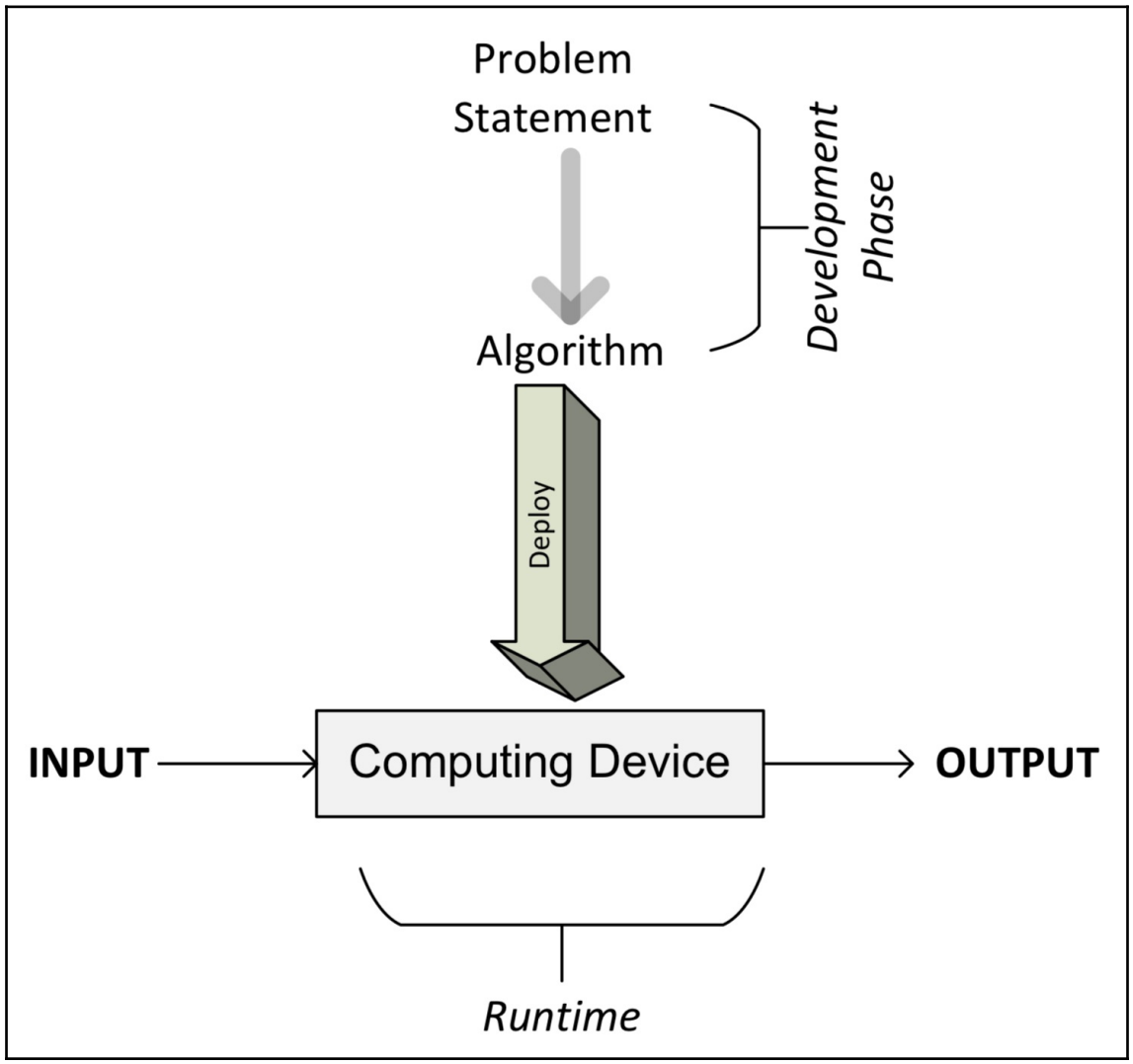
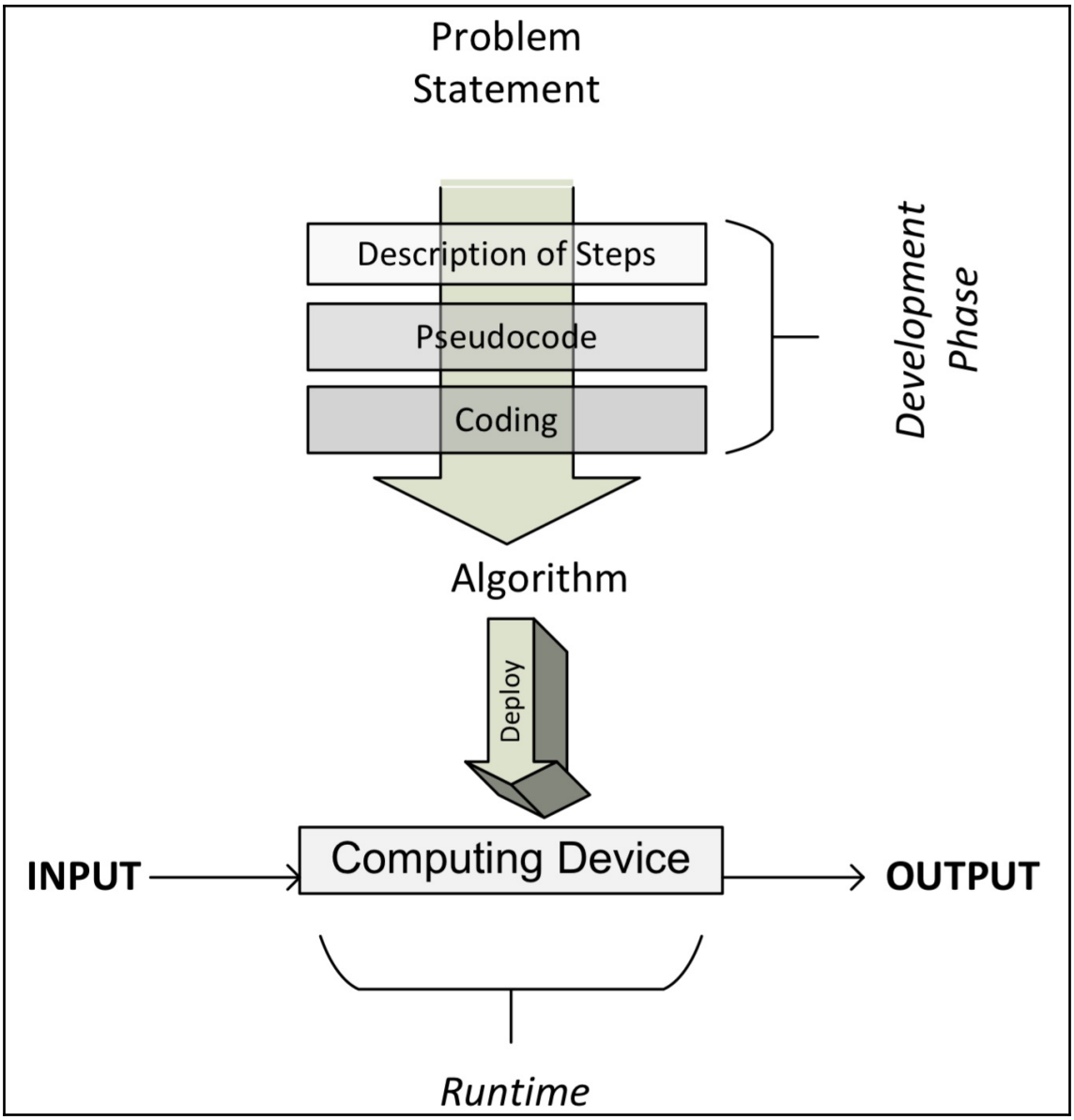
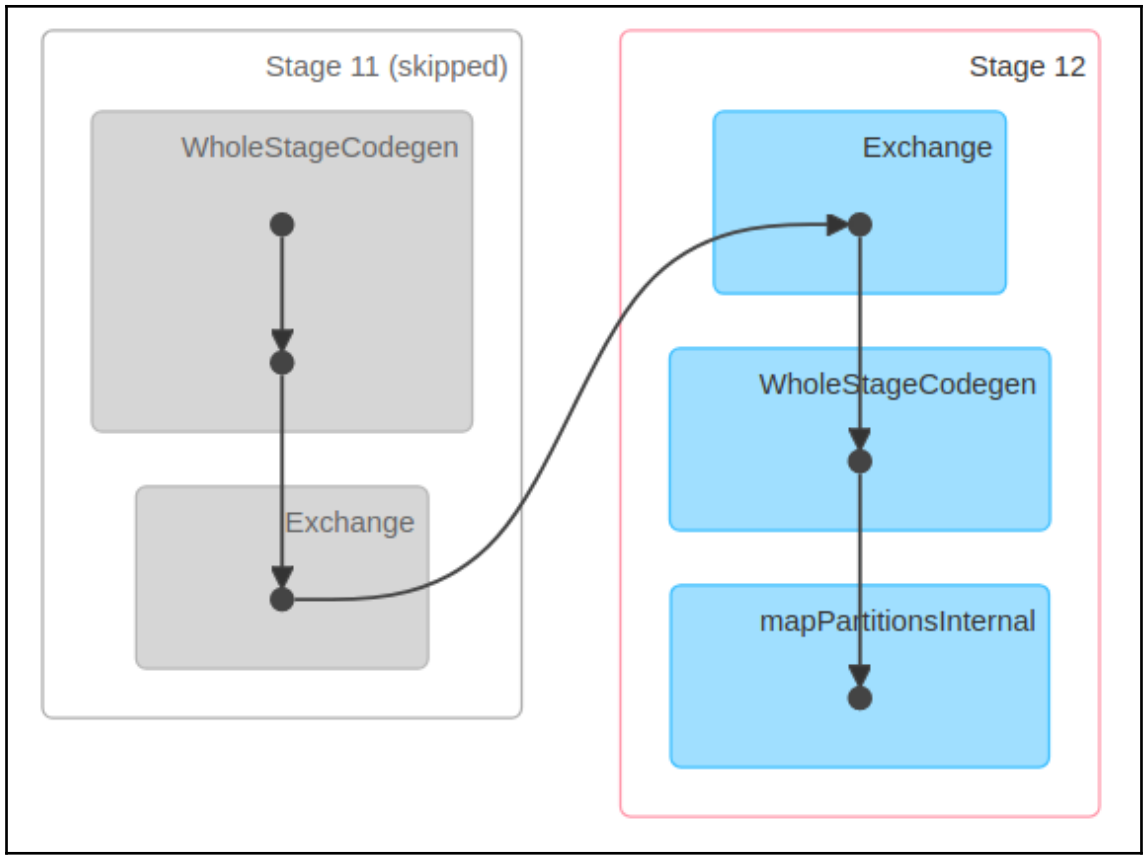
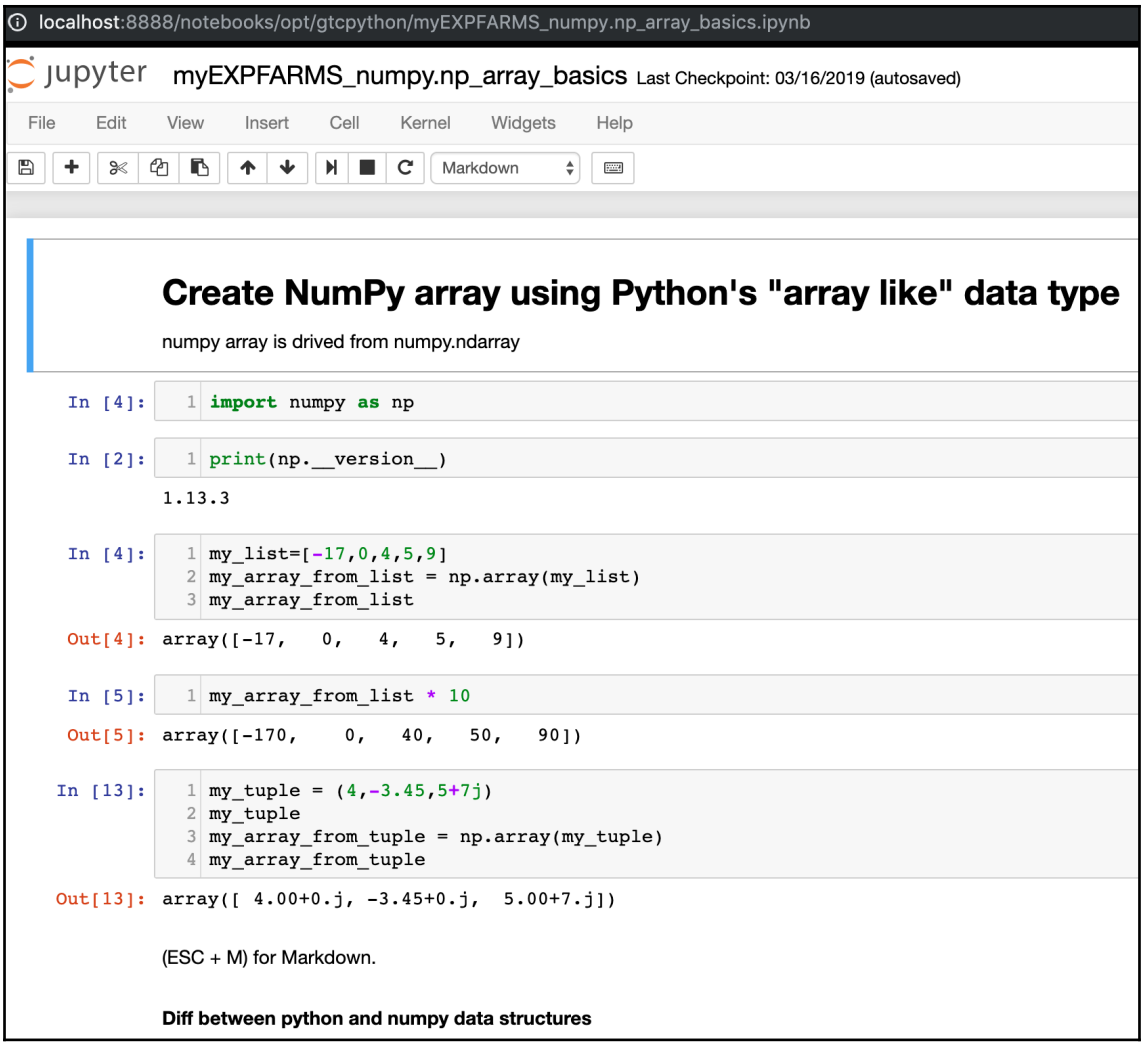
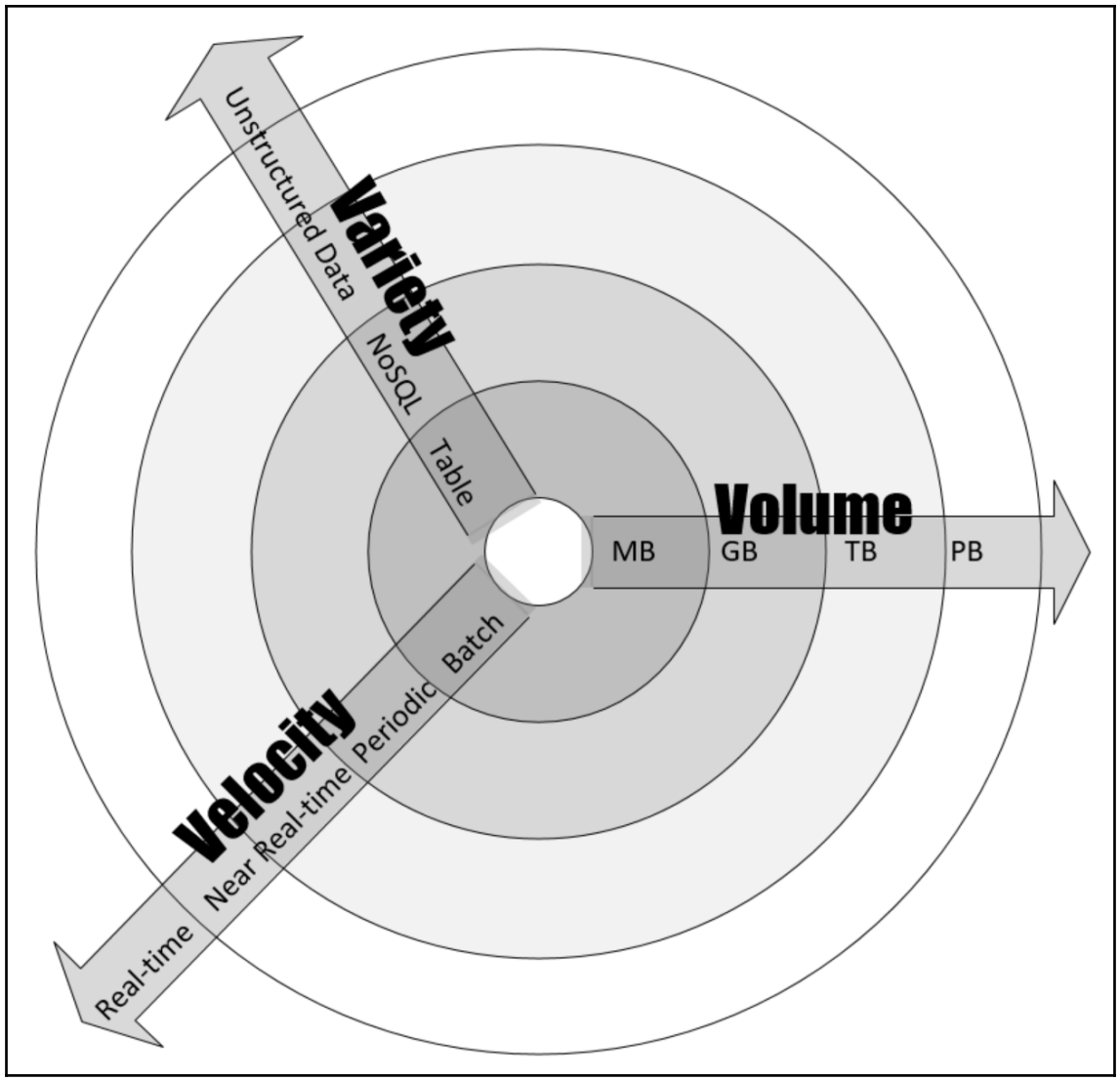
Chapter 1: Overview of Algorithms

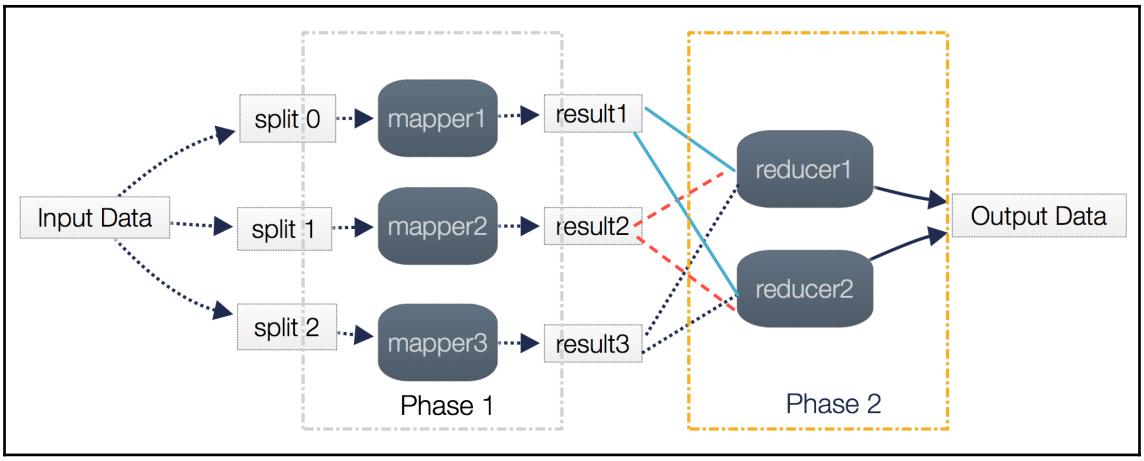



Problem
Statement
Description of Steps
Pseudocode
Coding
Development
Phase
Algorithm
Deploy
INPUT
Computing Device
OUTPUT
Runtime


Stage 11 (skipped)
WholeStageCodegen
Exchange
Stage 12
Exchange
WholeStageCodegen
mapPartitionsInternal


localhost:8888/notebooks/opt/gtcpython/myEXPFARMS_numpy.np_array_basics.ipynb
jupyter
myEXPFARMS_numpy.np_array_basics
Last Checkpoint: 03/16/2019 (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Markdown
Create NumPy array using Python's "array like" data type
numpy array is drived from numpy.ndarray
In [4]:
import numpy as np
In [2]:
print(np.__version__)
1.13.3
In [4]:
my_list=[-17,0,4,5,9]
my_array_from_list = np.array(my_list)
my_array_from_list
Out[4]:
array([-17, 0, 4, 5, 9])
In [5]:
my_array_from_list * 10
Out[5]:
array([-170, 0, 40, 50, 90])
In [13]:
my_tuple = (4,-3.45,5+7j)
my_tuple
my_array_from_tuple = np.array(my_tuple)
my_array_from_tuple
Out[13]:
array([4.00+0.j, -3.45+0.j, 5.00+7.j])
(ESC + M) for Markdown.
Diff between python and numpy data structures


Variety
Unstructured Data
NoSQL
Table
Volume
MB
GB
TB
PB
Velocity
Real-time
Near Real-time
Periodic
Batch

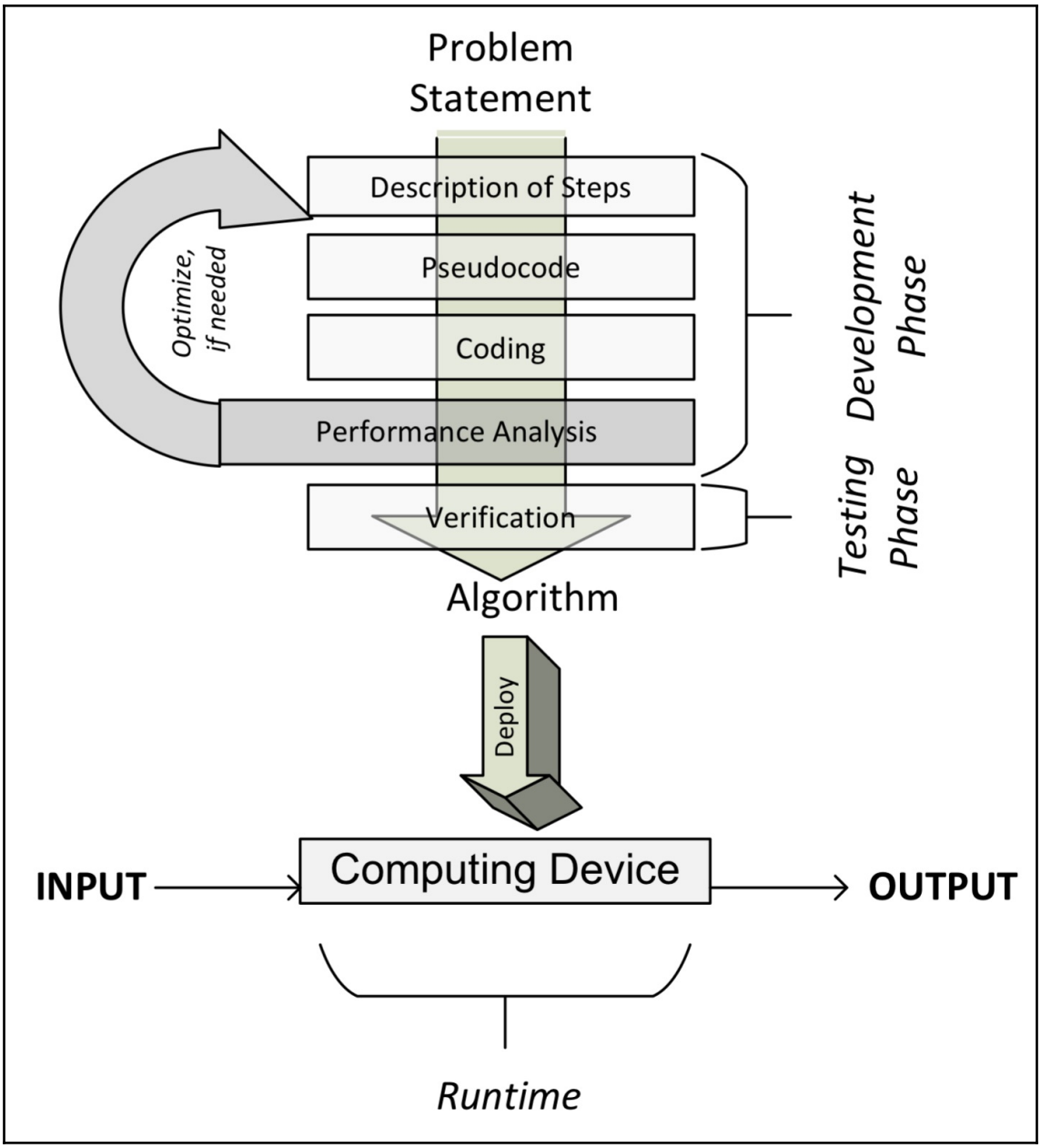


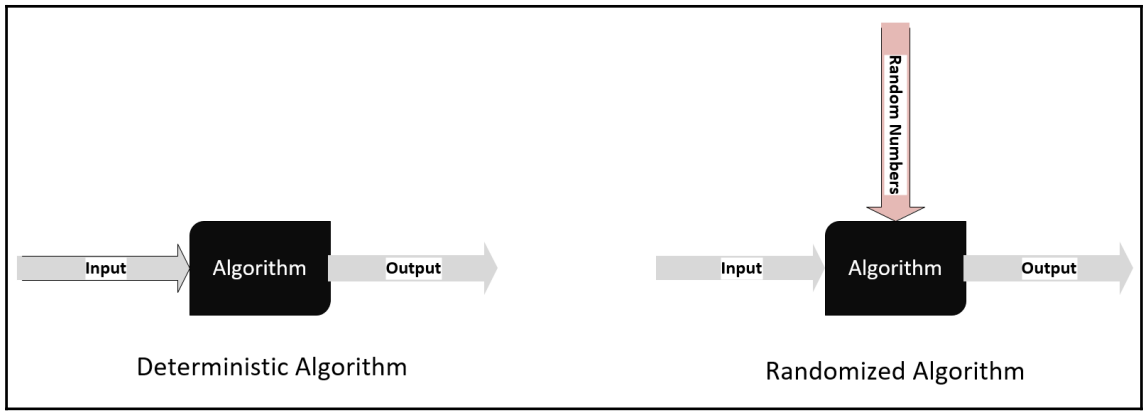

```
In [2]: 1 getFirst([1,2,3])
Out[2]: 1

In [3]: 1 getFirst([1,2,3,4,5,6,7,8,9,10])
Out[3]: 1
```

```
In [5]: 1 getSum([1,2,3])
Out[5]: 6

In [6]: 1 getSum([1,2,3,4])
Out[6]: 10
```

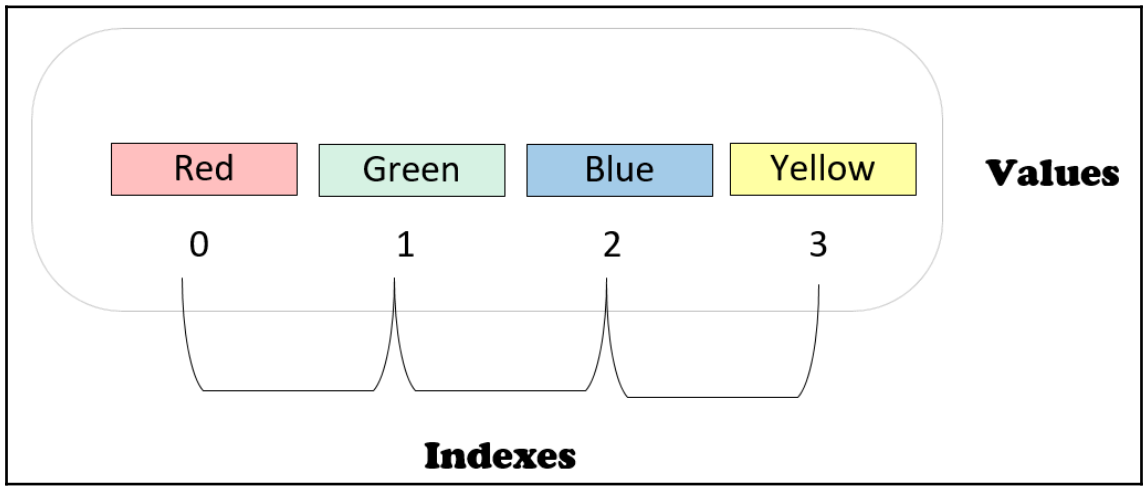
```
In [8]:  1 getSum([[1,2],[3,4]])
Out[8]: 10

In [9]:  1 getSum([[1,2,3],[4,5,6]])
Out[9]: 21
```

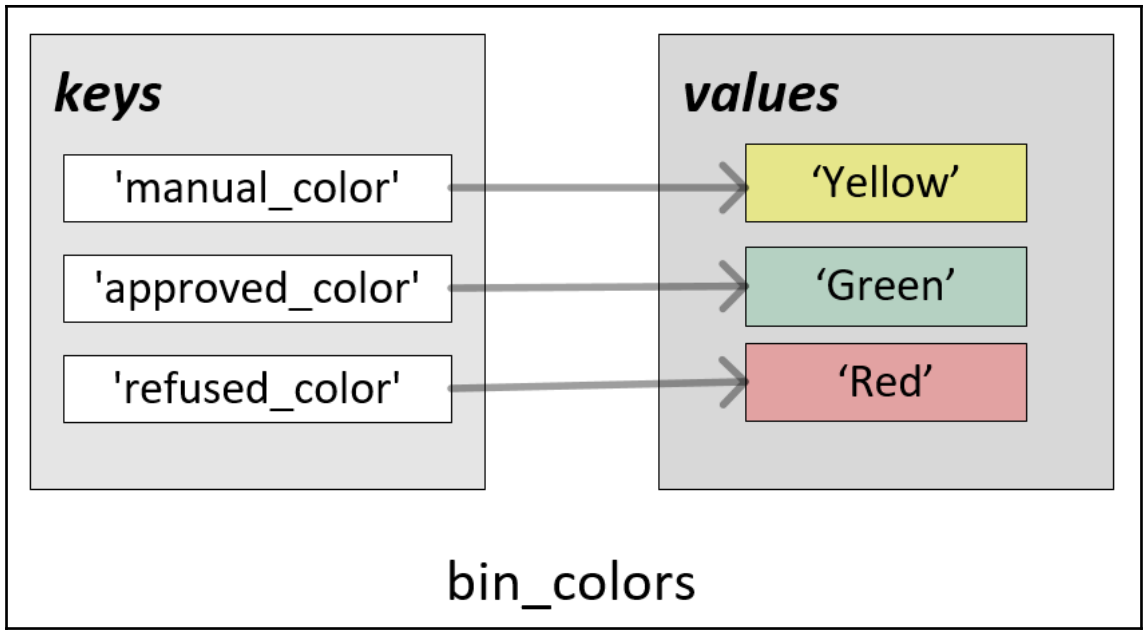
```
In [11]:  1 searchBinary([8,9,10,100,1000,2000,3000], 10)
          2
Out[11]: True

In [12]:  1 searchBinary([8,9,10,100,1000,2000,3000], 5)
Out[12]: False
```

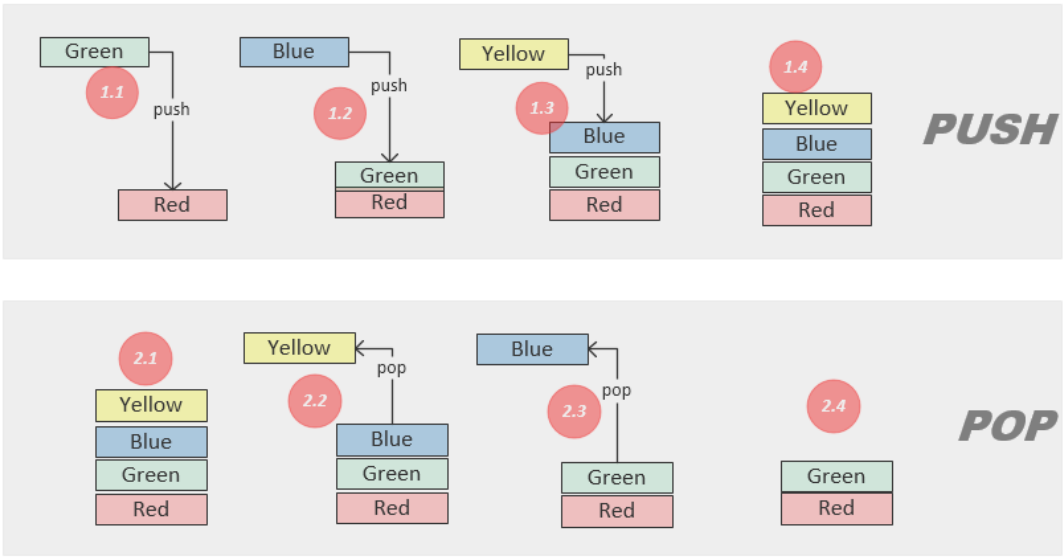

Problem Statement
Description of Steps
Pseudocode
Coding
Performance Analysis
Verification
Optimize, if needed
Development Phase
Testing Phase
Algorithm
Deploy
INPUT
Computing Device
OUTPUT
Runtime


Input
Algorithm
Output
Deterministic Algorithm
Random Numbers
Input
Algorithm
Output
Randomized Algorithm

Chapter 2: Data Structures Used in Algorithms







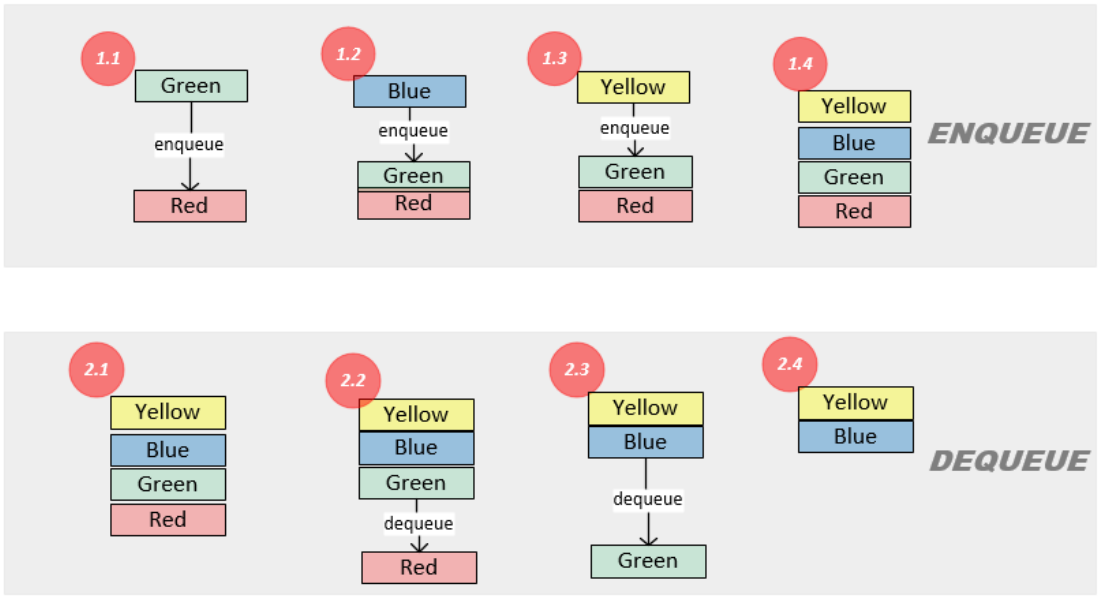
Stack

Populate the stack

```
In [2]: stack=Stack()
        stack.push('Red')
        stack.push('Green')
        stack.push("Blue")
        stack.push("Yellow")
```

Pop

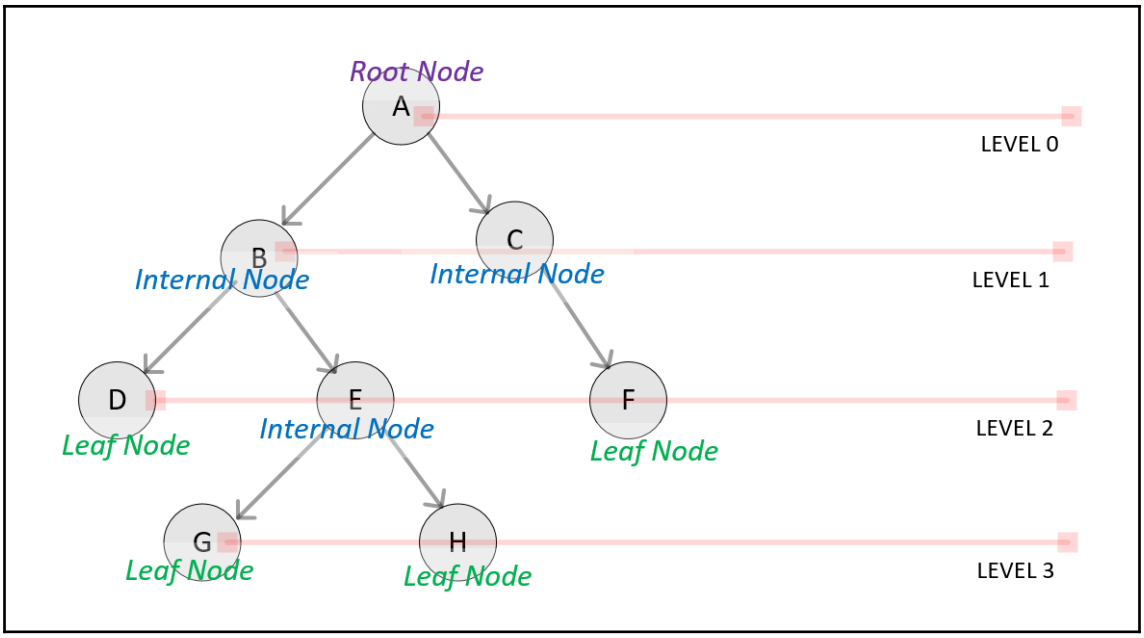
```
In [3]: stack.pop()
Out[3]: 'Yellow'

In [7]: stack.isEmpty()
Out[7]: False
```

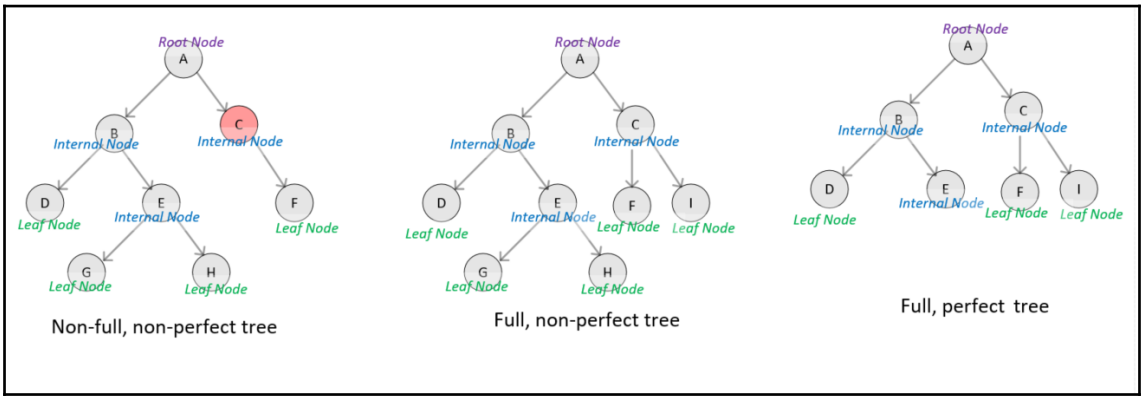

1.1
Green
enqueue
Red
1.2
Blue
enqueue
Green
Red
1.3
Yellow
enqueue
Green
Red
1.4
Yellow
Blue
Green
Red
ENQUEUE
2.1
Yellow
Blue
Green
Red
2.2
Yellow
Blue
Green
dequeue
Red
2.3
Yellow
Blue
dequeue
Green
2.4
Yellow
Blue
DEQUEUE

Queue

Using Queue Class

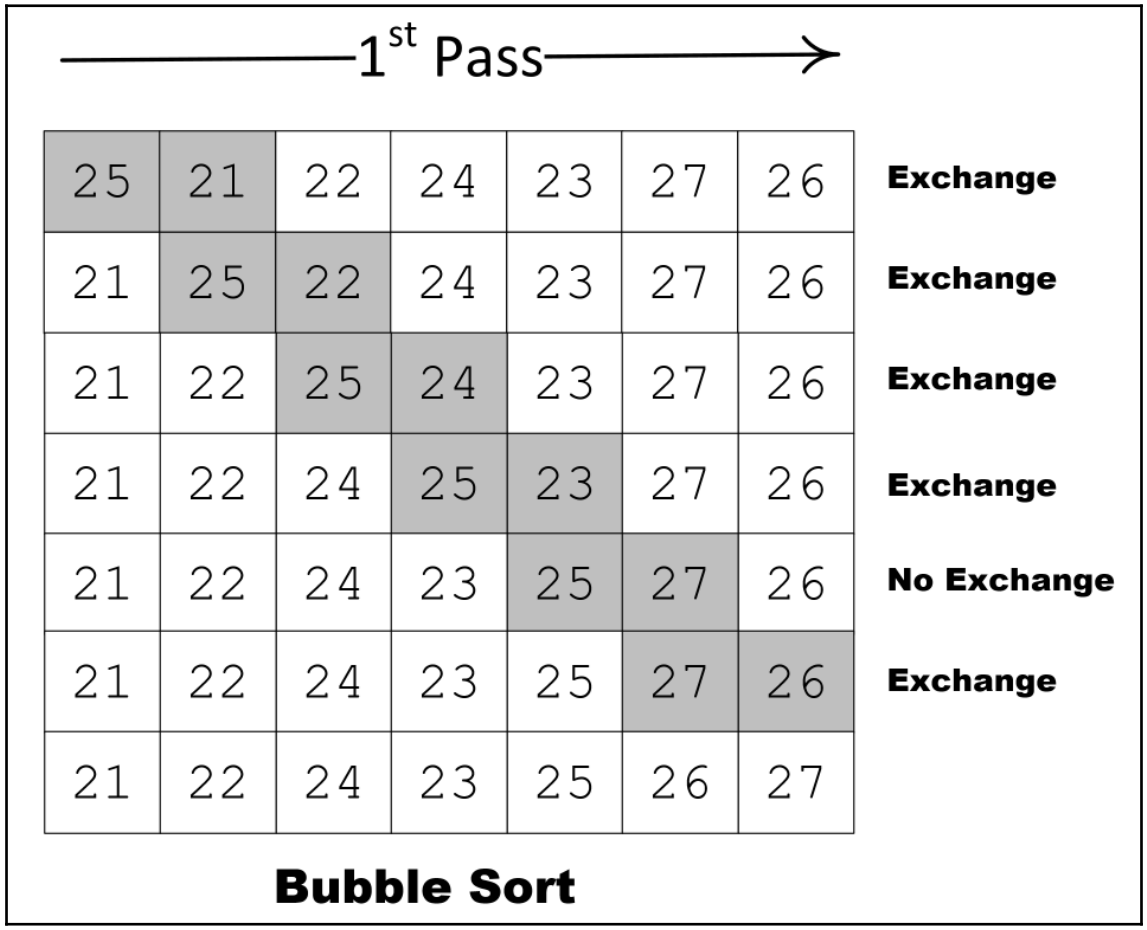
```
In [2]: queue = Queue()

In [3]: queue.enqueue('Red')

In [4]: queue.enqueue('Green')

In [5]: queue.enqueue('Blue')

In [6]: queue.enqueue('Yellow')

In [7]: print(queue.size())
        4

In [8]: print(queue.dequeue())
        Red

In [9]: print(queue.dequeue())
        Green
```


Root Node
A
LEVEL 0
B
Internal Node
C
Internal Node
LEVEL 1
D
Leaf Node
E
Internal Node
F
Leaf Node
LEVEL 2
G
Leaf Node
H
Leaf Node
LEVEL 3


Non-full, non-perfect tree
Full, non-perfect tree
Full, perfect tree

Chapter 3: Sorting and Searching Algorithms

```
In [104]:  1  print(var1,var2)

           2 1
```



In [91]:

```
lastElementIndex = len(list)-1
print(0,list)
for idx in range(lastElementIndex):
                if list[idx]>list[idx+1]:
                    list[idx],list[idx+1]=list[idx+1],list[idx]
                print(idx+1,list)
```

```
0 [25, 21, 22, 24, 23, 27, 26]
1 [21, 25, 22, 24, 23, 27, 26]
2 [21, 22, 25, 24, 23, 27, 26]
3 [21, 22, 24, 25, 23, 27, 26]
4 [21, 22, 24, 23, 25, 27, 26]
5 [21, 22, 24, 23, 25, 27, 26]
6 [21, 22, 24, 23, 25, 26, 27]
```

In [5]:

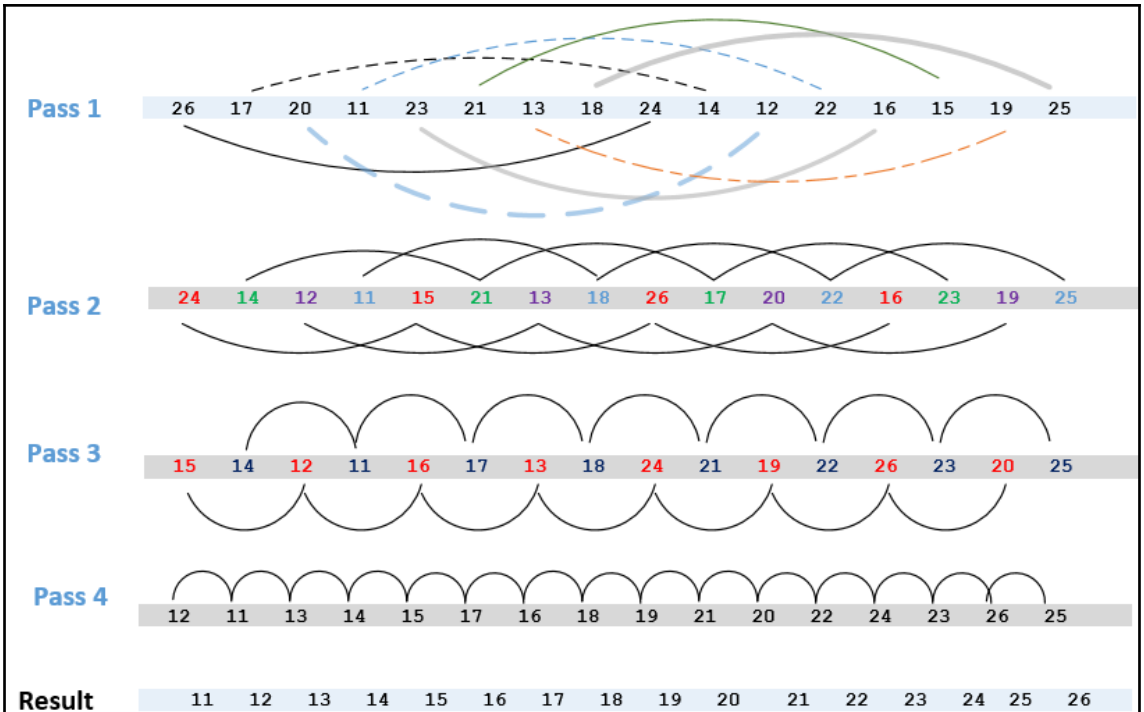
```
def BubbleSort(list):
# Excahnge the elements to arrange in order
    lastElementIndex = len(list)-1
    for passNo in range(lastElementIndex,0,-1):
        for idx in range(passNo):
            if list[idx]>list[idx+1]:
                list[idx],list[idx+1]=list[idx+1],list[idx]
    return list
```

25	26	22	24	27	23	21	**Insert 25**
25	26	22	24	27	23	21	**Insert 26**
22	25	26	24	27	23	21	**Insert 22**
22	24	25	26	27	23	21	**Insert 24**
22	24	25	26	27	23	21	**Insert 27**
22	23	24	25	26	27	21	**Insert 23**
21	22	23	24	25	26	27	**Insert 21**

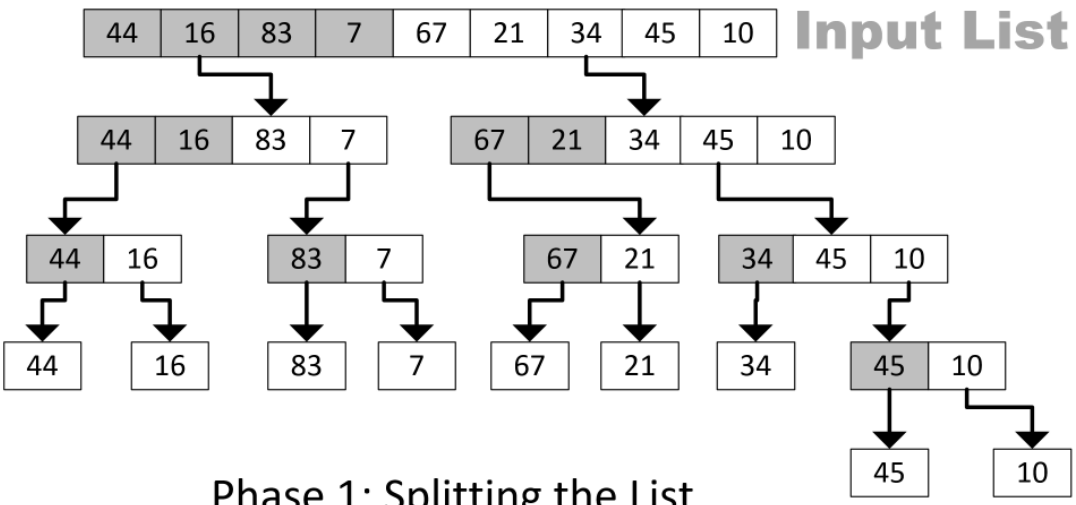
Insertion Sort

```
In [134]:
1 list = [25,26,22,24,27,23,21]

In [135]:
1 InsertionSort(list)
2 print(list)

[21, 22, 23, 24, 25, 26, 27]
```

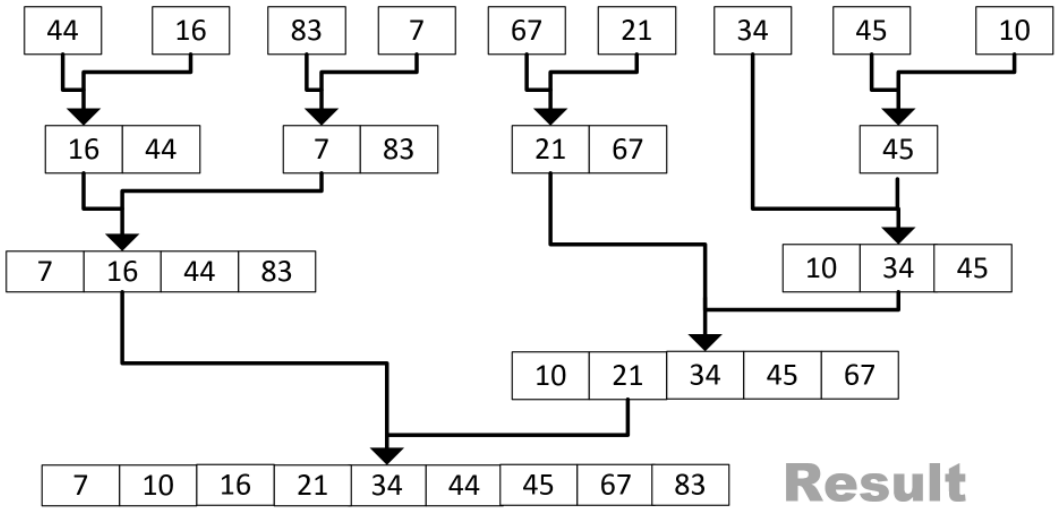
$$w(N) = \sum_{i=1}^{N-1} i = \frac{(N-1)N}{2} = \frac{N^2 - N}{2}$$



Passes in the Shell Sort Algorithm



Phase 1: Splitting the List



Phase 2: Merging the Splits

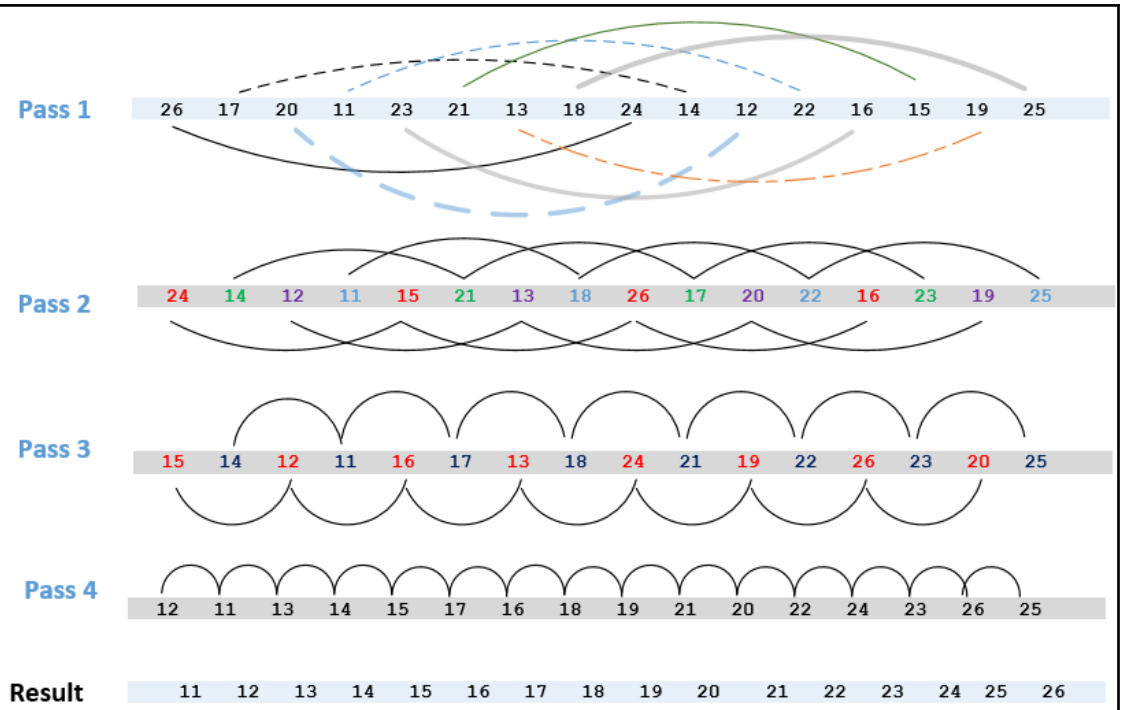
In [6]:

```
def MergeSort(list):
    if len(list)>1:
        mid = len(list)//2 #splits list in half
        left = list[:mid]
        right = list[mid:]

        MergeSort(left) #repeats until length of each list is 1
        MergeSort(right)

        a = 0
        b = 0
        c = 0
        while a < len(left) and b < len(right):
            if left[a] < right[b]:
                list[c]=left[a]
                a = a + 1
            else:
                list[c]=right[b]
                b = b + 1
            c = c + 1
        while a < len(left):
            list[c]=left[a]
            a = a + 1
            c = c + 1

        while b < len(right):
            list[c]=right[b]
            b = b + 1
            c = c + 1
    return list
```

In [180]:

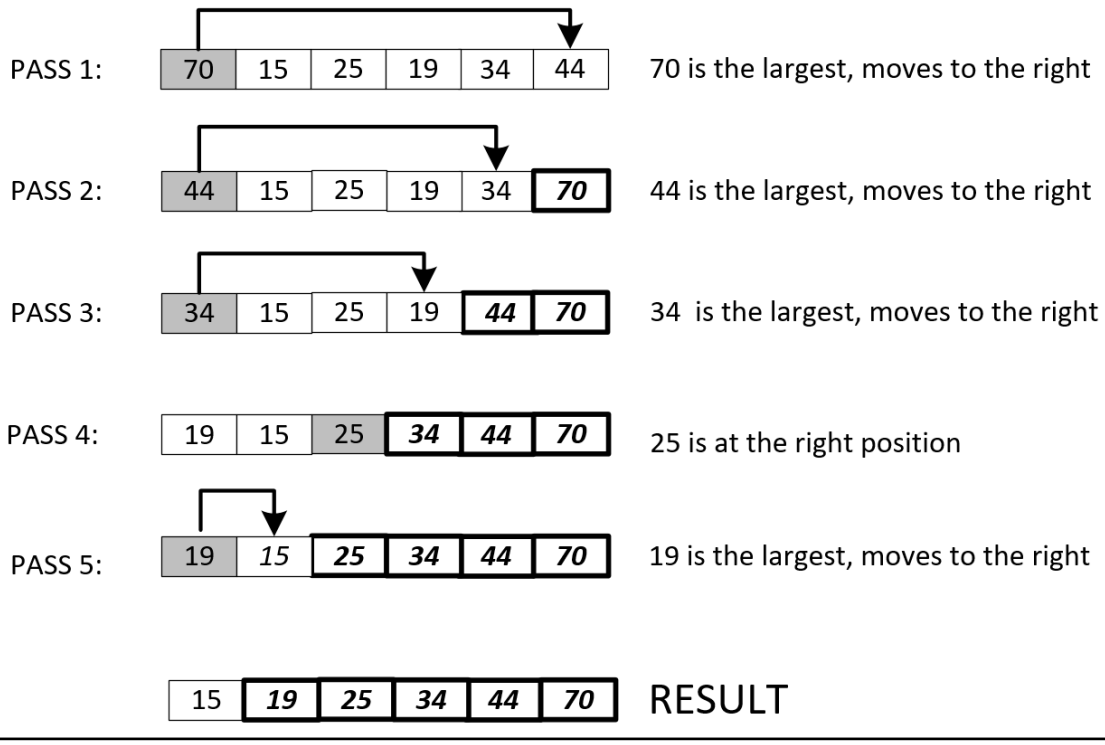
```
1 list = [44,16,83,7,67,21,34,45,10]
2 MergeSort(list)
3 print(list)
4
5
```

```
[7, 10, 16, 21, 34, 44, 45, 67, 83]
```



Passes in the Shell Sort Algorithm

```
In [119]:
1 list = [26,17,20,11,23,21,13,18,24,14,12,22,16,15,19,25]

In [120]:
1 shellSort(list)
2 print(list)

[11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
```



In [202]:

```
1 list = [70,15,25,19,34,44]
2 SelectionSort(list)
3 print(list)
```

```
[15, 19, 25, 34, 44, 70]
```

```
1 list = [12, 33, 11, 99, 22, 55, 90]
2 print(LinearSearch(list, 12))
3 print(LinearSearch(list, 91))
```

```
True
False
```

In [14]:

```
1 list = [12, 33, 11, 99, 22, 55, 90]
2 sorted_list = BubbleSort(list)
3 print(BinarySearch(list, 12))
4 print(BinarySearch(list, 91))
```

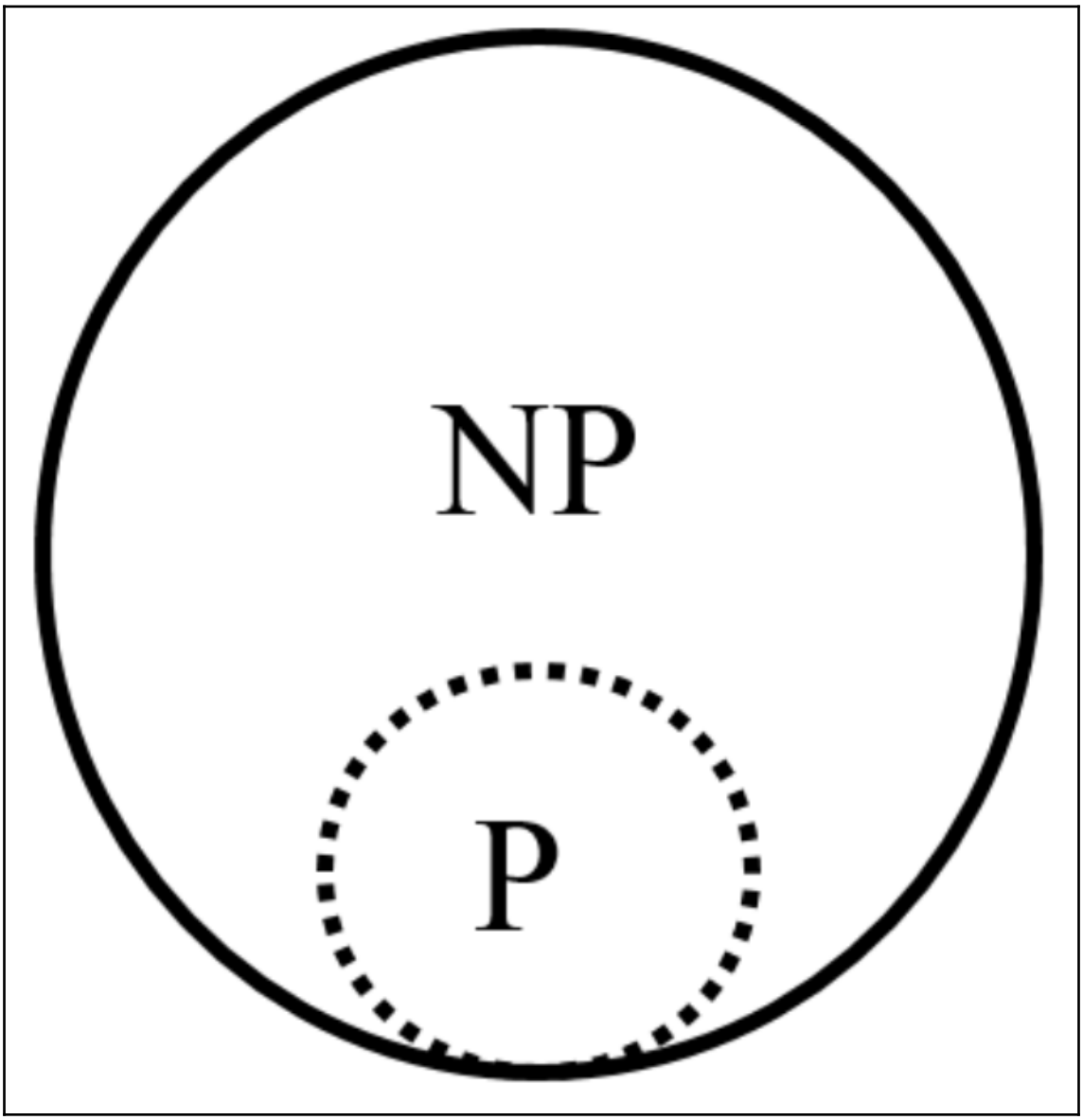
```
True
False
```

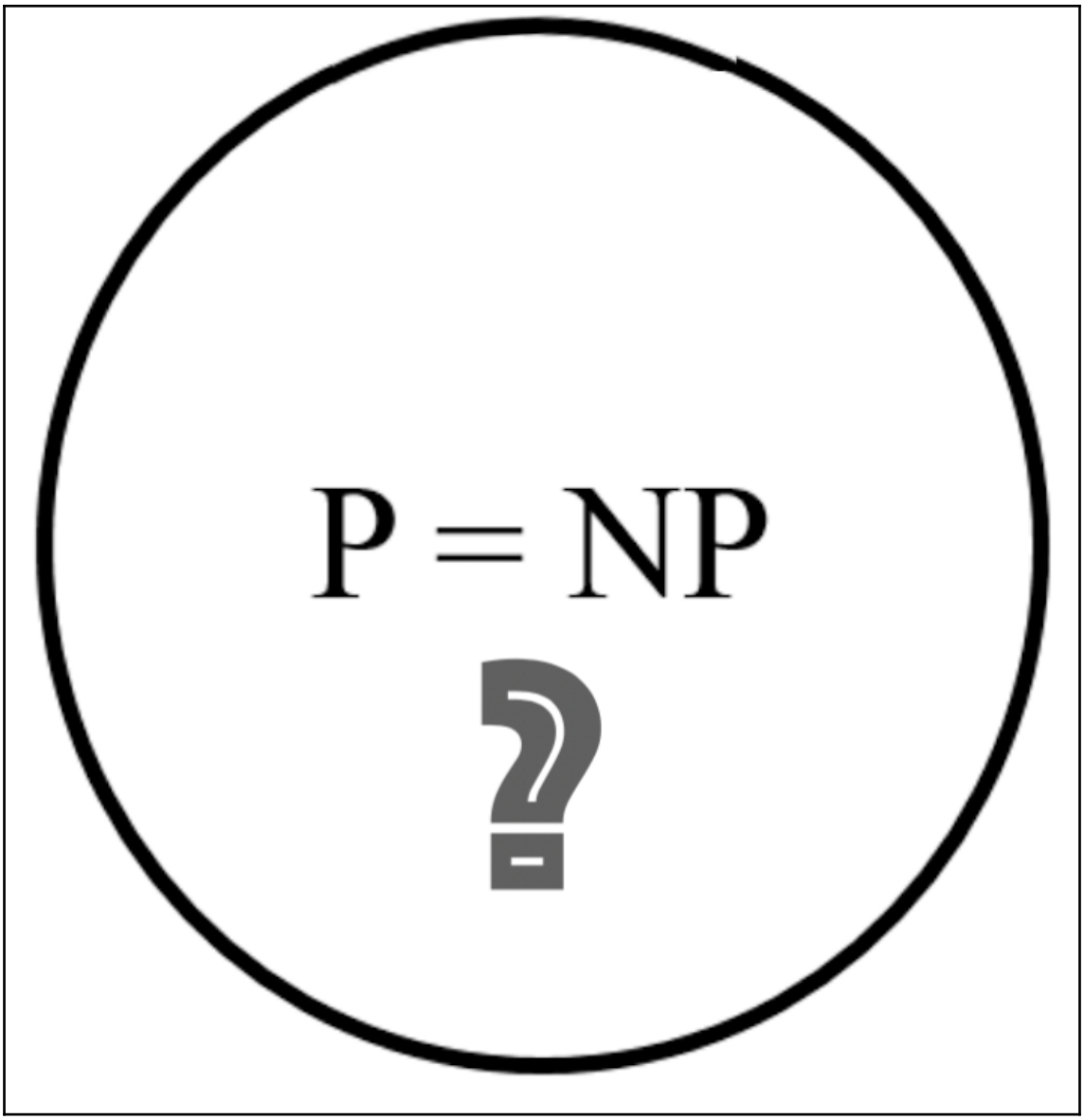
In [16]:

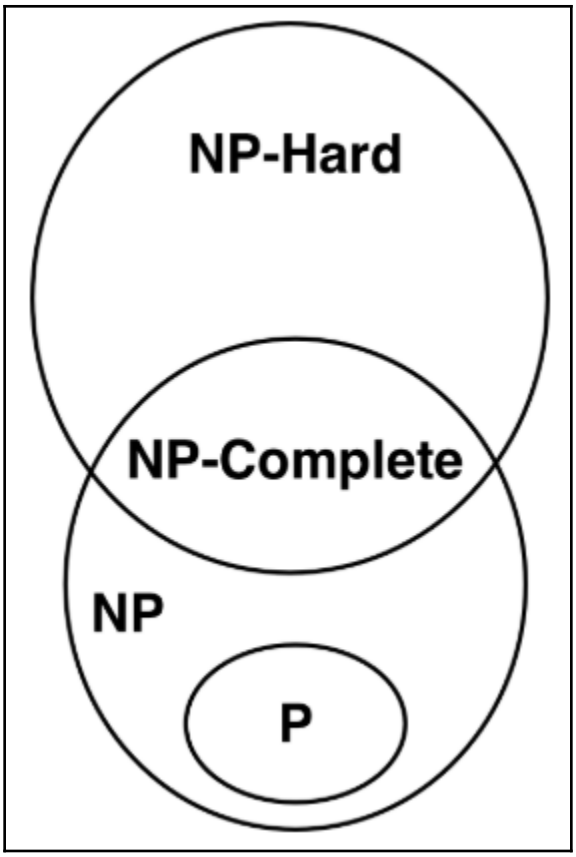
```
1 list = [12, 33, 11, 99, 22, 55, 90]
2 sorted_list = BubbleSort(list)
3 print(IntPolsearch(list, 12))
4 print(IntPolsearch(list,91))
```

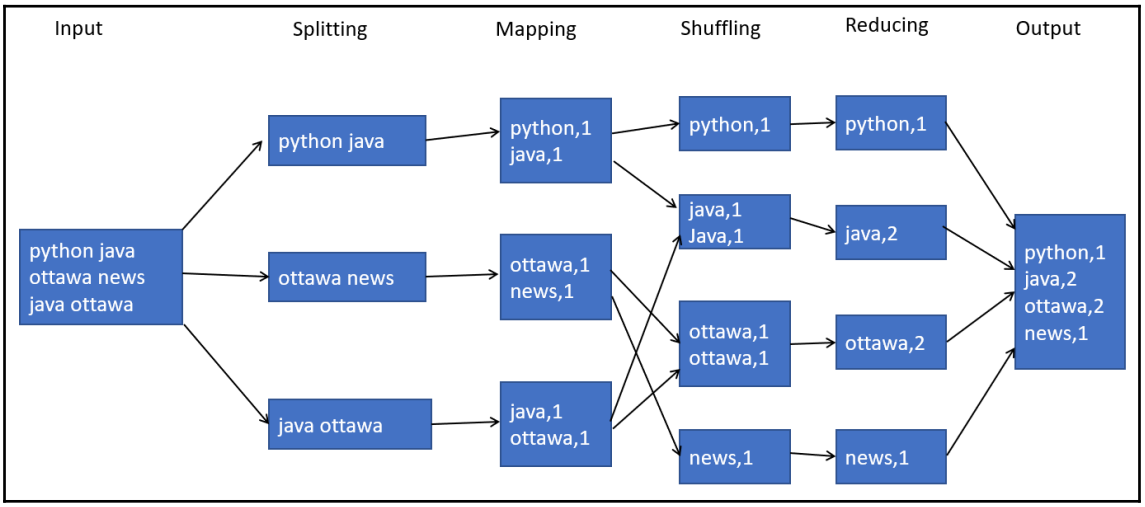
```
True
False
```

Chapter 4: Designing Algorithms




P = NP


NP-Hard
NP-Complete
NP
P


Input
Splitting
Mapping
Shuffling
Reducing
Output
python java
ottawa news
java ottawa
python java
ottawa news
java ottawa
python,1
java,1
ottawa,1
news,1
java,1
ottawa,1
python,1
java,1
Java,1
ottawa,1
ottawa,1
news,1
python,1
java,2
ottawa,2
news,1
python,1
java,2
ottawa,2
news,1

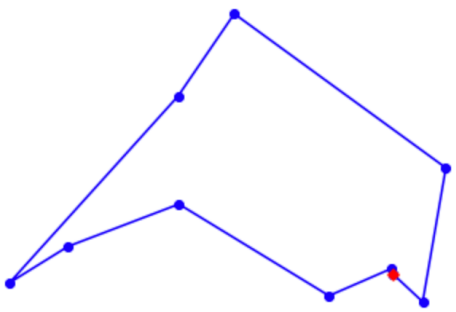
```
In [19]: wordPairs = wordsRDD.map(lambda w: (w, 1))
         print (wordPairs.collect())

         [('python', 1), ('java', 1), ('ottawa', 1), ('ottawa', 1), ('java', 1), ('news', 1)]
```

```
In [20]: wordCountsCollected = wordPairs.reduceByKey(lambda x,y: x+y)
         print(wordCountsCollected.collect())

         [('python', 1), ('java', 2), ('ottawa', 2), ('news', 1)]
```

```
[ ]  from time import clock
     from collections import Counter
     def tsp(algorithm, cities):
         t0   = clock()
         tour = algorithm(cities)
         t1   = clock()
         assert Counter(tour) == Counter(cities) # Every city appears exactly once in tour
         visualize_tour(tour)
         print("{}: {} cities ⇒ tour length {:.0f} (in {:.3f} sec)".format(
               name(algorithm), len(tour), distance_tour(tour), t1 - t0))

     def name(algorithm): return algorithm.__name__.replace('_tsp', '')
```

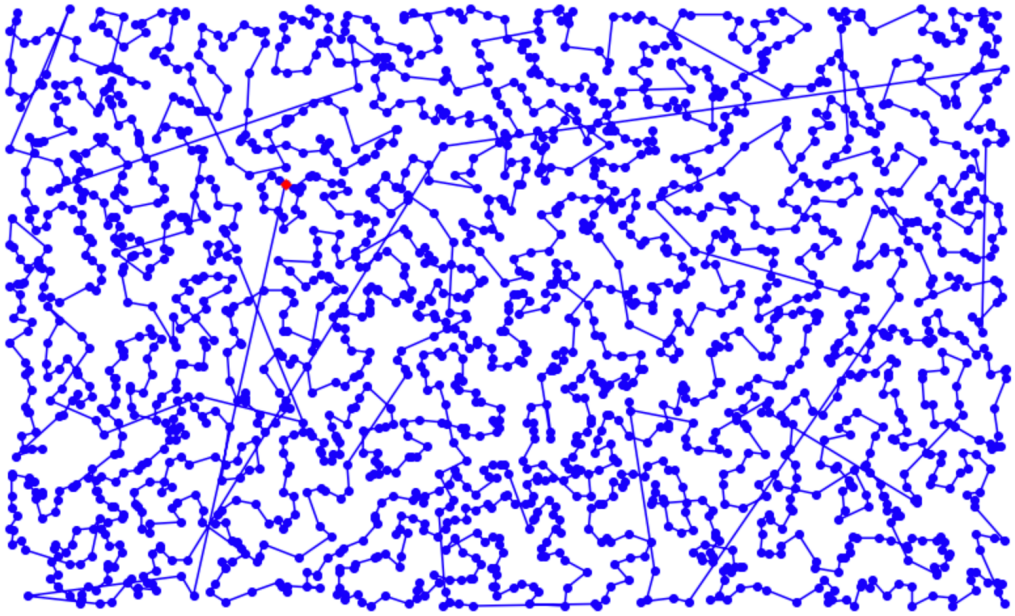
```
[ ]  tsp(brute_force, generate_cities(10))
```

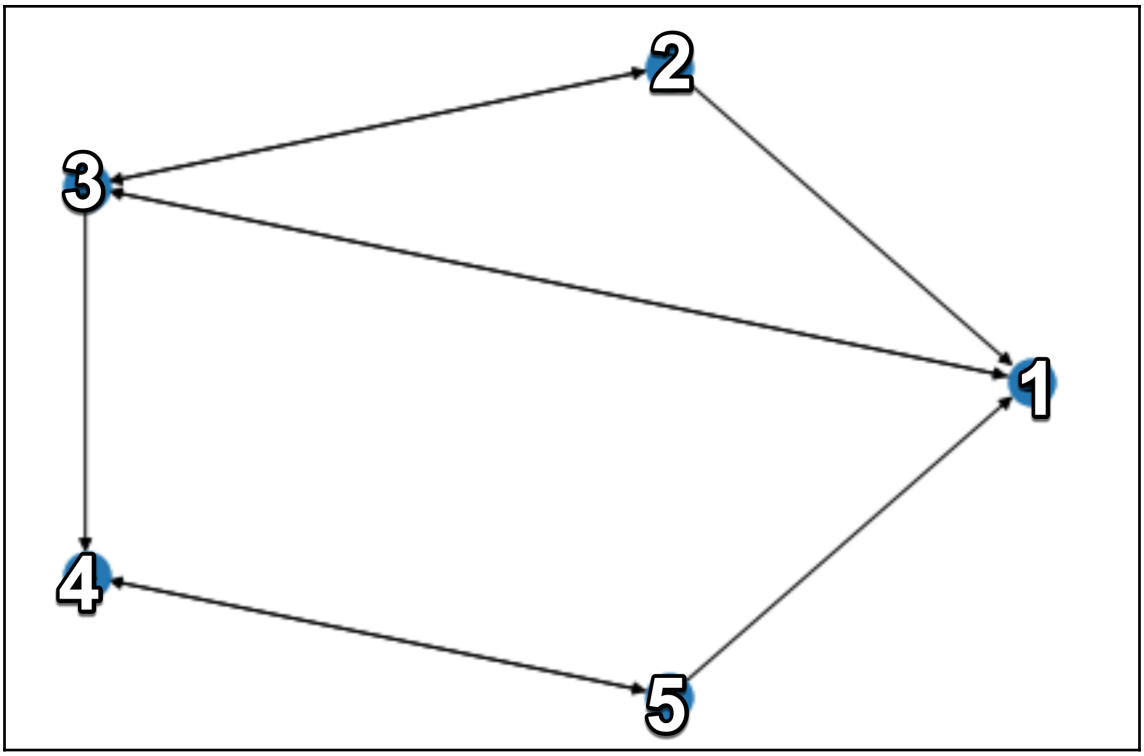
```
brute_force: 10 cities ⇒ tour length 1218 (in 10.962 sec)
```



```
tsp(greedy_algorithm, generate_cities(2000))
```

```
nn: 1991 cities ⇒ tour length 15846 (in 0.514 sec)
```






```
def createPageRank(aGraph):
  nodes_set = len(aGraph)
  M = nx.to_numpy_matrix(aGraph)
  outwards = np.squeeze(np.asarray(np.sum(M, axis=1)))
  prob_outwards = np.array(
   [1.0/count
    if count>0 else 0.0 for count in outwards])
  G = np.asarray(np.multiply(M.T, prob_outwards))
  p = np.ones(nodes_set) / float(nodes_set)
  if np.min(np.sum(G,axis=0)) < 1.0:
    print ('WARN: G is substochastic')
  return G, p
```

```
[6]  G, p = createPageRank(myWeb)
     print (G)
```

```
[[0.        0.5       0.33333333 0.        0.5       ]
 [0.        0.        0.33333333 0.        0.        ]
 [1.        0.5       0.         0.        0.        ]
 [0.        0.        0.33333333 0.        0.5       ]
 [0.        0.        0.         1.        0.        ]]
```

1 2 3 4 5

```
In [147]: # Print our decision variable values
          print (A.varValue)
          print (B.varValue)
```

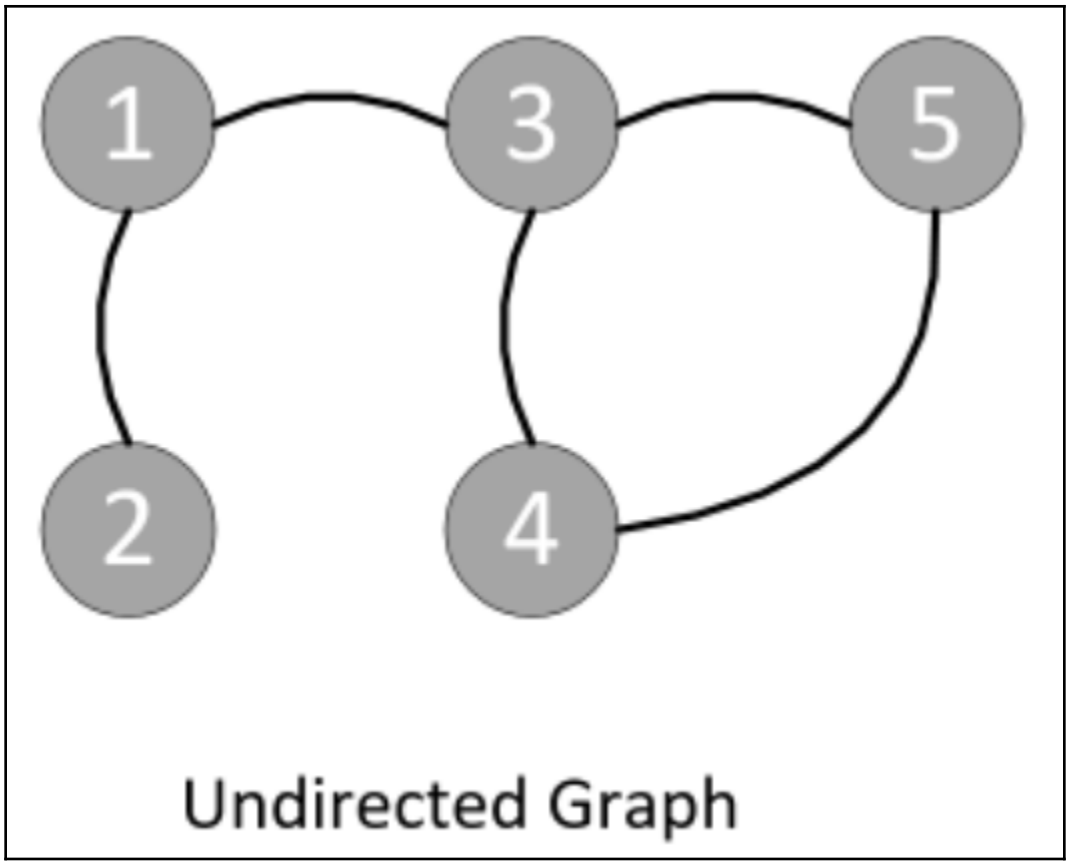
```
6.0
1.0
```

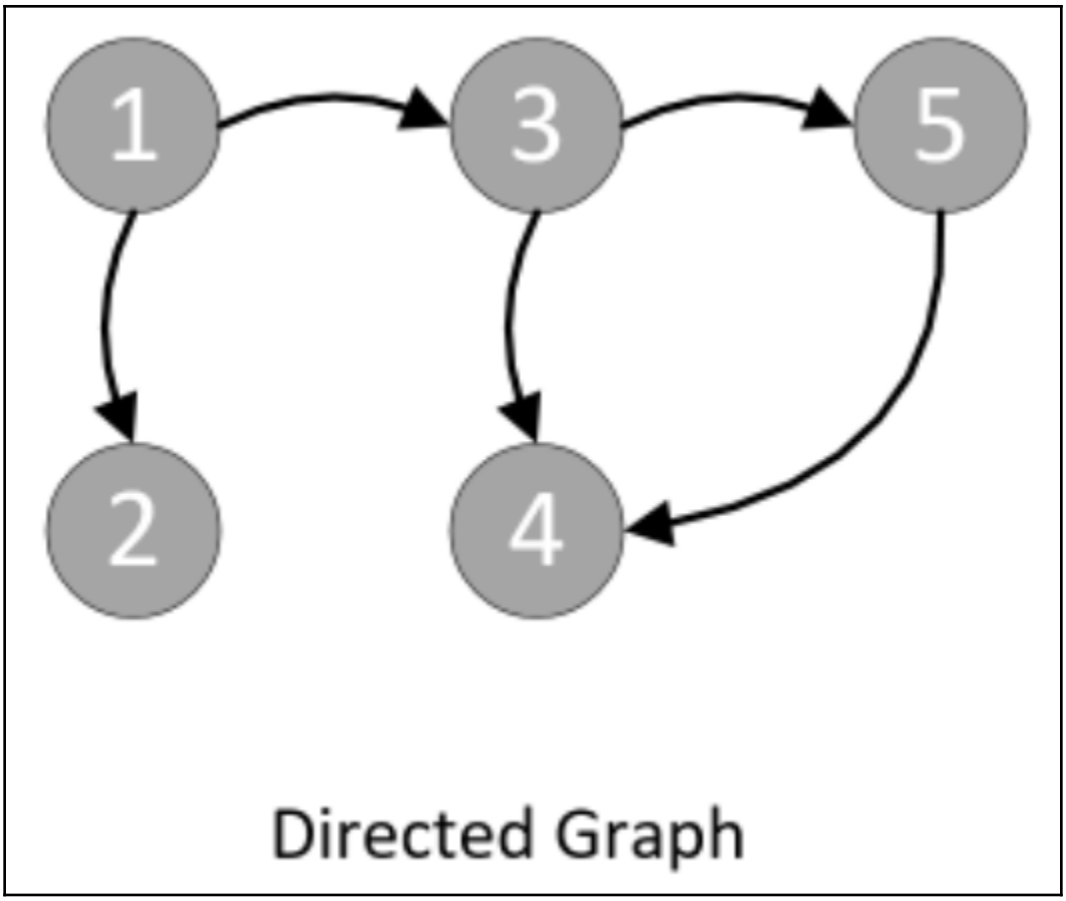
```
In [148]: # Print our objective function value
          print (pulp.value(model.objective))
```

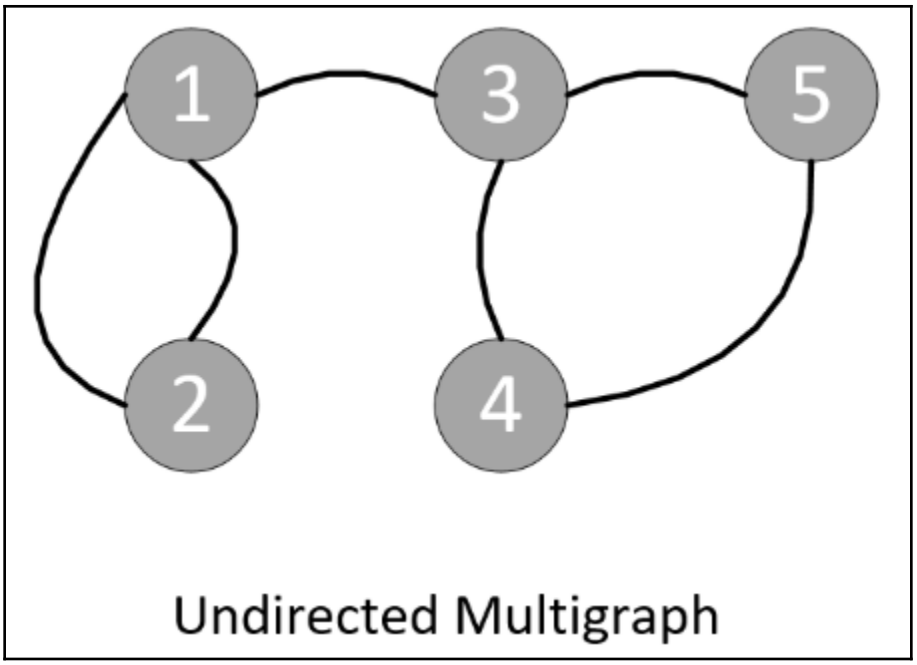
```
32500.0
```

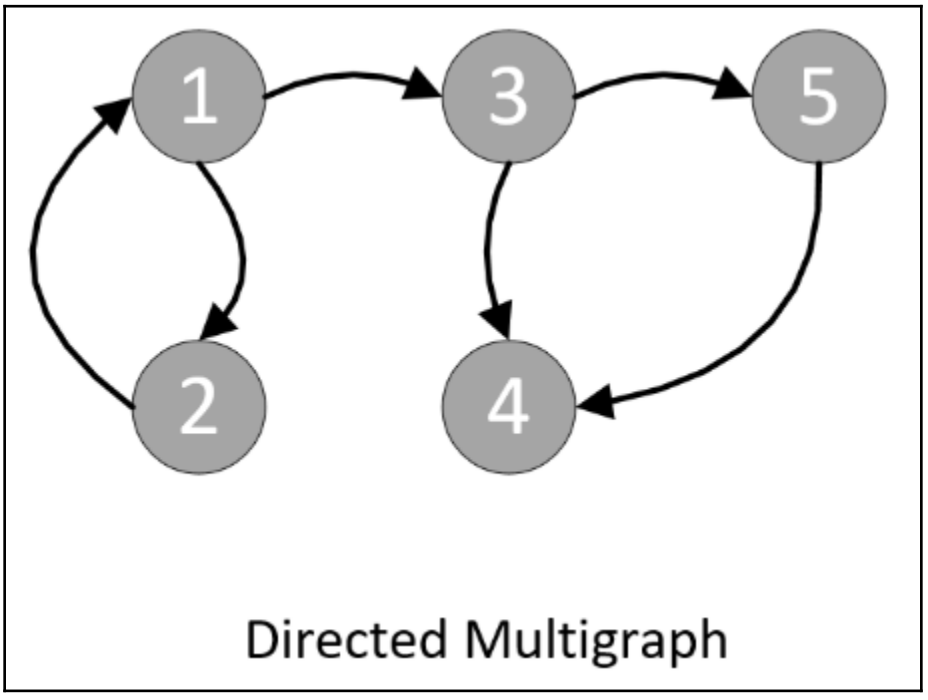
```
In [5]: 1 list(G.nodes)
Out[5]: ['Mike', 'Amine', 'Wassim', 'Nick']

In [6]: 1 list(G.edges)
Out[6]: [('Mike', 'Amine')]
```

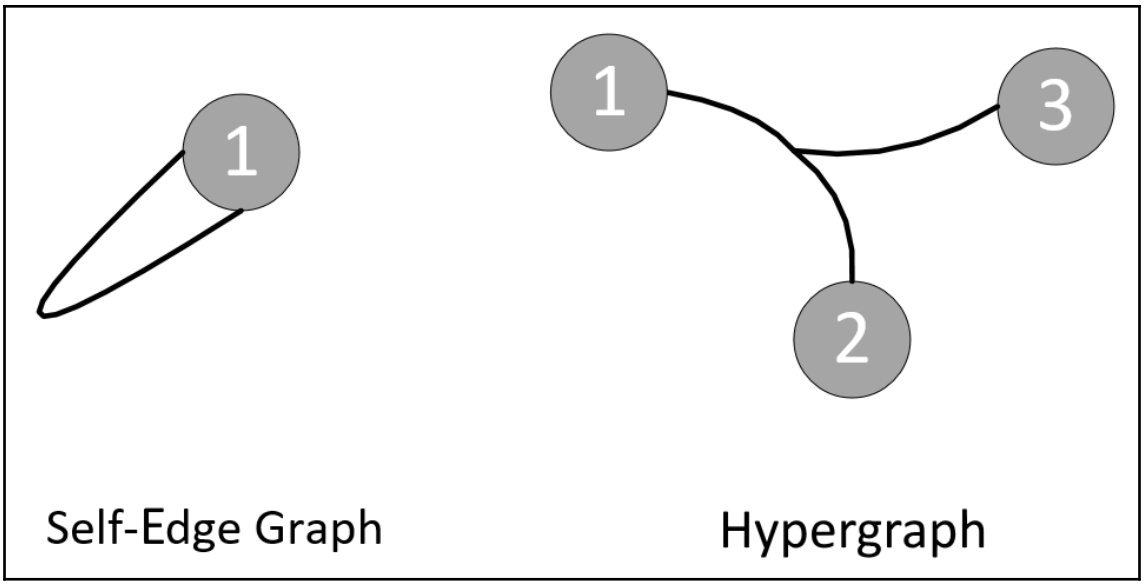
```
In [9]: 1 list(G.edges)
Out[9]: [('Mike', 'Amine'), ('Amine', 'Imran')]
```

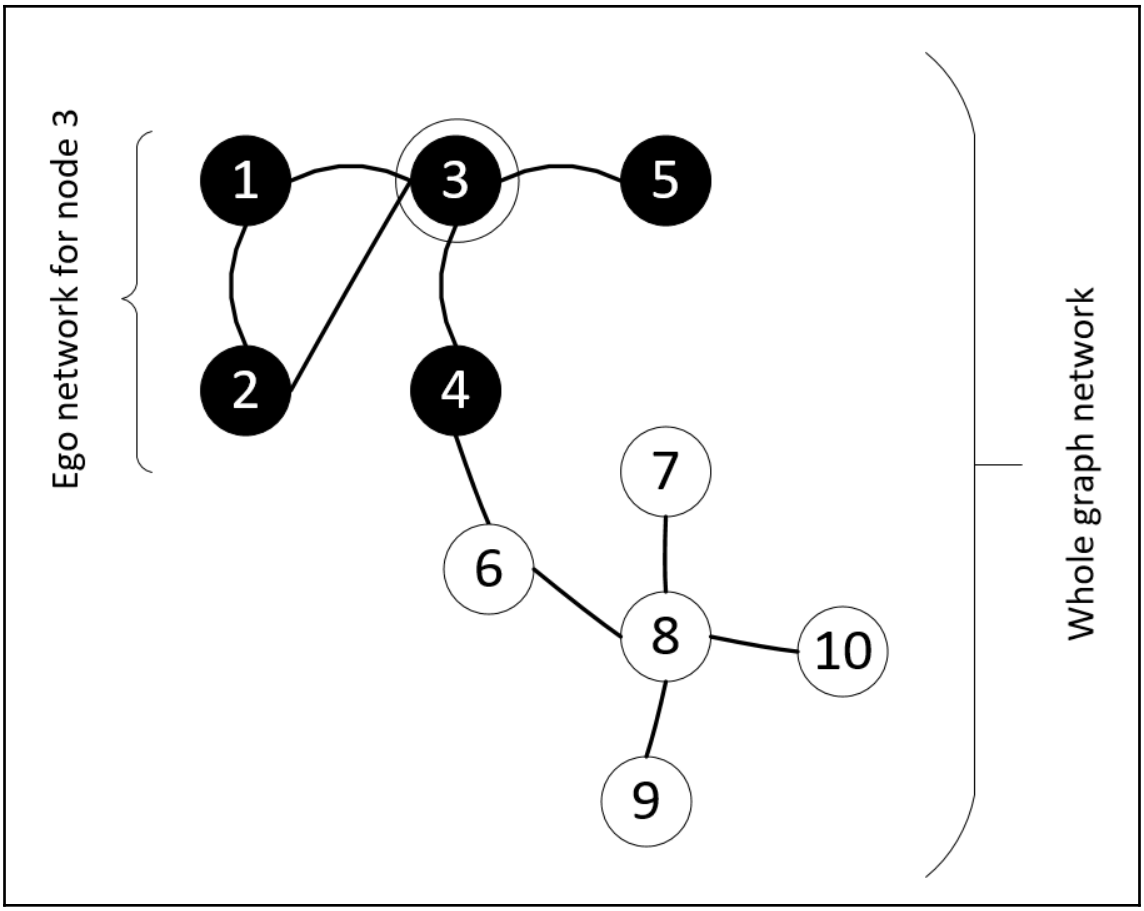

1
3
5
2
4
Undirected Graph


1
3
5
2
4
Directed Graph


1
3
5
2
4
Undirected Multigraph


1
3
5
2
4
Directed Multigraph

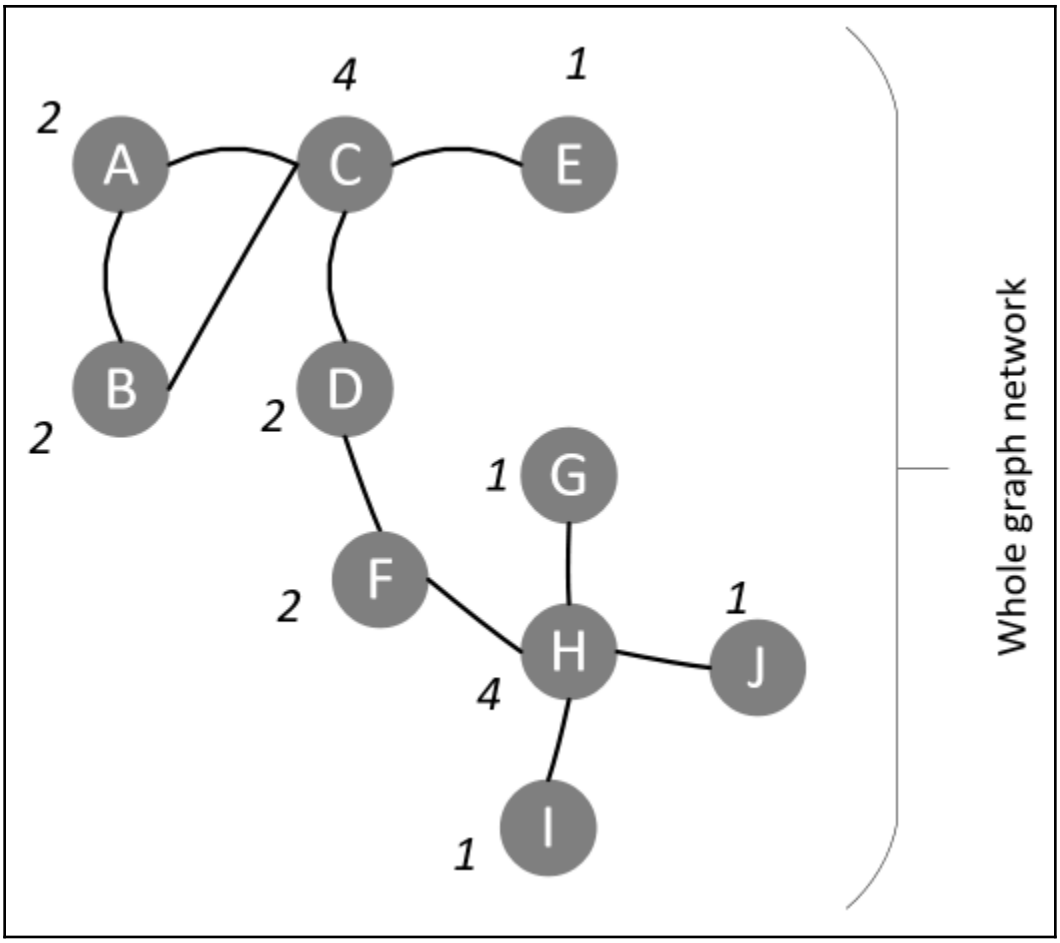

1
1
3
2
Self-Edge Graph
Hypergraph



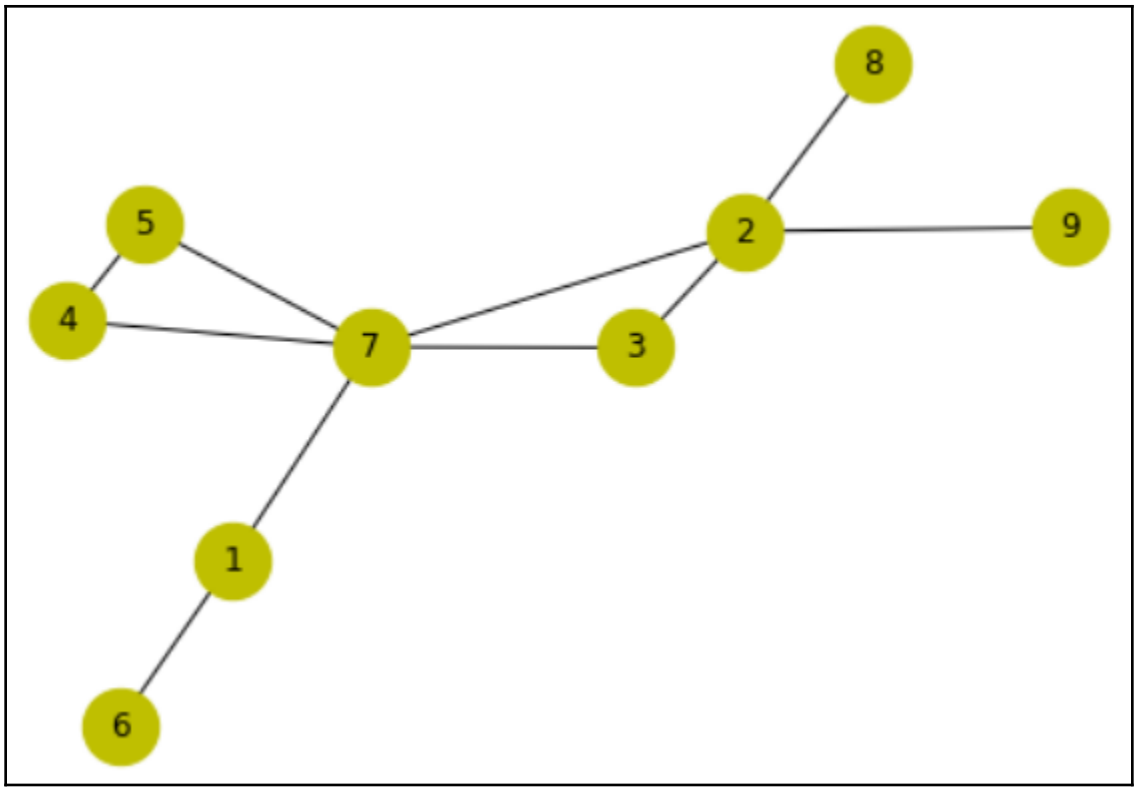
$$Edges_{total} = \binom{N}{2} = \frac{N(N-1)}{2}$$

$$density = \frac{Edges_{observed}}{Edges_{total}}$$

$$C_{DC_a} = \frac{deg(a)}{|\mathcal{V}| - 1}$$



$$C_{DC_c} = \frac{deg(c)}{|\mathscr{V}| - 1} = \frac{4}{10 - 1} = 0.44$$


8
5
2
9
4
7
3
1
6

```
In [8]:
1 nx.degree_centrality(G)
```

```
Out[8]: {1: 0.25,
 2: 0.5,
 3: 0.25,
 4: 0.25,
 5: 0.25,
 6: 0.125,
 7: 0.625,
 8: 0.125,
 9: 0.125}
```

```
In [9]:
1 nx.betweenness_centrality(G)
```

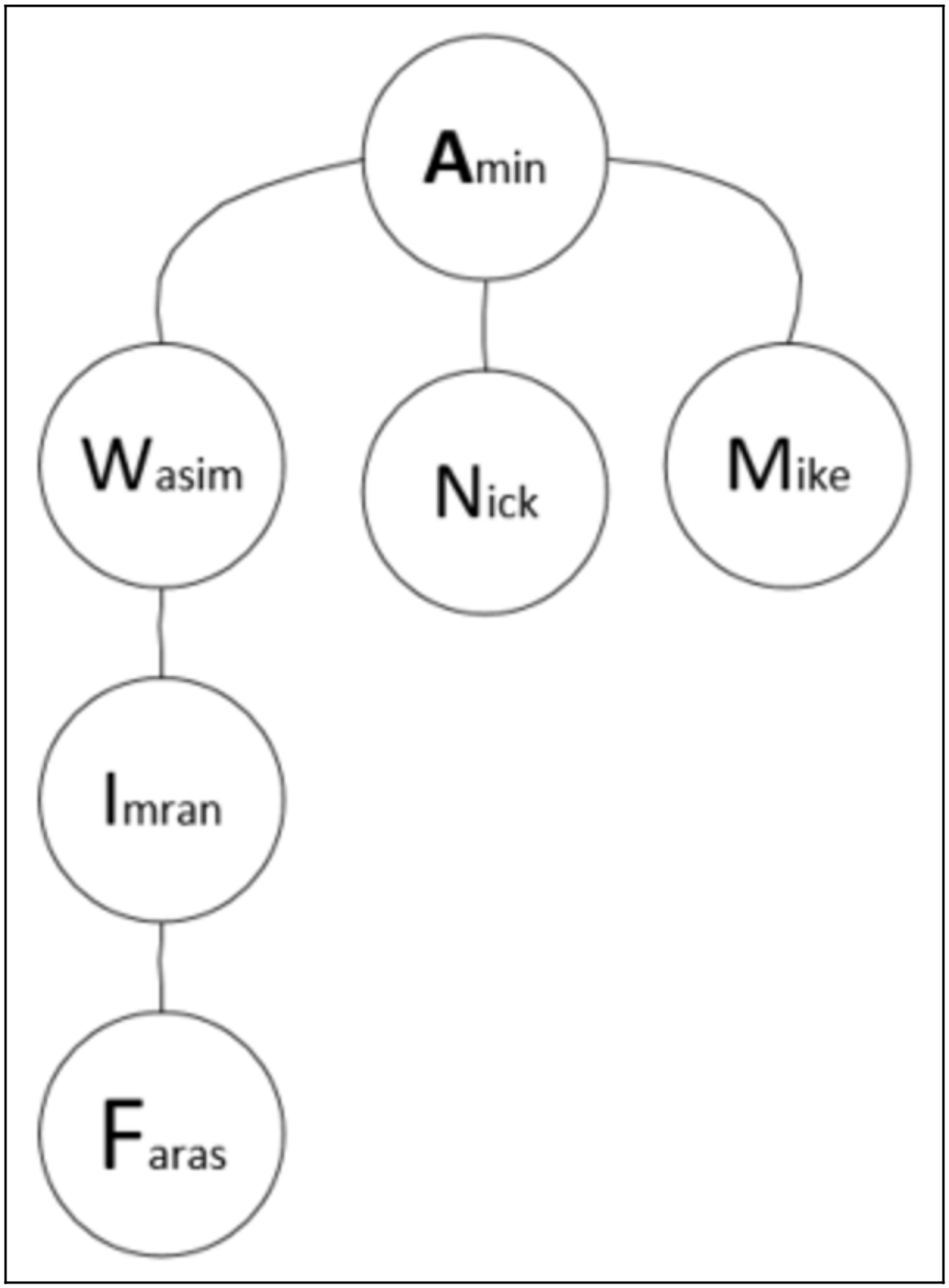
```
Out[9]: {1: 0.25,
 2: 0.46428571428571425,
 3: 0.0,
 4: 0.0,
 5: 0.0,
 6: 0.0,
 7: 0.7142857142857142,
 8: 0.0,
 9: 0.0}
```

```
In [10]:
1 nx.closeness_centrality(G)
```

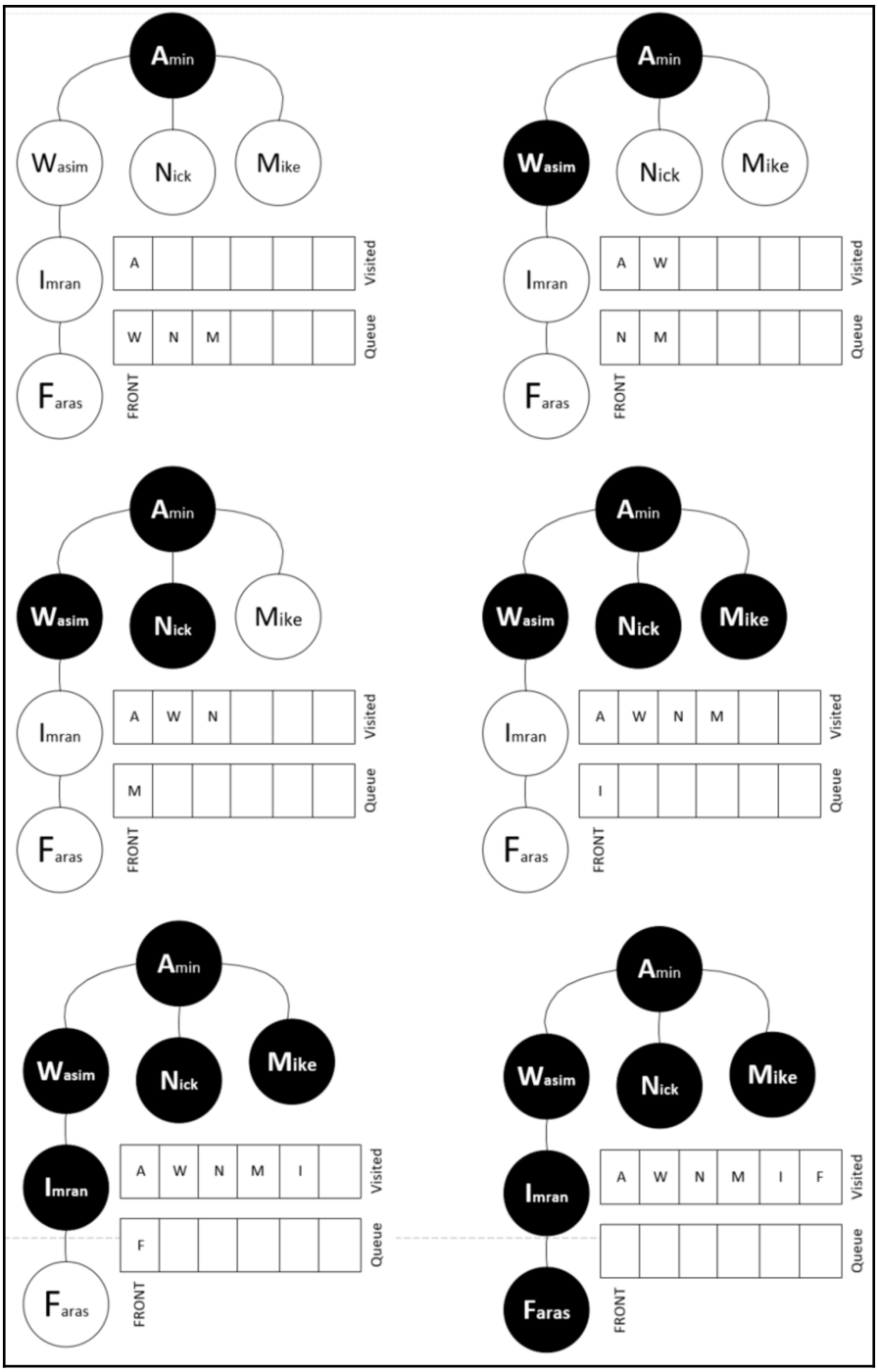
```
Out[10]: {1: 0.5,
 2: 0.6153846153846154,
 3: 0.5333333333333333,
 4: 0.47058823529411764,
 5: 0.47058823529411764,
 6: 0.34782608695652173,
 7: 0.7272727272727273,
 8: 0.4,
 9: 0.4}
```

```
In [11]:
1 centrality = nx.eigenvector_centrality(G)
2 sorted((v, '{:0.2f}'.format(c)) for v, c in centrality.items())
```

```
Out[11]: [(1, '0.24'),
 (2, '0.45'),
 (3, '0.36'),
 (4, '0.32'),
 (5, '0.32'),
 (6, '0.08'),
 (7, '0.59'),
 (8, '0.16'),
 (9, '0.16')]
```

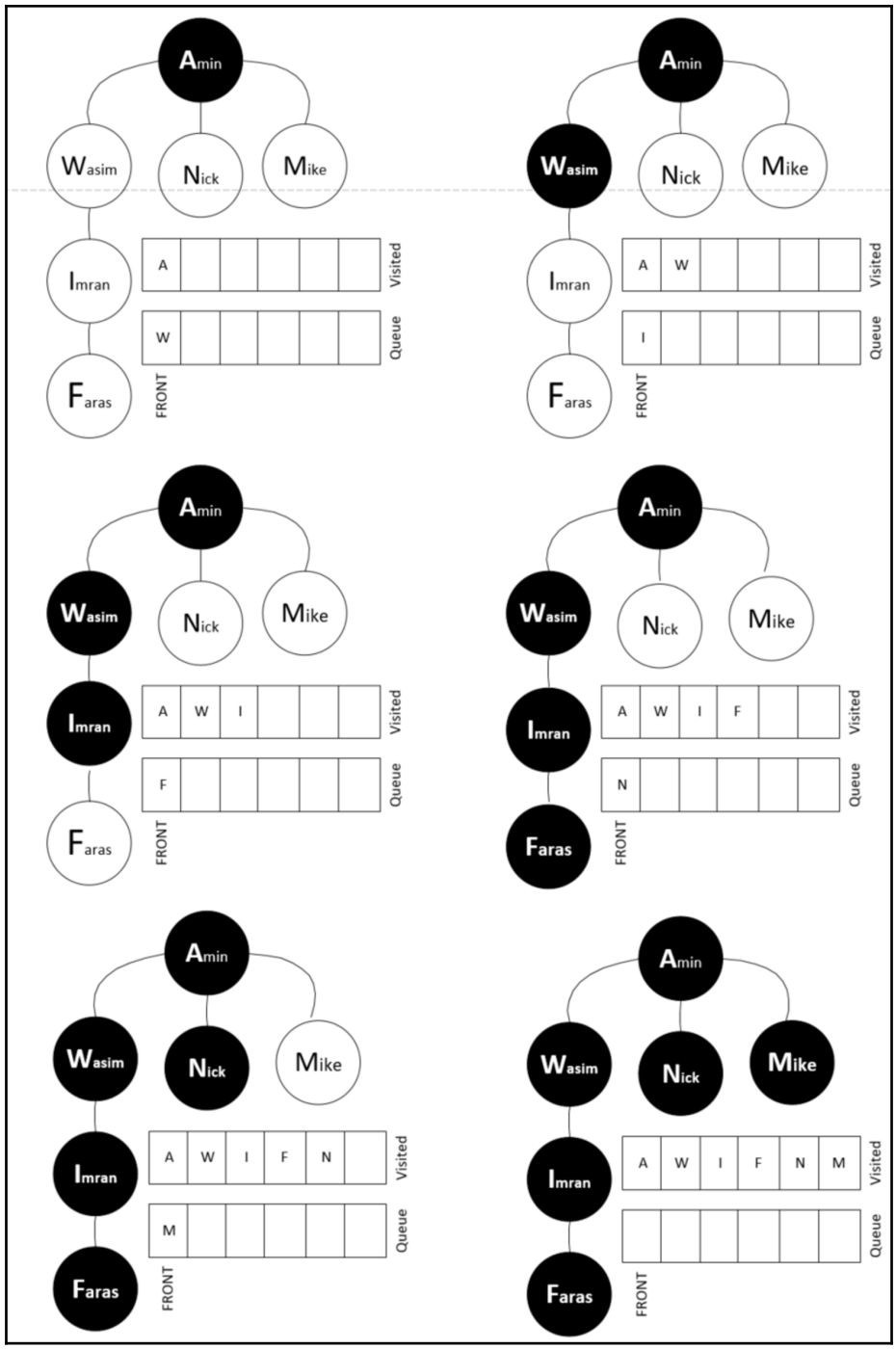

Amin
Wasim
Nick
Mike
Imran
Faras

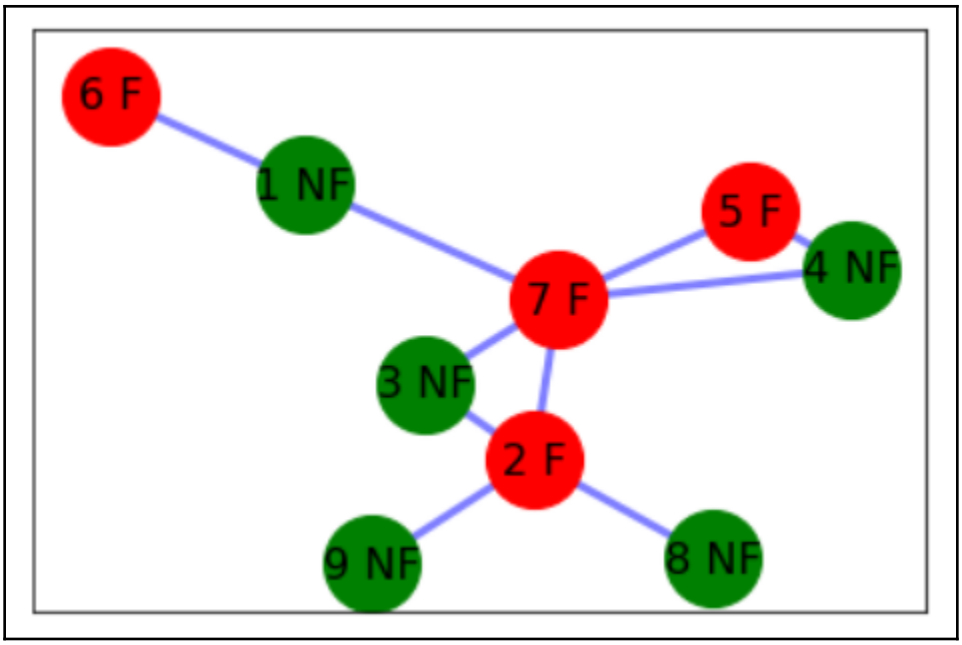
```
def bfs(graph, start):
    visited = []
    queue = [start]

    while queue:
        node = queue.pop(0)
        if node not in visited:
            visited.append(node)
            neighbours = graph[node]
            for neighbour in neighbours:
                queue.append(neighbour)
    return visited
```


Amin
Wasim
Nick
Mike
Imran
Faras
A
W
N
M
I
F
Visited
Queue
FRONT

```
In [97]: bfs(graph,'Amin')
Out[97]: ['Amin', 'Wasim', 'Nick', 'Mike', 'Imran', 'Faras']
```

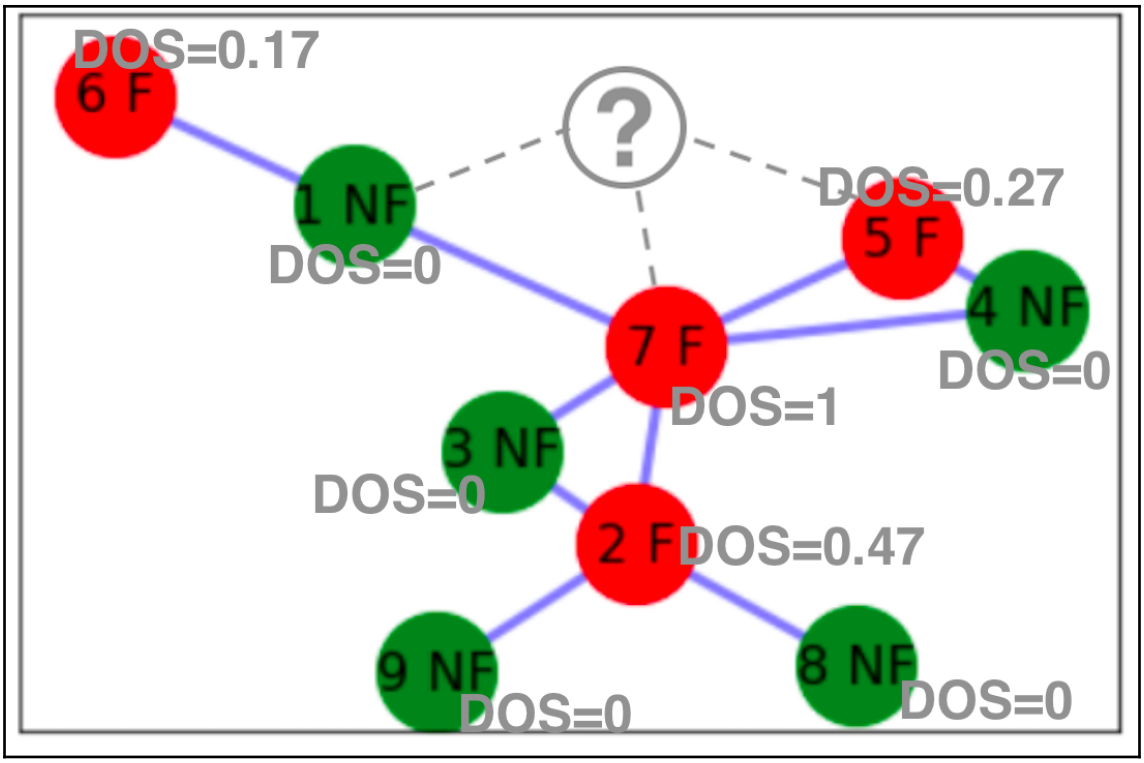
```
Out[94]: {'Amin', 'Faras', 'Imran', 'Mike', 'Nick', 'Wasim'}
```


Amin
Wasim
Nick
Mike
Imran
Faras
A
W
I
F
N
M
Visited
Queue
FRONT



$$P(F|q) = \frac{1}{degree_q} \sum_{n_j \in Neighborhood_n | class(n_j)=F} w(n, n_j) DOS_{normalized_j}$$

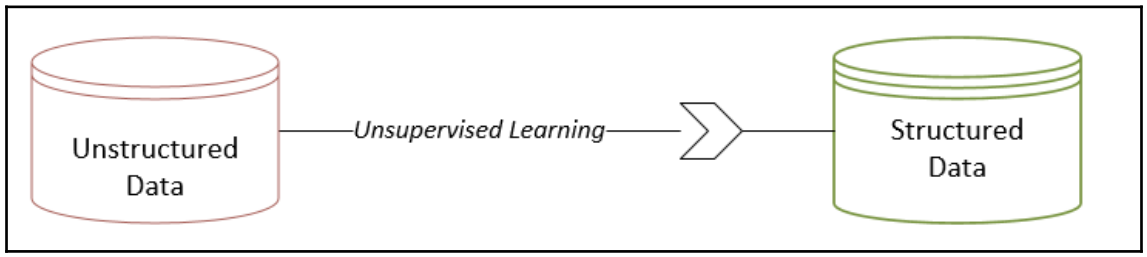
$$P(F|q) = \frac{1+1}{3} = \frac{2}{3} = .67$$

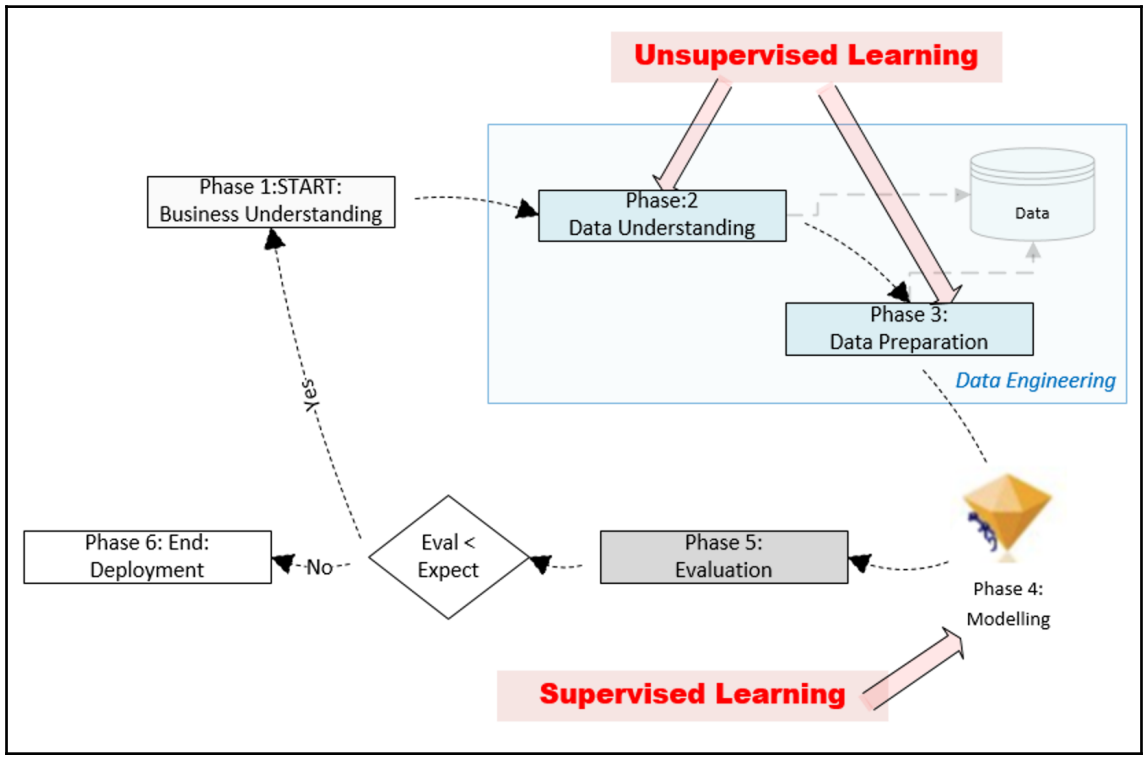


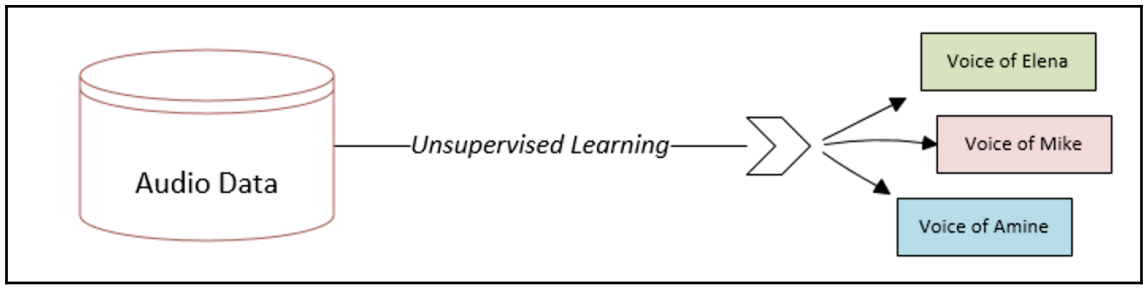
$$DOS_k = \frac{1}{degree_k} \sum_{n_j \in Neighborhood_n} w(n, n_j) DOS_{normalized_j}$$

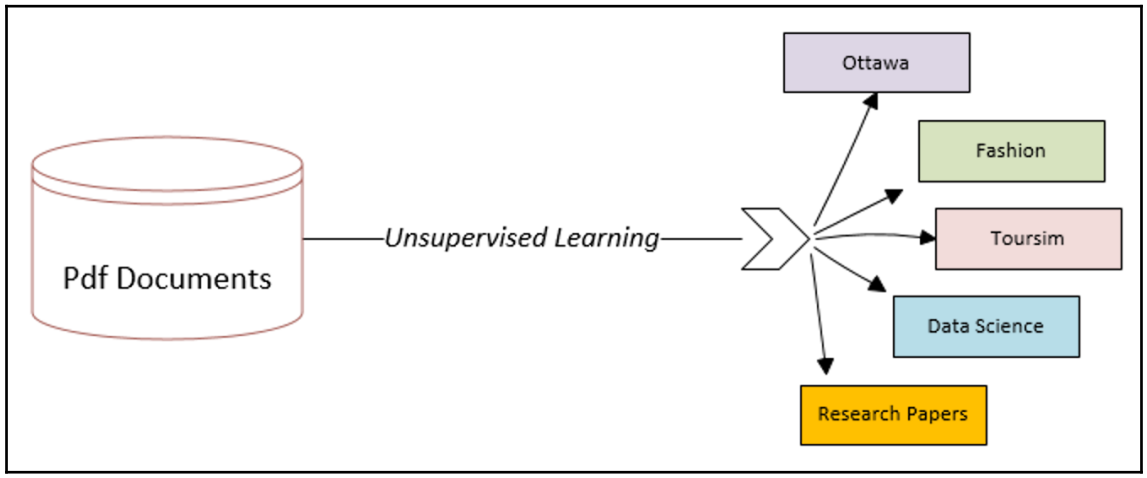
$$DOS_k = \frac{(0 + 1 + 0.27)}{3} = 0.42$$

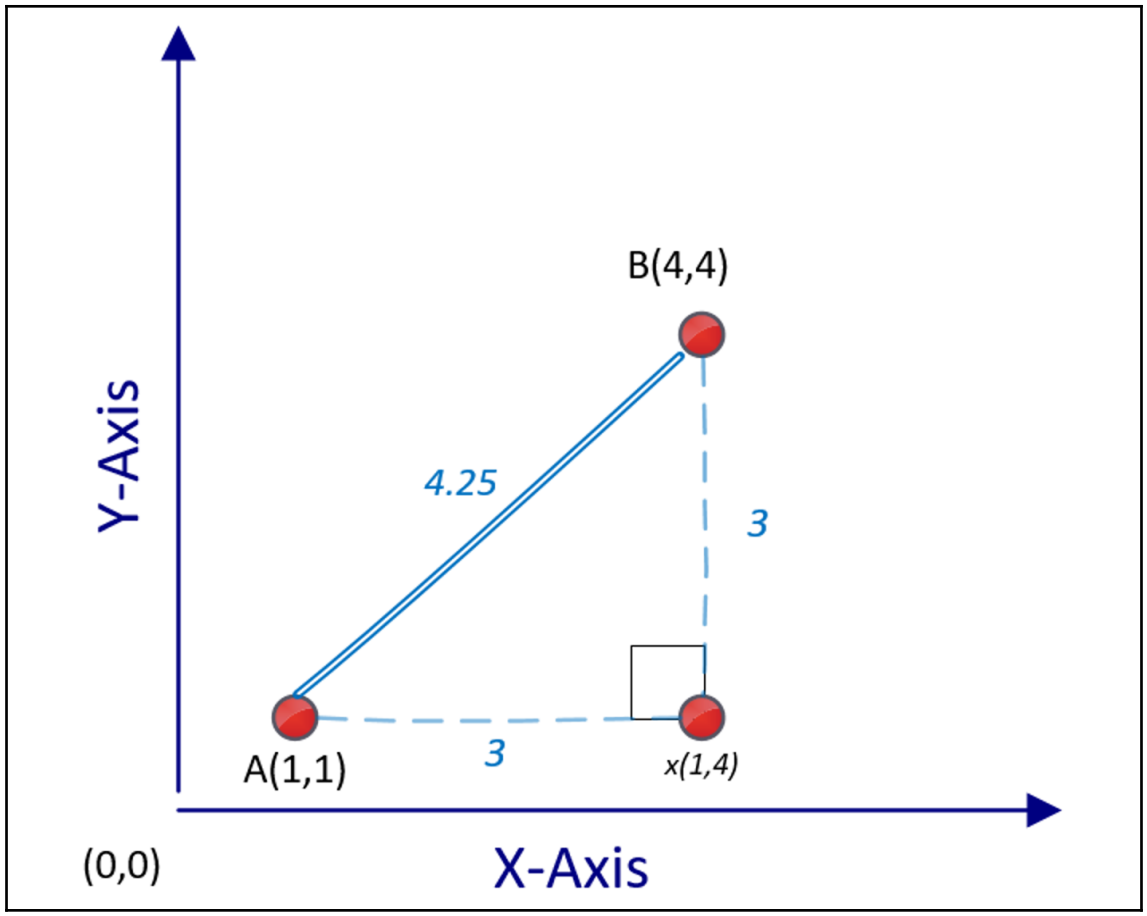
Chapter 6: Unsupervised Machine Learning Algorithms





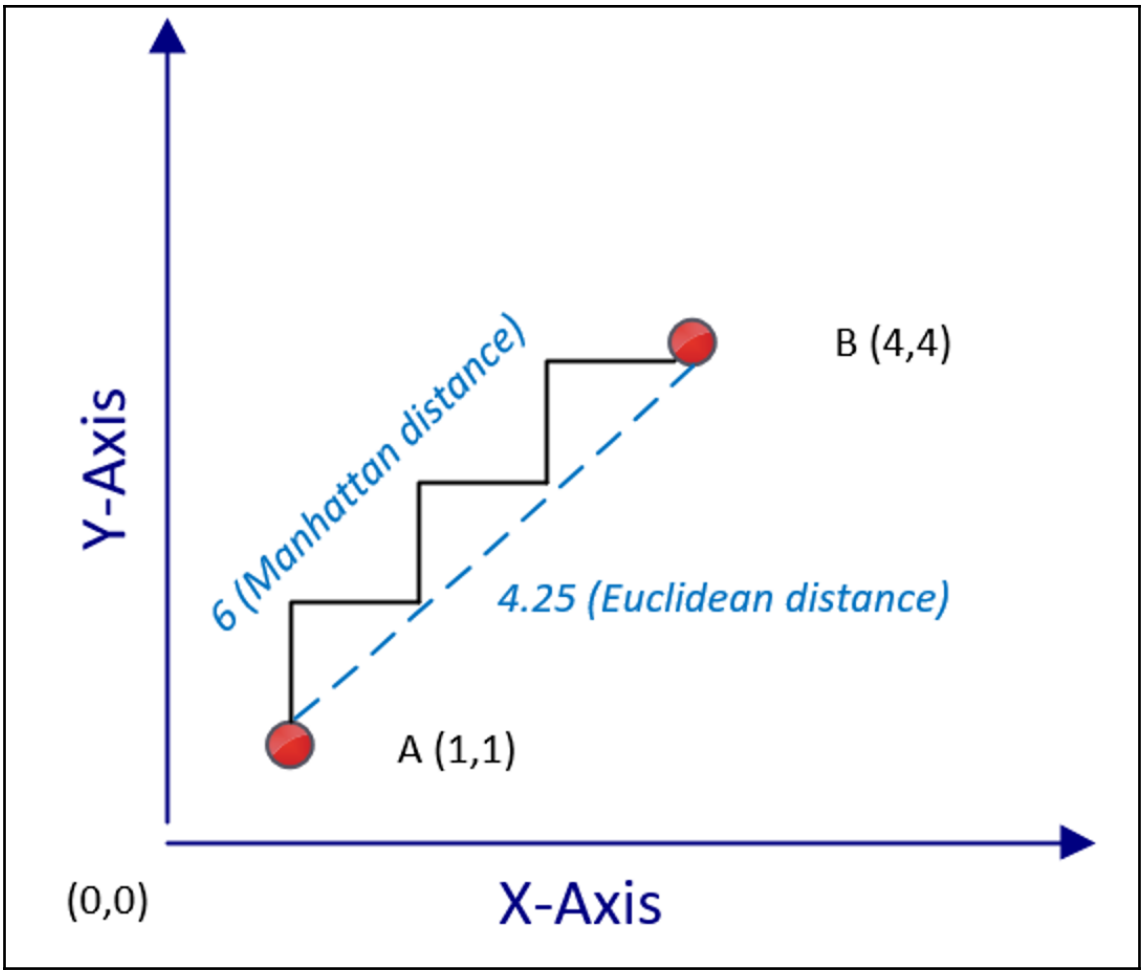

Audio Data
Unsupervised Learning
Voice of Elena
Voice of Mike
Voice of Amine

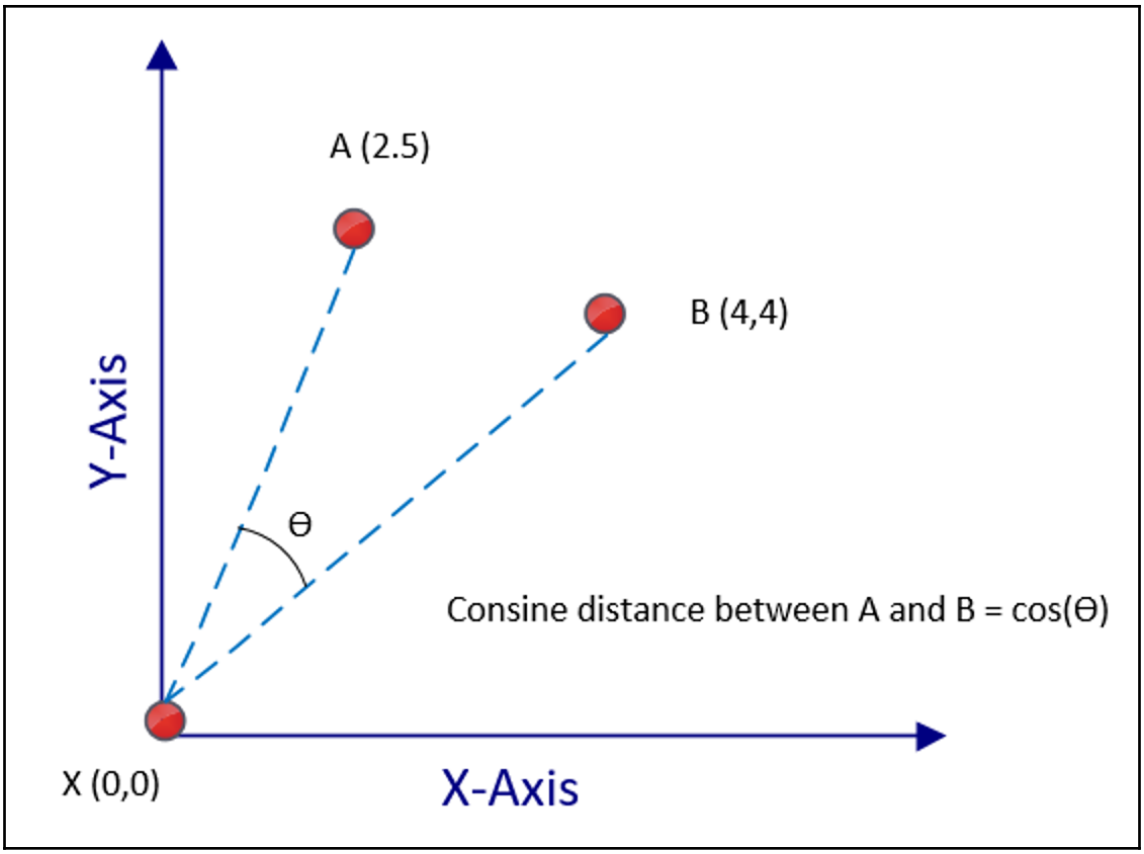

Pdf Documents
Unsupervised Learning
Ottawa
Fashion
Toursim
Data Science
Research Papers

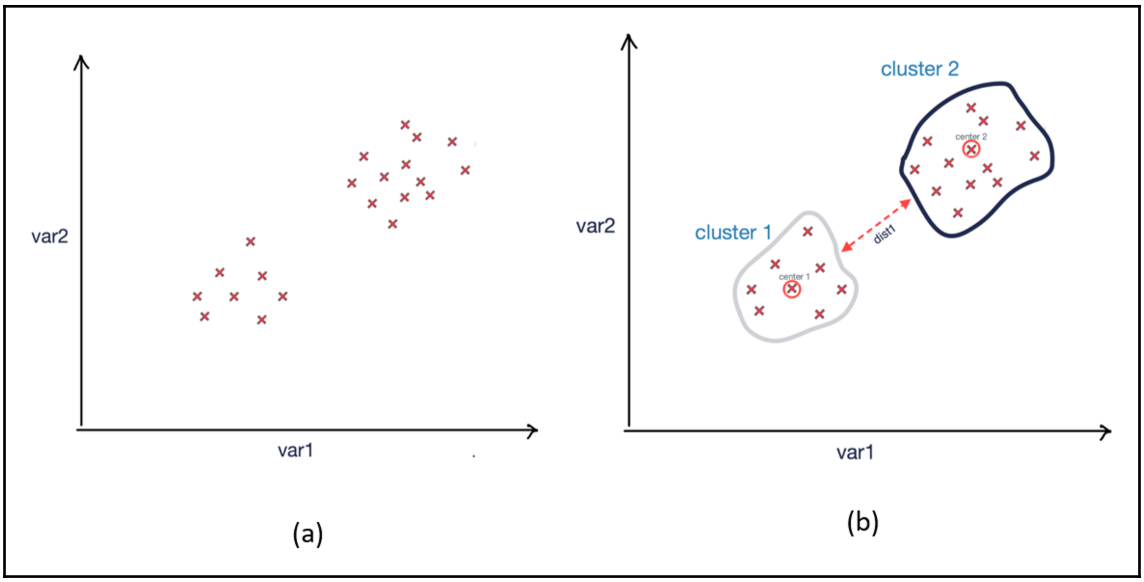


$$d(A,B) = \sqrt{(a_2 - b_2)^2 + (a_1 - b_1)^2} = \sqrt{(4-1)^2 + (4-1)^2} = \sqrt{9+9} = 4.25$$

$$d(A,B) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

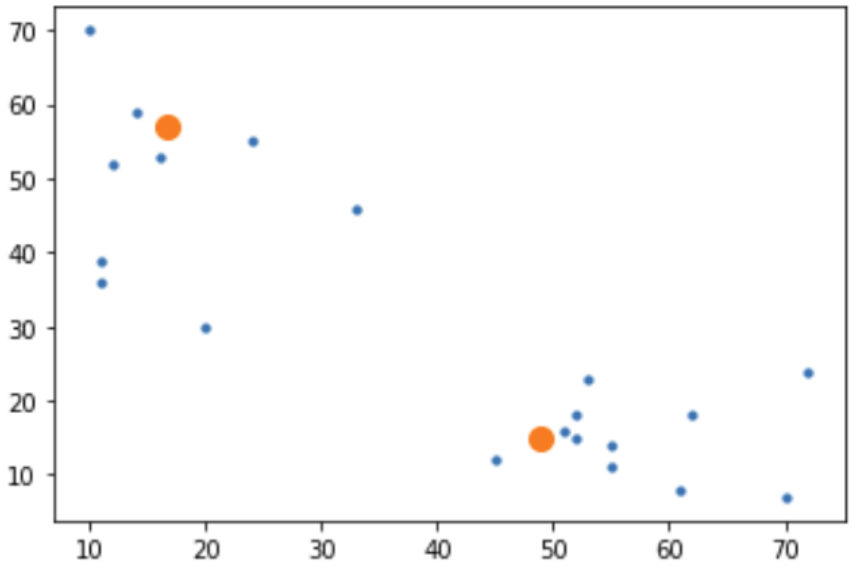

Y-Axis
6 (Manhattan distance)
B (4,4)
4.25 (Euclidean distance)
A (1,1)
(0,0)
X-Axis

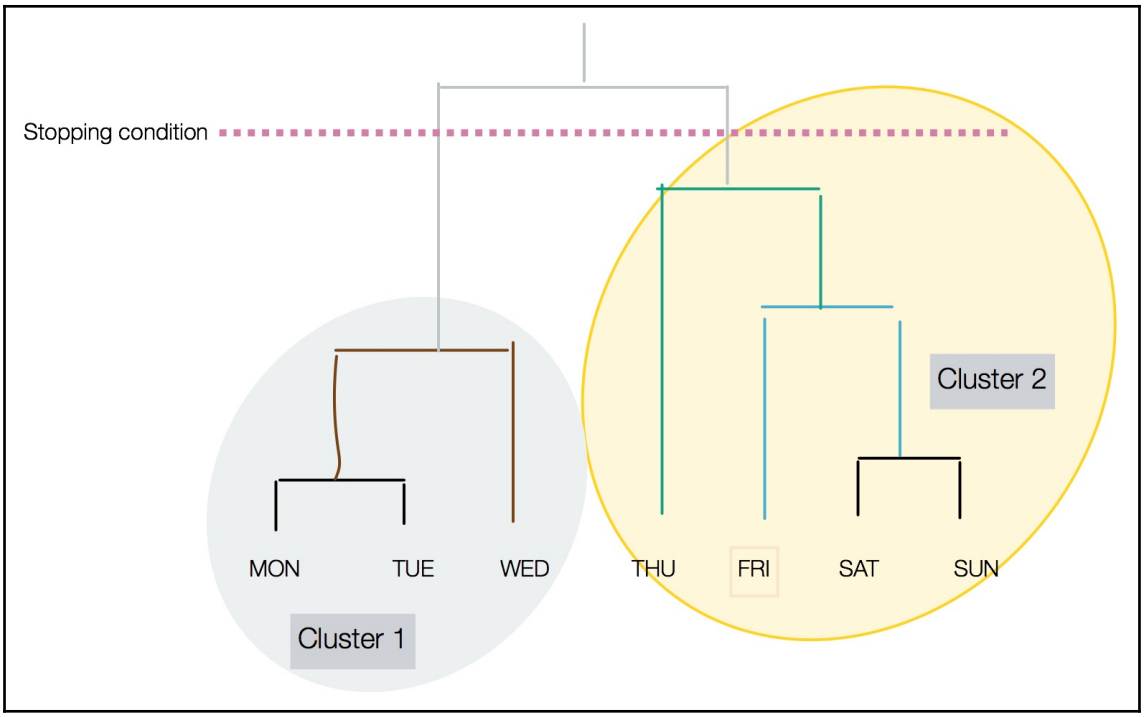

A (2.5)
B (4,4)
Y-Axis
Θ
Consine distance between A and B = cos(Θ)
X (0,0)
X-Axis



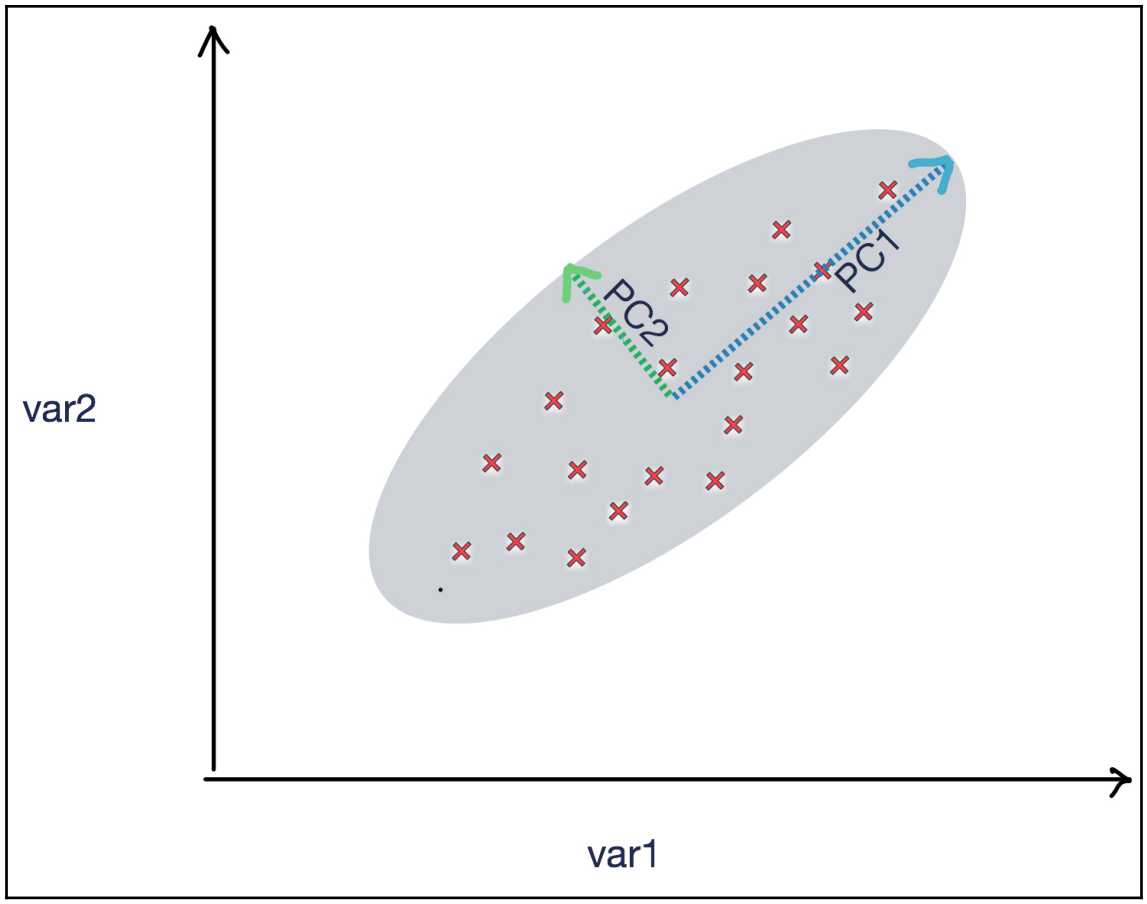

```
In [3]: print(labels)

[1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1]

In [6]:
print(centroids)

[[57.09090909 15.09090909]
 [16.77777778 48.88888889]]
```

In [16]:
```
import matplotlib.pyplot as plt
plt.scatter(dataset['x'],dataset['y'], s=10)
plt.scatter(centers[0],centers[1],s=100)
plt.show()
```






```
In [3]:  1 print(cluster.labels_)

[0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0]
```




```
In [36]: print(pd.DataFrame(pca.components_,columns=X.columns))
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	
0	0.361387	-0.084523	0.856671	0.358289	Coefficients for PC1
1	0.656589	0.730161	-0.173373	-0.075481	Coefficients for PC2
2	-0.582030	0.597911	0.076236	0.545831	Coefficients for PC3
3	-0.315487	0.319723	0.479839	-0.753657	Coefficients for PC4

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	PC1	PC2	PC3	PC4
0	5.1	3.5	1.4	0.2	2.818240	5.646350	-0.659768	0.031089
1	4.9	3.0	1.4	0.2	2.788223	5.149951	-0.842317	-0.065675
2	4.7	3.2	1.3	0.2	2.613375	5.182003	-0.613952	0.013383
3	4.6	3.1	1.5	0.2	2.757022	5.008654	-0.600293	0.108928
4	5.0	3.6	1.4	0.2	2.773649	5.653707	-0.541773	0.094610
...	...	...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	7.446475	5.514485	-0.454028	-0.392844
146	6.3	2.5	5.0	1.9	7.029532	4.951636	-0.753751	-0.221016
147	6.5	3.0	5.2	2.0	7.266711	5.405811	-0.501371	-0.103650
148	6.2	3.4	5.4	2.3	7.403307	5.443581	0.091399	-0.011244
149	5.9	3.0	5.1	1.8	6.892554	5.044292	-0.268943	0.188390

```
In [37]: print(pca.explained_variance_ratio_)
```

```
[0.92461872 0.05306648 0.01710261 0.00521218]
```

$$\pi$$

$$\pi$$

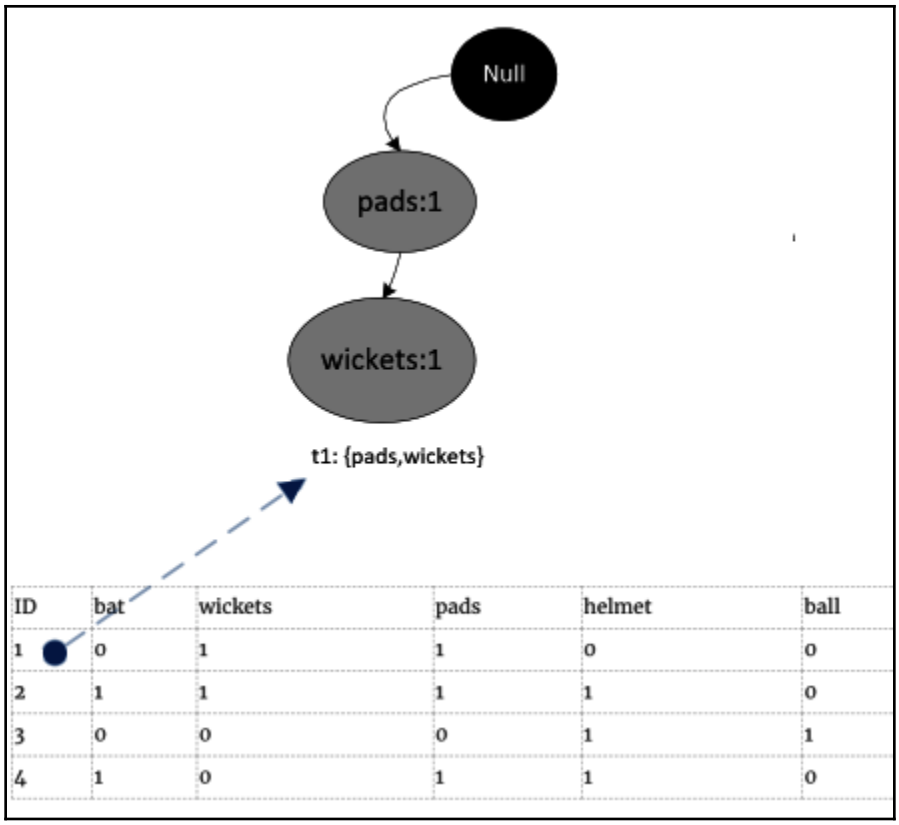
$$X \bigcap Y = \varnothing$$

$$support(itemset_a) = \frac{numItemset_a}{num_{total}}$$

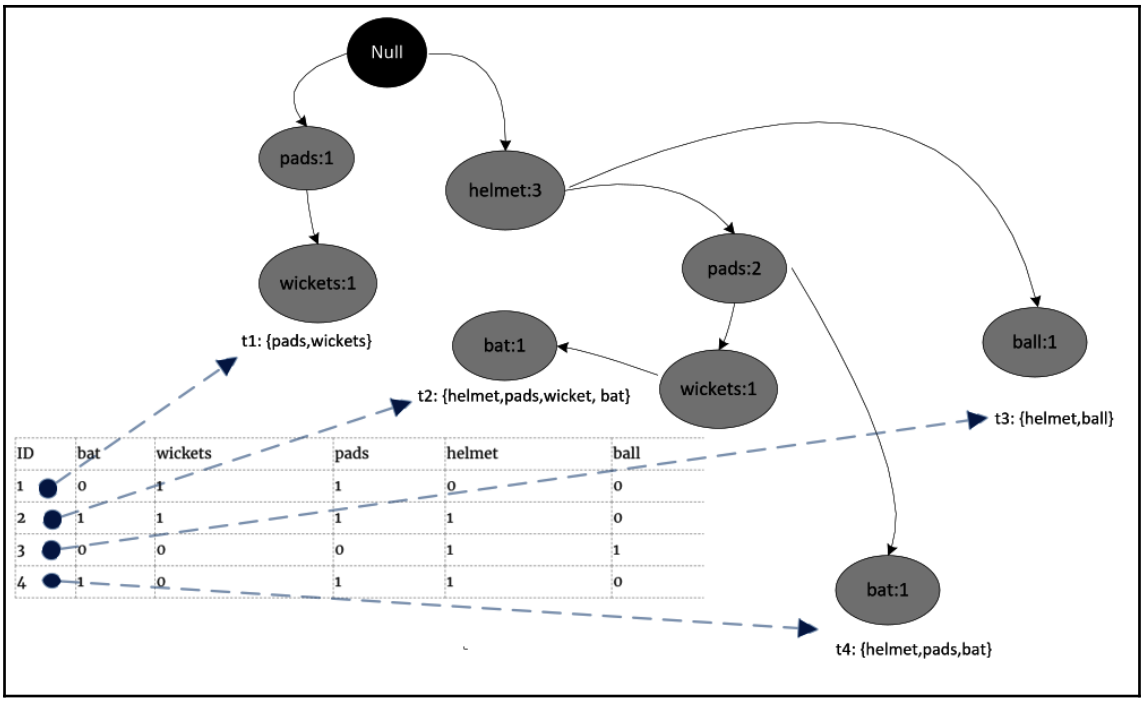
$$confidence(X \Rightarrow Y) = \frac{support(X \cup Y)}{support(X)}$$

$$confidence(helmet, ball \Rightarrow wickets) = \frac{support(helmet, ball \cup wickets)}{support(helmet, ball)} = \frac{\frac{1}{6}}{\frac{2}{6}} = 0.5$$

$$Lift(X \Rightarrow Y) = \frac{support(X \cup Y)}{support(X) \times support(Y)}$$



ID	bat	wickets	pads	helmet	ball
1	0	1	1	0	0
2	1	1	1	1	0
3	0	0	0	1	1
4	1	0	1	1	0

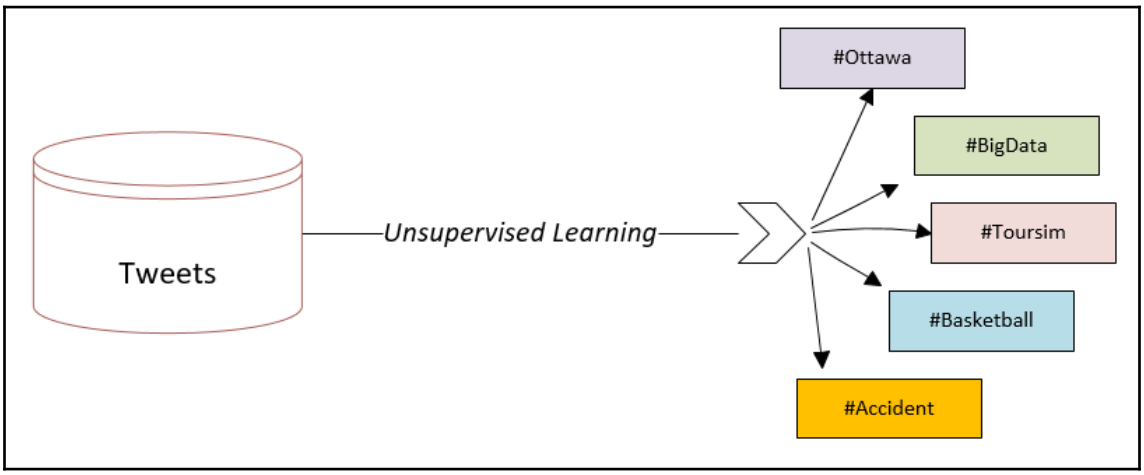


ID	bat	wickets	pads	helmet	ball
1	0	1	1	0	0
2	1	1	1	1	0
3	0	0	0	1	1
4	1	0	1	1	0

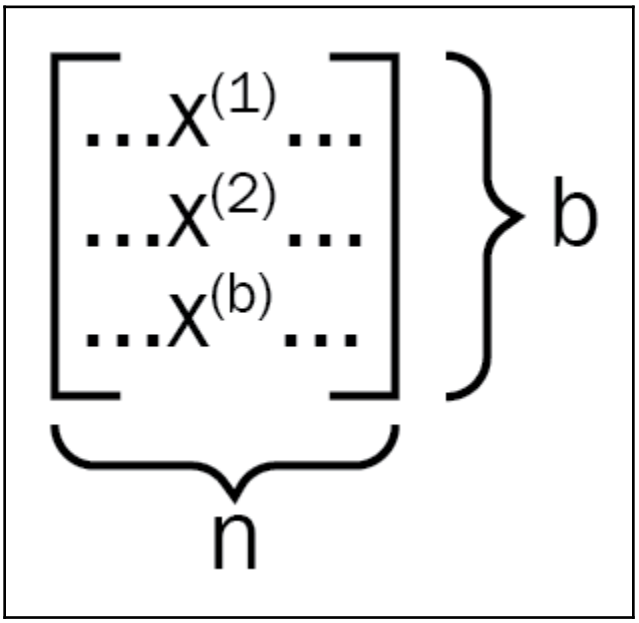
```
In [39]: patterns
```

```
Out[39]: {('pad',): 1,
          ('helmet', 'pad'): 1,
          ('wickets',): 2,
          ('pads', 'wickets'): 2,
          ('bat', 'wickets'): 1,
          ('helmet', 'wickets'): 1,
          ('bat', 'pads', 'wickets'): 1,
          ('helmet', 'pads', 'wickets'): 1,
          ('bat', 'helmet', 'wickets'): 1,
          ('bat', 'helmet', 'pads', 'wickets'): 1,
          ('bat',): 2,
          ('bat', 'helmet'): 2,
          ('bat', 'pads'): 2,
          ('bat', 'helmet', 'pads'): 2,
          ('pads',): 3,
          ('helmet',): 3,
          ('helmet', 'pads'): 2}
```

```
In [22]: rules = fp.generate_association_rules(patterns,0.3)
         rules

Out[22]: {('helmet',): (('pads',), 0.6666666666666666),
          ('pad',): (('helmet',), 1.0),
          ('pads',): (('helmet',), 0.6666666666666666),
          ('wickets',): (('bat', 'helmet', 'pads'), 0.5),
          ('bat',): (('helmet', 'pads'), 1.0),
          ('bat', 'pads'): (('helmet',), 1.0),
          ('bat', 'wickets'): (('helmet', 'pads'), 1.0),
          ('pads', 'wickets'): (('bat', 'helmet'), 0.5),
          ('helmet', 'pads'): (('bat',), 1.0),
          ('helmet', 'wickets'): (('bat', 'pads'), 1.0),
          ('bat', 'helmet'): (('pads',), 1.0),
          ('bat', 'helmet', 'pads'): (('wickets',), 0.5),
          ('bat', 'helmet', 'wickets'): (('pads',), 1.0),
          ('bat', 'pads', 'wickets'): (('helmet',), 1.0),
          ('helmet', 'pads', 'wickets'): (('bat',), 1.0)}
```



Chapter 7: Traditional Supervised Learning Algorithms

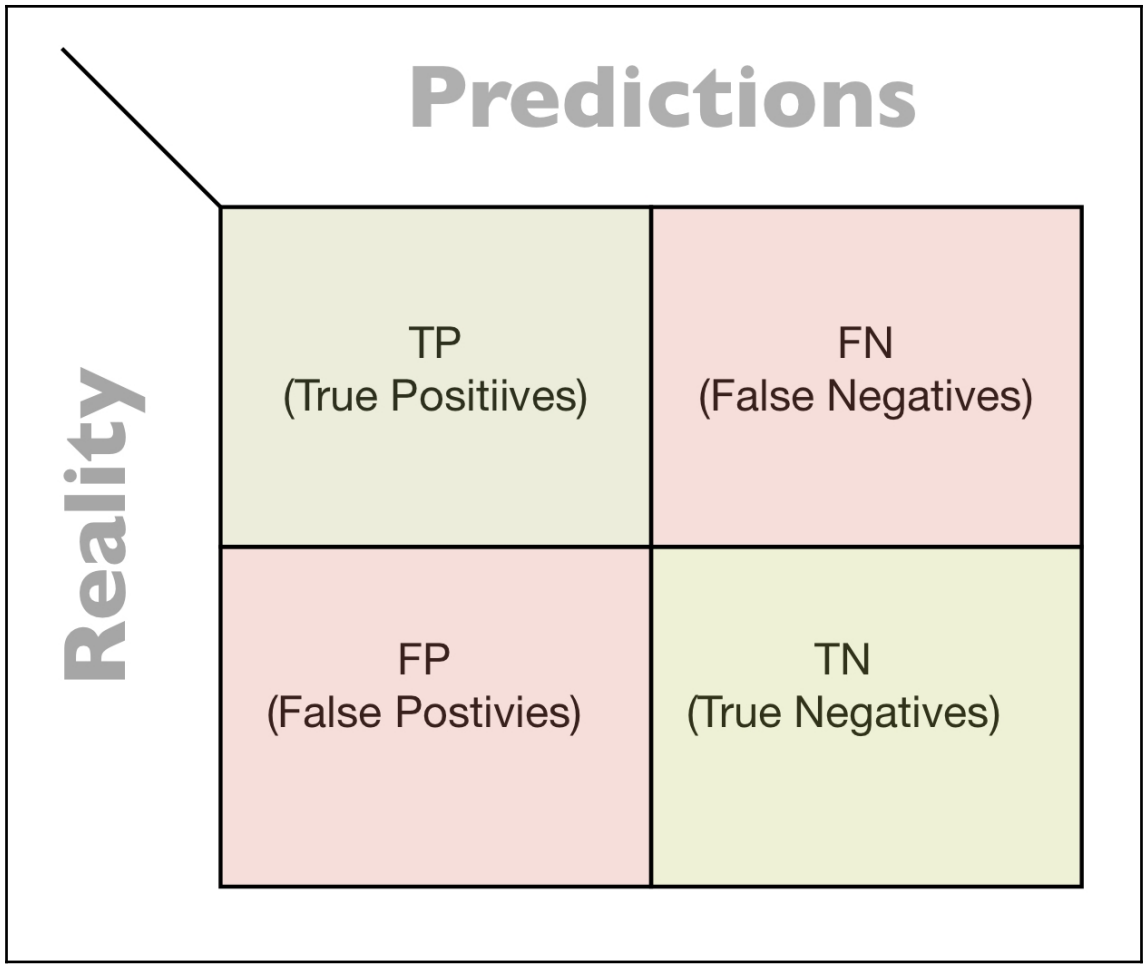


$$\acute{y}(i) \approx y(i); for 1 \leq i \leq b$$

$$\acute{y} = P(y = 1|x) : where; x \in \Re^{n_x}$$

	Gender	Age	Estimated Salary	Purchased
0	Male	19	19,000	0
1	Male	35	20,000	0
2	Female	26	43,000	0
3	Female	27	57,000	0
4	Male	19	76,000	0

	Female	Male	Age	Estimated Salary	Purchased
0	0.0	1.0	19	19,000	0
1	0.0	1.0	35	20,000	0
2	1.0	0.0	26	43,000	0
3	1.0	0.0	27	57,000	0
4	0.0	1.0	19	76,000	0

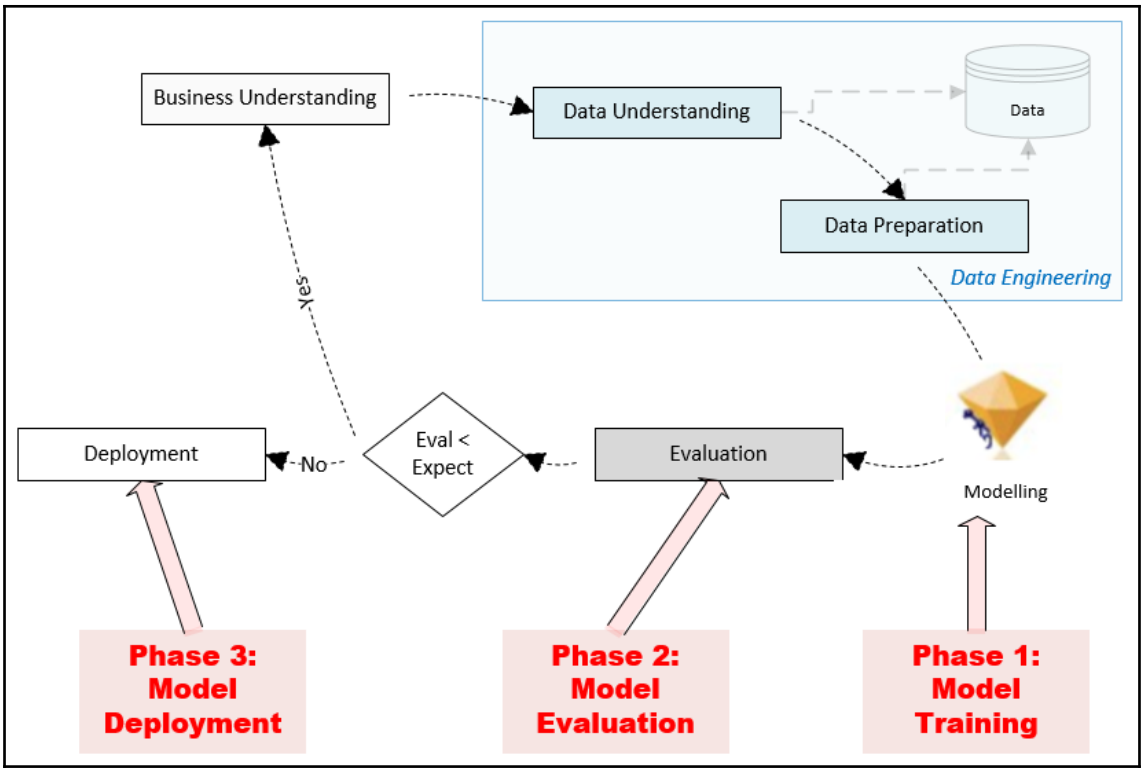


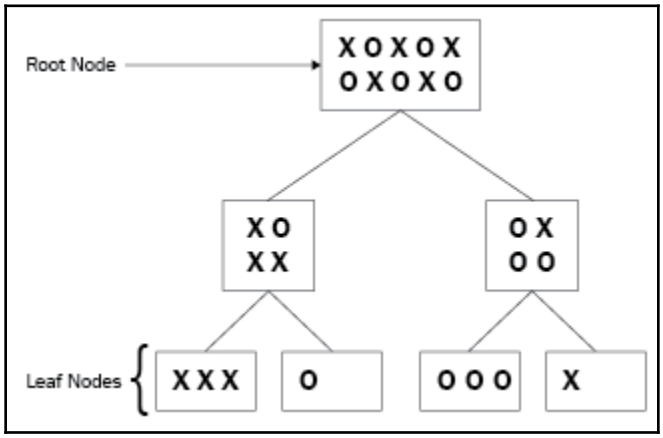
$$\frac{TP + TN}{TP + TN + FP + FN}$$

$$\frac{TP}{TP + FN} = \frac{CorrectlyFlagged}{CorrectlyFlagged + Misses}$$

$$\frac{TP}{TP + FP} = \frac{CorrectlyFlagged}{CorrectlyFlagged + WronglyFlagged}$$

$$2\left(\frac{Precision * Recall}{Precision + Recall}\right)$$


Business Understanding
Data Understanding
Data
Data Preparation
Data Engineering
Yes
Deployment
No
Eval < Expect
Evaluation
Modelling
Phase 3: Model Deployment
Phase 2: Model Evaluation
Phase 1: Model Training


Root Node
X O X O X
O X O X O
X O
X X
O X
O O
Leaf Nodes
X X X
O
O O O
X

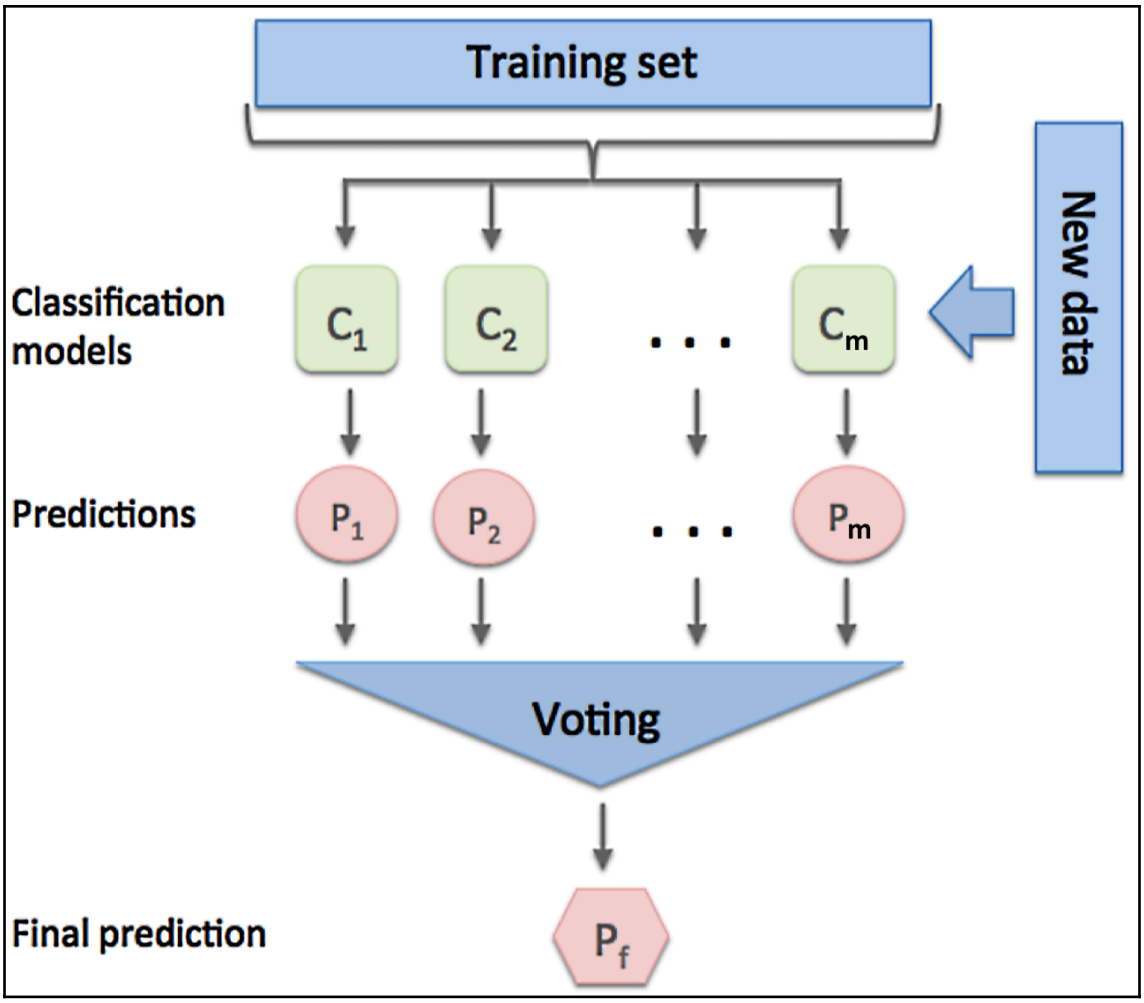
```
Out[22]: array([[64,  4],
                [ 2, 30]])
```

```
0.94 0.9375 0.8823529411764706
```

```
In [20]: from xgboost import XGBClassifier
         classifier = XGBClassifier()
         classifier.fit(X_train, y_train)

Out[20]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                       colsample_bynode=1, colsample_bytree=1, gamma=0,
                       learning_rate=0.1, max_delta_step=0, max_depth=3,
                       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                       nthread=None, objective='binary:logistic', random_state=0,
                       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                       silent=None, subsample=1, verbosity=1)
```

```
Out[21]: array([[64,  4],
                [ 3, 29]])
```

```
0.93 0.90625 0.8787878787878788
```



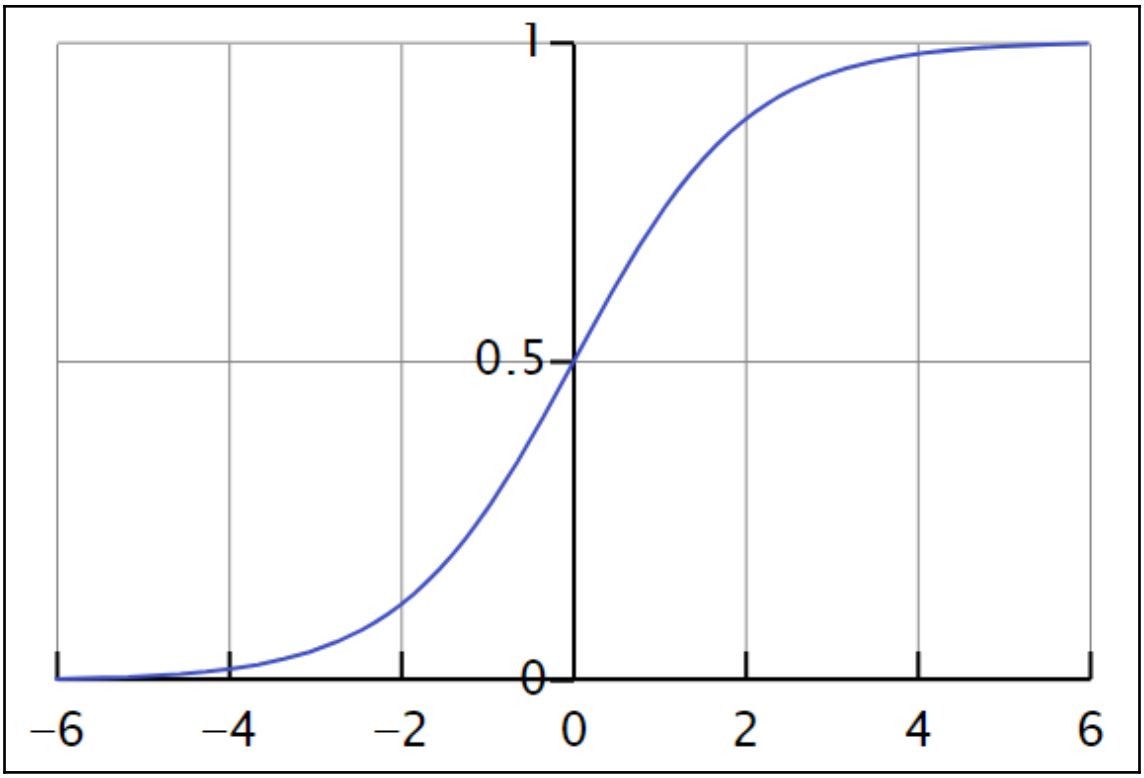

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, max_depth = 4,criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[9]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                       max_depth=4, max_features='auto', 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, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

```
Out[10]: array([[64,  4],
                [ 3, 29]])
```

```
0.93 0.90625 0.8787878787878788
```

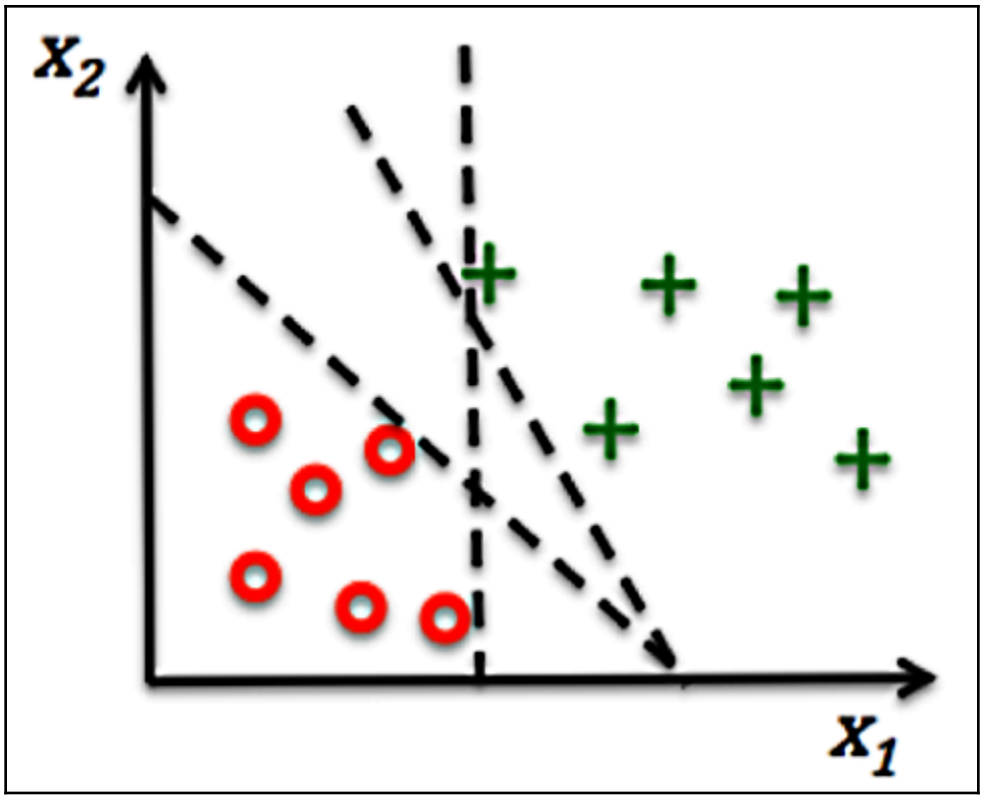
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

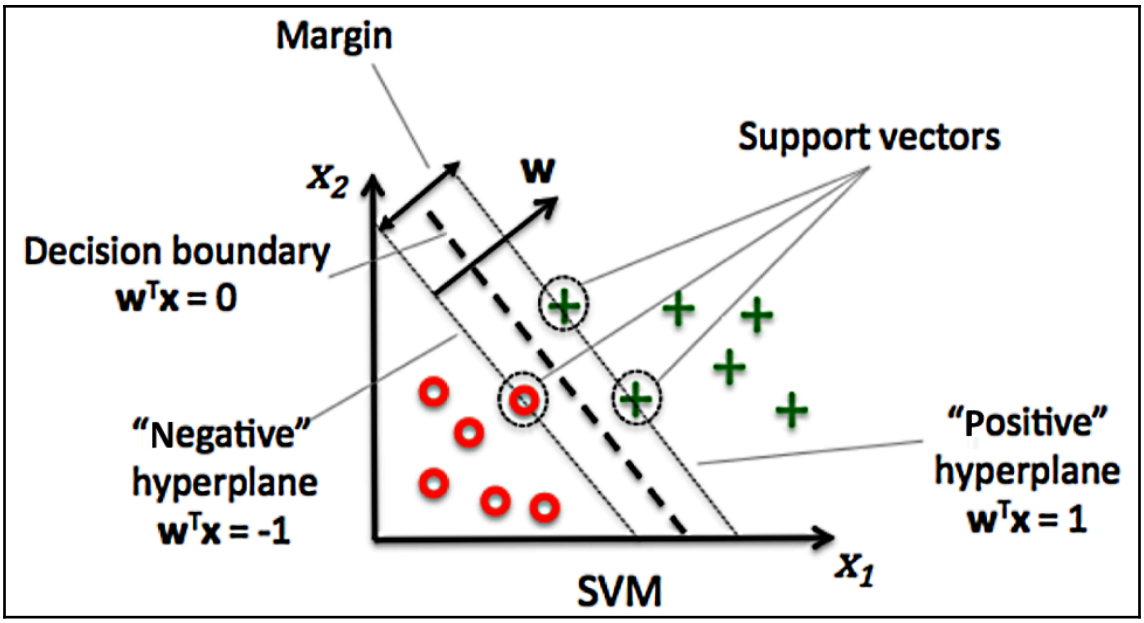


$$Cost(w,b) = \frac{1}{b}\sum Loss(\acute{y}^{(i)}, y^{(i)})$$

```
Out[11]: array([[65,  3],
                [ 6, 26]])
```

```
0.91 0.8125 0.896551724137931
```


x_2
x_1




```
Out[9]: array([[66,  2],
               [ 9, 23]])
```

```
0.89 0.71875 0.92
```

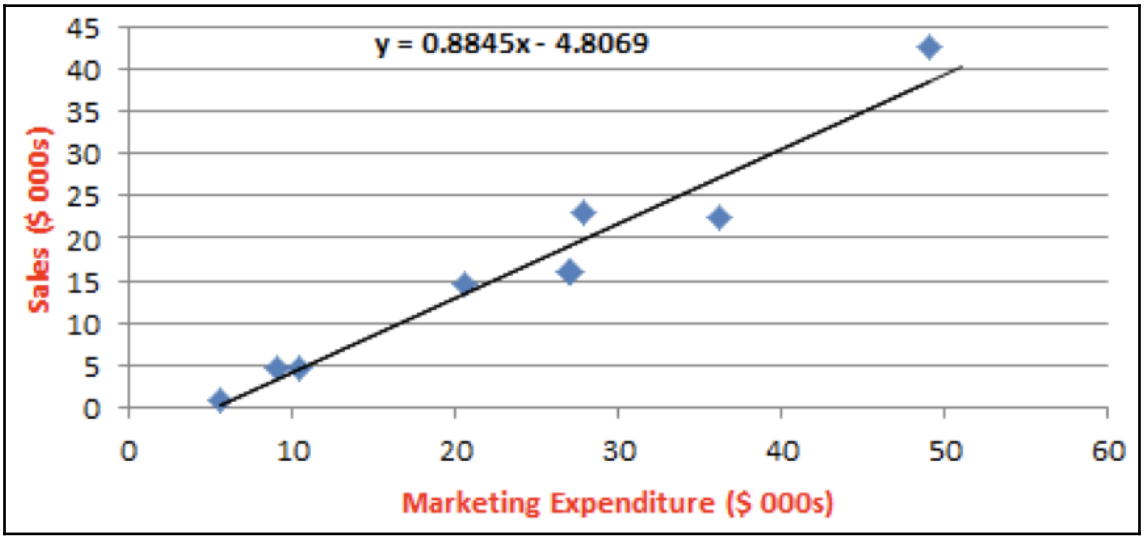
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

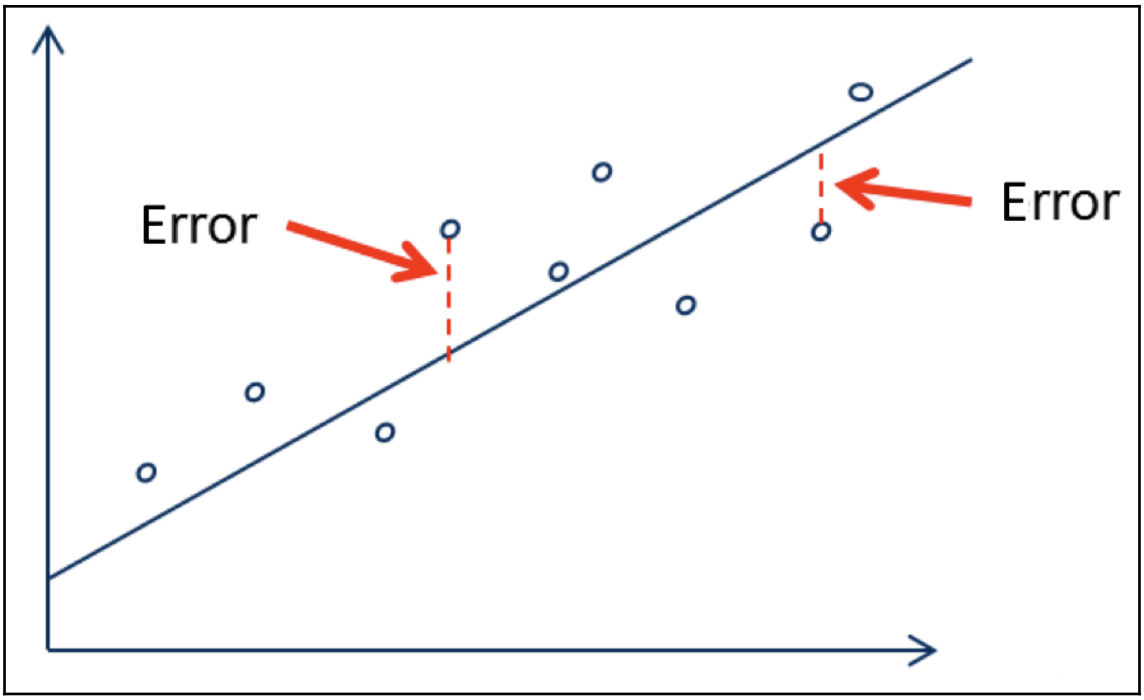
```
Out[10]: array([[66,  2],
                [ 6, 26]])
```

```
0.92 0.8125 0.9285714285714286
```

	NAME	CYLINDERS	DISPLACEMENT	HORSEPOWER	WEIGHT	ACCELERATION	MPG
0	chevrolet chevelle malibu	8	307.0	130	3504	12.0	18.0
1	buick skylark 320	8	350.0	165	3693	11.5	15.0
2	plymouth satellite	8	318.0	150	3436	11.0	18.0
3	amc rebel sst	8	304.0	150	3433	12.0	16.0
4	ford torino	8	302.0	140	3449	10.5	17.0

$$\acute{y} = (X)w + \alpha$$





$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\dot{y}^{(i)} - y^{i})^2}$$

```
Out[10]: 4.362141296771179
```

```
In [43]: from sklearn.tree import DecisionTreeRegressor
         regressor = DecisionTreeRegressor(max_depth=3)
         regressor.fit(X_train, y_train)

Out[43]: DecisionTreeRegressor(criterion='mse', 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')
```

```
Out[45]: 5.2771702288377
```

```
In [5]: from sklearn import ensemble
        params = {'n_estimators': 500, 'max_depth': 4, 'min_samples_split': 2,
                  'learning_rate': 0.01, 'loss': 'ls'}
        regressor = ensemble.GradientBoostingRegressor(**params)

        regressor.fit(X_train, y_train)

Out[5]: GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                                  learning_rate=0.01, loss='ls', 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, n_estimators=500,
                                  n_iter_no_change=None, presort='auto',
                                  random_state=None, subsample=1.0, tol=0.0001,
                                  validation_fraction=0.1, verbose=0, warm_start=False)
```

```
Out[7]: 4.034836373089085
```

```
In [63]: df.columns

Out[63]: Index(['Date', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
                'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
                'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
                'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
                'Temp3pm', 'RainToday', 'RISK_MM', 'RainTomorrow'],
               dtype='object')
```

```
In [124]: df.iloc[:,0:12].head()
```

Out[124]:

	Date	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm
0	2007-11-01	8.0	24.3	0.0	3.4	6.3	7	30.0	12	7	6.0	20
1	2007-11-02	14.0	26.9	3.6	4.4	9.7	1	39.0	0	13	4.0	17
2	2007-11-03	13.7	23.4	3.6	5.8	3.3	7	85.0	3	5	6.0	6
3	2007-11-04	13.3	15.5	39.8	7.2	9.1	7	54.0	14	13	30.0	24
4	2007-11-05	7.6	16.1	2.8	5.6	10.6	10	50.0	10	2	20.0	28

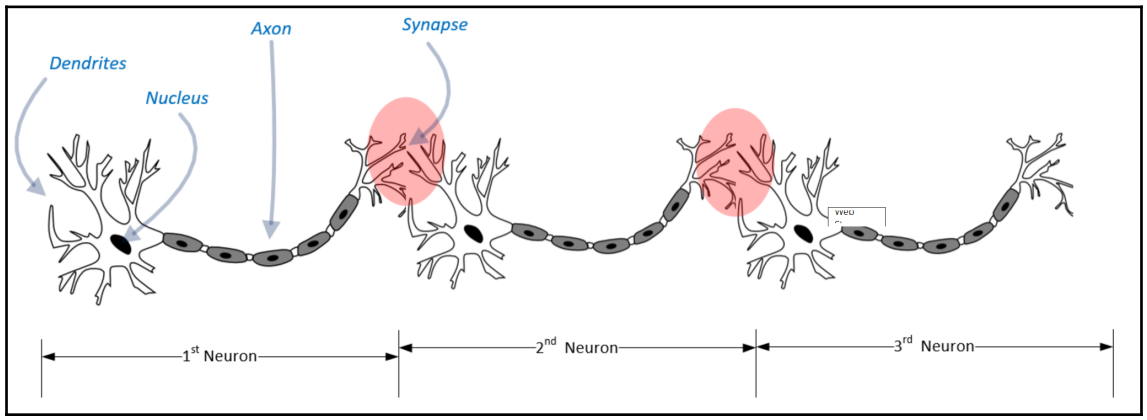
```
In [127]: df.iloc[:,12:25].head()
```

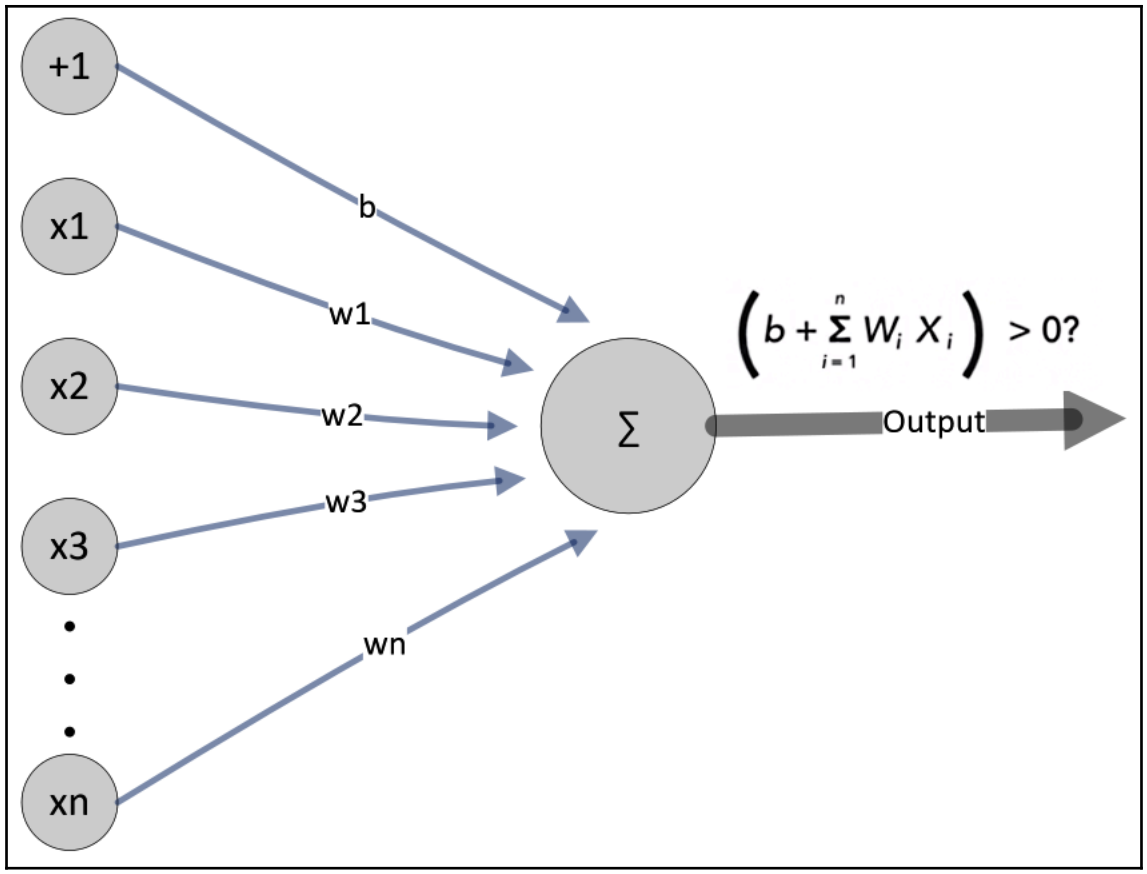
Out[127]:

	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RISK_MM	RainTomorrow
0	68	29	1019.7	1015.0	7	7	14.4	23.6	0	3.6	1
1	80	36	1012.4	1008.4	5	3	17.5	25.7	1	3.6	1
2	82	69	1009.5	1007.2	8	7	15.4	20.2	1	39.8	1
3	62	56	1005.5	1007.0	2	7	13.5	14.1	1	2.8	1
4	68	49	1018.3	1018.5	7	7	11.1	15.4	1	0.0	0

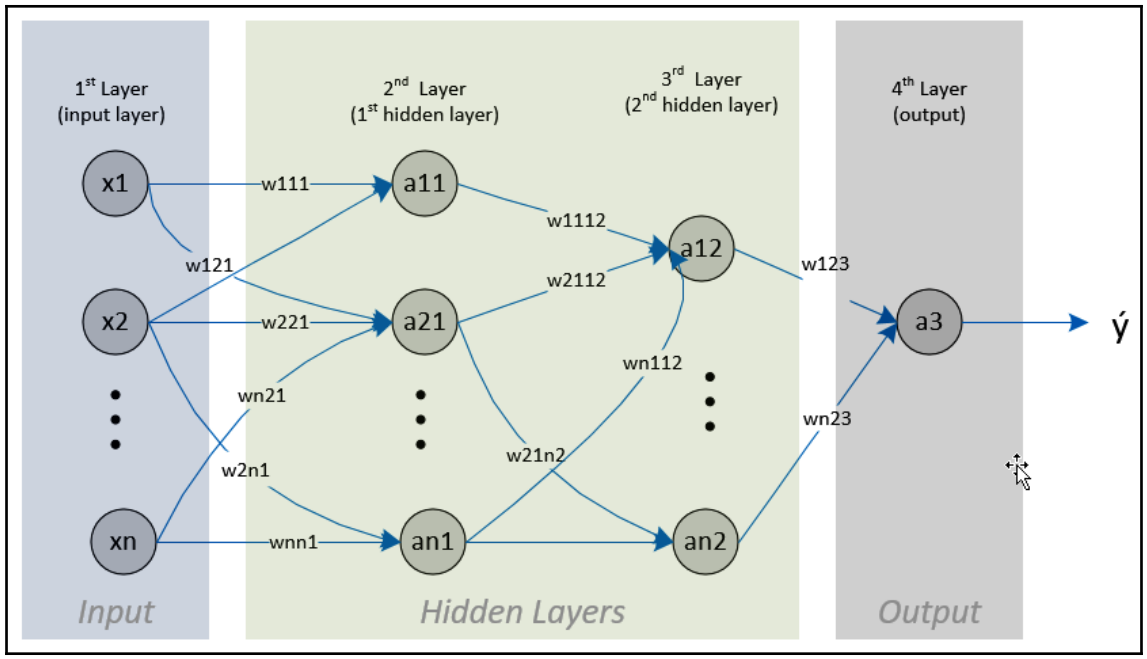
```
In [89]: predict = model.predict(train_y)

In [90]: from sklearn.metrics import accuracy_score

In [91]: accuracy_score(predict , test_y)

Out[91]: 0.9696969696969697
```

Chapter 8: Neural Network Algorithms

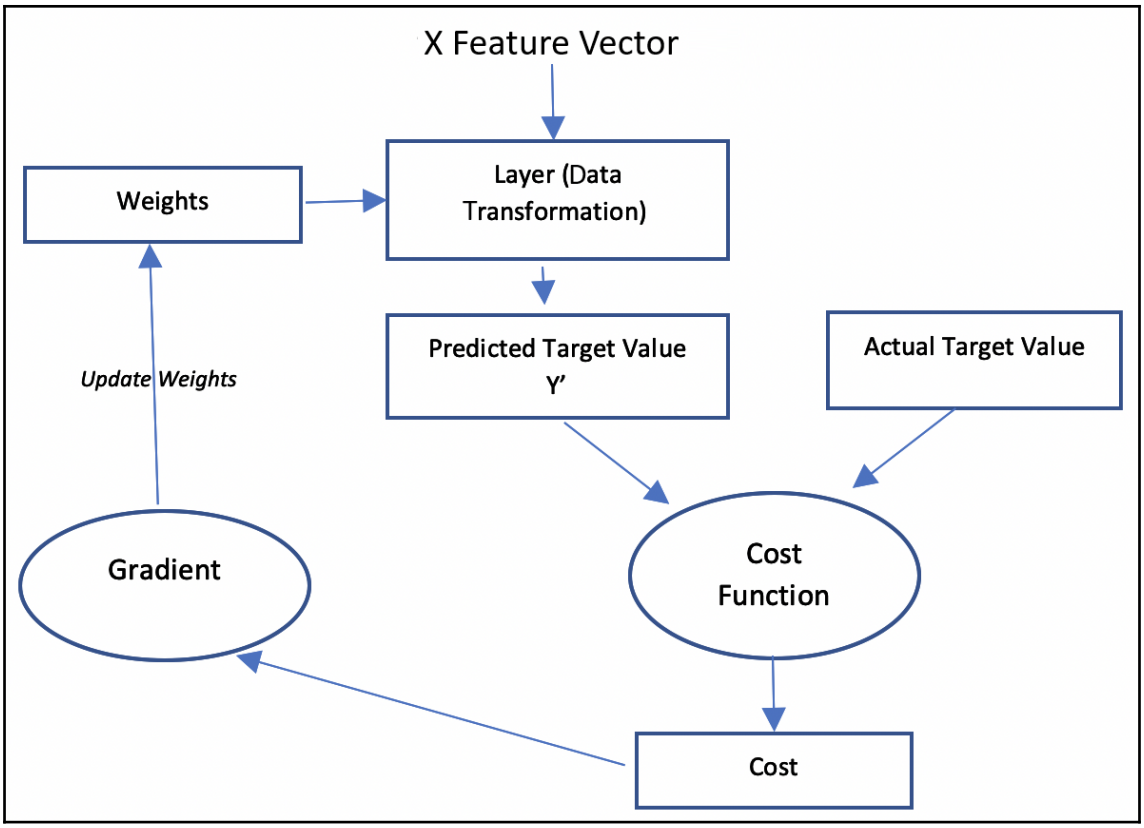


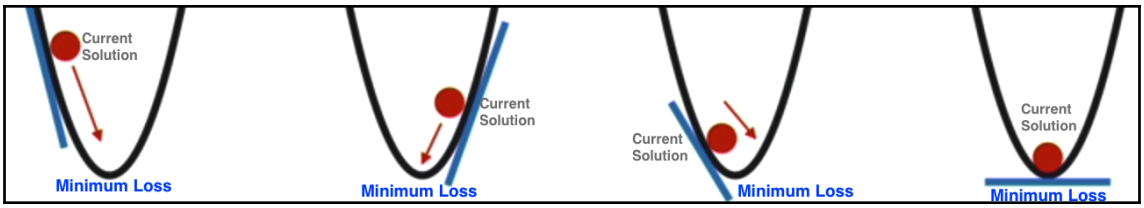


$$\left(b + \sum_{i=1}^{n} w_i x_i\right) > 0?$$

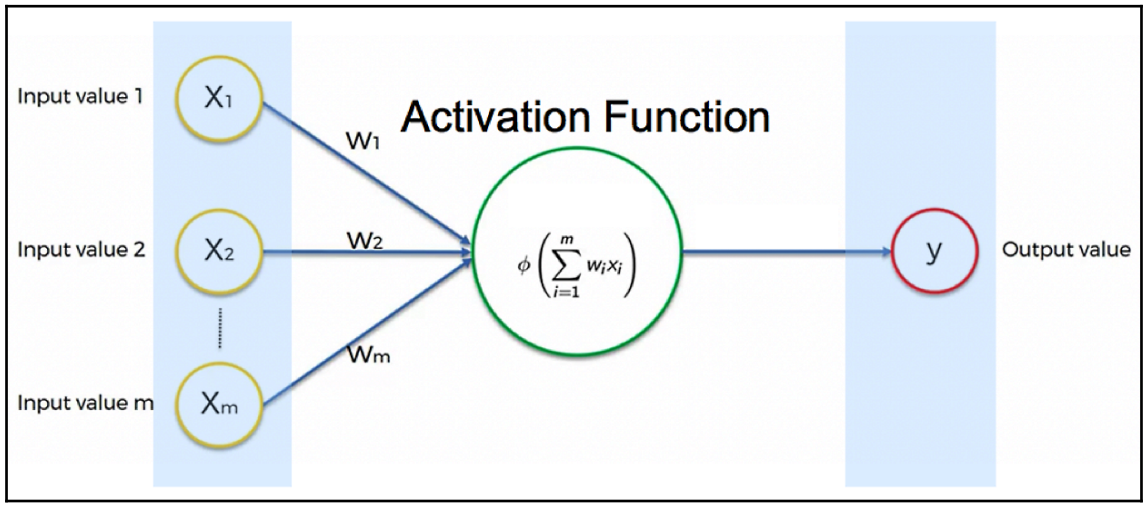

1st Layer
(input layer)
2nd Layer
(1st hidden layer)
3rd Layer
(2nd hidden layer)
4th Layer
(output)
x1
x2
xn
a11
a21
an1
a12
an2
a3
ý
w111
w121
w221
wn21
w2n1
wnn1
w1112
w2112
wn112
w21n2
w123
wn23
Input
Hidden Layers
Output

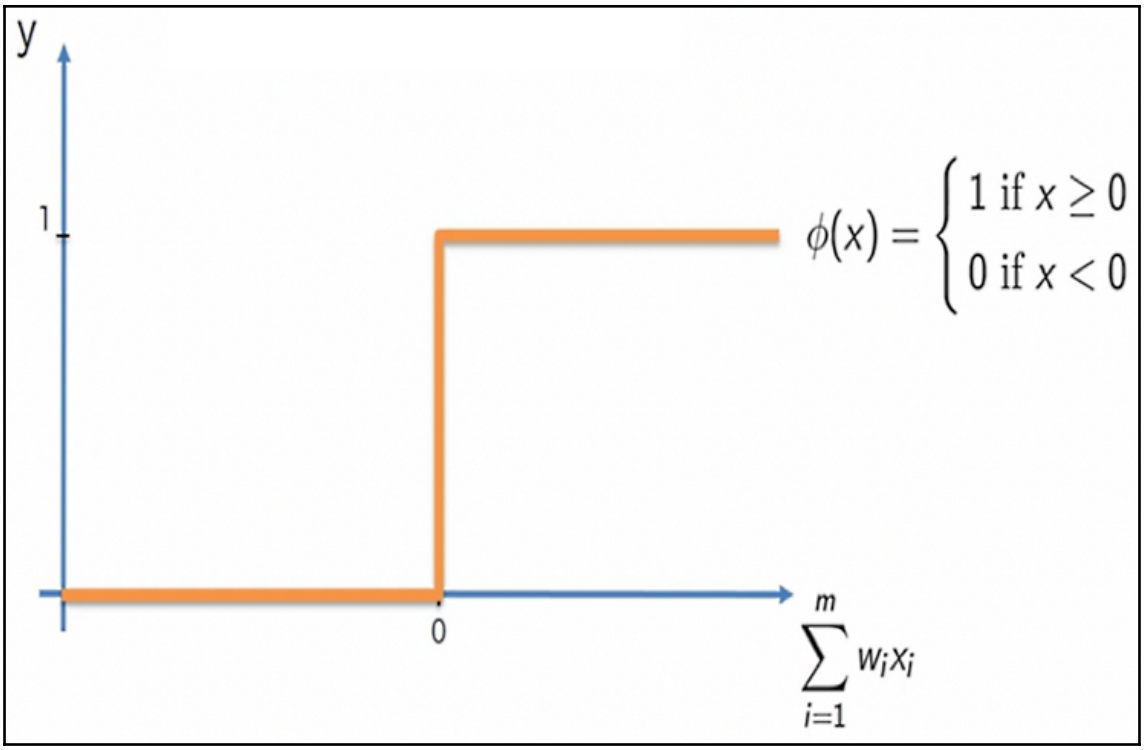
$$n_h^{[l]}$$

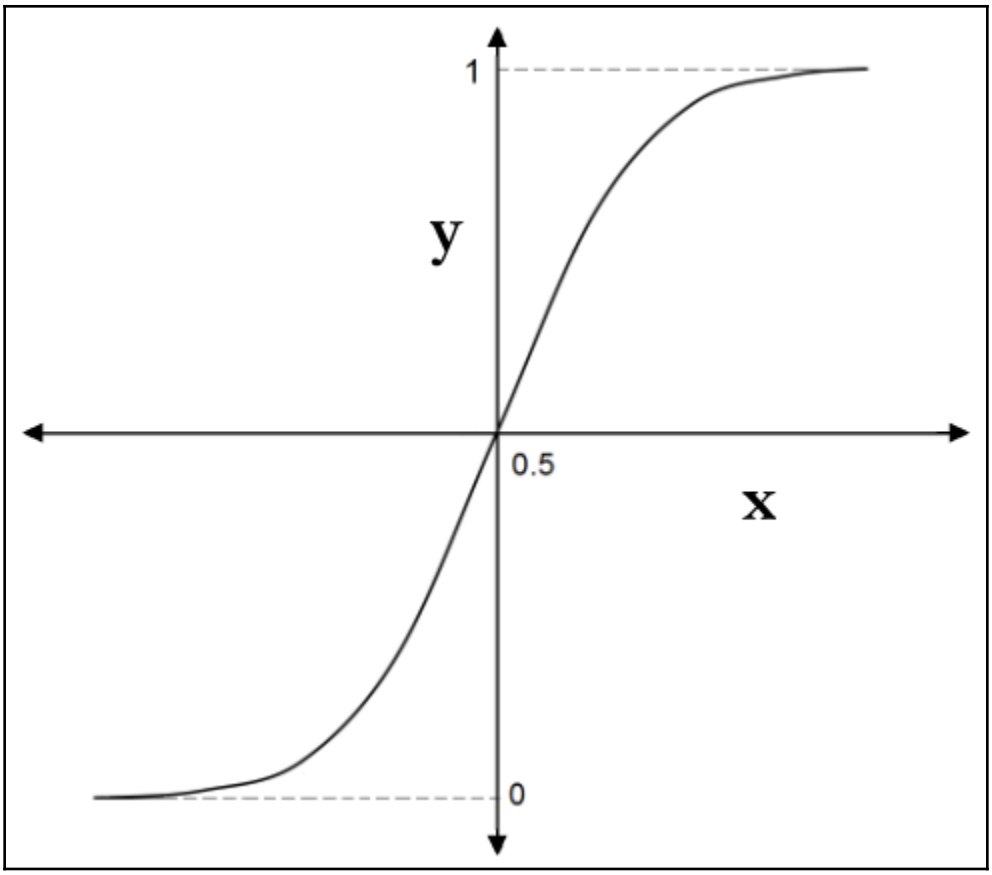

X Feature Vector
Weights
Layer (Data Transformation)
Predicted Target Value Y'
Actual Target Value
Update Weights
Gradient
Cost Function
Cost


Current Solution
Minimum Loss
Current Solution
Minimum Loss
Current Solution
Minimum Loss
Current Solution
Minimum Loss

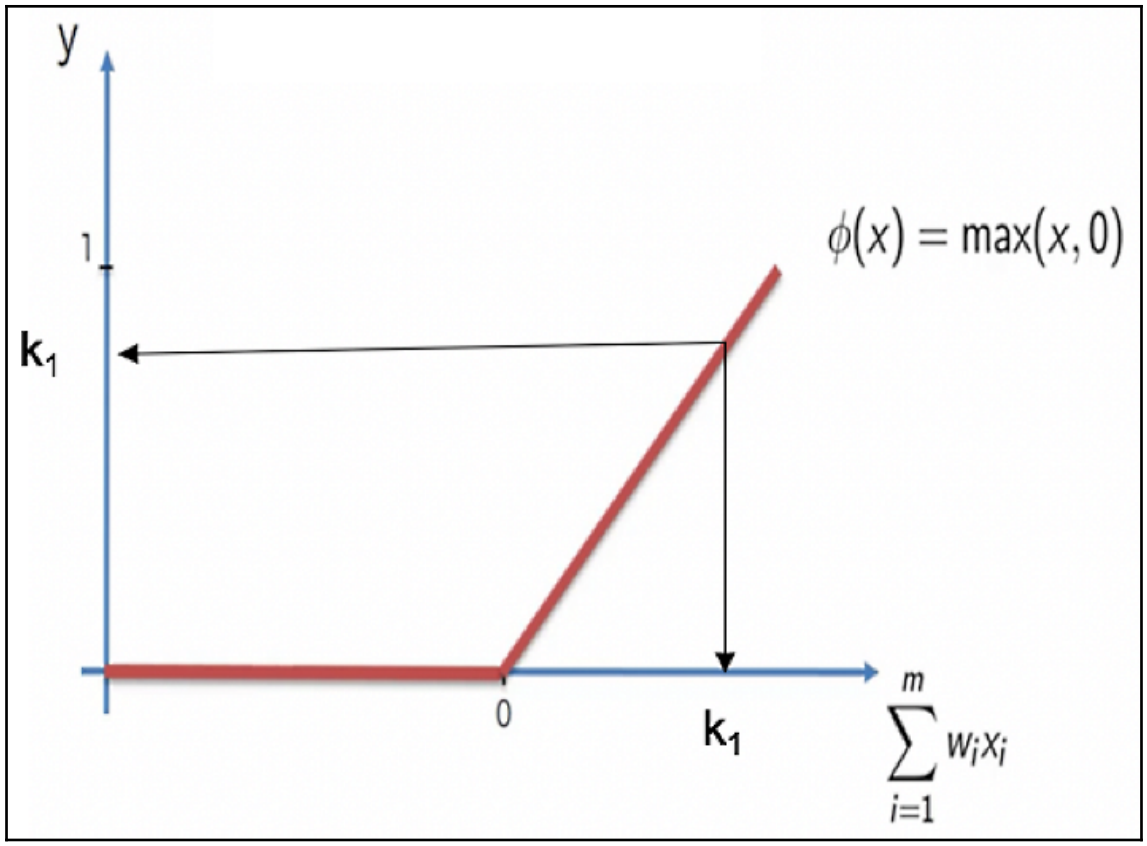
$$gradient = \frac{\triangle y}{\triangle x}$$




y
1
0
$\phi(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$
$\sum_{i=1}^{m} w_i x_i$



$$y = f(x) = \frac{1}{1 + e^{-x}}$$



$$y = f(x) = 0;$$

$$x < 0$$

$$y = f(x) = x$$

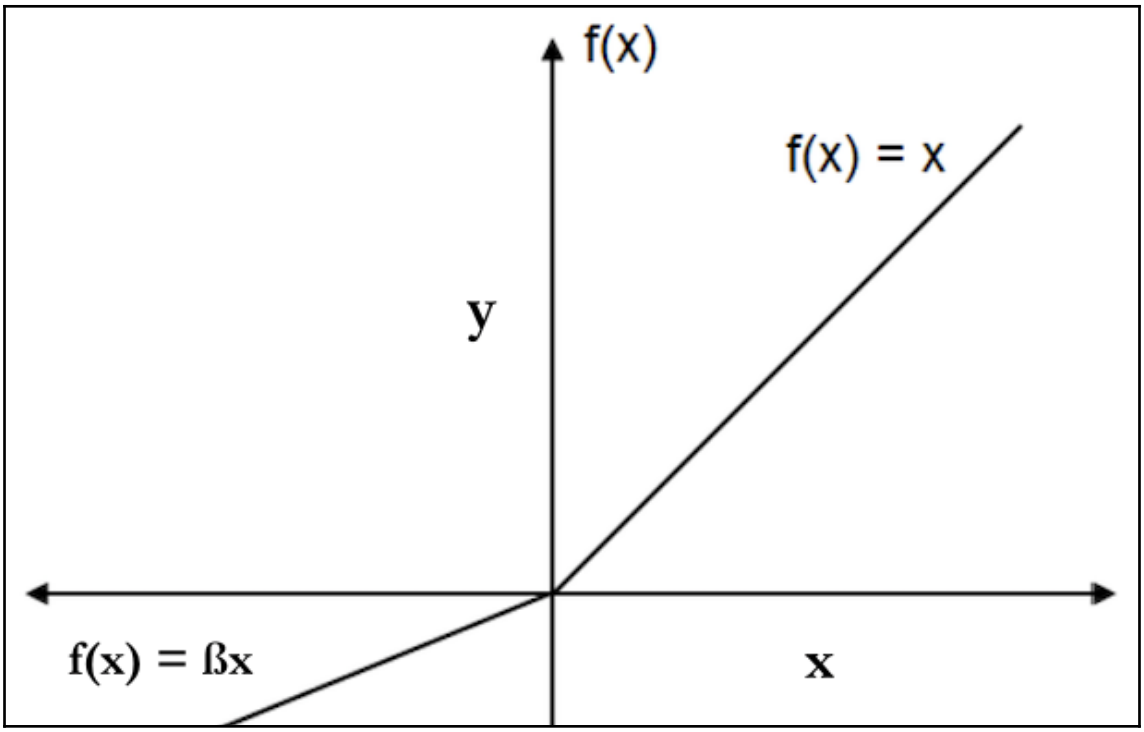
$$x >== 0$$

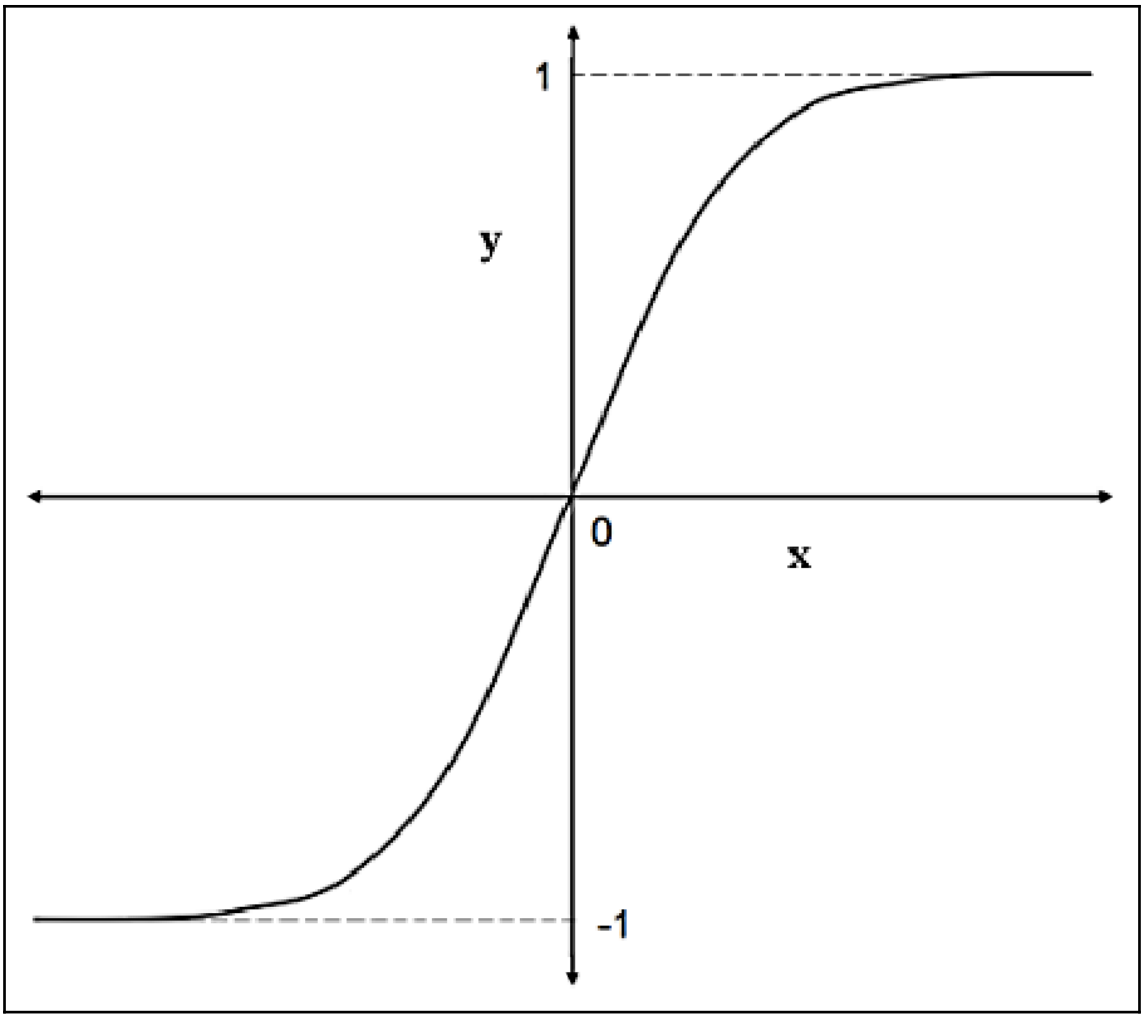
$$y = f(x) = ßx$$

$$x < 0$$

$$y = f(x) = x$$

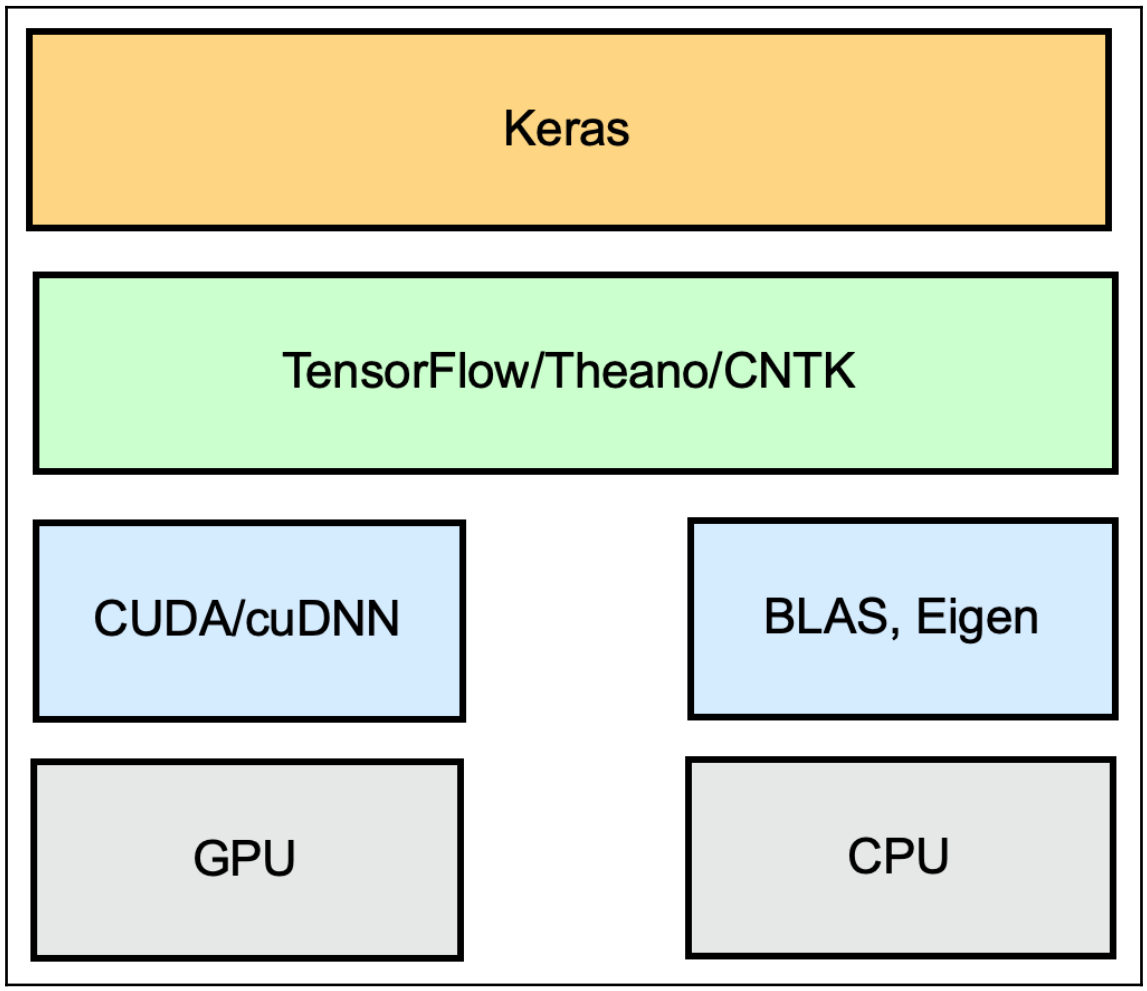
$$x >== 0$$


f(x)
f(x) = x
y
f(x) = ßx
x



$$y = f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

$$prob^{(s)} = \frac{e^{x^s}}{\sum_{i=1}^{n} e^{x^i}}$$



Import Packages

```
[ ] import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Activation, Dropout
    from tensorflow.keras.datasets import mnist
```

Load Data

Let us load the mnist dataset

```
[ ] (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

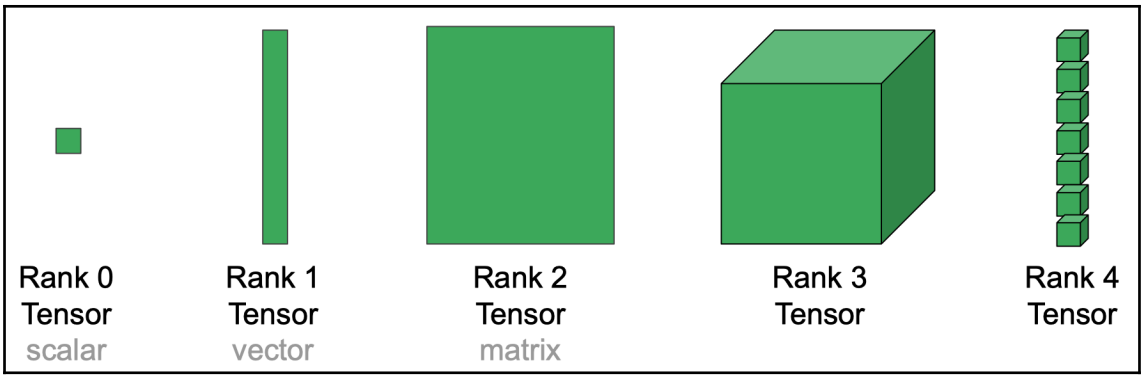
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

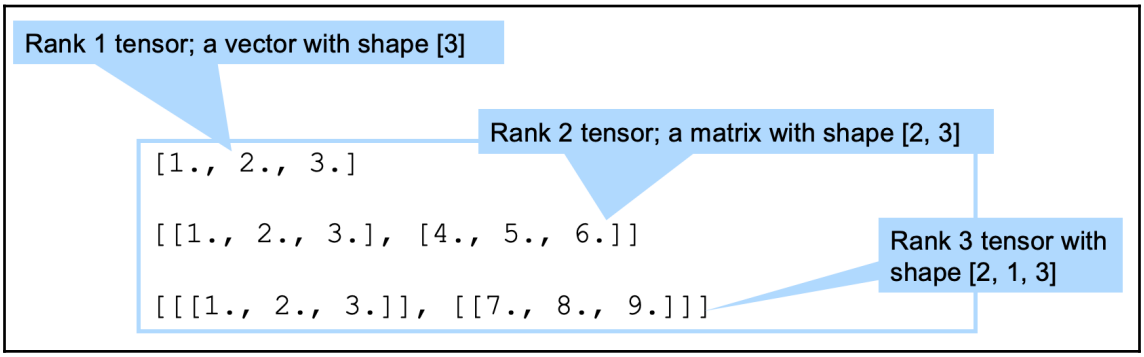
```
[ ] model = tf.keras.models.Sequential([
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(128, activation='relu'),
            tf.keras.layers.Dropout(0.15),
            tf.keras.layers.Dense(128, activation='relu'),
            tf.keras.layers.Dropout(0.15),
            tf.keras.layers.Dense(10, activation='softmax'),
            ])
```

```
inputs = tf.keras.Input(shape=(128,128))
x = tf.keras.layers.Flatten()(inputs)
x = tf.keras.layers.Dense(512, activation='relu',name='d1')(x)
x = tf.keras.layers.Dropout(0.2)(x)
predictions = tf.keras.layers.Dense(10,activation=tf.nn.softmax, name='d2')(x)
model = tf.keras.Model(inputs=inputs, outputs=predictions)
```

```
optimiser = tf.keras.optimizers.RMSprop
model.compile (optimizer= optimiser, loss='mse', metrics = ['accuracy'])
```

```
model.fit(x_train, y_train, batch_size=128, epochs=10)
```






```
In [13]: print("Define constant tensors")
         a = tf.constant(2)
         print("a = %i" % a)
         b = tf.constant(3)
         print("b = %i" % b)

         Define constant tensors
         a = 2
         b = 3
```

In [14]:
```
print("Running operations, without tf.Session")
c = a + b
print("a + b = %i" % c)
d = a * b
print("a * b = %i" % d)
```

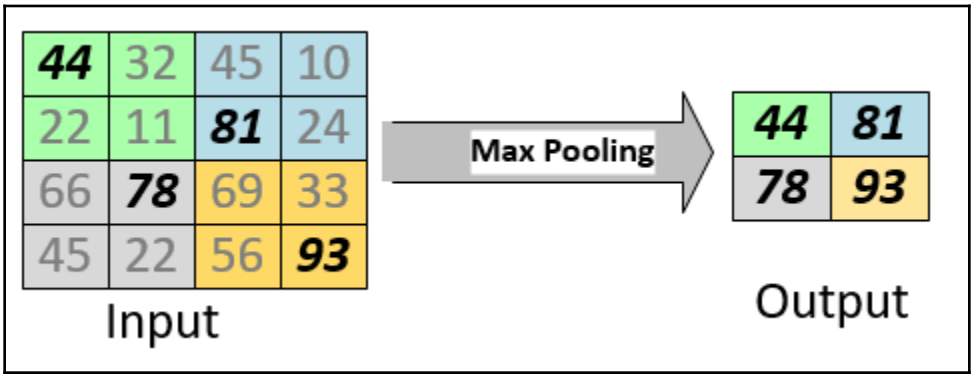
```
Running operations, without tf.Session
a + b = 5
a * b = 6
```

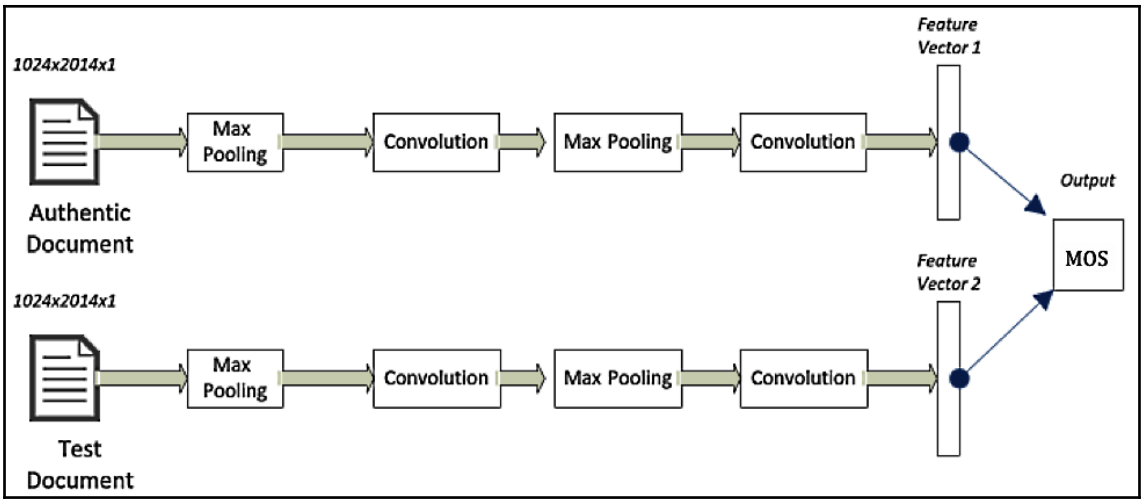
In [16]:
```
c = a + b
print("a + b = %s" % c)
```

```
a + b = Tensor("add:0", shape=(2, 2), dtype=float32)
```

In [17]:
```
d = tf.matmul(a, b)
print("a * b = %s" % d)
```

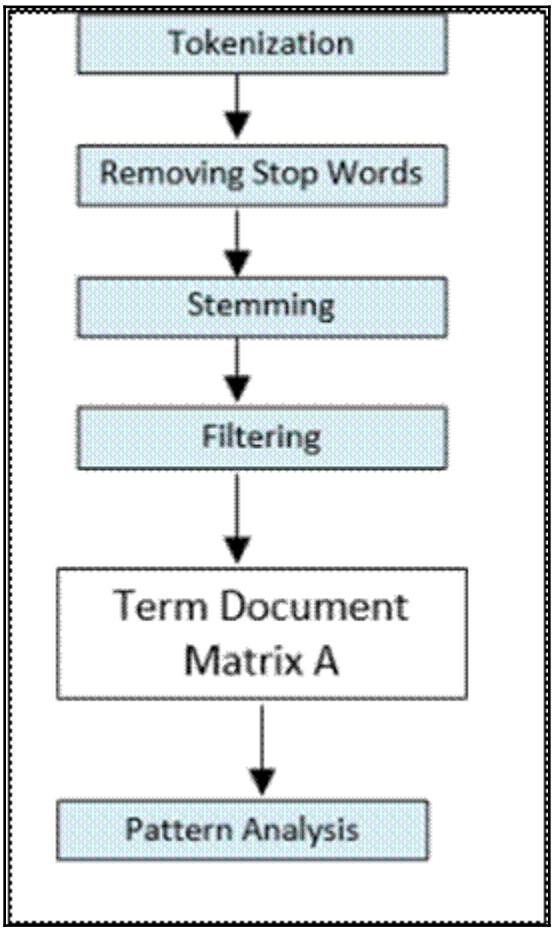
```
a * b = Tensor("MatMul:0", shape=(2, 2), dtype=float32)
```






```
[10] # Build the model
     model = tf.keras.models.Model([input_a, input_b], measureOfSimilarity)
     # Train
     model.compile(loss='binary_crossentropy',optimizer=tf.keras.optimizers.Adam(),metrics=['accuracy'])

     model.fit([train_pairs[:, 0], train_pairs[:, 1]], tr_labels,
               batch_size=128,epochs=10,validation_data=([test_pairs[:, 0], test_pairs[:, 1]], test_labels))

     Epoch 1/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.3459 - accuracy: 0.8500 - val_loss: 0.2652 - val_accuracy: 0.9105
     Epoch 2/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.1773 - accuracy: 0.9337 - val_loss: 0.1685 - val_accuracy: 0.9508
     Epoch 3/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.1215 - accuracy: 0.9563 - val_loss: 0.1301 - val_accuracy: 0.9610
     Epoch 4/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.0956 - accuracy: 0.9665 - val_loss: 0.1087 - val_accuracy: 0.9685
     Epoch 5/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.0790 - accuracy: 0.9724 - val_loss: 0.1104 - val_accuracy: 0.9669
     Epoch 6/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.0649 - accuracy: 0.9770 - val_loss: 0.0949 - val_accuracy: 0.9715
     Epoch 7/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.0568 - accuracy: 0.9803 - val_loss: 0.0895 - val_accuracy: 0.9722
     Epoch 8/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.0513 - accuracy: 0.9823 - val_loss: 0.0807 - val_accuracy: 0.9770
     Epoch 9/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.0439 - accuracy: 0.9847 - val_loss: 0.0916 - val_accuracy: 0.9737
     Epoch 10/10
     847/847 [==============================] - 6s 7ms/step - loss: 0.0417 - accuracy: 0.9853 - val_loss: 0.0835 - val_accuracy: 0.9749
     <tensorflow.python.keras.callbacks.History at 0x7ff1218297b8>
```

Chapter 9: Algorithms for Natural Language Processing

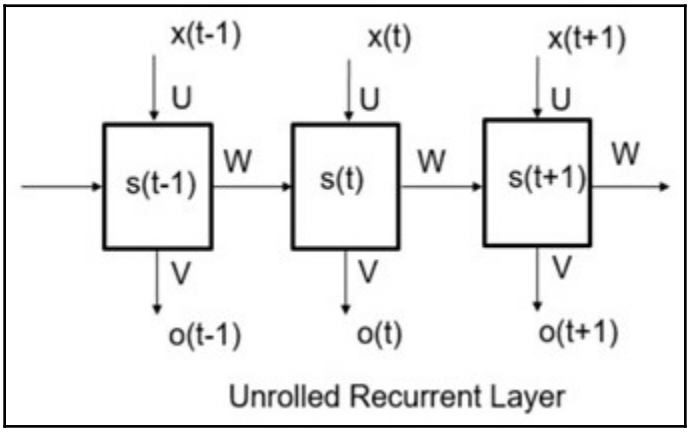


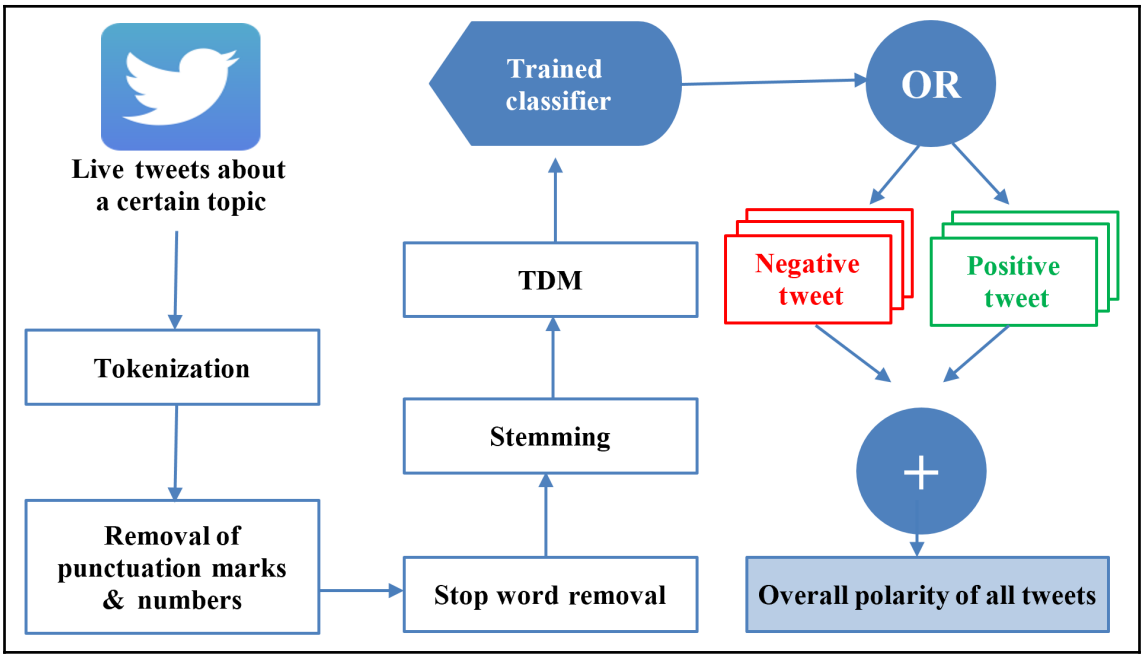
```
In [2]: # Importing the dataset
        dataset = pd.read_csv('Restaurant_Reviews.tsv', delimiter = '\t', quoting = 3)
        dataset.head()
```

Out[2]:

	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

```
In [18]: # Making the Confusion Matrix
         from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)

In [19]: cm

Out[19]: array([[55, 42],
                [12, 91]])
```



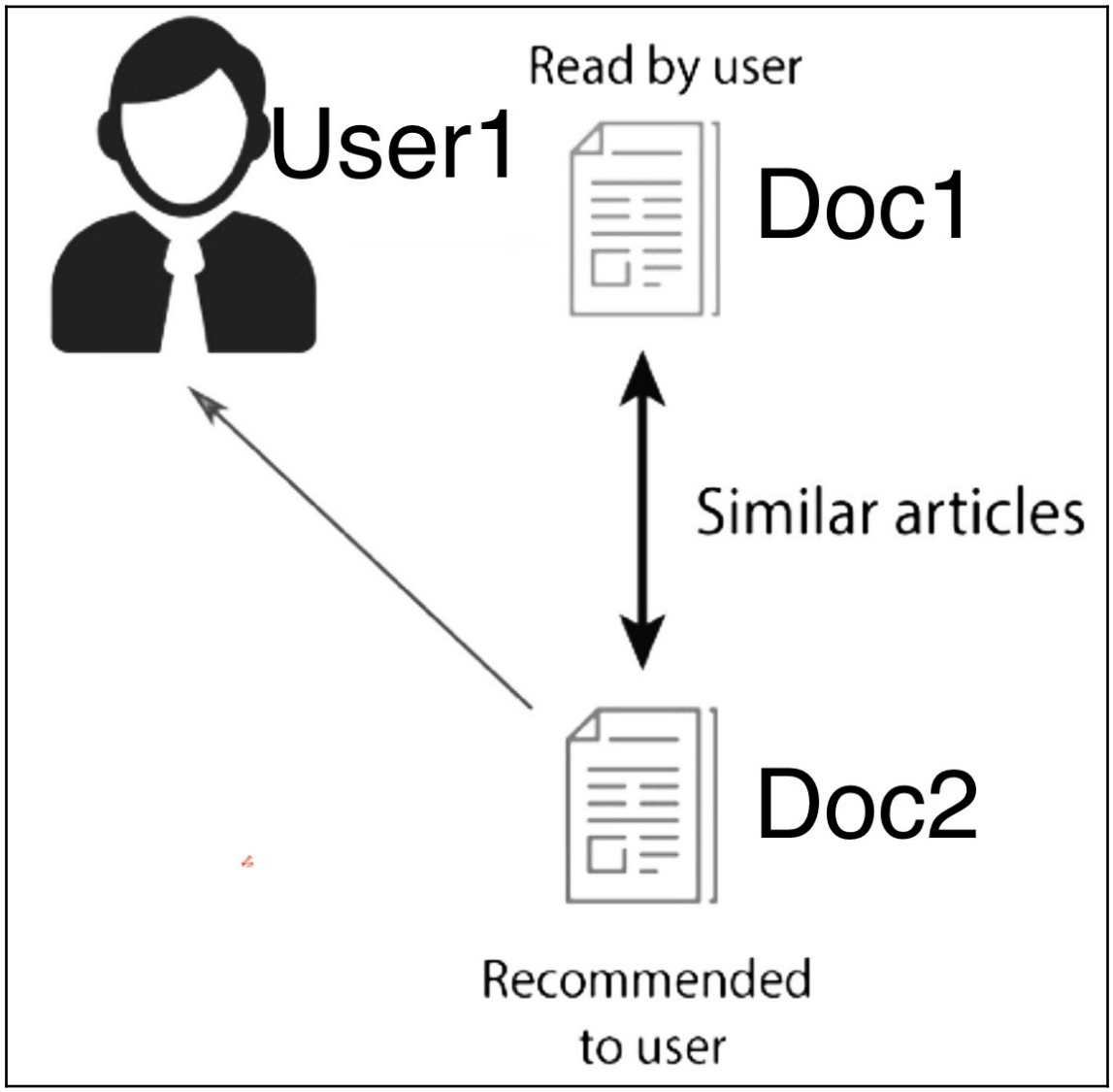


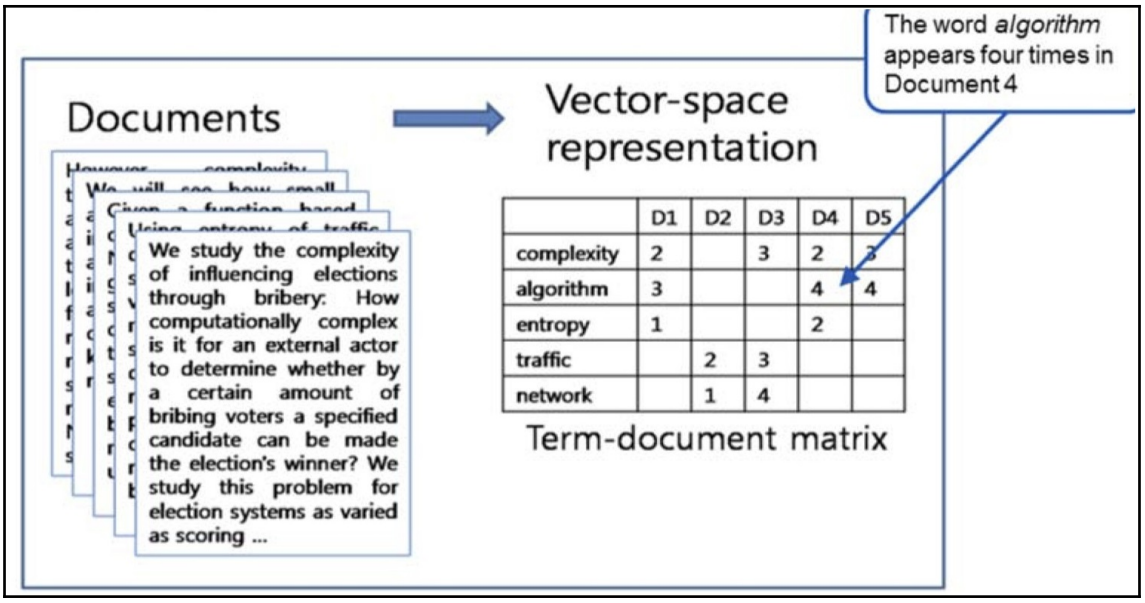
	label	review
0	neg	how do films like mouse hunt get into theatres...
1	neg	some talented actresses are blessed with a dem...
2	pos	this has been an extraordinary year for austra...
3	pos	according to hollywood movies made in last few...
4	neg	my first press screening of 1998 and already i...

```
In [2]: len(df)
Out[2]: 2000
```

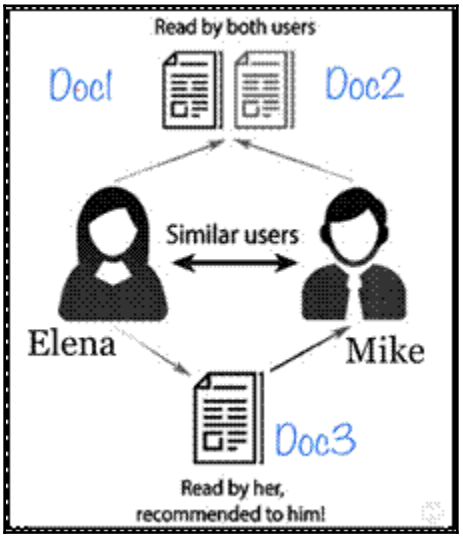
```
In [23]: from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

In [24]: print(confusion_matrix(y_test,predictions))

[[259  23]
 [102 198]]

In [25]: print(classification_report(y_test,predictions))

              precision    recall  f1-score   support

         neg       0.72      0.92      0.81       282
         pos       0.90      0.66      0.76       300

    accuracy                           0.79       582
   macro avg       0.81      0.79      0.78       582
weighted avg       0.81      0.79      0.78       582


In [26]: print(accuracy_score(y_test,predictions))

0.7852233676975945
```

Chapter 10: Recommendation Engines



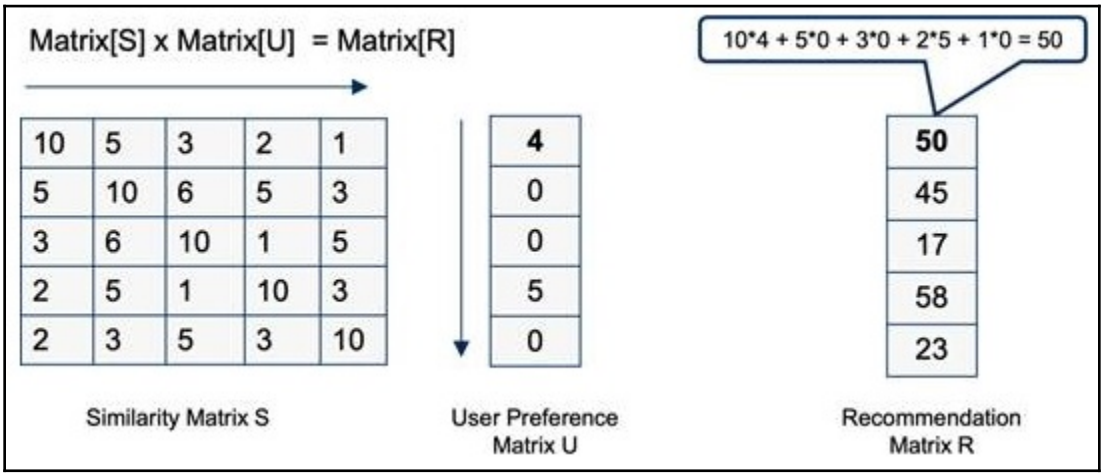


	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		



	Item 1	Item 2	Item 3	Item 4	Item 5
Item 1	10	5	3	2	1
Item 2	5	10	6	5	3
Item 3	3	6	10	1	5
Item 4	2	5	1	10	3
Item 5	1	3	5	3	10

Item 1	4
Item 2	0
Item 3	0
Item 4	5
Item 5	0



Out[5]:

	userId	movieId	rating	timestamp	title	genres
0	1	1	4.0	964982703	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	5	1	4.0	847434962	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	7	1	4.5	1106635946	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
3	15	1	2.5	1510577970	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
4	17	1	4.5	1305696483	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy

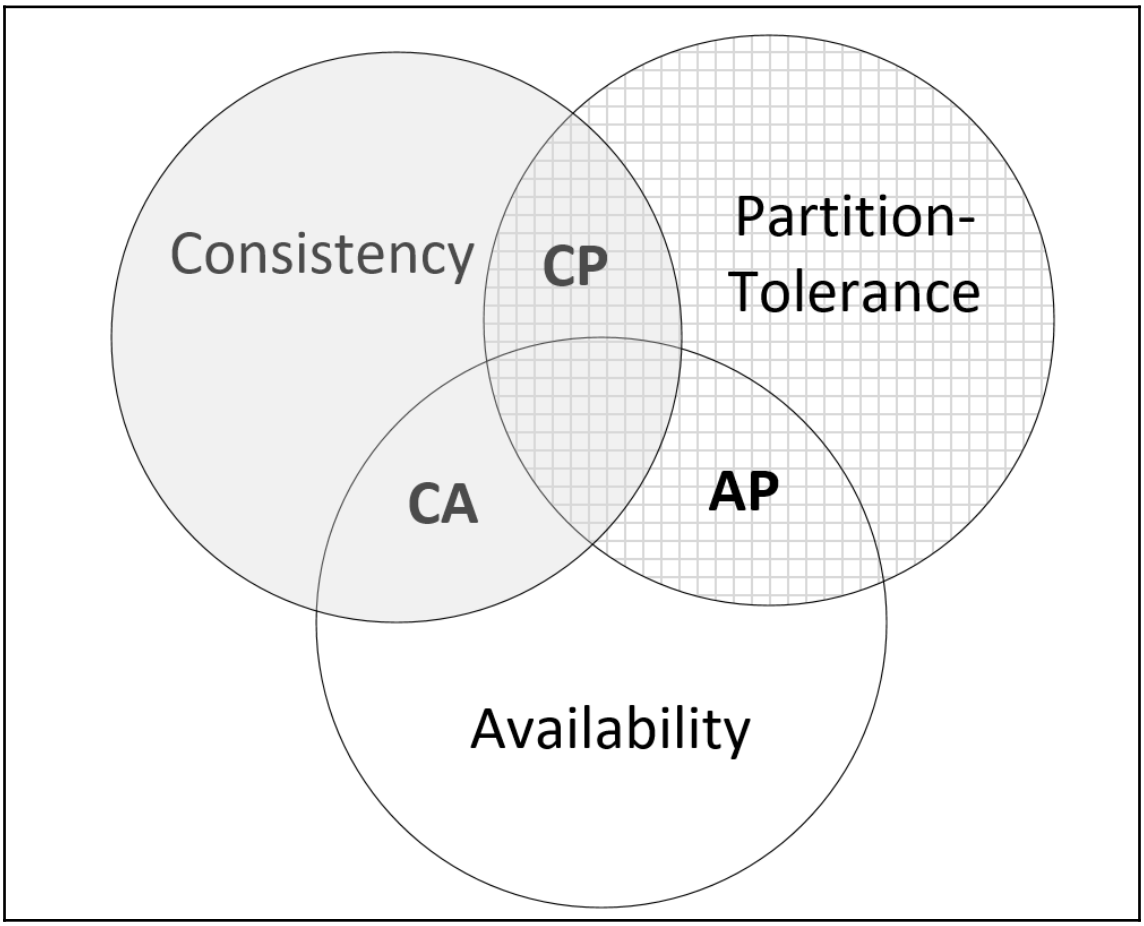
Out[6]:

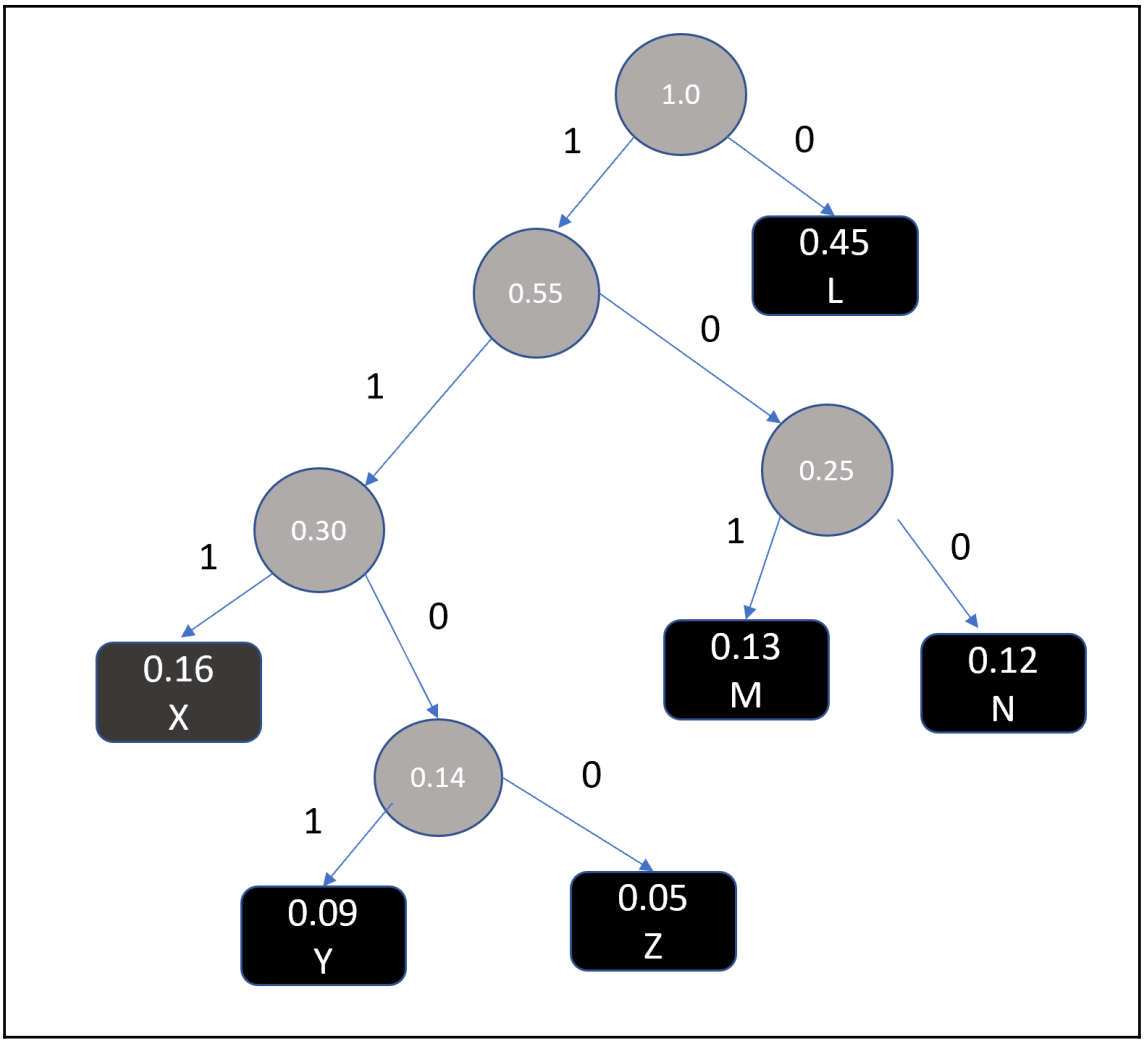
title	rating	number_of_ratings
'71 (2014)	4.0	1
'Hellboy': The Seeds of Creation (2004)	4.0	1
'Round Midnight (1986)	3.5	2
'Salem's Lot (2004)	5.0	1
'Til There Was You (1997)	4.0	2

Out[12]:

title	correlation	number_of_ratings
'burbs, The (1989)	0.353553	17
(500) Days of Summer (2009)	0.131120	42
*batteries not included (1987)	0.785714	7
10 Things I Hate About You (1999)	0.265637	54
10,000 BC (2008)	-0.075431	17

Chapter 11: Data Algorithms

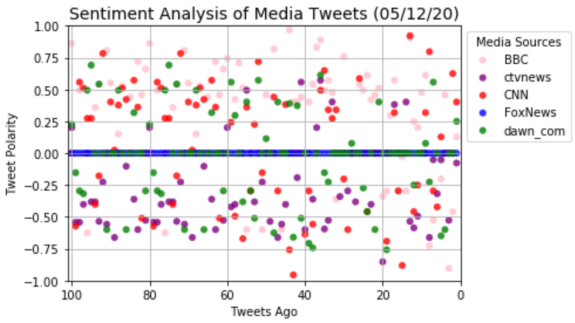



1.0
1
0
0.45
L
0.55
0
1
0.25
0.30
1
1
0
0
0.16
X
0.13
M
0.12
N
0.14
0
1
0.09
Y
0.05
Z

In [12]:
```
# We start extracting 100 tweets from each of the news sources
print("...STARTING..... collecting tweets from sources")

# Let us define an array to hold the sentiments
array_sentiments = []

for user in news_sources:
    count_tweet=100  # Setting the twitter count at 100
    print("Start tweets from %s"%user)
    for x in range(5):     # Extracting 5 pages of tweets
        public_tweets=api.user_timeline(user,page=x)
        # For each tweet
        for tweet in public_tweets:
            #Calculating the compound,+ive,-ive and neutral value for each tweet
            compound = analyzer.polarity_scores(tweet["text"])["compound"]
            pos = analyzer.polarity_scores(tweet["text"])["pos"]
            neu = analyzer.polarity_scores(tweet["text"])["neu"]
            neg = analyzer.polarity_scores(tweet["text"])["neg"]

            array_sentiments.append({"Media":user,
                                     "Tweet Text":tweet["text"],
                                     "Compound":compound,
                                     "Positive":pos,
                                     "Negative":neg,
                                     "Neutral":neu,
                                     "Date":tweet["created_at"],
                                     "Tweets Ago":count_tweet})

            count_tweet-=1

print("DONE with extracting tweets")
```

```
...STARTING..... collecting tweets from sources
Start tweets from @BBC
Start tweets from @ctvnews
Start tweets from @CNN
Start tweets from @FoxNews
Start tweets from @dawn_com
DONE with extracting tweets
```

In [21]:
```
for media in source:
    mydf=sentiments_df[sentiments_df["Media"]==media]
    plt.scatter(mydf["Tweets Ago"],mydf["Compound"], marker="o", linewidth=0, alpha=0.8, label=media,
                facecolors=mydf.Media.map({"BBC": "pink", "ctvnews" : "purple",  "CNN": 'red',
                                           "FoxNews":"blue","dawn_com":"green"}))

plt.legend(bbox_to_anchor = (1,1),title="Media Sources")
plt.title("Sentiment Analysis of Media Tweets (%s)" % (time.strftime("%x")), fontsize=14)
plt.xlabel("Tweets Ago")
plt.ylabel("Tweet Polarity")
plt.xlim(101,0)
plt.ylim(-1,1)
plt.grid(True)
plt.savefig("Output/Sentiment Analysis of Media Tweets.png",bbox_inches='tight')
plt.show()
```

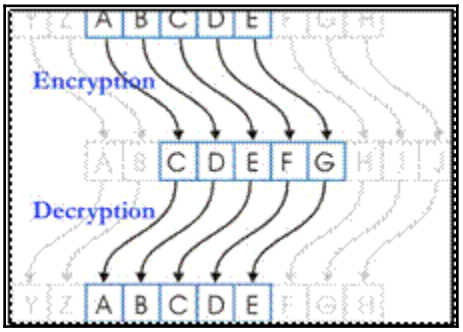


In [22]:
```
# Calculating the mean for each Media channel and storing to a dataframe
means_media_trends=sentiments_df.groupby("Media").mean()["Compound"].to_frame()
#Resetting the index
means_media_trends.reset_index(inplace=True)

means_media_trends
```

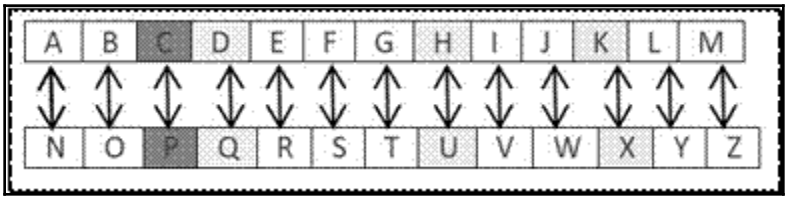
Out[22]:

	Media	Compound
0	BBC	0.234137
1	CNN	0.057011
2	FoxNews	0.000000
3	ctvnews	-0.194618
4	dawn_com	-0.025389

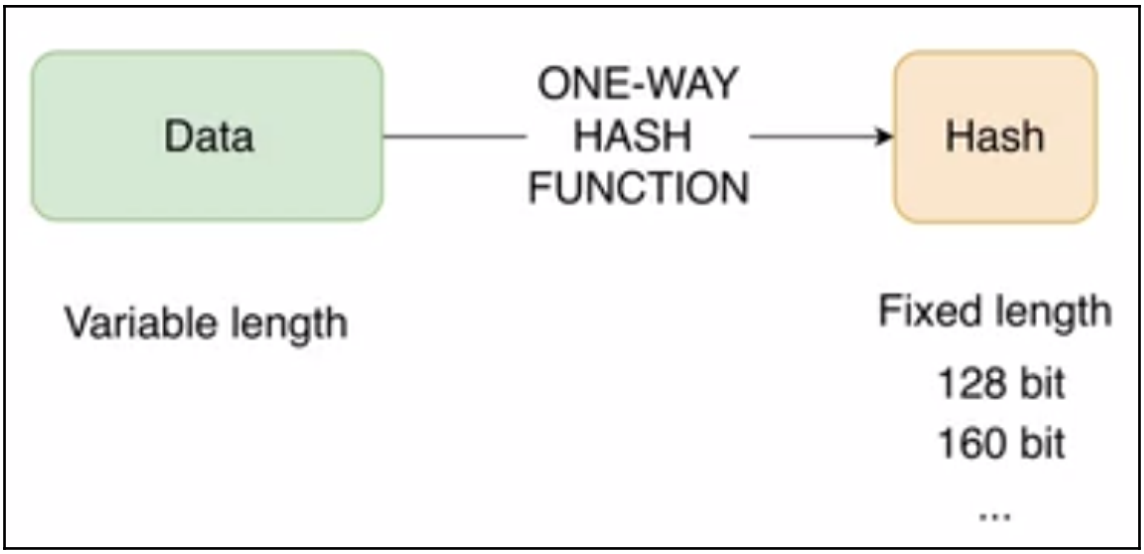
Chapter 12: Cryptography




```
In [37]: print(C)

         FDOP
```



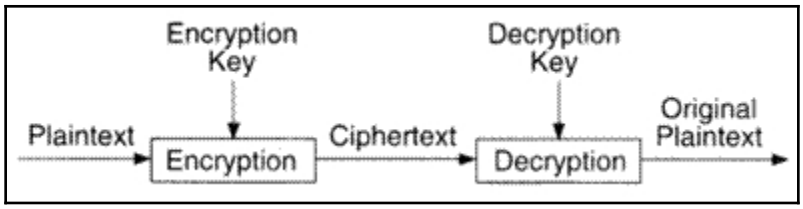

```
In [2]: print(C)

        PNYZ
```




```
In [36]: myHash
Out[36]: '$1$a6sQqHlF$j5iHhbCzmOzVwrxWDxnUu.'
```

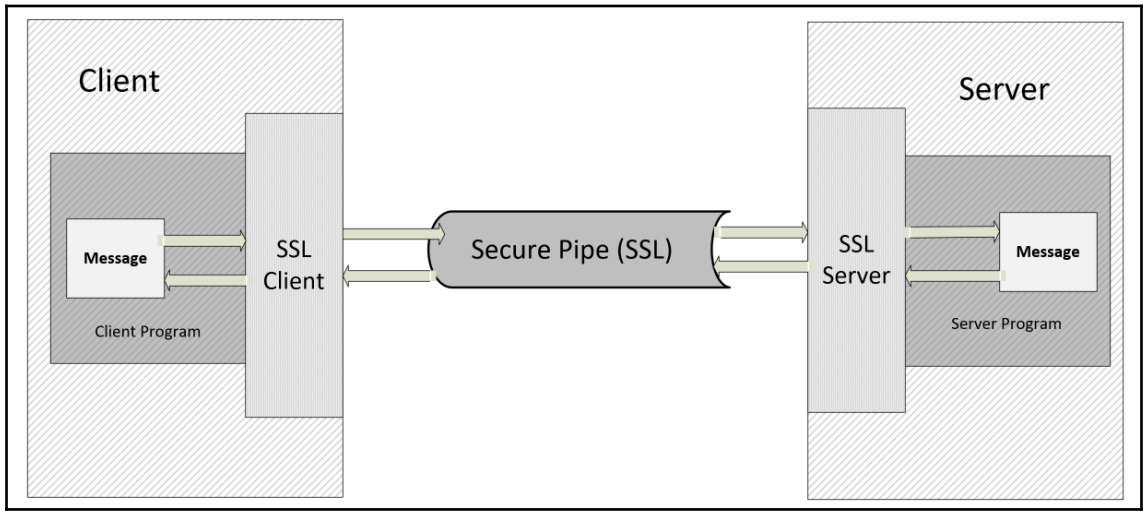
```
In [37]: md5_crypt.verify("myPassword", myHash)
Out[37]: True

In [38]: md5_crypt.verify("myPassword2", myHash)
Out[38]: False
```

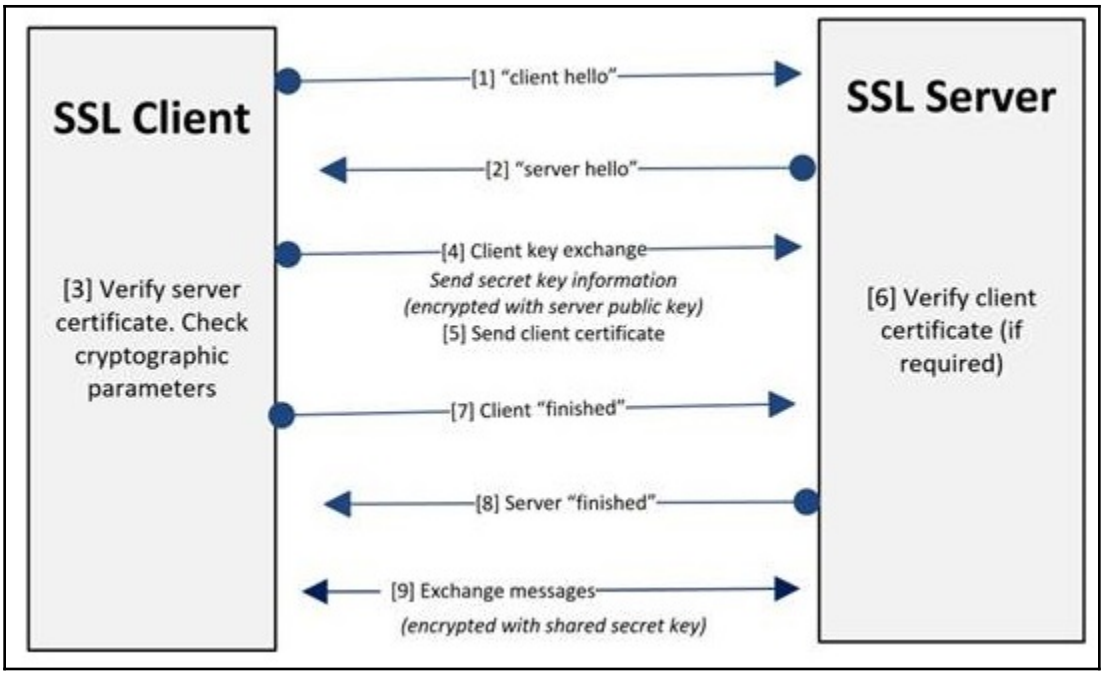
```
In [13]: myHash
Out[13]: '$6$qIo0foX5$a.RA/OyedLnLEnWovzqngCqhyy3EfqRtwacvWKsIoYSvYgRxCRetM3XSwrgMxwdPqZt4KfbXzCp6yNyxI5j6o/'
```

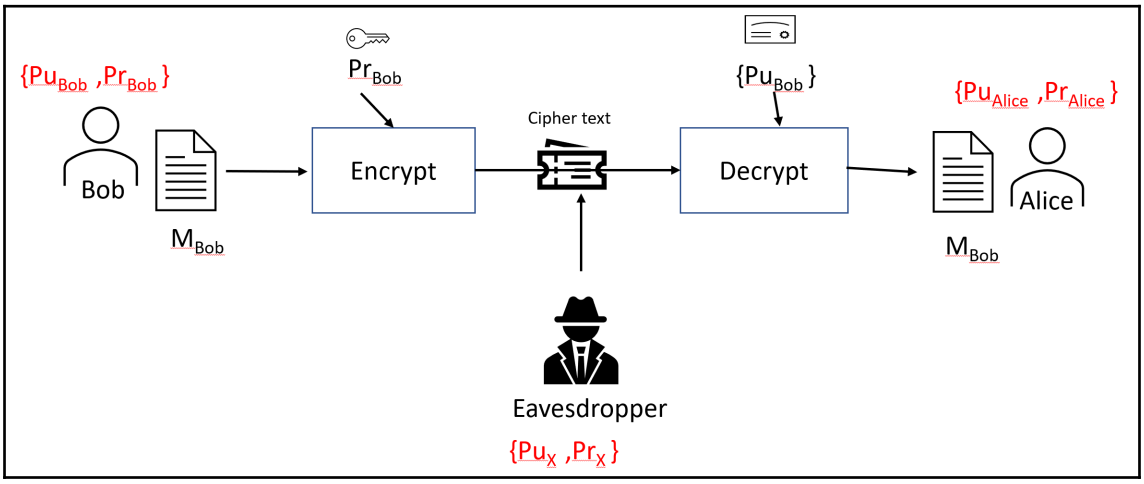



```
In [29]: key = Fernet.generate_key()
         print(key)

         b'NbzXiNqKR25SEv_O8EpuW2Lr_QO2vDStTDV4ex4WA5U='
```

```
In [46]: print(decrypted)

         b'Ottawa is really cold'
```

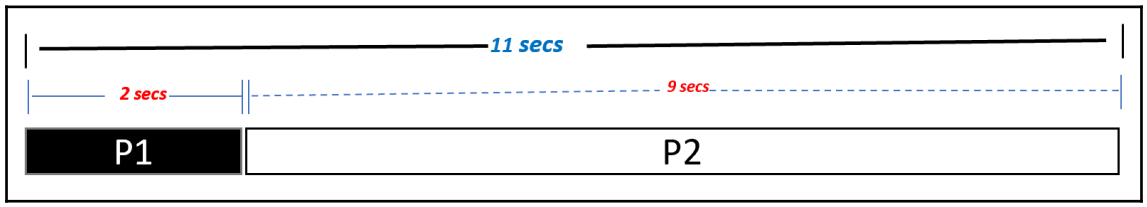



SSL Client
[3] Verify server certificate. Check cryptographic parameters
SSL Server
[6] Verify client certificate (if required)
[1] "client hello"
[2] "server hello"
[4] Client key exchange
Send secret key information
(encrypted with server public key)
[5] Send client certificate
[7] Client "finished"
[8] Server "finished"
[9] Exchange messages
(encrypted with shared secret key)


{PuBob ,PrBob }
PrBob
{PuBob }
{PuAlice ,PrAlice }
Bob
MBob
Encrypt
Cipher text
Decrypt
MBob
Alice
Eavesdropper
{PuX ,PrX }

```
In [21]: loaded_model = pickle.load(open(filename_destination, 'rb'))
         result = loaded_model.score(X_test, y_test)
         print(result)

         0.9473684210526315
```

Chapter 13: Large-Scale Algorithms

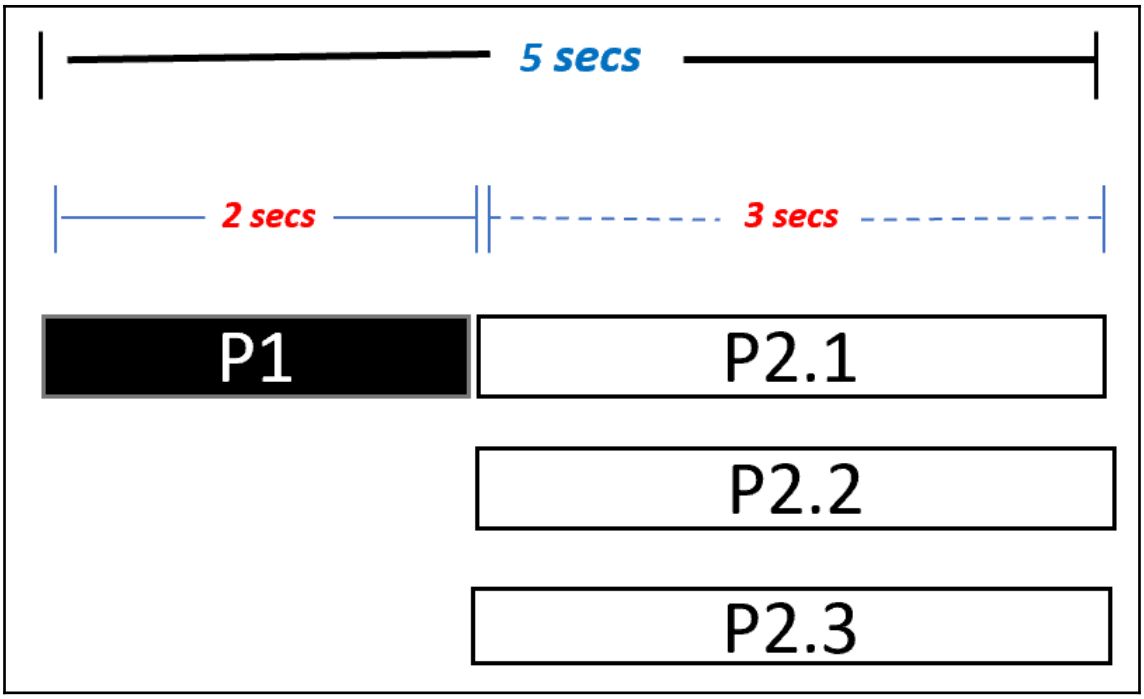


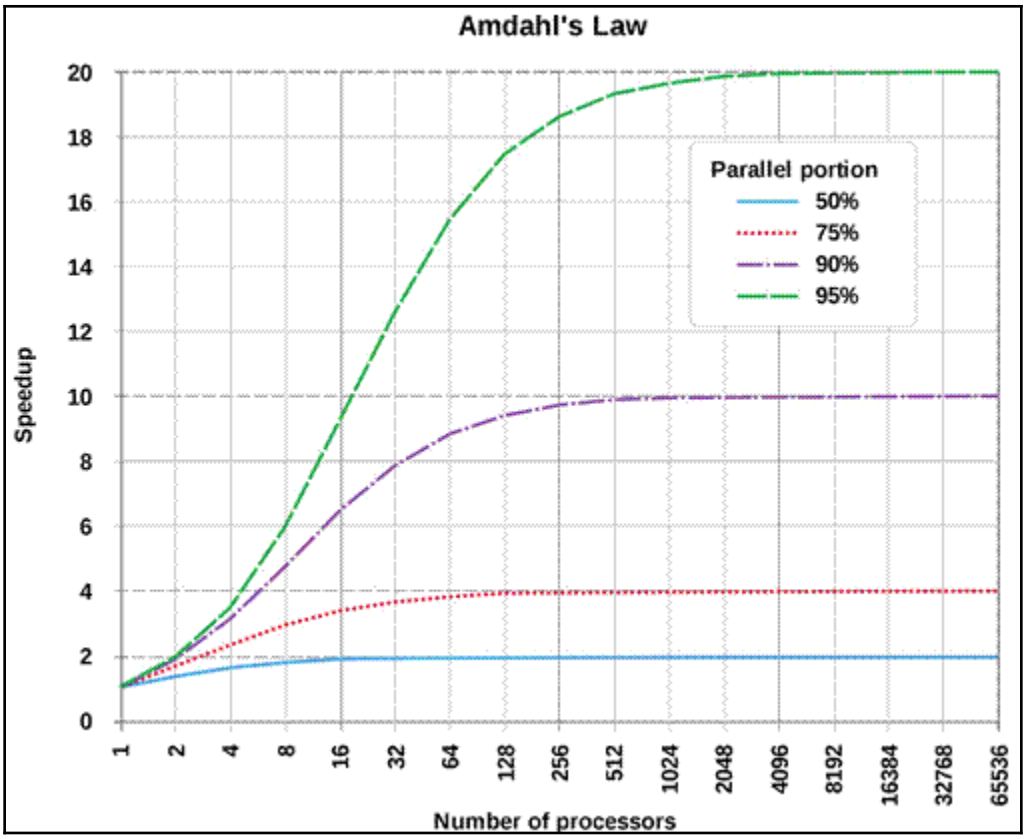
$$T_{par}(P) = T_{seq}(P1) + \frac{1}{s}T_{seq}(P2)$$

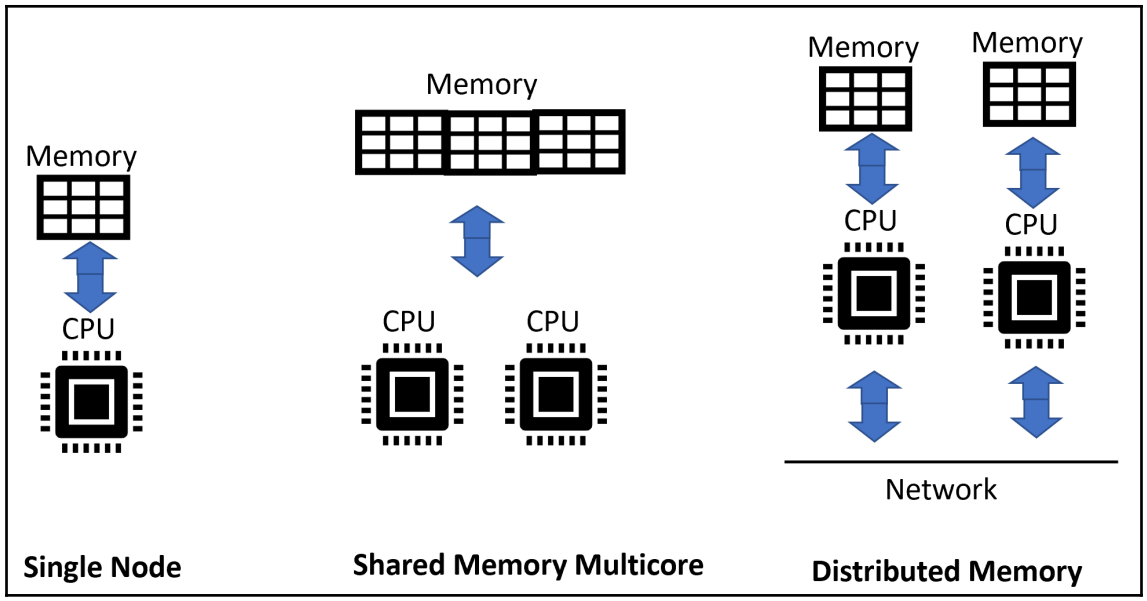
$$S(P) = \frac{T_{seq}(P)}{T_{par}(P)}$$

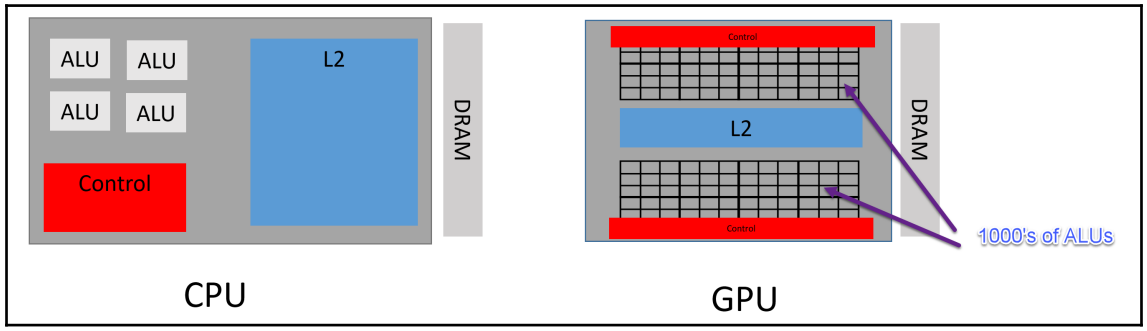
$$b = \frac{T_{seq}(P2)}{T_{seq}(P)}$$

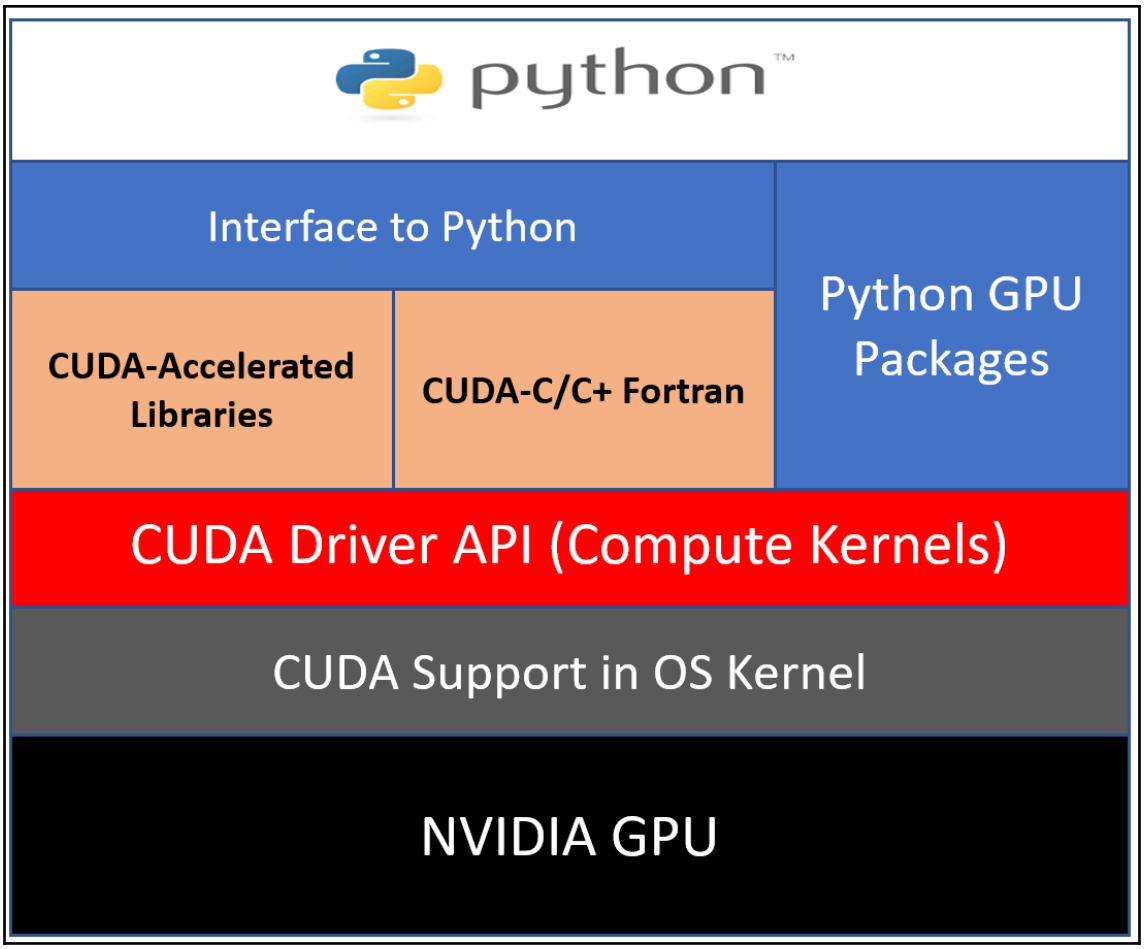
$$S(P) = \frac{1}{1 - b + \frac{b}{s}}$$


5 secs
2 secs
3 secs
P1
P2.1
P2.2
P2.3

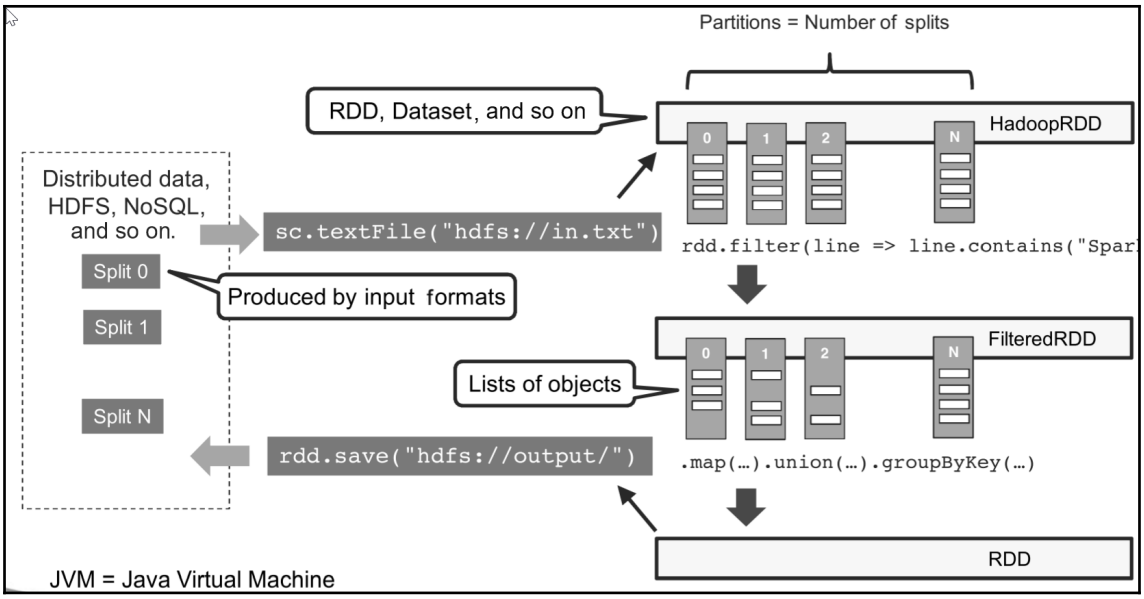

Amdahl's Law
Speedup
Number of processors
Parallel portion
50%
75%
90%
95%
0
2
4
6
8
10
12
14
16
18
20
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536


Memory
CPU
Single Node
Memory
CPU
CPU
Shared Memory Multicore
Memory
Memory
CPU
CPU
Network
Distributed Memory


ALU
ALU
ALU
ALU
Control
L2
DRAM
CPU
Control
L2
Control
DRAM
1000's of ALUs
GPU

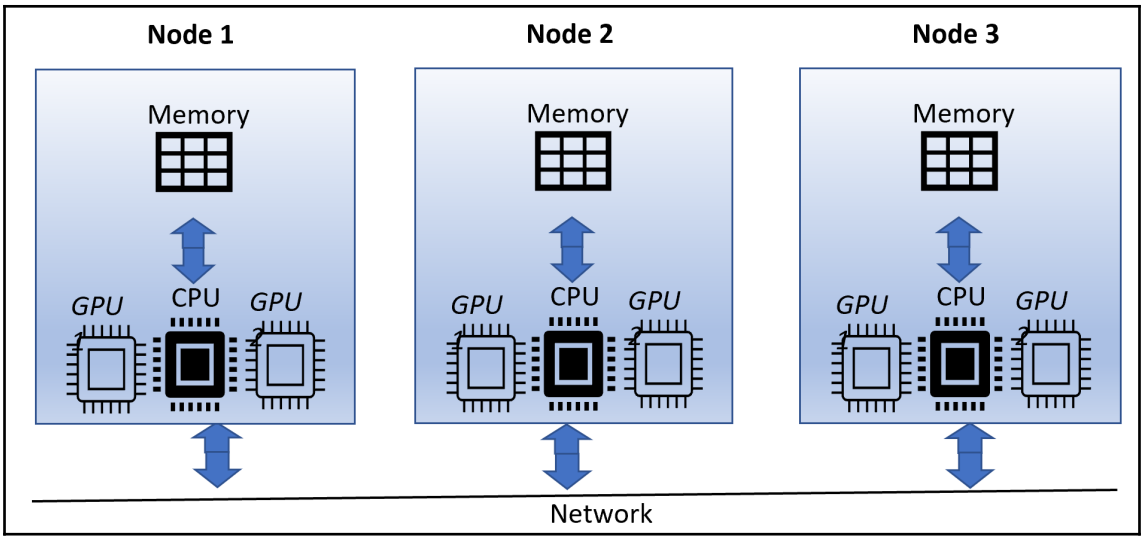



```
1.130657434463501
0.012250661849975586
```

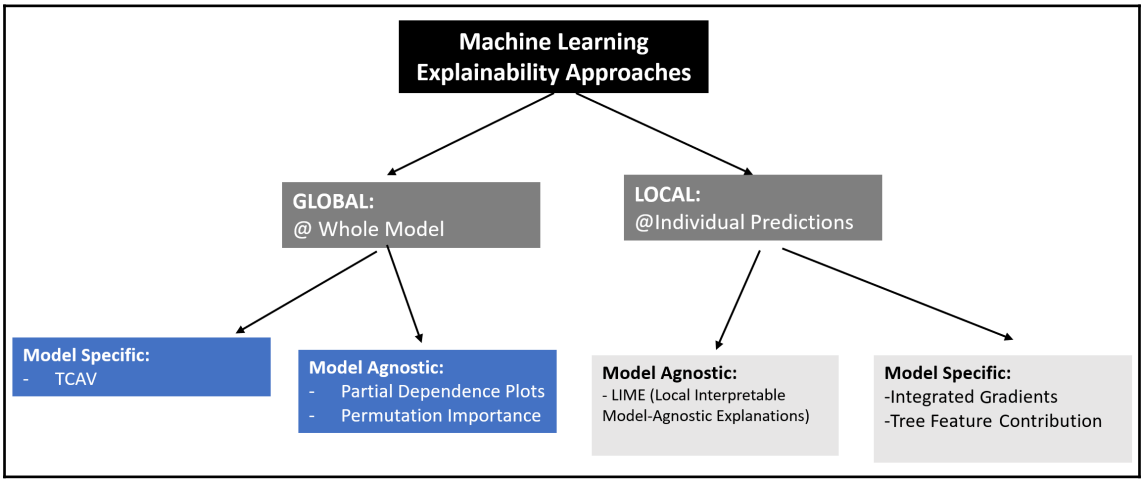

Partitions = Number of splits
RDD, Dataset, and so on
HadoopRDD
0
1
2
N
Distributed data, HDFS, NoSQL, and so on.
sc.textFile("hdfs://in.txt")
rdd.filter(line => line.contains("Spar
Split 0
Produced by input formats
Split 1
FilteredRDD
Lists of objects
Split N
rdd.save("hdfs://output/")
.map(…).union(…).groupByKey(…)
RDD
JVM = Java Virtual Machine

```
In [3]: df.columns

Out[3]: ['pickup_datetime',
         'dropoff_datetime',
         'pickup_longitude',
         'pickup_latitude',
         'dropoff_longitude',
         'dropoff_latitude',
         'passenger_count',
         'trip_distance',
         'payment_type',
         'fare_amount',
         'tip_amount',
         'tolls_amount',
         'total_amount']
```

```
In [9]: data=spark.sql("SELECT payment_type,Count(*) AS COUNT,AVG(fare_amount),
                    AVG(tip_amount) AS AverageFare from main GROUP BY payment_type")
        data.show()

+------------+-----+------------------+-----------------+
|payment_type|COUNT|  avg(fare_amount)|      AverageFare|
+------------+-----+------------------+-----------------+
|         CRD|10000|32.384988999999784| 7.61713200000006|
|         Cas| 3080| 34.64730519480518|7.497457792207749|
+------------+-----+------------------+-----------------+
```

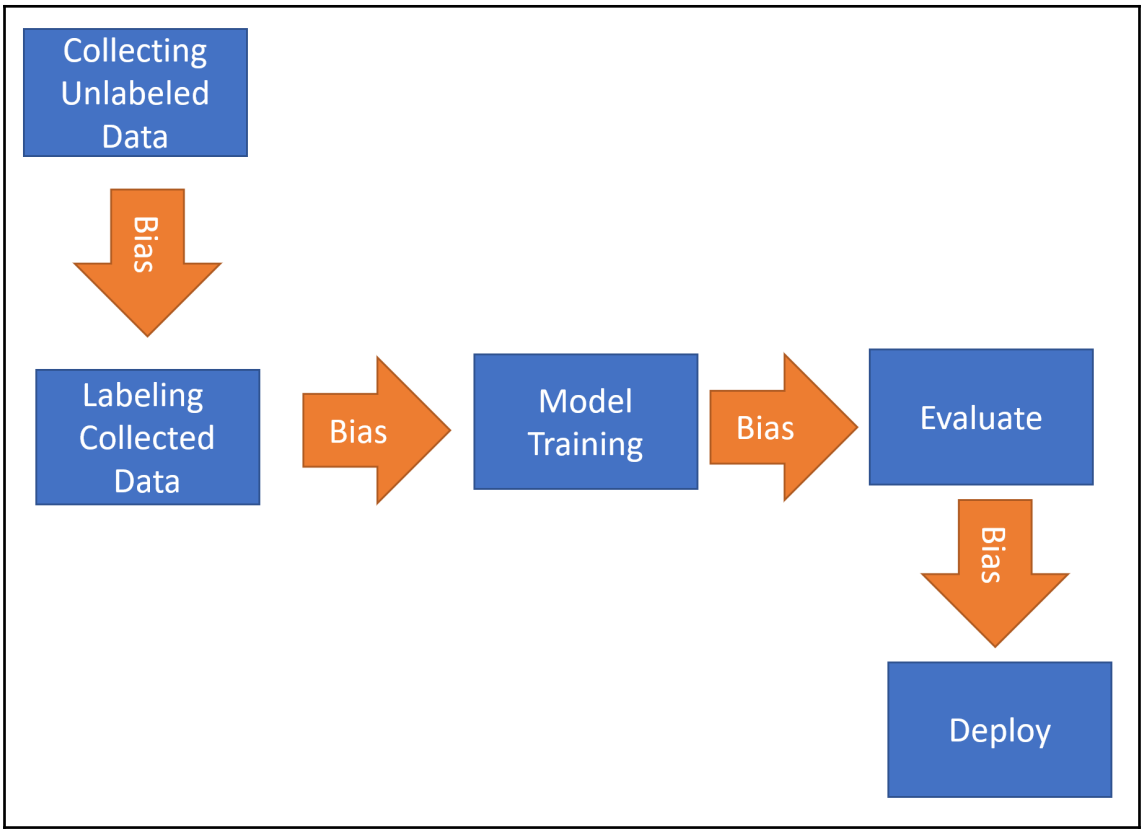

Node 1
Node 2
Node 3
Memory
Memory
Memory
GPU
CPU
GPU
GPU
CPU
GPU
GPU
CPU
GPU
Network

Chapter 14: Practical Considerations




```
In [2]: pkl_file = open("housing.pkl","rb")
        housing = pickle.load(pkl_file)
        pkl_file.close()
        housing['feature_names']

Out[2]: array(['crime_per_capita', 'zoning_prop', 'industrial_prop',
               'nitrogen_oxide', 'number_of_rooms', 'old_home_prop',
               'distance_from_city_center', 'high_way_access',
               'property_tax_rate', 'pupil_teacher_ratio', 'low_income_prop',
               'lower_status_prop', 'median_price_in_area'], dtype='<U25')
```

In [9]:
```
for i in [1, 35]:
    exp = myexplainer.explain_instance(X_test[i], regressor.predict,
            num_features=10)
    exp.as_pyplot_figure()
    plt.tight_layout()
```


Collecting Unlabeled Data
Bias
Labeling Collected Data
Bias
Model Training
Bias
Evaluate
Bias
Deploy