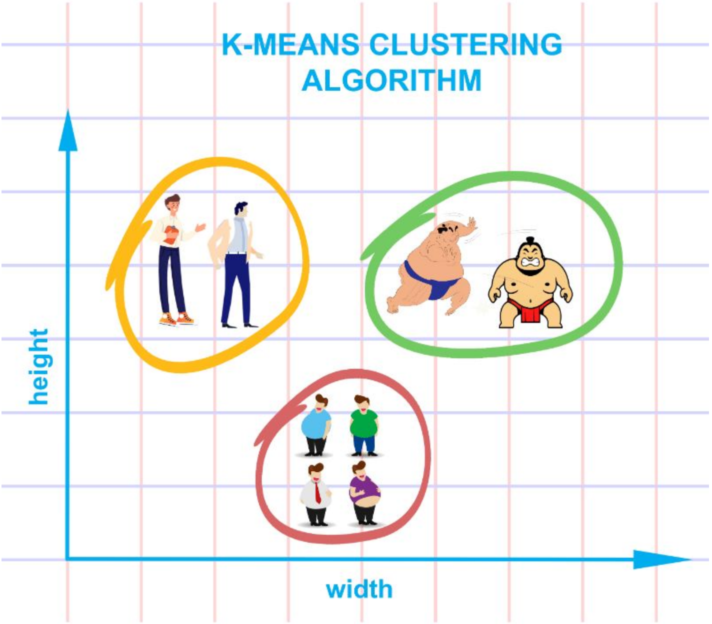
UNSUPERVISED LEARNING

Clustering Algorithms

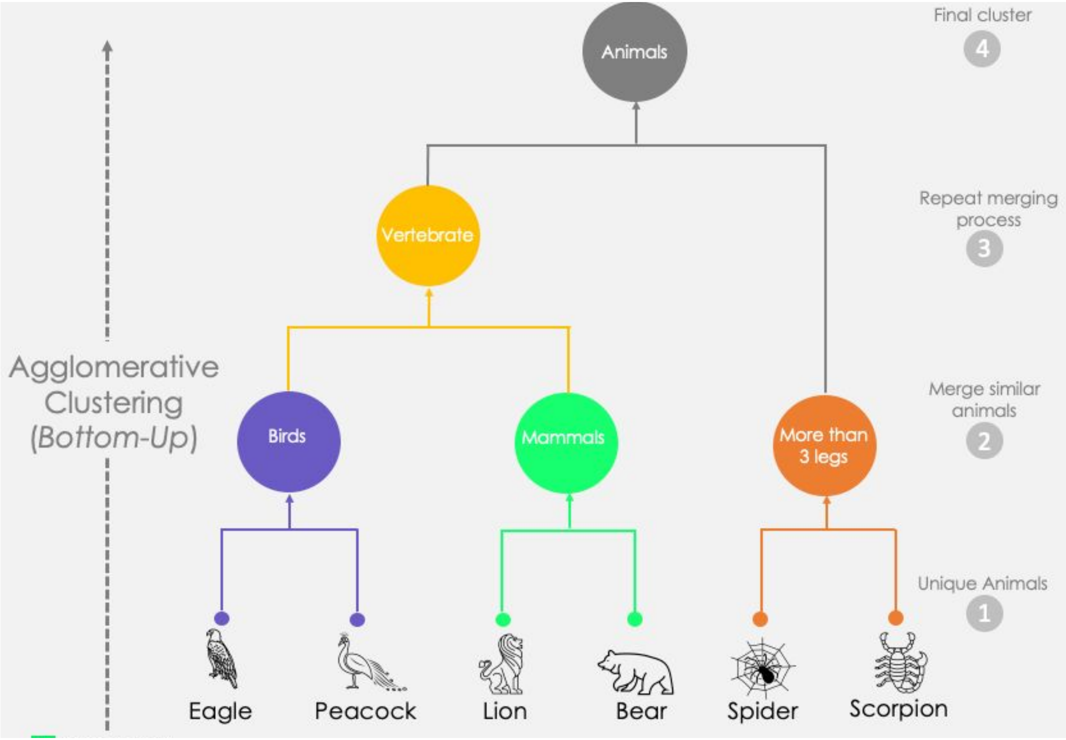
K-Means Clustering

```python
from sklearn.cluster import KMeans

data = list(zip(x, y))
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



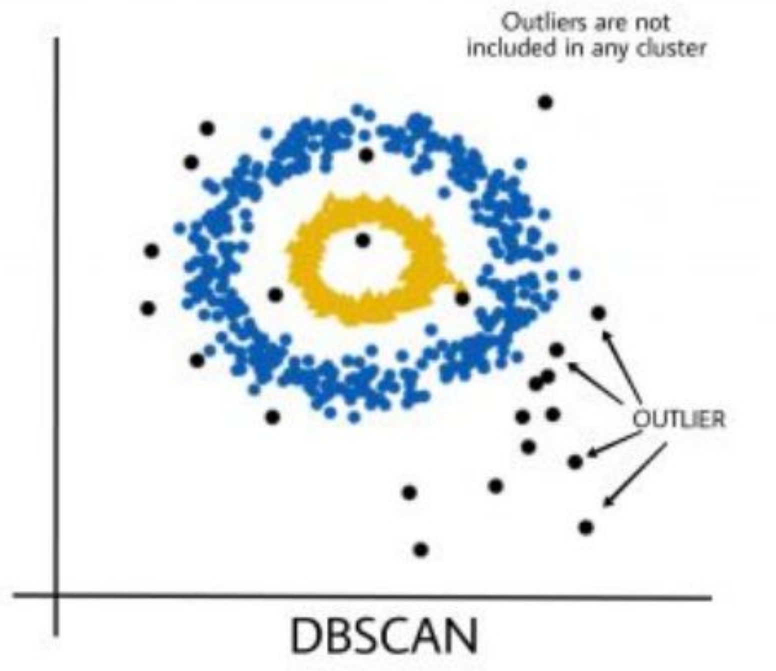
Hierarchical Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy
import dendrogram, linkage

x = [4, 5, 10, 4, 3, 11, 14 , 6,
10, 12]
y = [21, 19, 24, 17, 16, 25, 24,
22, 21, 21]

data = list(zip(x, y))

linkage_data = linkage(data,
method='ward',
metric='euclidean')
dendrogram(linkage_data)

plt.show()
```



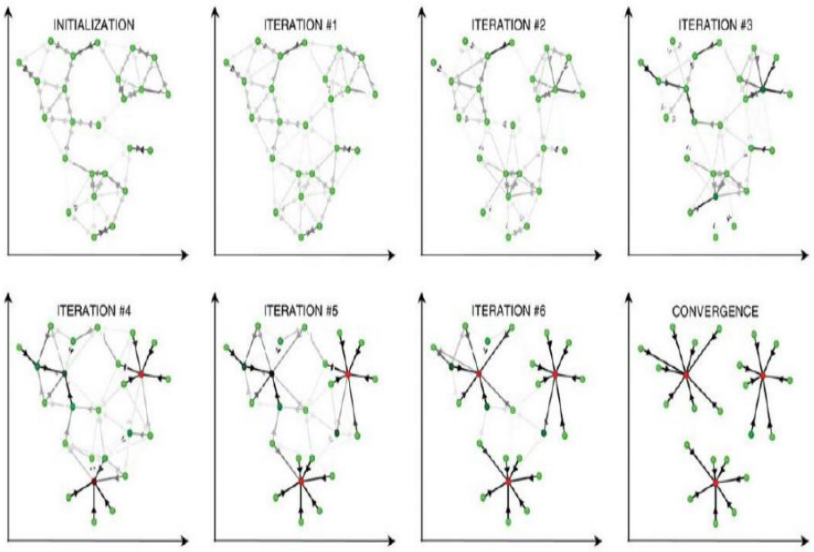
DBSCAN

```python
import numpy as np

from sklearn import metrics
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.3, min_samples=10).fit(X)
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)
```



Affinity Propegation

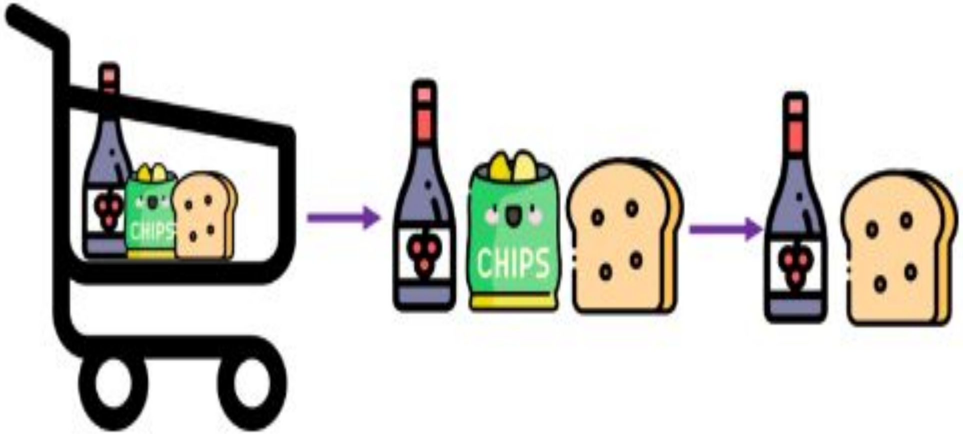
```
from sklearn.cluster import AffinityPropagation
from sklearn import metrics
from sklearn.datasets.samples_generator import
make_blobs

centers = [[1, 1], [-1, -1], [1, -1], [-1, -1]]
X, labels_true = make_blobs(n_samples = 400, centers
= centers, cluster_std = 0.5, random_state = 0)

af = AffinityPropagation(preference =-50).fit(X)
cluster_centers_indices = af.cluster_centers_indices_
labels = af.labels_

n_clusters_ = len(cluster_centers_indices)
```



Associative Algorithms

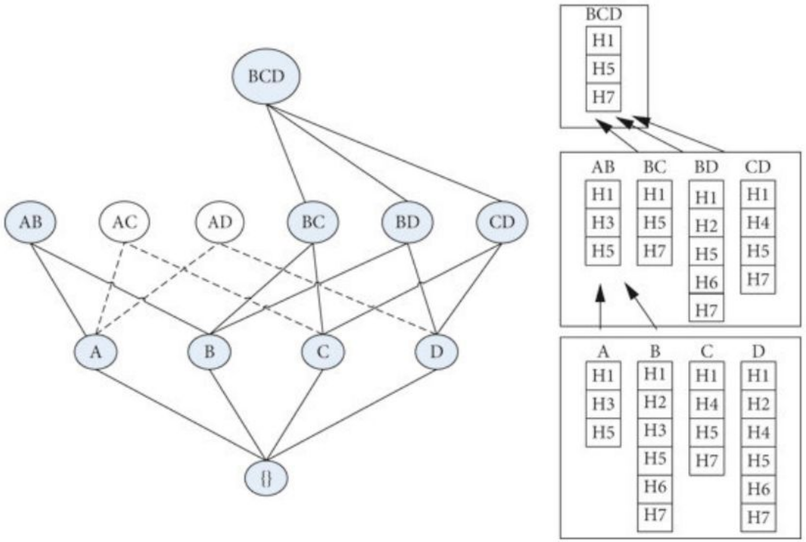
Apriori Algorithm

```python
# Build the model
frq_items1 = AP(basket1_France, min_support = 0.05, use_colnames = True)

# Collect the inferred rules in a dataframe
rules1 = AR(frq_items1, metric = "lift", min_threshold = 1)
rules1 = rules1.sort_values(['confidence', 'lift'], ascending = [False, False])
print(rules1.head())
```



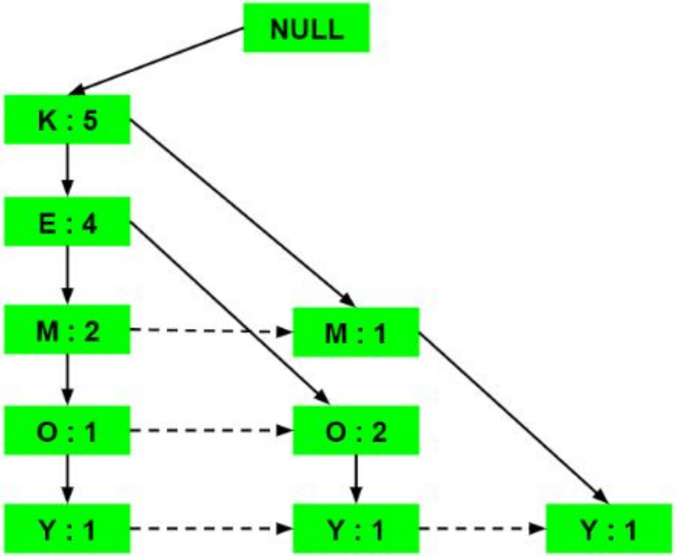
Eclat Algorithm

```
from pyECLAT import ECLAT

# create an instance of eclat
my_eclat = ECLAT(data=data, verbose=True)

# fit the algorithm
rule_indices, rule_supports =
my_eclat.fit(min_support=min_support,

min_combination=min_n_products,

max_combination=max_length)
```



FP Growth

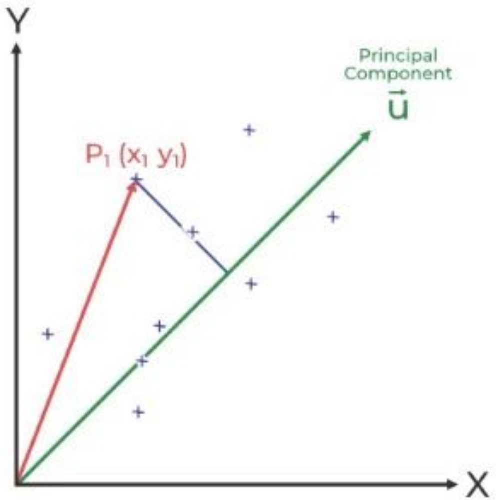
```python
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth

# Sample transaction dataset
dataset = [['Apple', 'Banana', 'Egg'],
           ['Banana', 'Egg', 'Milk'],
           ['Apple', 'Banana'],
           ['Banana', 'Milk']]

# Convert dataset to one-hot encoded format
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Mining frequent itemsets using FP-Growth
frequent_itemsets = fpgrowth(df, min_support=0.3,
use_colnames=True)

# Display the frequent itemsets
print(frequent_itemsets)
```



Dimensionality Reduction

Principal Component Analysis

```
# Matrix multiplication or
dot Product
Z_pca = Z @ pca_component
# Rename the columns name
Z_pca.rename({'PC1':
'PCA1', 'PC2': 'PCA2'},
axis=1, inplace=True)
# Print the  Principal
Component values
print(Z_pca)
```



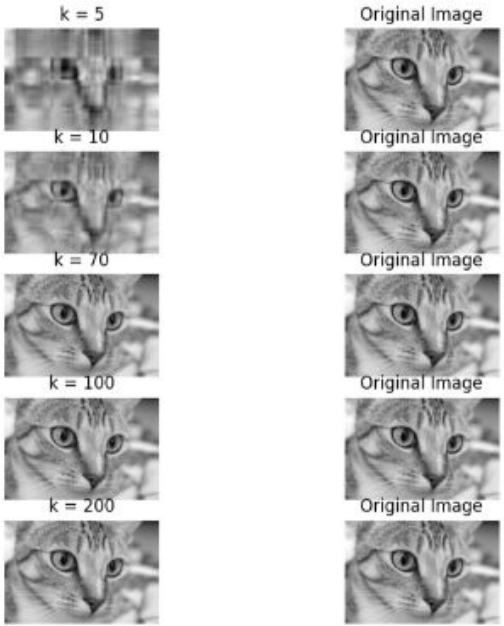
$$\mathrm{Proj}_{P_1}\vec{u} = \frac{P_1 \cdot \vec{u}}{|u|}$$

$$= P_1 \cdot \vec{u} \quad .....\vec{u} \rightarrow \text{Unit Vector}$$

Singular Value Decomposition

```
# Imports

from skimage.color import rgb2gray
from skimage import data
import matplotlib.pyplot as plt
import numpy as np
from scipy.linalg import svd

"""
SVD on image compression
"""
cat = data.chelsea()
plt.imshow(cat)
# convert to grayscale
gray_cat = rgb2gray(cat)

# calculate the SVD and plot the image
U, S, V_T = svd(gray_cat, full_matrices=False)
S = np.diag(S)
fig, ax = plt.subplots(5, 2, figsize=(8, 20))

curr_fig = 0
```

```
for r in [5, 10, 70, 100, 200]:
	cat_approx = U[:, :r] @ S[0:r, :r] @ V_T[:r,
:]
	ax[curr_fig][0].imshow(cat_approx,
cmap='gray')
	ax[curr_fig][0].set_title("k = "+str(r))
	ax[curr_fig, 0].axis('off')
	ax[curr_fig][1].set_title("Original Image")
	ax[curr_fig][1].imshow(gray_cat,
cmap='gray')
	ax[curr_fig, 1].axis('off')
	curr_fig += 1
plt.show()
```

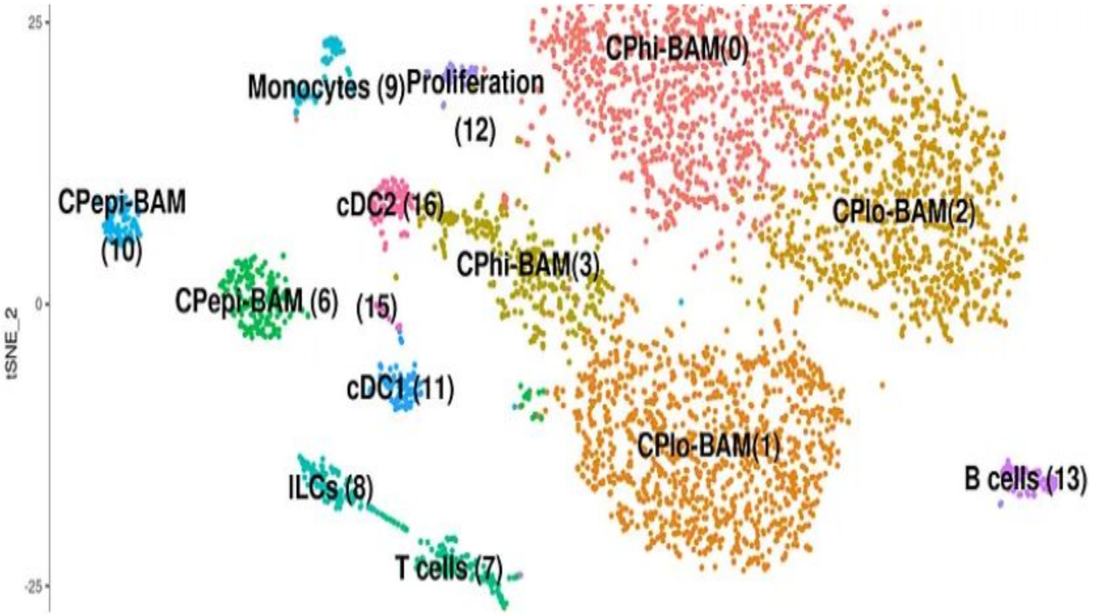


Independent Component Analysis

```
import numpy as np
from sklearn.decomposition import FastICA
import matplotlib.pyplot as plt
ica = FastICA(n_components=3)
S_ = ica.fit_transform(X) # Estimated sources

# Plot the results
plt.figure(figsize=(8, 6))
plt.subplot(3, 1, 1)
plt.title('Original Sources')
plt.plot(S)
plt.subplot(3, 1, 2)
plt.title('Observed Signals')
plt.plot(X)
plt.subplot(3, 1, 3)
plt.title('Estimated Sources (FastICA)')
plt.plot(S_)
plt.tight_layout()
plt.show()
```

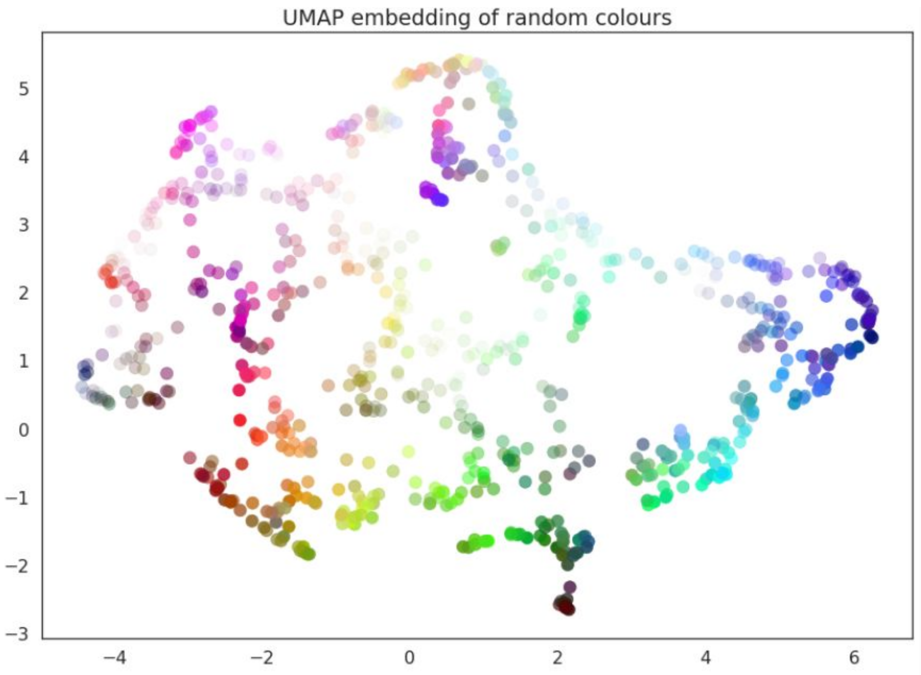
t-SNE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

data_1000 = standardized_data[0:1000, :]
labels_1000 = labels[0:1000]

model = TSNE(n_components = 2, random_state = 0)
tsne_data = model.fit_transform(data_1000)
tsne_data = np.vstack((tsne_data.T, labels_1000)).T
tsne_df = pd.DataFrame(data = tsne_data,columns
=("Dim_1", "Dim_2", "label"))

sn.scatterplot(data=tsne_df, x='Dim_1', y='Dim_2',
                         hue='label', palette="bright")
plt.show()
```



UMAP

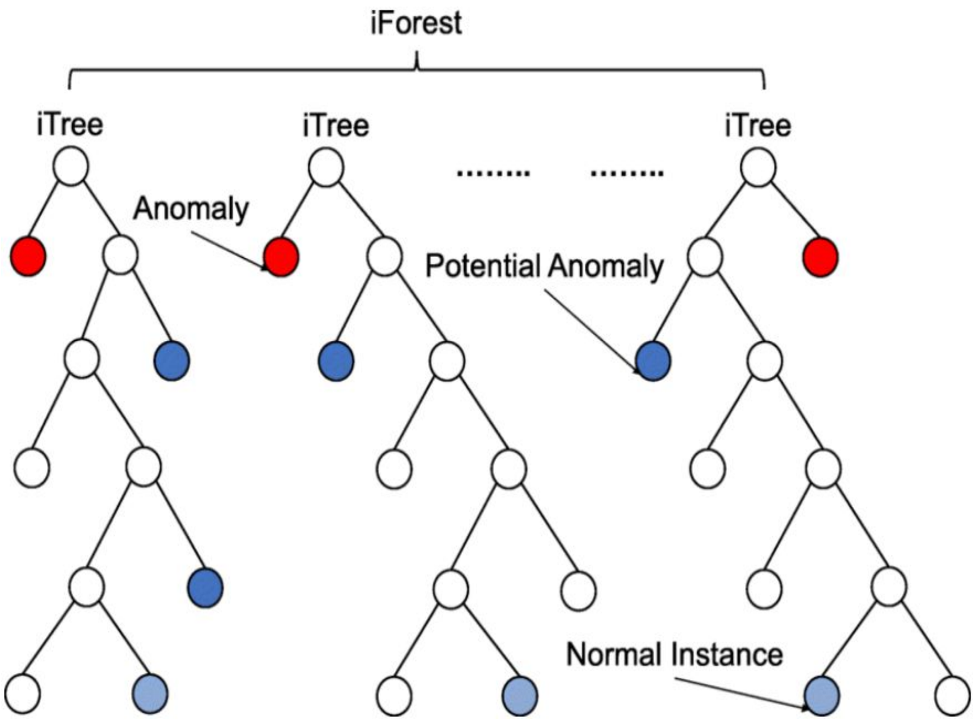
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
import umap
%matplotlib inline
fit = umap.UMAP()
%time u = fit.fit_transform(data)
CPU times: user 7.73 s, sys: 211 ms, total: 7.94 s
Wall time: 6.8 s
plt
plt.sc after(u[:,0],c=data)
plt.title('UMAP embedding of random colours');
```



Anomaly Detection

Isolation Forest

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

df = pd.read_csv('salary.csv')
df.head(10)
model=IsolationForest(n_estimators=50,
max_samples='auto',
contamination=float(0.1),max_features=1.0)
model.fit(df[['salary']])
```



One - Class SVM

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn import svm
%matplotlib inline

clf = svm.OneClassSVM()
clf.fit(X_train)
```