```
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('weatherAUS.csv')
print('The Shape of The Data ',mData.shape)
mData.info()
```

```
The Shape of The Data  (142193, 24)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 24 columns):
Date             142193 non-null object
Location         142193 non-null object
```

Preprocessing the Data

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

```
def Preprocessing_Data(data):
  mDataDroped = data.drop(columns=['Date','RISK_MM','Sunshine','Evaporation','Cloud3pm','Cloud9am','Pressure9am','Pressure3pm
                          ,axis=1)
  mDataDroped = mDataDroped.dropna(how='any')
  mDataDroped= mDataDroped.reset_index()
  Y = mDataDroped['RainTomorrow']
  X = mDataDroped.drop(columns=['RainTomorrow'])

  col_names = X.select_dtypes("object").columns
  X = pd.get_dummies(X, columns=col_names)
  scaler = preprocessing.MinMaxScaler().fit(X)
  x_normalized = scaler.transform(X)
  newData = pd.DataFrame(columns= X.columns,data=x_normalized)

  newData['RainTomorrow'] = Y.map({'Yes': 1, 'No': 0})
  # ---------------------------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 newData , scaler
```

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

	index	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Temp
0	0.000000	0.516509	0.523629	0.001632	0.289062	0.211765	0.258824	0.71	0.22	0.508
1	0.000007	0.375000	0.565217	0.000000	0.289062	0.023529	0.235294	0.44	0.25	0.514
2	0.000014	0.504717	0.576560	0.000000	0.304688	0.200000	0.282353	0.38	0.30	0.594
3	0.000021	0.417453	0.620038	0.000000	0.132812	0.105882	0.082353	0.45	0.16	0.533
4	0.000028	0.613208	0.701323	0.002720	0.265625	0.058824	0.211765	0.82	0.33	0.527
5	0.000035	0.544811	0.652174	0.000544	0.382812	0.200000	0.258824	0.55	0.23	0.586
6	0.000042	0.537736	0.563327	0.000000	0.335938	0.211765	0.258824	0.49	0.19	0.533
7	0.000049	0.382075	0.595463	0.000000	0.218750	0.047059	0.176471	0.48	0.19	0.495
8	0.000056	0.429245	0.693762	0.000000	0.570312	0.058824	0.305882	0.42	0.09	0.537
9	0.000063	0.509434	0.659735	0.003808	0.164062	0.152941	0.105882	0.58	0.27	0.575
10	0.000070	0.516509	0.665406	0.000000	0.179688	0.176471	0.047059	0.48	0.22	0.582
11	0.000077	0.575472	0.500945	0.005985	0.187500	0.152941	0.129412	0.89	0.91	0.487
12	0.000084	0.575472	0.442344	0.042437	0.421875	0.305882	0.305882	0.76	0.93	0.518
13	0.000091	0.497642	0.487713	0.009793	0.289062	0.258824	0.211765	0.65	0.43	0.485
14	0.000105	0.533019	0.485822	0.000000	0.117188	0.105882	0.082353	0.69	0.82	0.514
15	0.000113	0.518868	0.523629	0.045702	0.437500	0.047059	0.211765	0.80	0.65	0.531
16	0.000120	0.464623	0.516068	0.028836	0.281250	0.258824	0.176471	0.47	0.32	0.478
17	0.000127	0.431604	0.574669	0.000000	0.148438	0.176471	0.047059	0.45	0.26	0.485
18	0.000134	0.471698	0.644612	0.000000	0.132812	0.082353	0.082353	0.56	0.28	0.554
19	0.000141	0.603774	0.714556	0.000000	0.281250	0.176471	0.235294	0.38	0.28	0.668
20	0.000148	0.683962	0.691871	0.000000	0.265625	0.200000	0.211765	0.54	0.24	0.654
21	0.000155	0.561321	0.674858	0.000000	0.203125	0.047059	0.129412	0.55	0.23	0.592

22	0.000162	0.497642	0.703214	0.000000	0.281250	0.023529	0.200000	0.49	0.17	0.605
23	0.000169	0.582547	0.731569	0.000000	0.218750	0.082353	0.129412	0.45	0.19	0.641
24	0.000183	0.674528	0.708885	0.000000	0.320312	0.129412	0.329412	0.56	0.15	0.670
25	0.000190	0.665094	0.604915	0.000000	0.304688	0.200000	0.329412	0.49	0.22	0.607
26	0.000197	0.495283	0.548204	0.003264	0.335938	0.105882	0.235294	0.78	0.70	0.415
27	0.000204	0.483491	0.551985	0.002176	0.250000	0.176471	0.176471	0.48	0.28	0.508
28	0.000211	0.466981	0.591682	0.000000	0.382812	0.200000	0.341176	0.46	0.26	0.567
29	0.000218	0.426887	0.542533	0.000000	0.265625	0.200000	0.105882	0.44	0.22	0.466
...	...	...	...	...	...	...	...	...	...	
121760	0.999789	0.544811	0.587902	0.000000	0.234375	0.200000	0.211765	0.61	0.36	0.476
121761	0.999803	0.419811	0.620038	0.000000	0.164062	0.047059	0.105882	0.36	0.16	0.502
121762	0.999810	0.389151	0.555766	0.000000	0.203125	0.105882	0.129412	0.46	0.25	0.443
121763	0.999817	0.500000	0.510397	0.000000	0.234375	0.200000	0.129412	0.59	0.34	0.445
121764	0.999824	0.422170	0.519849	0.000000	0.218750	0.129412	0.176471	0.62	0.32	0.400
121765	0.999831	0.327830	0.478261	0.000000	0.304688	0.211765	0.305882	0.56	0.32	0.386
121766	0.999838	0.332547	0.457467	0.000000	0.281250	0.235294	0.258824	0.61	0.22	0.356
121767	0.999845	0.235849	0.453686	0.000000	0.234375	0.176471	0.235294	0.45	0.18	0.295
121768	0.999852	0.290094	0.465028	0.000000	0.250000	0.200000	0.176471	0.42	0.22	0.333
121769	0.999859	0.306604	0.446125	0.000000	0.187500	0.129412	0.152941	0.42	0.26	0.352
121770	0.999866	0.316038	0.482042	0.000000	0.234375	0.176471	0.258824	0.38	0.11	0.369
121771	0.999873	0.228774	0.483932	0.000000	0.281250	0.129412	0.305882	0.29	0.06	0.318
121772	0.999880	0.212264	0.500945	0.000000	0.281250	0.129412	0.211765	0.27	0.19	0.352
121773	0.999887	0.294811	0.468809	0.000000	0.250000	0.129412	0.282353	0.58	0.26	0.343

Data Helping Functions

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

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

Results Variables

```
myResultsDF = pd.DataFrame()
myResultsDF['Metric'] = ['completeness_score','adjusted_rand_score','fowlkes_mallows_score','silhouette_score','calinski_hara
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.values, 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,2)


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	0.082771	0.083151	0.084843

PCA -> K_Mean Test

3	silhouette_score	0.109456	0.109493	0.115697

```
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,2,50)


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	0.082771	0.083151	0.084843	0.082771	0.083151	0.084843
1	adjusted_rand_score	0.199819	0.200140	0.201305	0.199819	0.200140	0.201305
2	fowlkes_mallows_score	0.723365	0.723724	0.723783	0.723365	0.723724	0.723783
3	silhouette_score	0.109456	0.109493	0.115697	0.109456	0.109493	0.115697
4	calinski_harabaz_score	11087.223740	4418.774036	2351.679476	11087.223740	4418.774036	2351.679476
5	CentroidRMSE	0.054197	0.054454	0.061929	0.054197	0.054454	0.061929

AutoEncoder - > K_mean Test

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


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,50,'sigmoid',2)
```

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

	Metric	Kmean_Full	Kmean_FTrain_Test	Kmean_FK-folds	PCA_Full	PCA_Train_Test	PCA_K-folds	AE_KMean_Fu
0	completeness_score	0.082771	0.083151	0.084843	0.082771	0.083151	0.084843	0.0000
1	adjusted_rand_score	0.199819	0.200140	0.201305	0.199819	0.200140	0.201305	0.0007
2	fowlkes_mallows_score	0.723365	0.723724	0.723783	0.723365	0.723724	0.723783	0.5751
3	silhouette_score	0.109456	0.109493	0.115697	0.109456	0.109493	0.115697	0.0068
4	calinski_harabaz_score	11087.223740	4418.774036	2351.679476	11087.223740	4418.774036	2351.679476	744.8719
5	CentroidRMSE	0.054197	0.054454	0.061929	0.054197	0.054454	0.061929	0.0348

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=12)
    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,50,'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 dro
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_Fu
0	completeness_score	0.082771	0.083151	0.084843	0.082771	0.083151	0.084843	0.0000
1	adjusted_rand_score	0.199819	0.200140	0.201305	0.199819	0.200140	0.201305	0.0007
2	fowlkes_mallows_score	0.723365	0.723724	0.723783	0.723365	0.723724	0.723783	0.5751
3	silhouette_score	0.109456	0.109493	0.115697	0.109456	0.109493	0.115697	0.0068
4	calinski_harabaz_score	11087.223740	4418.774036	2351.679476	11087.223740	4418.774036	2351.679476	744.8719
5	CentroidRMSE	0.054197	0.054454	0.061929	0.054197	0.054454	0.061929	0.0348

Saving Results

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