```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

Reading the Data

```
mData = pd.read_csv('winequality-red.csv')
print('The Shape of The Data ',mData.shape)
mData.info()
```

```
The Shape of The Data  (1599, 12)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
fixed acidity           1599 non-null float64
volatile acidity        1599 non-null float64
citric acid             1599 non-null float64
residual sugar          1599 non-null float64
chlorides               1599 non-null float64
free sulfur dioxide     1599 non-null float64
total sulfur dioxide    1599 non-null float64
density                 1599 non-null float64
pH                      1599 non-null float64
sulphates               1599 non-null float64
alcohol                 1599 non-null float64
quality                 1599 non-null int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Preprocessing the Data

```
from sklearn import preprocessing
from sklearn.utils import resample


def Preprocessing_Data(data):
  data['quality'] = data["quality"] - 3
  Y = data['quality']
  X = data.drop(columns=['quality'])
  scaler = preprocessing.MinMaxScaler().fit(X)
  x_normalized = scaler.transform(X)
  newData = pd.DataFrame(columns= X.columns,data=x_normalized)
  newData['quality'] = Y
  # ---------------------------Ballancing the Data set----------------
  Categories = pd.Categorical(Y.astype('object')).categories
  numOfCategories = Categories.size
  CategorySizes = Y.value_counts()
  maxSize = Y.value_counts().max()
  mSampledDate =[]
  for Category in Categories :
    if (newData[Y==Category].shape[0] != maxSize):
      temp = resample(newData[Y==Category],
                      replace=True,     # sample with replacement
                      n_samples=(maxSize-data[Y==Category].shape[0]),
                      random_state=123) # reproducible results
      mSampledDate.append(temp)

  BallancedData=newData
  for Data in mSampledDate:
    BallancedData = pd.concat([BallancedData, Data])
  print("total shape after up sampling = ", BallancedData.shape)
  BallancedData = pd.DataFrame(BallancedData)
  BallancedData.index = np.array(range(0,BallancedData.shape[0]))

  return BallancedData , scaler
```

```
DataFinal,scaler = Preprocessing_Data(mData)
```

```
total shape after up sampling =  (4086, 12)
(4086, 12)
```

Data Helping Functions

```
def GetFeature_Output(data):
    y = data.quality
    x = data.drop(columns=['quality'])
    return x,y

def addOutput(x,y):
    x["quality"]=y
    return x
```

Results Variables

```
myResultsDF = pd.DataFrame()
myResultsDF['Metric'] = ['completeness_score','adjusted_rand_score','fowlkes_mallows_score','silhouette_score','calinski_ha
myResultsDF
```

```
                Metric
0   completeness_score
1  adjusted_rand_score
2 fowlkes_mallows_score
3     silhouette_score
4 calinski_harabaz_score
5         CentroidRMSE
```

Evaluation Functions

```
#Centroid Evaluation Function
def CentroidRMSE(Features,Clutered1,Clustered2):
    #print("C1 IN ",Clutered1.shape)
    #print("C2 IN ",Clustered2.shape)

    X1 = Features.copy()
    X1["C1"] = Clutered1
    C1Means = X1.groupby("C1").mean().sort_values(X1.columns[0])

    X2 = Features.copy()
    X2["C2"] = Clustered2
    C2Means = X2.groupby("C2").mean().sort_values(X2.columns[0])
    #print("C1 IN Arranged ",C1Means.shape)
    #print("C2 IN Arranged",C2Means.shape)
    #if(C1Means.shape[0]>C2Means.shape[0]):
    #  diff = C2Means.shape[0] - C1Means.shape[0]
    #  C1Means = C1Means[:diff]
    #  print("C1 IN Changed",C1Means.shape)
    #elif(C1Means.shape[0]<C2Means.shape[0]):
    #  diff = C1Means.shape[0] - C2Means.shape[0]
    #  C2Means = C2Means[:diff]
    #  print("C2 IN Changed ",C2Means.shape)


    RMSE = np.sqrt(metrics.mean_squared_error(C1Means,C2Means))
    return RMSE


def Evalution(X,Y,Y_Pred):
  result = []
  result.append(metrics.completeness_score(Y, Y_Pred))
  result.append(metrics.adjusted_rand_score(Y, Y_Pred))
  result.append(metrics.fowlkes_mallows_score(Y,Y_Pred))
  result.append(metrics.silhouette_score(X, Y_Pred, metric='euclidean'))
  result.append(metrics.calinski_harabaz_score(X, Y_Pred))
  result.append(CentroidRMSE(X,Y,Y_Pred))
  return result
```

K_Mean Test

```
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.model_selection import train_test_split,StratifiedKFold
```

```
def K_Mean_Test(data,clusters_num):
  # full Data
  X , Y = GetFeature_Output(data)
  Clusterer = KMeans(n_clusters=clusters_num, random_state=0).fit(X)
  Y_Result = Clusterer.predict(X)
  Test_Result = Evalution(X,Y.values,Y_Result)
  # Train_ Test
  x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.4, random_state=0 )
  Clusterer = KMeans(n_clusters=clusters_num, random_state=0).fit(x_train)
  Y_Result = Clusterer.predict(x_test)
  result_train_test = Evalution(x_test, y_test, Y_Result)


  KfordScorer = []
  k_fold = StratifiedKFold(n_splits=5)
  Clusterer =  KMeans(n_clusters=clusters_num, random_state=0)
  for train_indices, test_indices in k_fold.split(X,Y):
    Clusterer.fit(X.iloc[train_indices])
    Y_Result = Clusterer.predict(X.iloc[test_indices])
    x_test = X.iloc[test_indices]
    y_test = Y.iloc[test_indices]
    resultTemp = Evalution(x_test, y_test, Y_Result)
    KfordScorer.append(resultTemp)
  results_KFolds= np.mean(KfordScorer,axis=0)

  return Test_Result,result_train_test,results_KFolds
```

```
r1,r2,r3 = K_Mean_Test(DataFinal,6)
```

```
myResultsDF['Kmean_Full'] = r1
myResultsDF['Kmean_FTrain_Test'] = r2
myResultsDF['Kmean_FK-folds'] = r3
myResultsDF
```

	Metric	Kmean_Full	Kmean_FTrain_Test	Kmean_FK-folds
0	completeness_score	0.220732	0.196039	0.254787
1	adjusted_rand_score	0.142946	0.112454	0.164138
2	fowlkes_mallows_score	0.300667	0.276166	0.323162
3	silhouette_score	0.234202	0.227988	0.239240
4	calinski_harabaz_score	1032.076059	401.171639	206.378573
5	CentroidRMSE	0.129906	0.144322	0.163664

PCA -> K_Mean Test

```
from sklearn.decomposition import PCA
```

```
def PCA_K_Mean_Test(data,clusters_num,PCA_comp_num):

  # full Data
  X_Original,Y_Original = GetFeature_Output(data)
  scikit_pca = PCA(n_components=PCA_comp_num)
  X = scikit_pca.fit_transform(X_Original)
  X = pd.DataFrame(X)
  Clusterer = KMeans(n_clusters=clusters_num, random_state=0).fit(X)
  Y_Result = Clusterer.predict(X)
  Test_Result = Evalution(X_Original,Y_Original.values,Y_Result)



  # Train_ Test
  x_train, x_test, y_train, y_test = train_test_split(X, Y_Original, test_size=0.4, random_state=0 )
  Clusterer = KMeans(n_clusters=clusters_num, random_state=0).fit(x_train)
  Y_Result = Clusterer.predict(x_test)
  result_train_test = Evalution(X_Original.iloc[x_test.index], y_test, Y_Result)


  KfordScorer = []
  k_fold = StratifiedKFold(n_splits=5)
  Clusterer =  KMeans(n_clusters=clusters_num, random_state=0)
  for train_indices, test_indices in k_fold.split(X,Y_Original):
    Clusterer.fit(X.iloc[train_indices])
    Y_Result = Clusterer.predict(X.iloc[test_indices])
    x_test = X.iloc[test_indices]
    y_test = Y_Original.iloc[test_indices]
    resultTemp = Evalution(X_Original.iloc[x_test.index], y_test, Y_Result)
    KfordScorer.append(resultTemp)
  results_KFolds= np.mean(KfordScorer,axis=0)

  return Test_Result,result_train_test,results_KFolds
```

```
r1,r2,r3 = PCA_K_Mean_Test(DataFinal,6,5)
```

```
myResultsDF['PCA_Full'] = r1
myResultsDF['PCA_Train_Test'] = r2
```

```
myResultsDF['PCA_K-folds'] = r3
myResultsDF
```

	Metric	Kmean_Full	Kmean_FTrain_Test	Kmean_FK-folds	PCA_Full	PCA_Train_Test	PCA_K-folds
0	completeness_score	0.220732	0.196039	0.254787	0.218227	0.214853	0.232714
1	adjusted_rand_score	0.142946	0.112454	0.164138	0.141182	0.136143	0.140458
2	fowlkes_mallows_score	0.300667	0.276166	0.323162	0.297863	0.293402	0.298038
3	silhouette_score	0.234202	0.227988	0.239240	0.233839	0.227327	0.228807
4	calinski_harabaz_score	1032.076059	401.171639	206.378573	1032.396724	401.099790	204.639195
5	CentroidRMSE	0.129906	0.144322	0.163664	0.144000	0.125932	0.132044

AutoEncoder - > K_mean Test

```
from keras.models import Model
from keras.layers import Input, Dense, Dropout ,BatchNormalization
```

```
Using TensorFlow backend.
```

```
def create_simple_AE(X,Y, enc_size, activation_H, activation_out):


  in_layer = Input(shape=(X.shape[1],))
  enc_layer = Dense(enc_size, activation=activation_H)(in_layer)
  dec_layer = Dense(X.shape[1], activation=activation_out)(enc_layer)

  AE = Model(in_layer, dec_layer)
  Enc = Model(in_layer, enc_layer)

  AE.compile(optimizer='adam', loss='mean_squared_error')

  x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0 )
  AE.fit(x_train, x_train,
         epochs=50,
         batch_size=50,
         shuffle=True,
         verbose=0,
         validation_data=(x_test, x_test))


  return AE, Enc


def AE_Kmean_Test(data,encSize,activation,clusters_num):

  # full Data
  X_Original,Y_Original = GetFeature_Output(data)
  AE,Enc = create_simple_AE(X_Original,Y_Original,encSize,activation,activation)

  X = pd.DataFrame(Enc.predict(X_Original))
  Clusterer = KMeans(n_clusters=clusters_num, random_state=0).fit(X)
  Y_Result = Clusterer.predict(X)
  Test_Result = Evalution(X_Original,Y_Original.values,Y_Result)




  # Train_ Test
  x_train, x_test, y_train, y_test = train_test_split(X_Original, Y_Original, test_size=0.4, random_state=0)

  AE,Enc = create_simple_AE(x_train,y_train,encSize,activation,activation)
  X_train_encoded = pd.DataFrame(Enc.predict(x_train))
  X_test_encoded = pd.DataFrame(Enc.predict(x_test))

  Clusterer = KMeans(n_clusters=clusters_num, random_state=0).fit(X_train_encoded)
  Y_Result = Clusterer.predict(X_test_encoded)
  result_train_test = Evalution(X_Original.iloc[x_test.index], y_test, Y_Result)


  # K-Folds
  KfordScorer = []
  k_fold = StratifiedKFold(n_splits=5)
  for train_indices, test_indices in k_fold.split(X_Original,Y_Original):
    AE,Enc = create_simple_AE(X_Original.iloc[train_indices],Y_Original.iloc[train_indices],encSize,activation,activation)
    X_train_encoded = pd.DataFrame(Enc.predict(X_Original.iloc[train_indices]))
    X_test_encoded = pd.DataFrame(Enc.predict(X_Original.iloc[test_indices]))

    Clusterer = KMeans(n_clusters=clusters_num, random_state=0).fit(X_train_encoded)
    Y_Result = Clusterer.predict(X_test_encoded)
    y_test = Y_Original.iloc[test_indices]
    resultTemp = Evalution(X_Original.iloc[test_indices], y_test, Y_Result)
    KfordScorer.append(resultTemp)
  results_KFolds= np.mean(KfordScorer,axis=0)

  return Test_Result,result_train_test,results_KFolds
```

```
r1,r2,r3 = AE_Kmean_Test(DataFinal,10,'sigmoid',6)


myResultsDF['AE_KMean_Full'] = r1
myResultsDF['AE_KMean__Train_Test'] = r2
myResultsDF['AE_KMean__K-folds'] = r3
myResultsDF
```

```
›/python3.6/dist-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.:

›lacer.
›/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
```

an_FTrain_Test	Kmean_FK-folds	PCA_Full	PCA_Train_Test	PCA_K-folds	AE_KMean_Full	AE_KMean__Train_Test	AE_KMean__K-folds
0.196039	0.254787	0.218227	0.214853	0.232714	0.231459	0.198721	0.260030
0.112454	0.164138	0.141182	0.136143	0.140458	0.166723	0.130478	0.170808
0.276166	0.323162	0.297863	0.293402	0.298038	0.309826	0.284217	0.319184
0.227988	0.239240	0.233839	0.227327	0.228807	0.194350	0.171895	0.202508
401.171639	206.378573	1032.396724	401.099790	204.639195	994.602390	357.920717	195.884810
0.144322	0.163664	0.144000	0.125932	0.132044	0.132300	0.114560	0.124141

AutoEncoder with SoftMax layer

```
from keras.activations import softmax


def create_softmax_AE(X,Y, enc_size, activation_H, activation_out):
  in_layer = Input(shape=(X.shape[1],))
  enc_layer = Dense(enc_size, activation=activation_H)(in_layer)
  enc_layer = BatchNormalization()(enc_layer)
  enc_layer= Dropout(0.7)(enc_layer)
  enc_layer = Dense(Y.value_counts().size, activation='softmax')(enc_layer)
  enc_layer = BatchNormalization()(enc_layer)
  enc_layer = Dropout(0.7)(enc_layer)
  dec_layer = Dense(enc_size, activation=activation_out)(enc_layer)
  dec_layer = BatchNormalization()(dec_layer)
  dec_layer = Dropout(0.7)(dec_layer)
  dec_layer = Dense(X.shape[1], activation=activation_out)(dec_layer)

  AE = Model(in_layer, dec_layer)
  Enc = Model(in_layer, enc_layer)

  AE.compile(optimizer='adam', loss='mean_squared_error')

  x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0 )
  AE.fit(x_train, x_train,
         epochs=50,
         batch_size=50,
         shuffle=True,
         verbose=0,
         validation_data=(x_test, x_test))


  return AE, Enc


def AE_SoftMax_Test(data,encSize,activation):

  # full Data
  X_Original,Y_Original = GetFeature_Output(data)
  AE,Enc = create_softmax_AE(X_Original,Y_Original,encSize,activation,activation)

  EncOutput = pd.DataFrame(Enc.predict(X_Original))
  Y_Result = EncOutput.idxmax(axis=1)
  Test_Result = Evalution(X_Original,Y_Original.values,Y_Result)




  # Train_ Test
  x_train, x_test, y_train, y_test = train_test_split(X_Original, Y_Original, test_size=0.4, random_state=0)
  AE,Enc = create_softmax_AE(x_train,y_train,encSize,activation,activation)
  EncOutput = pd.DataFrame(Enc.predict(x_test))
  Y_Result = EncOutput.idxmax(axis=1)
  result_train_test = Evalution(X_Original.iloc[x_test.index], y_test, Y_Result.values)


  # K-Folds
  KfordScorer = []
  k_fold = StratifiedKFold(n_splits=5)
  for train_indices, test_indices in k_fold.split(X_Original,Y_Original):
    AE,Enc = create_softmax_AE(X_Original.iloc[train_indices],Y_Original.iloc[train_indices],encSize,activation,activation)
    EncOutput = pd.DataFrame(Enc.predict(X_Original.iloc[test_indices]))
    Y_Result = EncOutput.idxmax(axis=1)
    #print("C2" ,Y_Result.value_counts())
    y_test = Y_Original.iloc[test_indices]
    #print("C1",y_test.value_counts())
    resultTemp = Evalution(X_Original.iloc[test_indices], y_test, Y_Result.values)
    KfordScorer.append(resultTemp)
  results_KFolds= np.mean(KfordScorer,axis=0)

  return Test_Result,result_train_test,results_KFolds
```

```
r1,r2,r3 = AE_SoftMax_Test(DataFinal,10,'sigmoid')


myResultsDF['AE_Softmax_Full'] = r1
myResultsDF['AE_Softmax__Train_Test'] = r2
myResultsDF['AE_Softmax__K-folds'] = r3
myResultsDF
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: calling d
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

	Metric	Kmean_Full	Kmean_FTrain_Test	Kmean_FK-folds	PCA_Full	PCA_Train_Test	PCA_K-folds	AE_KMean_Full
0	completeness_score	0.220732	0.196039	0.254787	0.218227	0.214853	0.232714	0.23145
1	adjusted_rand_score	0.142946	0.112454	0.164138	0.141182	0.136143	0.140458	0.16672
2	fowlkes_mallows_score	0.300667	0.276166	0.323162	0.297863	0.293402	0.298038	0.30982
3	silhouette_score	0.234202	0.227988	0.239240	0.233839	0.227327	0.228807	0.19435
4	calinski_harabaz_score	1032.076059	401.171639	206.378573	1032.396724	401.099790	204.639195	994.60239
5	CentroidRMSE	0.129906	0.144322	0.163664	0.144000	0.125932	0.132044	0.13230

Saving Results

```
myResultsDF.to_csv("evaluation_results_UpSampling.csv")
```