```
## k-means clustering
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas import DataFrame
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.cluster import KMeans

# data
MA = pd.read_csv('MA_A.csv', index_col=0)
MA.drop(['C1', 'C2'], axis='columns', inplace=True)
x = MA.iloc[:,0:]

# pca
x_1 = StandardScaler().fit_transform(x)
pca = PCA(n_components=2)
principalComponents_x = pca.fit_transform(x_1)
principalDf_x = pd.DataFrame(data = principalComponents_x
                           , columns = ['component 1', 'component 2'])

# normalization
a = principalDf_x.values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
a_scaled = min_max_scaler.fit_transform(a)

# model
kmeans = KMeans(n_clusters=3)
kmeans.fit(a_scaled)
a_scaled = pd.DataFrame(a_scaled, columns=principalDf_x.columns)
predict = pd.DataFrame(kmeans.predict(a_scaled))
predict.columns=['predict']
r = pd.concat([a_scaled,predict],axis=1)

# visualization
colors = ("blue", "red", "green")
plt.scatter(r['component 1'],r['component 2'],c=r['predict'],alpha=0.7)
centers = pd.DataFrame(kmeans.cluster_centers_,columns=['component 1','component 2'])
center_x = centers['component 1']
center_y = centers['component 2']
plt.scatter(center_x,center_y,s=100,marker='D',c=colors)

# plot
plt.title('k-means on MA')
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.savefig('k-means.png', dpi = 300)
plt.show()
```

```
## Support Vector Machine
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas import DataFrame
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn import svm
from sklearn.svm import SVC
from mlxtend.plotting import plot_decision_regions

# data
MA = pd.read_csv('MA_A.csv', index_col=0)
y_sv = MA.loc[:,['C1']].values
y_1_sv = pd.DataFrame(data=y_sv, columns = ['C1'])
MA.drop(['C1', 'C2'], axis='columns', inplace=True)
x = MA.iloc[:,0:]

# pca
x_1 = StandardScaler().fit_transform(x)
pca = PCA(n_components=2)
principalComponents_x = pca.fit_transform(x_1)
principalDf_x = pd.DataFrame(data = principalComponents_x
                          , columns = ['component 1', 'component 2'])

# normalization
a = principalDf_x.values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
a_scaled = min_max_scaler.fit_transform(a)

# model
svm = SVC(C=10, kernel='linear')
svm.fit(a_scaled, y_1_sv.values.ravel())
svm.score(a_scaled, y_1_sv.values.ravel(), sample_weight=None)


# visualization
plot_decision_regions(a_scaled, y_1_sv.values.ravel(), clf=svm, legend=2)
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.title('SVM on MA')
plt.savefig('SVM.png', dpi = 300)
plt.show()


# cross validation
from sklearn.model_selection import cross_val_score

scores = cross_val_score(svm, a_scaled, y_1_sv.values.ravel(), cv=7)
print("cross validation:", scores)
```

```
## Deep Neural Network

from keras.models import Sequential
from keras.layers.core import Dense
from keras.utils import np_utils
from keras.layers.normalization import BatchNormalization
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# seed
seed = 0
np.random.seed(seed)

# data(load user data)
df = pd.read_csv('../dataset/MA_A.csv')
dataset = df.values
X_ = dataset[:,2:]
Y_obj = dataset[:,1]
Z_id = dataset[:,0]
X = preprocessing.normalize(X_)
e = LabelEncoder()
e.fit(Y_obj)
Y = e.transform(Y_obj)
Y = np_utils.to_categorical(Y)

# model
kfold = KFold(n_splits=7, shuffle=True, random_state=seed)

accuracy = []
loss = []

for train, validation in kfold.split(X, Y):
    model = Sequential()
    model.add(Dense(60, input_dim=X.shape[1], activation='relu'))
    model.add(Dense(60, activation='relu'))
    model.add(Dense(60, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.summary()
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    hist = model.fit(X[train], Y[train], epochs=1000, batch_size=10, verbose=0)

    k_accuracy = model.evaluate(X[validation], Y[validation])[1]
    k_loss = model.evaluate(X[validation], Y[validation])[0]

    accuracy.append(k_accuracy)
    loss.append(k_loss)


result = pd.DataFrame({'acc': accuracy, 'loss': loss})
print('₩nDNN Mean accuracy: ''%.2f' % (result["acc"].mean()*100))
print('₩nDNN Mean loss: ''%.2f' % (result["loss"].mean()*100))
```