KNN, Decision Tree, Random Forest, Logistic Regression, Linear Regression, K-means Clustering
- Artificial Neural Network (ANN)
- Activation Functions
- Loss function
- Optimizer
- Deep Neural Network (DNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transfer Learning

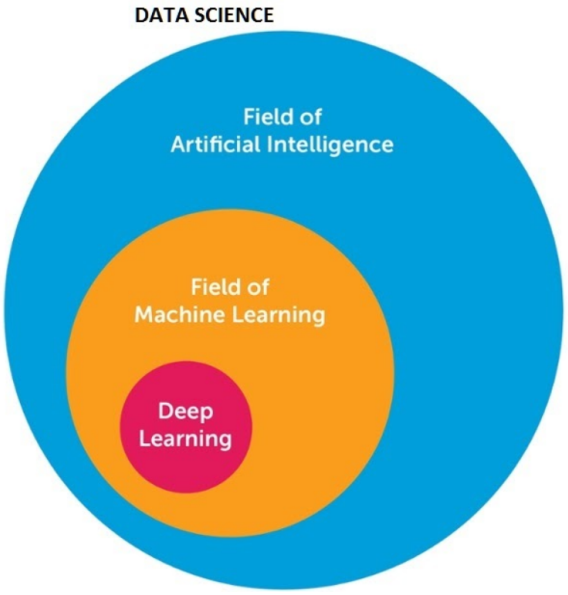
## Honey, do you know about Machine Learning?




## Many peopole around us expects us to know about Machine Learning. Why?

- Programming
- Statistics / Modeling
- Working with data all the times

## Big Picture of Data Science, AI, Machine Learrning, Deep Learning



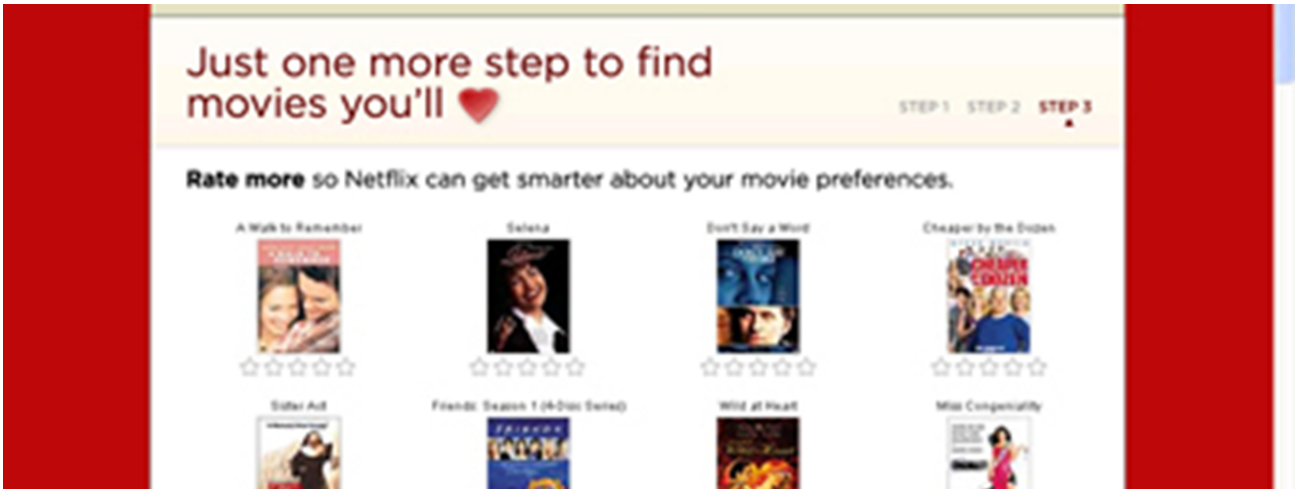

```
## Left aligned image
<img src="attachment:image.png" width="400" align="left">
```

- Data Science : a field of pertaining to data systems and processes, aimed at maintaining data sets and deriving meaning out of data.
- AI (Artificial Intelligence) : a filed of enabling machines to execute reasoning by replicating human intelligence.
- Machine Learning : a subset of AI that Machine/system can automatically learn and improve from experience.
- Deep Learning : a subset of Machine Learning that system uses artifical neural network to learn.

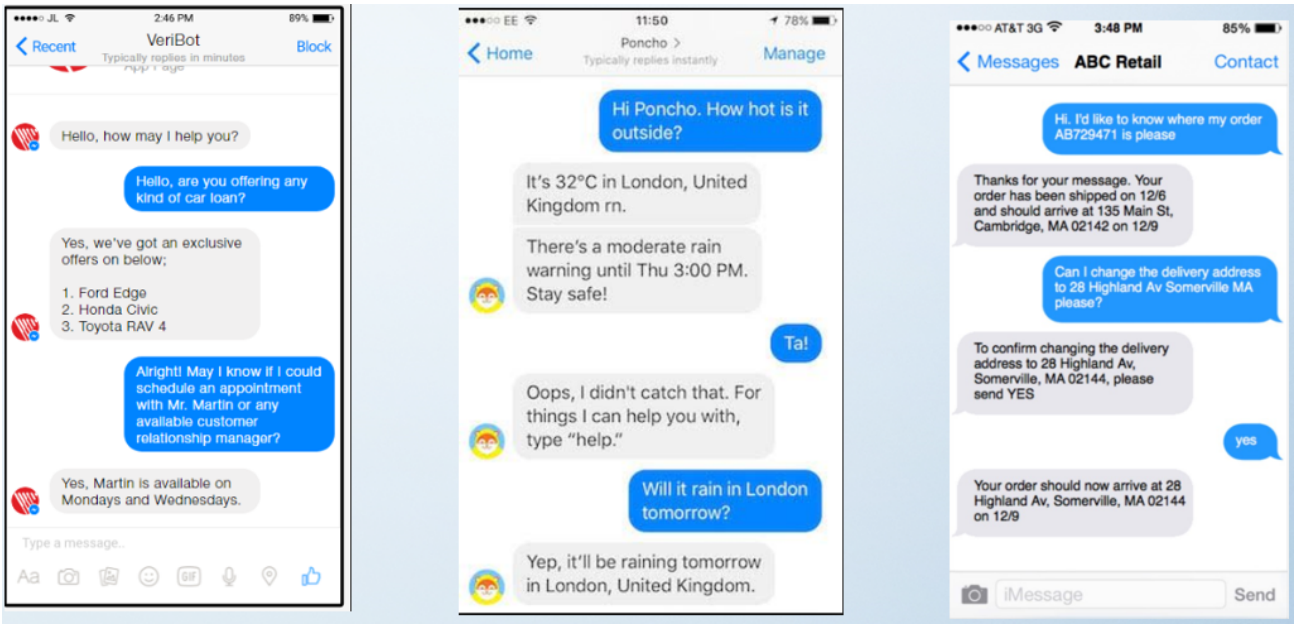
## Current AI Implementation

### Voice Regnition

### Recommendation - Amazon, Netflix, Spotify



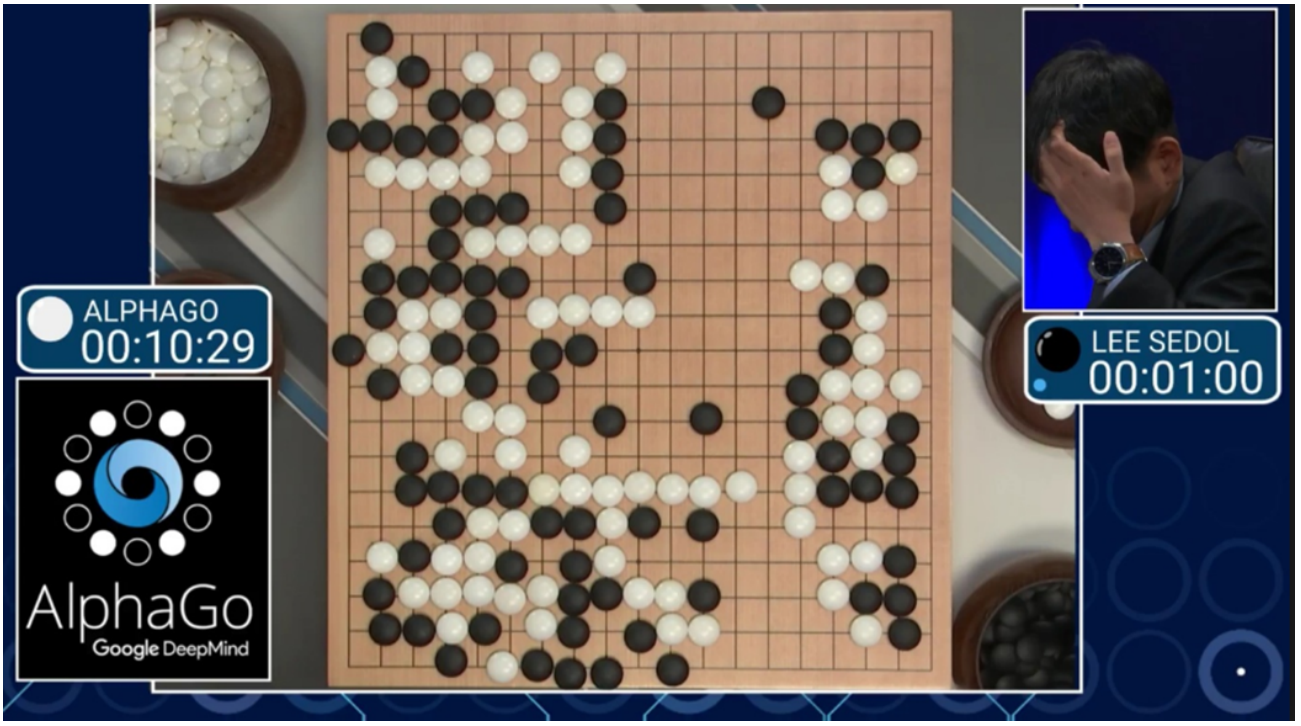

**Chatbot (Online Customer Service)**




**AI Assistant (AI System for accomplishing Real-World Tasks over the phone)**

**Amazon Go (Casher less Grocery Store)**

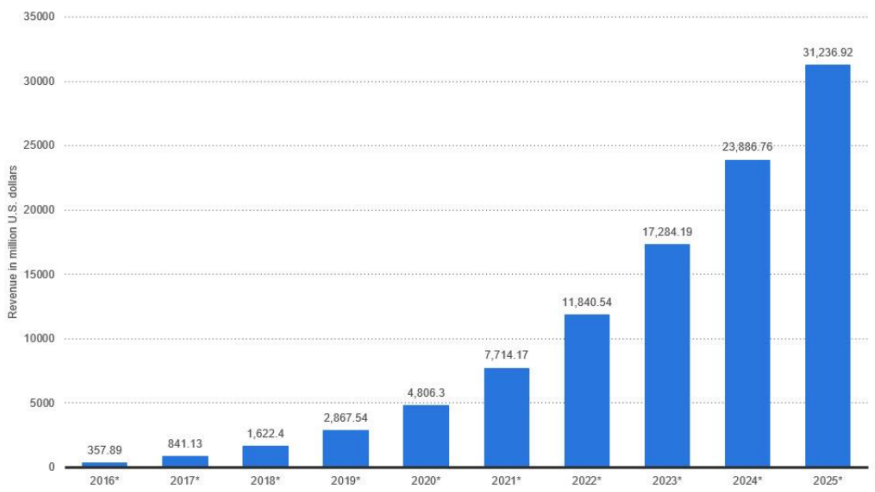
**AlphaGO – Beat "Go" World Champion**



### Why is AI(ML) so popular now?

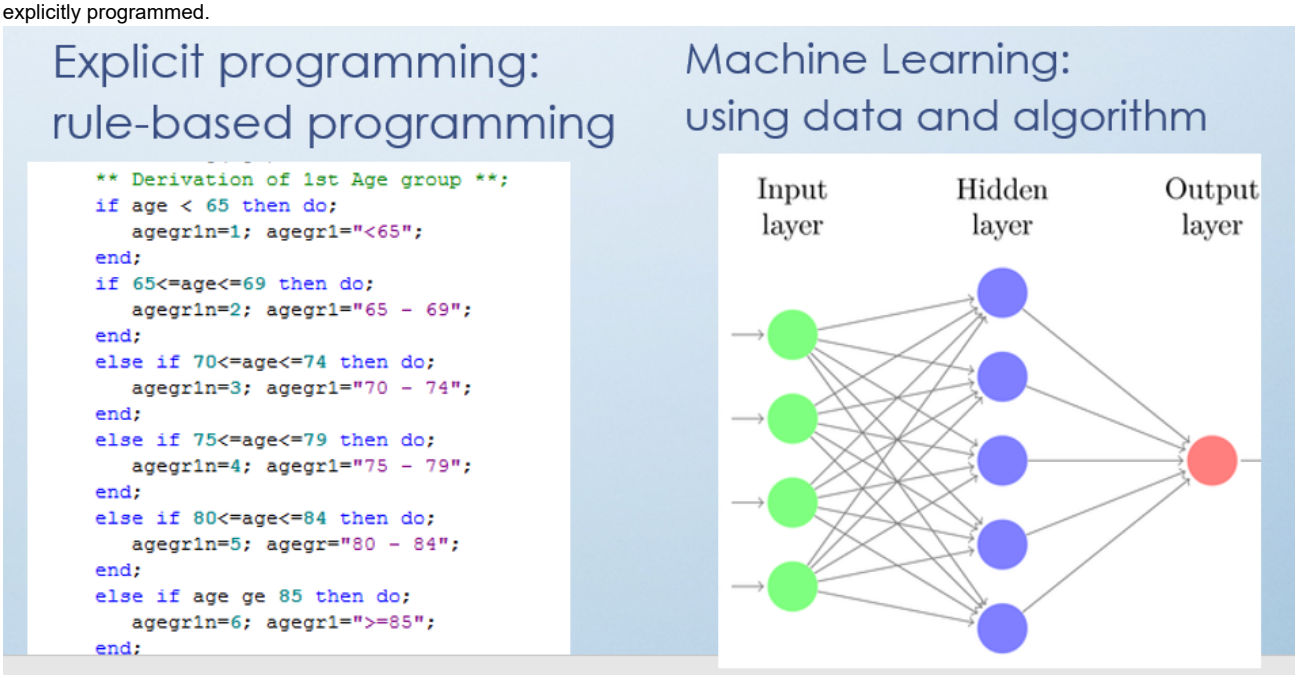
- Innovative
  - Can solve many complex business problems that were not resolved before
- New business development
  - Cost effective
- Automate a lot of works
  - Can replace or enhance human labors
  - “Pretty much anything that a normal person can do in <1 sec, we can now automate with AI” Andrew Ng
- Accurate
  - Better than humans in some applications

### AI Market Expected Growth




## Machine Learning Introduction

An application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.



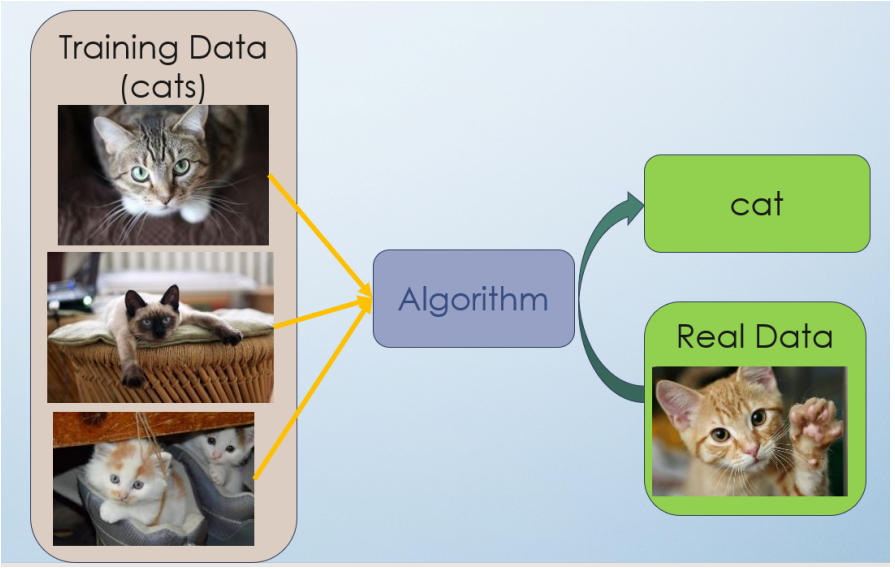

## What is it?

### How does human learn?

- Through Experiences

## How does Machine learn?

- Data
- Algorithm (Hypothesis Function)




## Machine Learning Functions

### Hypothesis function

- ML Algorithm/Model that we will train with data
- Linear Regression, Logistic Regression, Support Vector Machine, Decision Tree, Artificial Neural Network
- hθ(x) = θ0* x0 + θ1* x1 + θ2* x2 + θ3* x3 +,, + θn* xn (e.g., Y = 2x + 30)
- Parameters (ultimate goals of ML) : θ0, θ1, θ2 , θ3,,,θn

### Cost function (loss function)

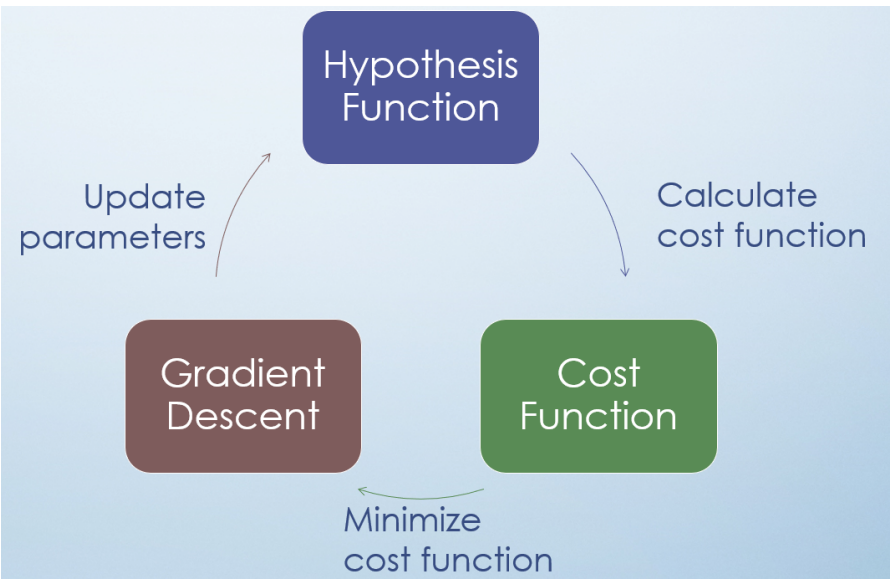
- To measure how well hypothesis function fits into data.
- Measured Difference between actual data point and hypothesized data point. (e.g., Y – hθ(x))
- J(θ0, θ1, θ2,,θn) = (1/2m)*sum[(Y - H)^2] for linear model

### Gradient Descent (optimizer)

- Engine that minimizes cost function
- Repeat
  - θ0 := θ0 – alpha* d/dθ0 * J(θ0, θ1, θ2,,θn)
  - θ1 := θ1 – alpha* d/dθ1 * J(θ0, θ1, θ2,,θn)
- Learning rate (alpha) – size of learning step in gradient descent

## How Machine Learning works

### Hypothesis function vs cost function vs gradient descent




## Teaching Young Childrens about Cat Pictures

1. Show first sets of pictures
2. Tell kids that some are “cat” pictures and some are NOT.
3. Explain why some are NOT “cat” pictures
4. Show next sets of pictures
5. Repeat 2-3
6. Keep showing CAT pictures using step 4 and 5 until kids understand “cat” pictures.

## Machine Learning Process

1. Select Data and hypothesis function
2. Start with random parameters of hypothesis function
3. Process first sets of data in hypothesis function
4. Find predicted values from hypothesis function
5. Calculate cost function ( actual – predicted)
6. Minimize cost function using Gradient Descent
7. Update parameters of hypothesis function based on cost function minimization
8. Repeat steps ( 3 and 7) with next sets of data until the cost function become 0 or at the lowest point.

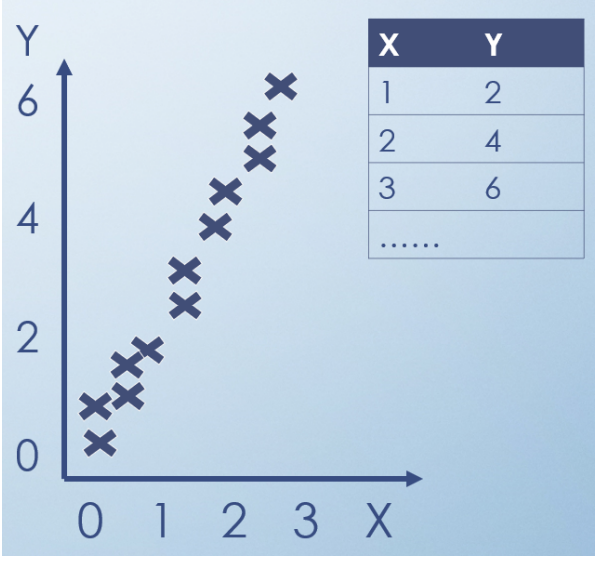
### 1. Select Data and Hypothesis Function

Data :

- Y = Survival Days in months
- X = number of drug exposures

Hypothesis function:

- Linear Regression ($h\theta(x) = \theta x$)




### 2. Start with random parameters of hypothesis function

hθ(x) = 4x at θ = 4

### 3 & 4 Process first set of data in hypothesis function and predict values from hypothesis function

hθ(x) = 4x at θ = 4

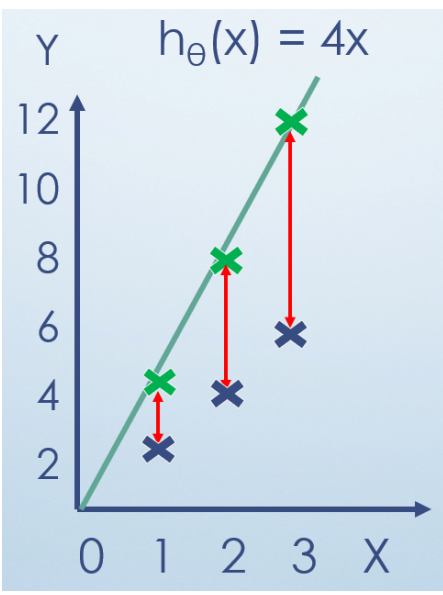
x = [1,2,3]

- at x = 1, y = 4
- at x = 2, y = 8
- at x = 3, y = 12

### 5. Calculate cost function : difference between actual and predicted

Cost function : J(θ)=(1/2m)*sum[(Y - H)^2]

J(4) =(1/6)*sum((2-4)^2 + (4-8)^2 + (6-12)^2) = 56/6 = 9.333

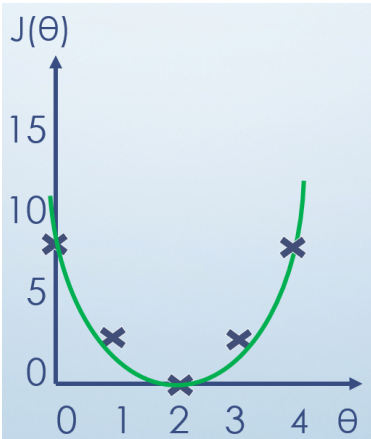



**Cost Function in different Parameter**

- J(0) =56/6 = 9.333

- J(1) =14/6 = 2.333
- J(2) =0/6 = 0
- J(3) =14/6 = 2.333
- J(4) =56/6 = 9.333

Optimum θ is 2 : Best fitted model is H = 2X.

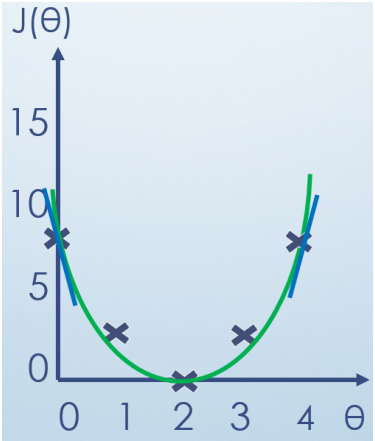



**Gradient Descent**

- Definition – Engine that minimize cost function.
- The process of getting to the lowest error value (optimum cost function)
- It is like walking down in the mountain to find the gold located in valley.

## 6. Minimize cost function using Gradient Descent

- Take the derivative of cost function since it will give the direction to move forward : d/dθ J(θ)
- Make steps down the cost function : θ := θ – d/dθ J(θ)
- The size of steps is determined by learning rate : θ := θ – alpha* d/dθ J(θ)




**Learning Rate ( alpha)**

At θ:= θ – alpha* d/dθJ(θ) at θ =4 assuming d/dθJ(4) = 2.

If the learning rate (alpha) = 0.1,
θ:= 4 - 0.1* 2 = 3.8
If the learning rate (alpha) = 0.5
θ:= 4 – 0.5* 2 = 3

## 7. Update parameters of hypothesis function based on cost function

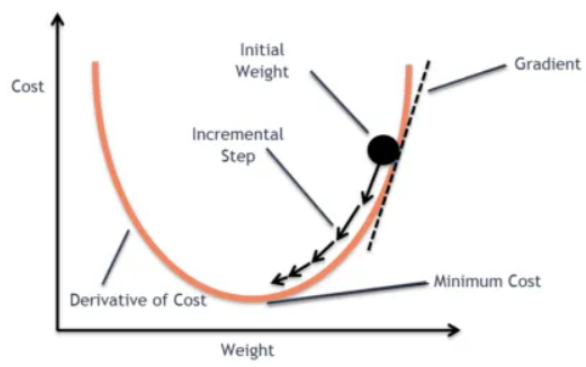
- Initial Hypothesis function - hθ(x) = 4x
- Cost function - J(θ) = (1/2m)*sum[(Y - h)^2] = 1/(*2*3)* sum(((2-4)^2 + (4-8)^2 + (6-12))^2)) = 56/6=9.333
- Gradient Descent – θ := θ – alpha* d/dθJ(4) = 4 – 0.1*2 = 3.8
- Modified Hypothesis function - hθ(x) = 3.8x

## 8.Repeat steps ( 3 and 7) with next sets of data

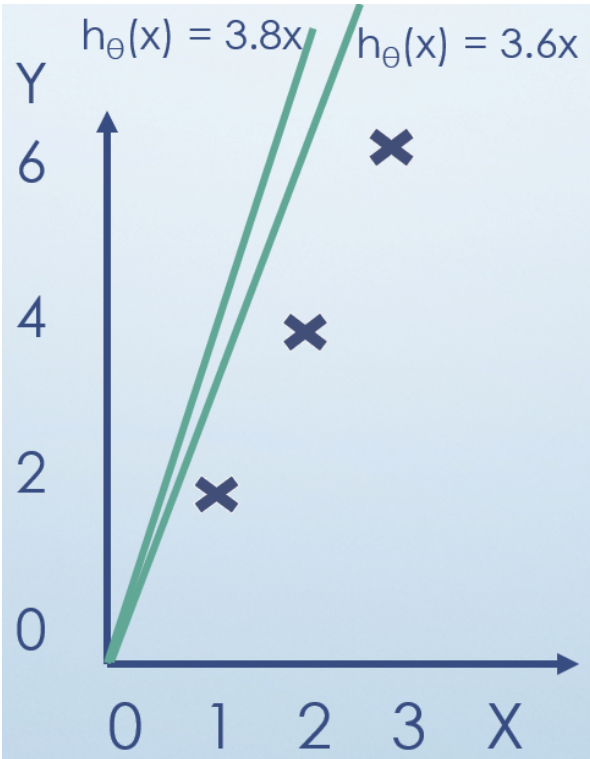
At second iteration,

- Initial Hypothesis function - hθ(x) = 3.8x
- Cost function - J(θ) = 1/(*23)* sum(((2-3.8)^2 + (4-7.6)^2 + (6-11.4)^2)) = 45.36/6 = 7.56
- Gradient Descent – θ := θ – alpha* d/dθ J(3.8) = 3.8 – 0.1*2 = 3.6
- Final Hypothesis function - hθ(x) = 3.6x

And the iteration will go on until the cost function gest to 0 like J(θ)=0 or it no longer decreases or no more data.




The cost function decreases and slope (θ) decreases as well to the optimum point.




## Machine Learning Types and Algorithms

Find the right job for right jobs/problems

### Machine Learning Types

Supervised

- Input data labeled : has correct answers
- Specific purpose
- Types
  - Classification for distinct output values
  - Regression for continuous output values

Unsupervised

- Input data not-labeled : No correct answers
- Exploratory
- Types : Clustering (clusters)

## Supervised Machine Learning

### Classification

- Categorical target

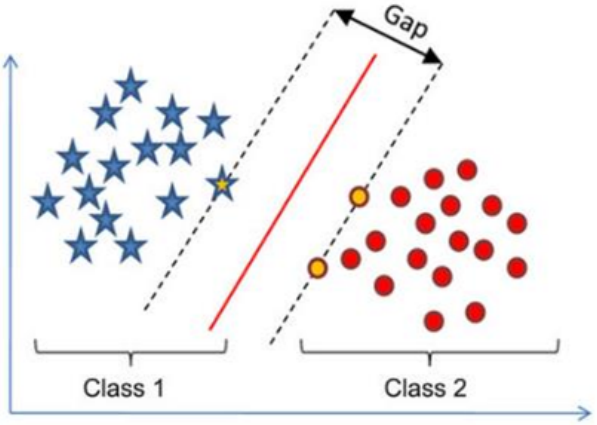
- Binary or Multiple Classification
- Example : Yes/No, 0 to 9, mild/moderate/severe
- Algorithms: SVM, K-nearest Neighbor, Naïve Bayes, Decision Tree, Random Forests, XGBoost, Logistic Regression

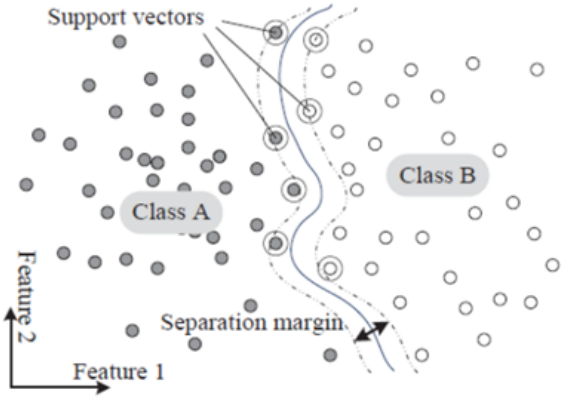
### Support Vector Machine (SVM)

- SVM is one of the most powerful classification model (both binary & multi), especially for complex, but small/mid-sized datasets.
- Best for binary classification
- Easy to train
- To find a line/ hyper-plane (in multidimensional space that separate outs classes) that maximizes margin between different classes.
  Pros:
- It works really well with clear margin of separation.
- It works well in high dimensional spaces.
- It works well where number of dimensions is greater than the number of samples.

Cons:

- It requires higher computation power, so it does not work well with larger data.
- It also doesn't perform very well when the data set has a lot of noises.




### Why Python for Machine Learning

Python is most popular / effective programming language for Machine Learning

Popular ML packages

- Scikit-Learn
  - simple and efficient ML tool, open-source
  - Classification, Regression, SVM, Clustering, Dimensionality Reduction, Decision Trees, Random Forests
- TensorFlow
  - developed by Google, open source since Nov.'2015
  - Artificial Neural Network, Convolutional NN, Recurrent NN, Autoencoder, Reinforcement Learning
- Keras

    - The official high-level API of TensorFlow
    - Easier and more friendly than TensorFlow
    - Developed by Google and contributed by MS, AWS
  - Pytorch
    - Dynamic ML model
    - Good with NLP

In [221]:
```
## import libraries
import numpy as np
import pandas as pd

## data visualization
import seaborn as sns
import matplotlib.pyplot as plt

## algorithm spliting train and test data
from sklearn.model_selection import train_test_split

## metric algorithm
from sklearn import metrics
```

In [222]:
```
## SVM Python Programming
### Import data
from sklearn import datasets
iris = datasets.load_iris()

X = iris["data"] # sepal length, sepal width, petal length, petal width
y = iris["target"] # 'setosa', 'versicolor', 'virginica'
```

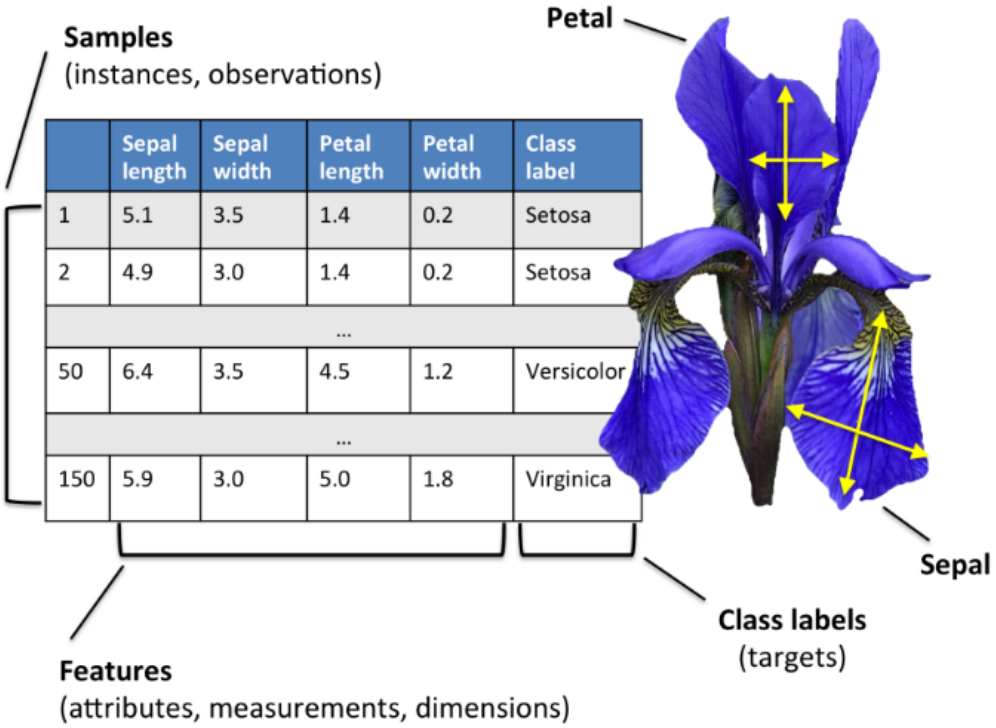
In [223]:
```
iris
```

Out[223]:
```
{'data': 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 [224]:
```
## Data Analysis
print("Shape of input data X   :", X.shape)
print("Shape of result(label) y:",y.shape)
print(X[[0, 1, 50, 149], :])
print(y[[0, 1, 50, 149]])
```

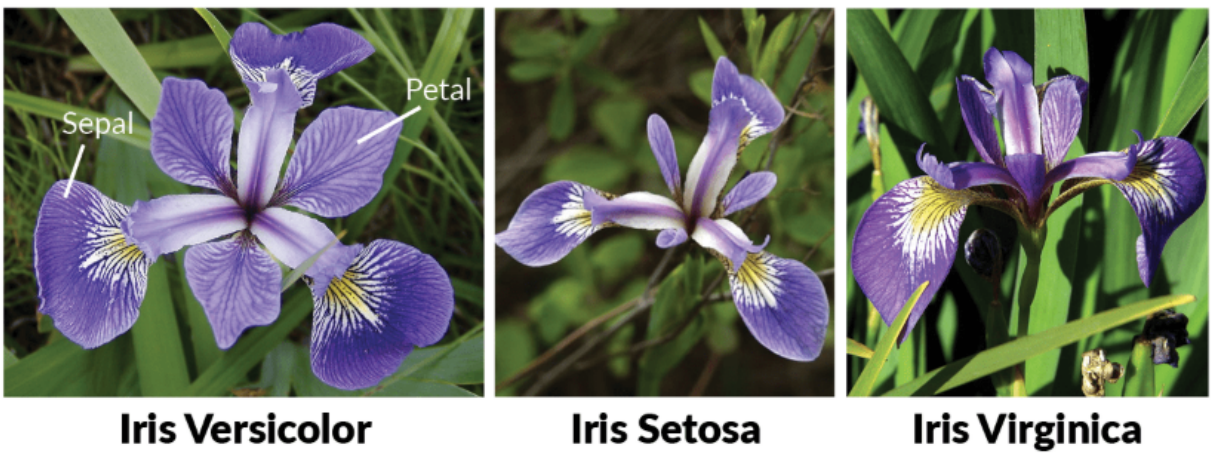
```
Shape of input data X   : (150, 4)
Shape of result(label) y: (150,)
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [7.  3.2 4.7 1.4]
 [5.9 3.  5.1 1.8]]
[0 0 1 2]
```

## Iris data description




| | Sepal length | Sepal width | Petal length | Petal width | Class label |
|---|---|---|---|---|---|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | Setosa |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 | Setosa |
| ... | | | | | |
| 50 | 6.4 | 3.5 | 4.5 | 1.2 | Versicolor |
| ... | | | | | |
| 150 | 5.9 | 3.0 | 5.0 | 1.8 | Virginica |

**Iris Data label: 0 = 'Setosa', 1 = 'Vesicolor', 2 = 'Virginica'**




In [225]:
```
## Set up dataframe for IRIS data
df_iris = pd.DataFrame(iris['data'], columns=iris['feature_names'])
df_iris.head()
```

Out[225]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) |
|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 |

In [226]:
```
## set up dataframe of input and target data
df_iris1 = pd.DataFrame(iris['data'], columns=iris['feature_names'])
df_iris2 = pd.DataFrame(iris['target'], columns=['target'])
print(df_iris1.head())
print(df_iris2.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
   target
0       0
1       0
2       0
3       0
4       0
```

In [227]:
```
## Converting factor preparation
num_to_char = {
    0: 'Setosa', 1:'Vesicolor', 2:'Virginica'
}
```

In [228]:
```
## joining by index number between
df_iris3 = df_iris1.join(df_iris2)
print(df_iris3.head())

## create new variable by converting target to species
df_iris3['species'] = df_iris3['target'].map(num_to_char)
df_iris3.head()
```

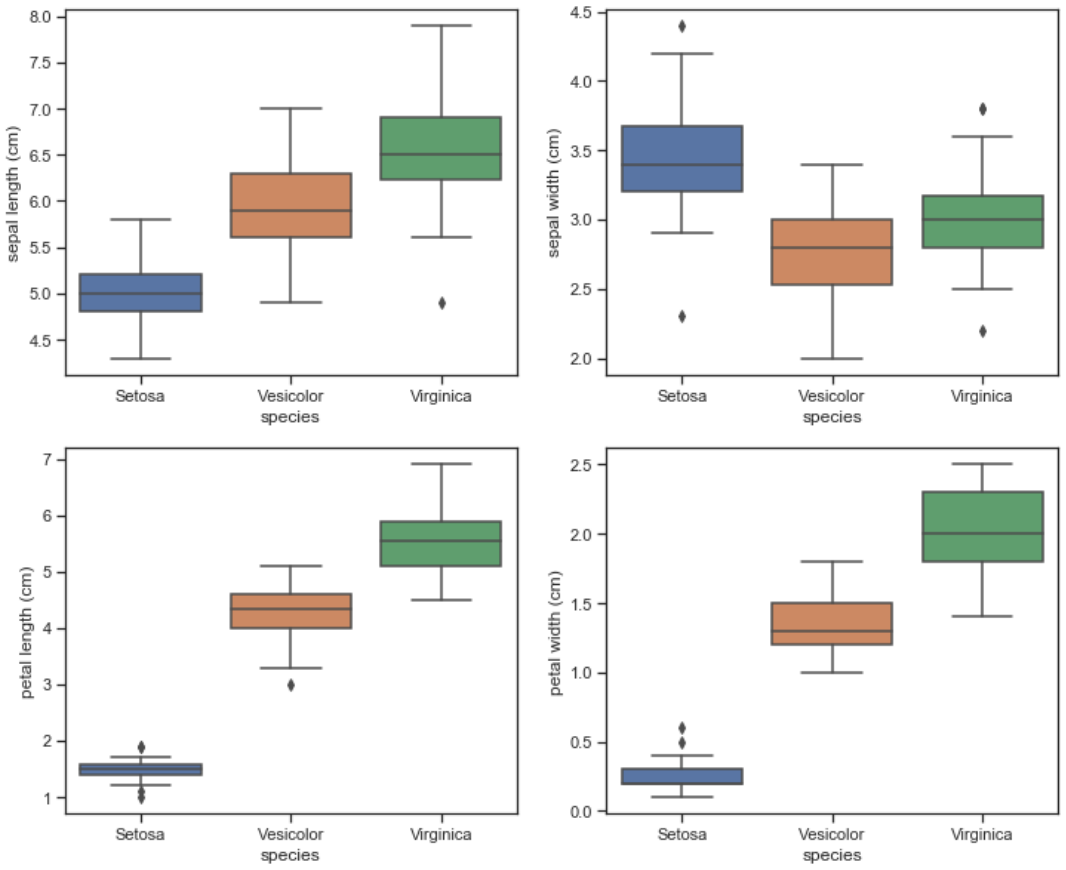
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

Out[228]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target | species |
|---|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 | Setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 | Setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 | Setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 | Setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 | Setosa |

In [229]:
```
## Data Visualization
sns.set(style="ticks")
plt.figure(figsize=(12,10))
plt.subplot(2,2,1)
sns.boxplot(x='species',y='sepal length (cm)',data=df_iris3)
plt.subplot(2,2,2)
sns.boxplot(x='species',y='sepal width (cm)',data=df_iris3)
plt.subplot(2,2,3)
sns.boxplot(x='species',y='petal length (cm)',data=df_iris3)
plt.subplot(2,2,4)
sns.boxplot(x='species',y='petal width (cm)',data=df_iris3)
plt.show()
```




In [230]:
```
## subsetting data 25 setosa, 25 versicolor and 25 virginica
X2 = np.concatenate((X[:25], X[50:75], X[100:125]), axis=0)
y2 = np.concatenate((y[:25], y[50:75], y[100:125]))
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2) ## splitting
```

In [231]:
```
## import modules
from sklearn.svm import SVC

## Pick up SVM Classifier model
svm_clf = SVC(kernel="linear", C=float("inf"))

## Train the model
svm_clf.fit(X_train, y_train)
```

Out[231]:
```
SVC(C=inf, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [232]:
```
## Predict using the trained SVM
svm_pred = svm_clf.predict((X_test))

print("Prection : ", svm_pred)
print("Test label: ", y_test)
```

```
Prection :  [1 1 1 2 2 0 1 0 0 2 0 2 1 0 2]
Test label:  [1 1 1 2 2 0 1 0 0 2 0 2 1 0 2]
```

In [233]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(svm_pred, y_test)
```

Out[233]:
```
1.0
```

In [234]:
```
print("Linear coefficient of SVM :", svm_clf.coef_)
print("Linear Intercept of SVM :", svm_clf.intercept_)
```

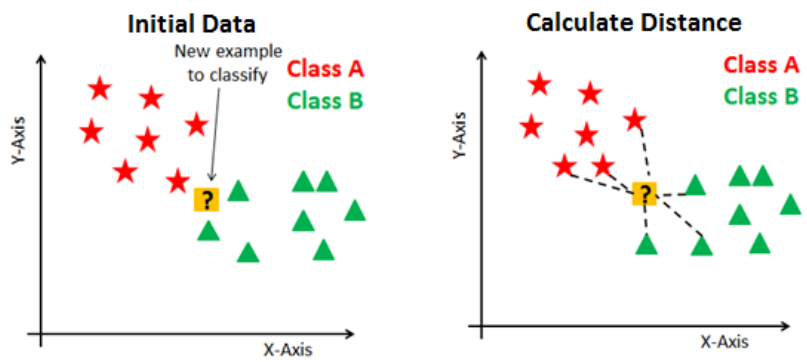
```
Linear coefficient of SVM : [[ 1.02119991e-02  5.37460967e-01 -8.27430707e-01 -3.81448839e-01]
 [-7.37511155e-03  1.78999904e-01 -5.38287977e-01 -2.92557588e-01]
 [ 1.26605136e+00  4.43154983e+00 -2.90569297e+00 -9.62159113e+00]]
Linear Intercept of SVM : [ 0.77190114  1.50820805 10.61541749]
```

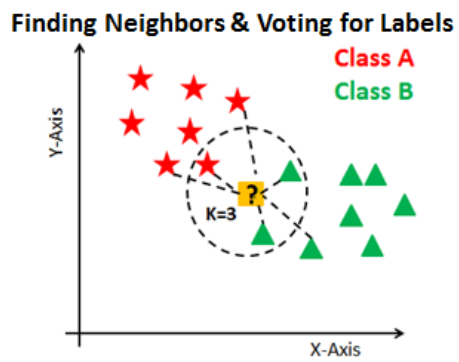
### KNN (K Nearest Neighbors)

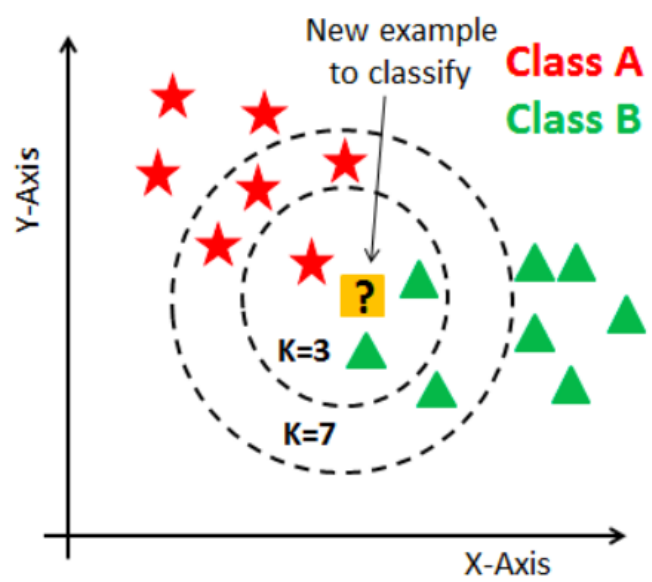
- A simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.
- Non-parametric & lazy algorithm
- KNN does not train KNN algorithm from training data, but memorize the location of training data itself.

How KNN works?

1. Calculate Distance between new data and training data
2. Find the class of closest neighbors (how many A and B in a given k)
3. Vote for labels which has most votes in a given k




Pros:

- The training phase is faster
- Simple and easy to implement
- Useful for a nonlinear data

Cons:

- Slower and costlier in terms of time and memory in prediction
- Large storage to store the entire training data

In [235]:
```
#import ML algorithm
from sklearn.neighbors import KNeighborsClassifier
```

In [236]:
```
#select and train model
#n_neighbors is the number of neighbor to use
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

Out[236]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [237]:
```
#predict output
knn_pred = knn.predict(X_test)

print("Prection from KNN : ", knn_pred)
print("Test label:         ", y_test)
```

```
Prection from KNN :  [1 1 1 2 2 0 1 0 0 2 0 2 1 0 2]
Test label:          [1 1 1 2 2 0 1 0 0 2 0 2 1 0 2]
```

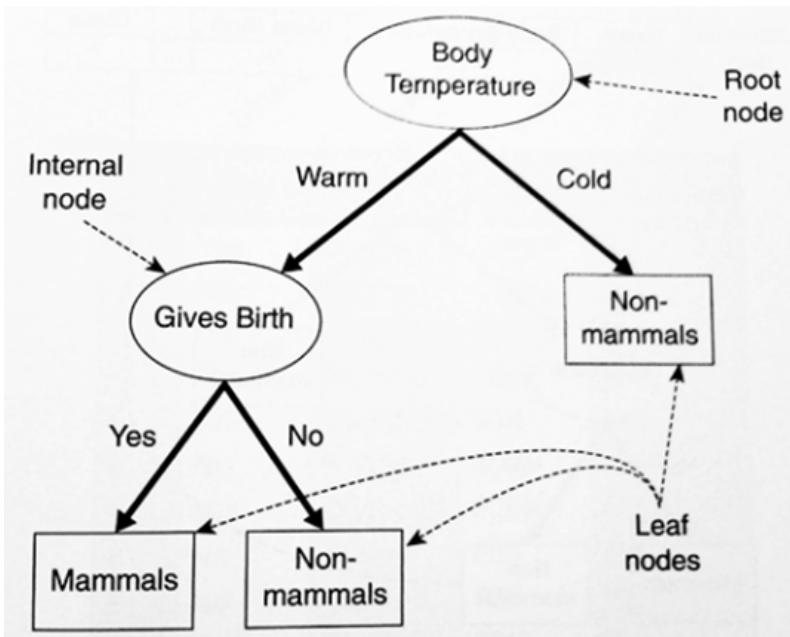
In [238]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(svm_pred, y_test)
```

Out[238]:
```
1.0
```

### Decision Trees

The widely used machine learning algorithm that use tree to visually and explicitly represent decision and its conditions.

It is used for both classification (more popular) and regression.




In [239]:
```
#import ML algorithm
from sklearn.tree import DecisionTreeClassifier
```

In [240]:
```
#select and train model
d_tree = DecisionTreeClassifier(max_depth=4)
d_tree.fit(X_train, y_train)
```

Out[240]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [241]:
```
#predict output
dtree_pred = d_tree.predict(X_test)

print("Prediction from Decision Tree : ", dtree_pred)
print("Test label:                     ", y_test)
```

```
Prediction from Decision Tree :  [1 1 1 2 2 0 1 0 0 2 0 2 1 0 2]
Test label:                      [1 1 1 2 2 0 1 0 0 2 0 2 1 0 2]
```

In [242]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(dtree_pred, y_test)
```

Out[242]: 1.0

In [243]:
```
from sklearn.tree.export import export_text
tree_rules = export_text(d_tree, feature_names=iris['feature_names'])
print(tree_rules)
```

```
|--- petal length (cm) <= 2.60
|   |--- class: 0
|--- petal length (cm) >  2.60
|   |--- petal width (cm) <= 1.65
|   |   |--- petal length (cm) <= 4.95
|   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.95
|   |   |   |--- class: 2
|   |--- petal width (cm) >  1.65
|   |   |--- petal length (cm) <= 4.85
|   |   |   |--- petal width (cm) <= 1.75
|   |   |   |   |--- class: 2
|   |   |   |--- petal width (cm) >  1.75
|   |   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.85
|   |   |   |--- class: 2
```

In [244]:
```
from sklearn import tree
import pydotplus
import collections
from IPython.display import Image
import graphviz

# Create DOT data
dot_data = tree.export_graphviz(d_tree, out_file=None,
                                feature_names=iris['feature_names'],
                                class_names=iris['target_names'])

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
#Image(graph.create_png())
```

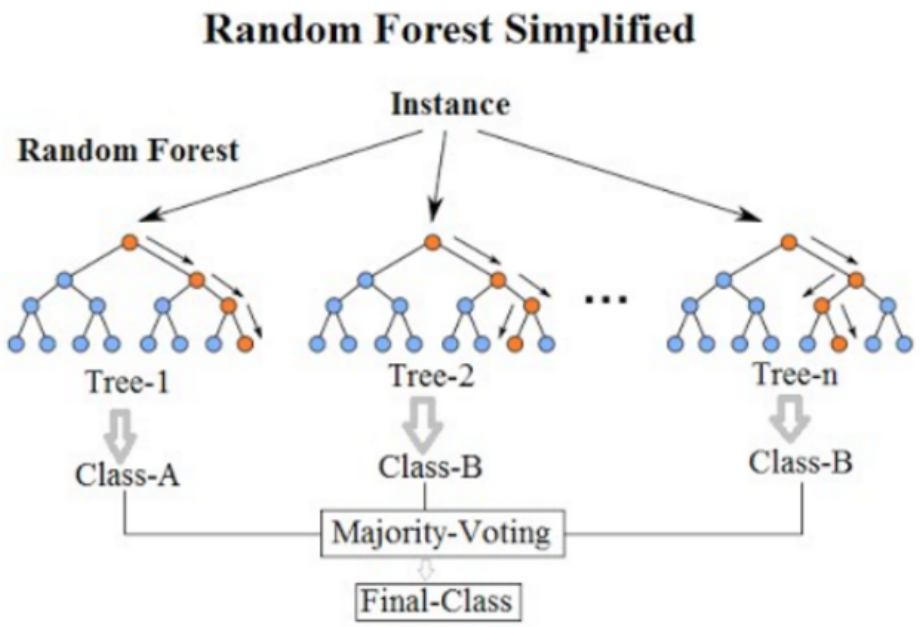
## Random Forests

The ensemble of decision trees that uses multiple decision trees.

Steps

1. Randomly choose bootstrap sample of size n (k samples from the training data).
2. Grow decision tree from bootstrap sample (sampling random sets of observations with replacement).
3. Repeat 1 and 2 on k times
4. Aggregate the prediction by each tree and choose the predictor with the majority vote. (e.g., Class A from tree1, Class B from tree 3 …)

Why Random Forest? Can build more robust model that has a better generalization performance and less subject to overfitting.




In [245]:
```
#import ML algorithm
from sklearn.ensemble import RandomForestClassifier
```

In [246]:
```
#select and train model (500 trees)
rdf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=15, n_jobs=1)
rdf.fit(X_train, y_train)
```

Out[246]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=15,
                       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=500, n_jobs=1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

In [247]:
```
#predict output
rdf_pred = rdf.predict(X_test)

print("Prediction from Random  Forest : ", rdf_pred)
print("Test label:                      ", y_test)
```

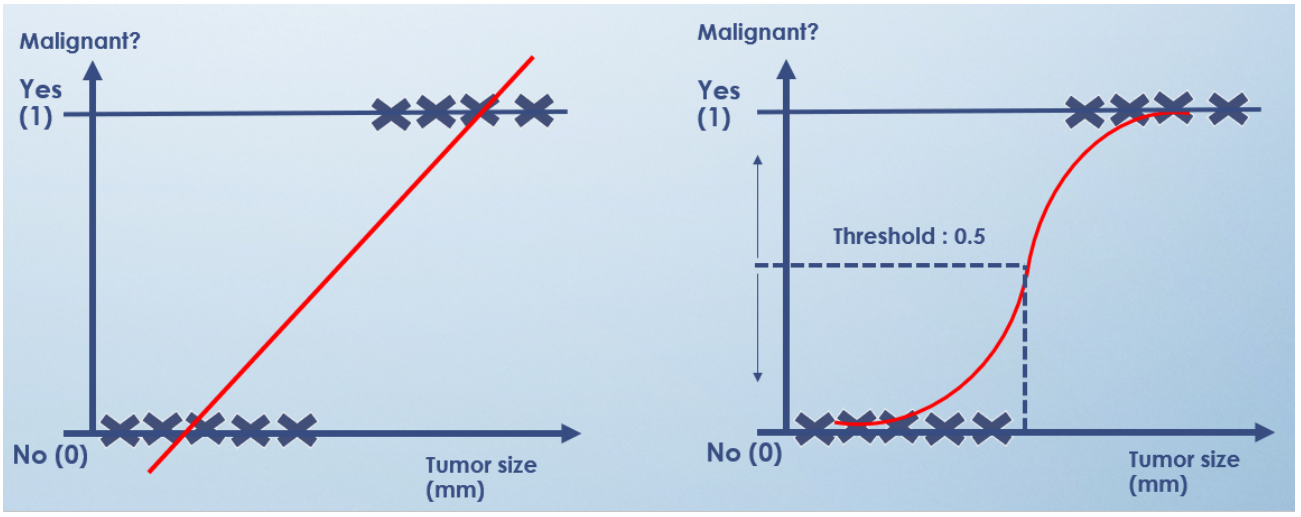
```
Prediction from Random  Forest :  [1 1 1 2 2 0 1 0 0 2 0 2 1 0 2]
Test label:                       [1 1 1 2 2 0 1 0 0 2 0 2 1 0 2]
```

In [248]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(rdf_pred, y_test)
```

Out[248]:
```
1.0
```

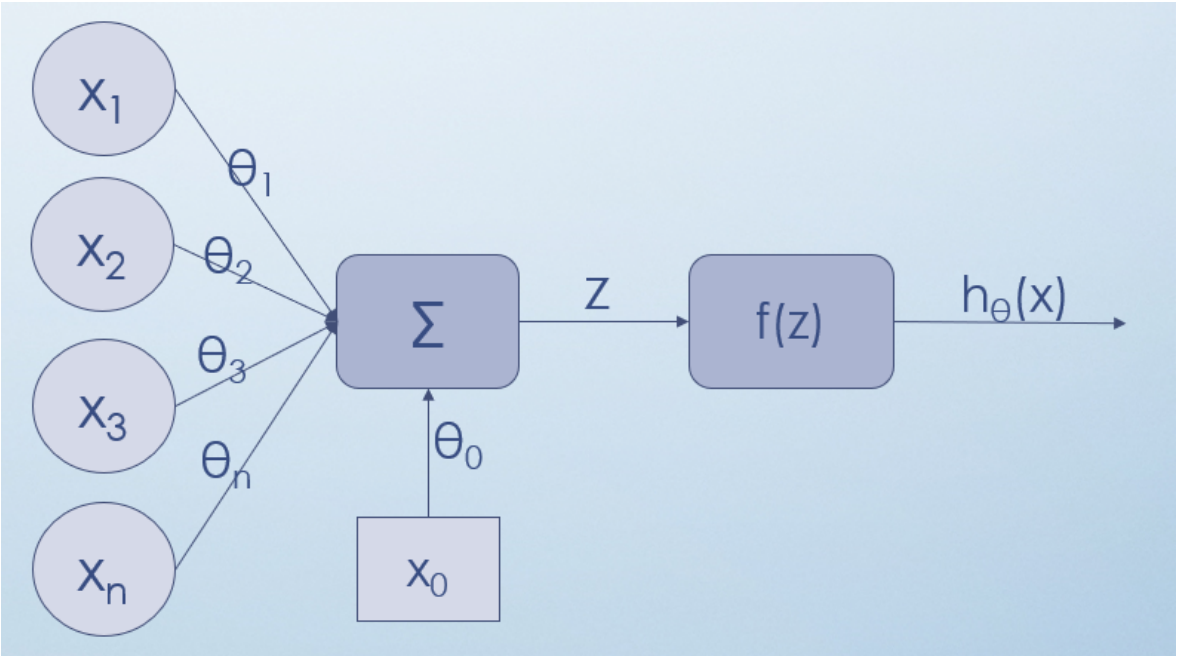
### Logistic Regression

- Output: Binary target (Yes/No)
- Input: Continuous/categorical variables
- Example : predicting if the tumor is malignant based on tumor size




### Logistic Regression Architecture

Hypothesis function : hθ(x) = 1 / (1 + e-z) where z= ( θ0x0 + θ1x1 + θ2x2 + θ3x3 +,, + θnxn )




In [249]:
```
## Prepare binary data
X_binary = X[y !=2]
y_binary = y[y !=2]

## Splitt 80 by 20
X_train_binary, X_test_binary, y_train_binary, y_test_binary = train_test_split(X_binary, y_binary, test_size=0.2)
```

In [250]:
```
print(y_train_binary)
print(np.unique(y_train_binary, return_counts=True))
```

```
[0 0 0 1 0 1 0 1 1 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0 0 1 1
 1 1 1 0 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0 1 0 0 1 1 1 1 0 0 0 0 1 1 1 0 0
 0 0 0 1 0 1]
(array([0, 1]), array([41, 39], dtype=int64))
```

In [251]:
```
#import ML algorithm
from sklearn.linear_model import LogisticRegression
```

In [252]:
```
#select and train model
lr = LogisticRegression()
lr.fit(X_train_binary, y_train_binary)
```

```
C:\Users\kevin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver wi
ll be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[252]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [253]:
```
#predict output
lr_pred = lr.predict(X_test_binary)

print("Prediction from Random  Forest : ", lr_pred)
print("Test label:                      ", y_test_binary)
```

```
Prediction from Random  Forest :  [0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 0 0]
Test label:                       [0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 0 0]
```

In [254]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(lr_pred, y_test_binary)
```

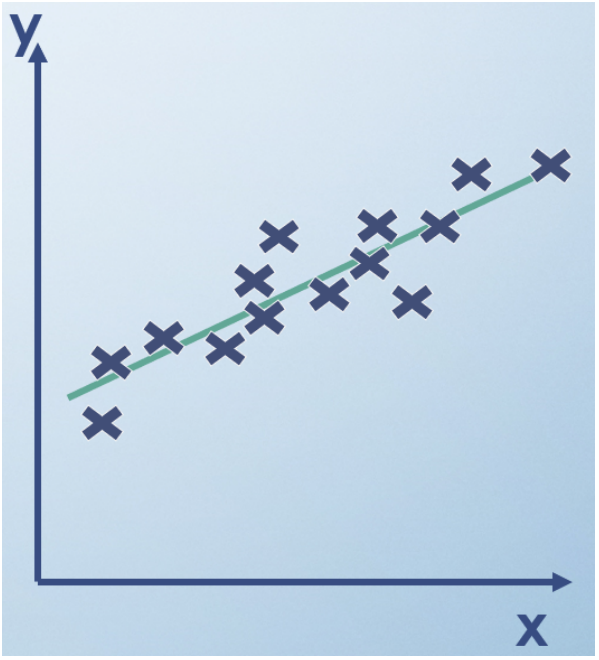
Out[254]:
```
1.0
```

In [255]:
```
## Coefficient
print("Weights of Logistic Regression   :", lr.coef_)
print("Intercept of Logistic Regression :",lr.intercept_)
```

```
Weights of Logistic Regression   : [[-0.3757827  -1.41160476  2.13463313  0.9228845 ]]
Intercept of Logistic Regression : [-0.23806821]
```

## Regression

- To predict values of a desired target quantity when the target quantity is continuous.
- Output: Continuous variables
- Example : predicting house price per sqft
- Types: Linear Regression, Polynomial Regression, Decision Tree, Random Forest
- Hypothesis function:
  - hθ(x) = θ0x0 + θ1x1 + θ2x2 + θ3x3 +,, + θnxn
  - y = 10 + 2x
- Accuracy of Regression model
  - Square Error(SE): sum (y – pred)2
  - Mean Square Error (MSE) = (1/n)*SE
  - Root Mean Square Error (RMSE) = square root of MSE

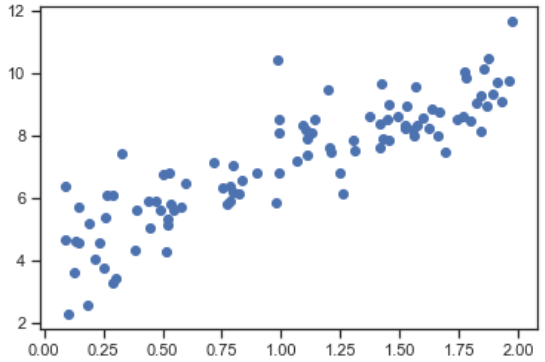



In [256]:
```
## Preparing Lineal Regression data
X_linear = 2 * np.random.rand(100, 1)
y_linear = 4 + 3 * X_linear + np.random.randn(100, 1)
```

In [257]:
```
## Visualization of data
import matplotlib.pyplot as plt
%matplotlib inline

plt.scatter(X_linear, y_linear)
```

Out[257]:
```
<matplotlib.collections.PathCollection at 0x221a891d400>
```




In [258]:
```
## train and test data
X_train_linear, X_test_linear, y_train_linear, y_test_linear = train_test_split(X_linear, y_linear, test_size=0.2,
```

In [259]:
```
#import ML algorithm
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
```

In [260]:
```
#select and train model
linear = linear_model.LinearRegression()
linear.fit(X_train_linear, y_train_linear)
```

Out[260]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [261]:
```
#predict output
linear_pred = linear.predict(X_test_linear)

print("Prediction from Linear Regression : ", linear_pred.flatten())
print("Test label:                        ", y_test_linear.flatten())
```

```
Prediction from Linear Regression :  [8.29387632 7.92828045 9.22243345 8.79440665 6.44861916 9.07417561
 8.99115368 5.847927   7.02091729 6.54776887 9.33963269 4.94429562
 8.57040977 8.68600915 6.39750959 9.46594515 7.62246563 4.50121931
 8.69912684 6.21286506]
Test label:                        [7.87839717 7.84695789 8.53366091 8.56305529 7.02899762 7.44612302
 8.77061253 6.47594712 6.81154255 6.53853609 9.84789295 6.0962674
 8.2371604  7.98626657 5.90463335 9.04502312 9.47153148 4.62542571
 9.56166866 7.14659478]
```

In [262]:
```
## Evaluation of trained SVM model
mean_squared_error(linear_pred, y_test_binary)
```

Out[262]:
```
52.599748730242084
```

In [263]:
```
## Coefficient
print("Weights of Linear Regression   :", linear.coef_)
print("Intercept of Linear Regression :", linear.intercept_)
```

```
Weights of Linear Regression   : [[2.93841348]]
Intercept of Linear Regression : [4.10066893]
```

## Unsupervised Machine Learning

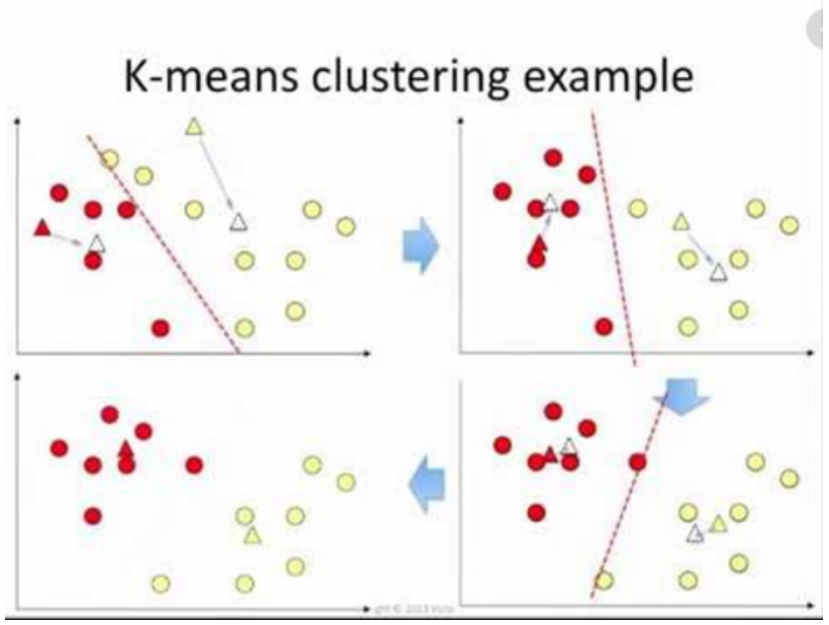
- Not-labeled input data : no correct answers
- Exploratory
- Clustering : the assignment of set of observations into subsets (clusters) so that data in the same clusters are more similar to each other than to those from different clusters.
- Algorithms : k-means
- Industry implementation - the grouping of documents, music, movies by different topics, finding customers that share similar - interests based on common purchase behaviors.
  - Customer segmentation
  - Document clustering
  - Image segmentation
  - Recommendation engines

### k-means Clustering

- The simplest and popular unsupervised machine learning algorithm
- To group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset.
- the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

How it works

1. Randomly pick k centroids as initial cluster centers.
2. Assign each sample to the nearest centroid.
3. Move the centroids to the center of the samples that were assigned to it.
4. Repeat 2 and 3 until the cluster assignments do not change under a tolerance.




In [264]:
```
#import ML algorithm
from sklearn.cluster import KMeans
```

In [265]:
```
# build k means model with 2 clusters
kmeans = KMeans(n_clusters = 3, tol=0.0001, max_iter=100)
kmeans.fit(X)
```

Out[265]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=100,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [266]:
```
#print the results of the value of centroids
print(kmeans.cluster_centers_) # centroids

#print the results of the value of centroids
print("prediction from Kmeans: /n", kmeans.labels_)
print("labels:                 /n", y)
```

```
[[5.006      3.428      1.462      0.246     ]
 [5.9016129  2.7483871  4.39354839 1.43387097]
 [6.85       3.07368421 5.74210526 2.07105263]]
prediction from Kmeans: /n [0 0 0 
```