Ismat Jarin

Final Project-Report

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

Problem 1

Linear Regression Wheat Data USA 2013-2019

Using the data in "USAFAO.csv" for total wheat production USA through 2013, project the wheat production for the USA in more recent years, 2013-2019. Use linear regression as well as a second order model. You need to find data from other sources for this. Get the most recent data you can find, e.g. 2018 or 2019 to check your projection. Comment on the results.

Solution:

Read the "USAFAO.csv" data table.

```
#bring the USAFAO.csv data as a table
import pandas as pd
import numpy as np
df = pd.read_csv('USAFAO.csv')
df.head(5)
```

Take the sum of feed and Food data of 'Wheat and Production' with respect to all the years:

```
df_wheatsum
```

		Y1961	Y1962	Y1963	Y1964	Y1965	Y1966	Y1967	Y1968	Y1969	Y1970	Y1971	Y1972	Y1973	Y1974	Y1975	Y1976
Area	**Item**																
United States of America	**Wheat and products**	14636	14258	14098	15019	17481	16214	14719	18099	19043	19078	21051	19441	17940	15474	16183	17943

Take Yin and Xin:

```
Yin=df_wheatsum.values  #take the wheat production values as an array which would be our target class
Yin=Yin.reshape(53,1)
```

2nd order model:

```
#calculate the matrix when M=2
#calculate the matrix A
Xtrain=calc_matrix(Xin,2)
Xtrain
```

Prediction value of wheat productions based on linear Regression from 2014-2019:

```
array([[31834.20319302],
       [31886.85421705],
       [31927.89608713],
       [31957.32880326],
       [31975.15236544],
       [31981.36677367]])
```

The actual value for Food and Feed from 2014-2017 of wheat product is:

Year	Value
2014	28744
2015	29930
2016	31087
2017	27565

Comment: From the above results we can observe that, through the prediction results are close, but they have some differences compared to actual result. The reasons behind this difference is that, We have only considered the year as the feature vectors to compute the production. There can be many other factors, for example, temperature, amount of rain, natural disasters than can be play important roles to wheat production. If we can consider those factors as a feature, then the prediction can be more accurate.

Problem 2

1. Create a string array of cereals: Wheat, Rice, Barley, Corn (Maize), Rye, Oats, Millet, Sorghum, Other Cereals. You can use a shorthand if it helps with plotting: Wh, Ri, Ba, Co, Ry, Oa, Mi, So, Ot

2. Bring in "USAFAO.csv" as a table. This is from the Food and Agriculture Organization of the United Nations on Kaggle. It only contains the data for the United States.

3. Convert the text labels for Cereal data in rows 1-17 into a categorical array of the finite 9 possibilities as in 1. above.

4. Remove data that doesn't fit our 9 categories. Note that row 114 and 115, titled: Cereals - Excluding Beer, is the sum of the cereal data above.

5. Create a column which calculates the average yield for each cereal over the last twenty years of data (combining feed and food data). In other words, get the 1994-2013 total numbers for each cereal and divide by 20 years to get the average yield per year for each cereal for the USA.

Solution:

Bring the table:

```
#bring the USAFAO.csv data as a table
import pandas as pd
import numpy as np
df = pd.read_csv('USAFAO.csv')
df.head(5)
```

creating a string array of cereals:

```
#create a array of cereals
cereals=['Wheat and products', 'Rice (Milled Equivalent)',
       'Barley and products', 'Maize and products', 'Rye and products',
       'Oats', 'Millet and products', 'Sorghum and products',
       'Cereals, Other']
```

Remove the data that does not fit into the 9 categories:

```
#Remove all the data that does nor fit into our 9 categories
data_UFAO=df.loc[df['Item'].isin(cereals)]
data_UFAO.head()
```

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude
0	USA	231	United States of America	2511	Wheat and products	5521	Feed	1000 tonnes	37.09	-95.71
1	USA	231	United States of America	2511	Wheat and products	5142	Food	1000 tonnes	37.09	-95.71

Average value of cereals data:

```
#The average cereal data of 20 years
data_UFAO_sum=pd.merge(data4,data_comb,left_on=['Item'],right_on=['Item'])
data_UFAO_sum.head(10)
```

	Area Abbreviation	Area Code	Area	Item Code	Item	Unit	latitude	longitude	sum
0	USA	231	United States of America	2511	Wheat and products	1000 tonnes	37.09	-95.71	30739.40
1	USA	231	United States of America	2805	Rice (Milled Equivalent)	1000 tonnes	37.09	-95.71	2108.30
2	USA	231	United States of America	2513	Barley and products	1000 tonnes	37.09	-95.71	2351.75
3	USA	231	United States of America	2514	Maize and products	1000 tonnes	37.09	-95.71	140556.20
4	USA	231	United States of America	2515	Rye and products	1000 tonnes	37.09	-95.71	189.20

Problem 3

1. Bring in "FAO.csv" as a table.

2. Group and Merge Data. Create a table which calculates the sum of the data grouped by "area code" In other words, get the 1994-2013 average yield for each cereal for each country, as in Problem 2 above.

Solution:

Read FAO.csv as a table:

```
#bring FAO.csv as table
data_FAO = pd.read_csv('FAO.csv',  encoding = "ISO-8859-1")
pd.options.mode.chained_assignment = None
data_FAO.head(5)
```

Calculate the average of cereal data based on area code:

```
#Add all the iteam (combing Food and Feed for each item)
dataT2=dataT1[['Area', 'Area Code','Item','sum']]
data_Tcomb=dataT2.groupby(['Area', 'Area Code','Item'])['sum'].agg('sum').reset_index()
data_Tcomb
```

```
#average over 20 years
data_Tcomb['sum']=data_Tcomb[['sum']]/20
data_Tcomb
```

	Area	Area Code	Item	sum
0	Afghanistan	2	Barley and products	257.35
1	Afghanistan	2	Cereals, Other	0.25
2	Afghanistan	2	Maize and products	279.45
3	Afghanistan	2	Millet and products	17.45
4	Afghanistan	2	Rice (Milled Equivalent)	400.10
...	...	...	...	...
1468	Zimbabwe	181	Oats	0.90
1469	Zimbabwe	181	Rice (Milled Equivalent)	43.65

```
#The average cereal data of 20 years
data_FAO_sum=pd.merge(dataT4,data_Tcomb,left_on=['Item','Area','Area Code'],right_on=['Item','Area','Area Code'])
data_FAO_sum
```

	Area Abbreviation	Area Code	Area	Item Code	Item	Unit	latitude	longitude	sum
0	AFG	2	Afghanistan	2511	Wheat and products	1000 tonnes	33.94	67.71	3332.55
1	AFG	2	Afghanistan	2805	Rice (Milled Equivalent)	1000 tonnes	33.94	67.71	400.10
2	AFG	2	Afghanistan	2513	Barley and products	1000 tonnes	33.94	67.71	257.35
3	AFG	2	Afghanistan	2514	Maize and products	1000 tonnes	33.94	67.71	279.45
4	AFG	2	Afghanistan	2517	Millet and products	1000 tonnes	33.94	67.71	17.45
...	...	...	...	...	...	...	...	...	...
1468	ZWE	181	Zimbabwe	2515	Rye and products	1000 tonnes	-19.02	29.15	0.00
1469	ZWE	181	Zimbabwe	2516	Oats	1000 tonnes	-19.02	29.15	0.90

Problem 4

1. Explore the data from Problem 2 by plotting the parallel coordinates with the cereals on the x-axis and the average yield of each cereal by country on the y-axis. Plot just a few countries for starters.

2. Normalize the cereal yield data so that it has zero mean and unit variance over the countries. In other words, Wheat, Rice, Barley, Corn (Maize), Rye, Oats, Millet, Sorghum, Other Cereals should each be zero mean and unit variance over the data set. Name this dataset CerealNorm. We are going to see if we can start to group countries into clusters based on this normalized data.

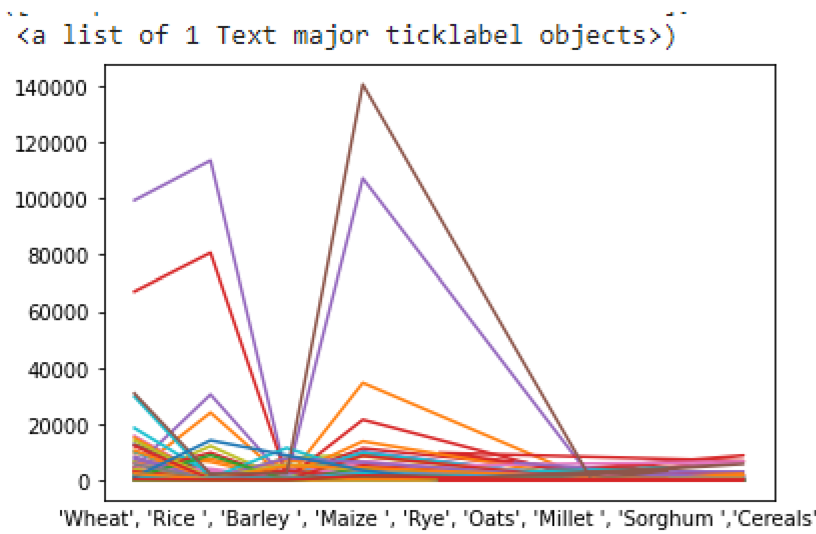
3. Scatter plot the first three dimensions using either multidimensional scaling or principal component analysis of CerealNorm.

4. Cluster the data using either kmeans or Guassian Mixture Model. Use 2 clusters. Label the 2 clusters: "high yield" and "normal yield"

5. Plot the parallel coordinates of the clusters.

Solution:

Explore the data by parallel plot:



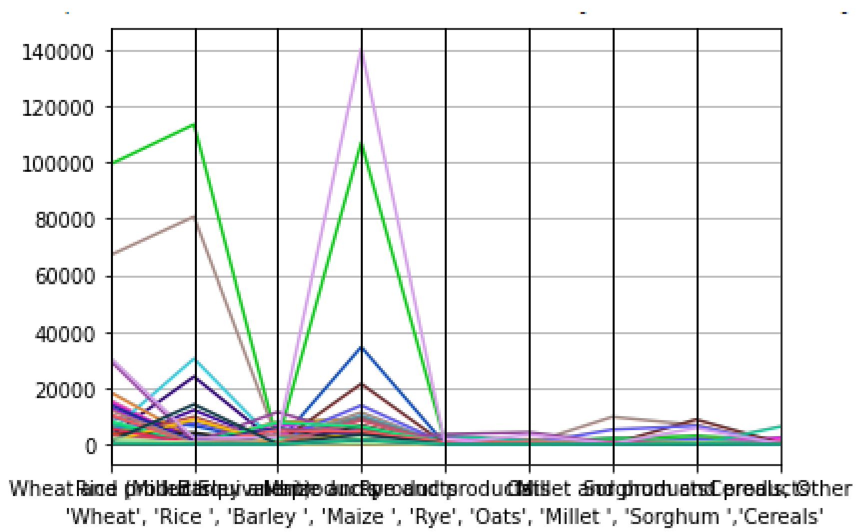
Arrange a dataset where columns represented the Cereals for every country:

```
#See the new dataset where We can observe Cereals per Country
Df_FAO
```

	country	Wheat and products	Rice (Milled Equivalent)	Barley and products	Maize and products	Rye and products	Oats	Millet and products	Sorghum and products	Cereals, Other
0	Afghanistan	3332	400	257	279	0	0	17	0	0
1	Albania	526	38	5	252	2	16	0	0	0
2	Algeria	6431	68	1065	1739	2	58	0	2	4
3	Angola	488	98	0	674	0	0	66	26	0
4	Antigua and Barbuda	4	1	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
169	Venezuela (Bolivarian Republic of)	1282	586	4	2432	1	40	0	370	17
170	Viet Nam	1026	14076	0	3443	0	0	1	0	1
171	Yemen	2225	262	32	298	0	0	70	356	2
172	Zambia	141	26	0	1434	0	0	19	19	0
173	Zimbabwe	308	43	6	1552	0	0	48	69	3

174 rows × 10 columns

Parallel plotting:



Normalized the data that zero means and unit variance exists over a country:

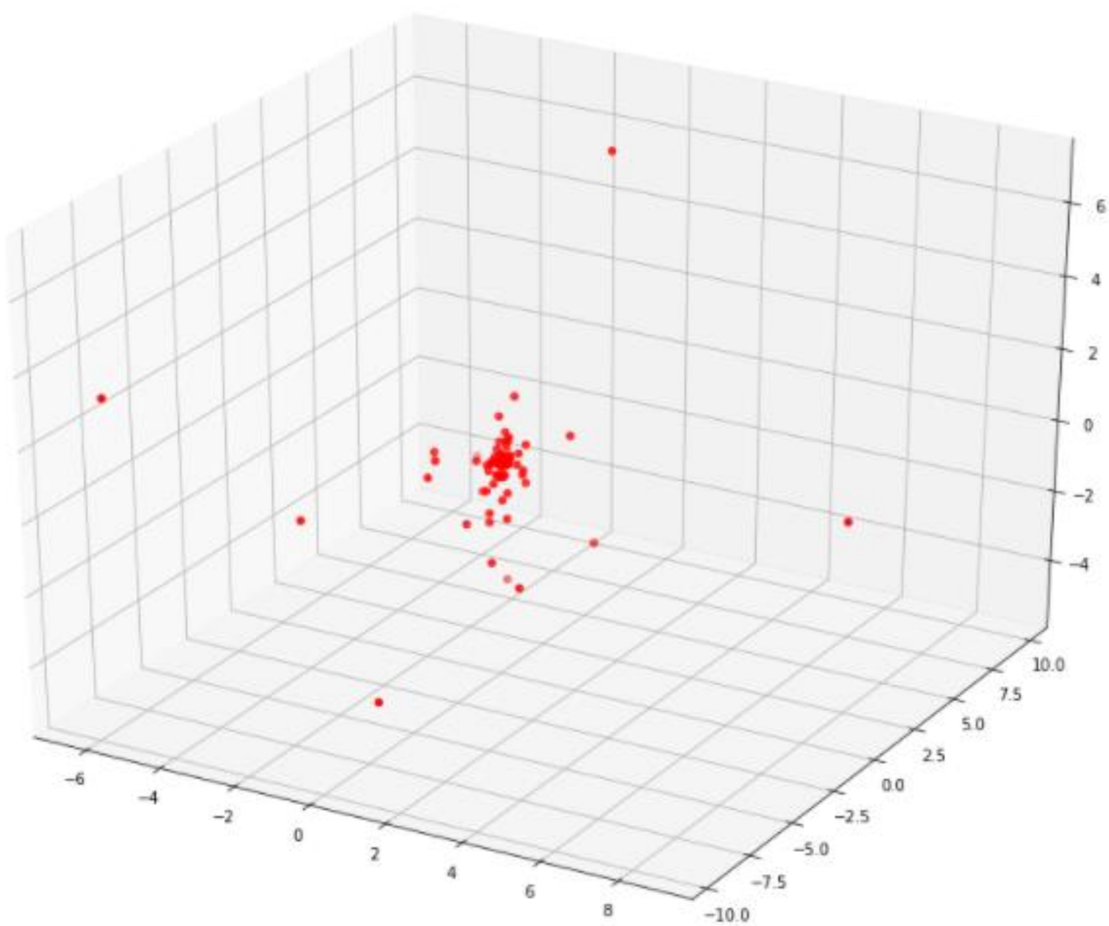
```
#normalize the data
#normalize the data mean=0, std=1 over countries
import pandas as pd
from sklearn import preprocessing
x=Df_FAO[cereals]
#normalized over the column (over wheat, Rice....etc for all country)
x_scaled = preprocessing.scale(x)
CerealNorm =  pd.DataFrame(x_scaled)
CerealNorm
```

	0	1	2	3	4	5	6	7	8
0	0.031055	-0.154504	-0.202917	-0.203190	-0.208304	-0.276496	-0.149090	-0.259207	-0.222188
1	-0.253268	-0.187586	-0.365711	-0.205165	-0.203468	-0.241067	-0.168998	-0.259207	-0.222188

Principle Component Analysis:

```
#Use Principal Component Analysis for Scatter Plot
from sklearn.decomposition import PCA
pca = PCA(svd_solver='full')
pca.fit(CerealNorm)
X = pca.transform(CerealNorm)
p=pca.explained_variance_ratio_
Src = pd.DataFrame(data=X)
#tempdata = [['pos']].join(Src)
print(Src)
```

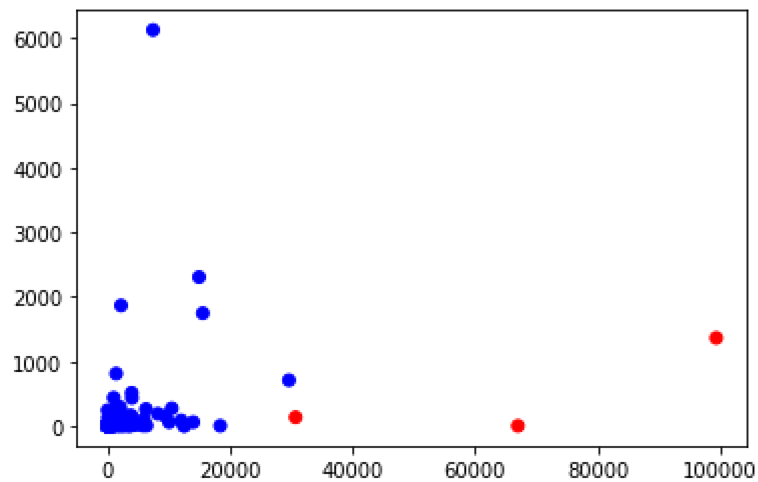
Plot 1st three components in a scatter plot:



K_mean Clustering, using k=2:

```
#k mean clustering when k=2
#Implementation of K-Means Clustering
sse = []
Xk=Df_FAO[cereals].values
model = KMeans(n_clusters = 2)
model.fit(DFC)                    #Use CerealNorm for normalized data
model.labels_
colormap = np.array(['Red', 'Blue'])
plt.scatter(Xk[:, 0], Xk[:, 8], c = colormap[model.labels_])
```

```
<matplotlib.collections.PathCollection at 0x7ff348c6e5c0>
```




```
model.labels_
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
      dtype=int32)
```

Model labels produced from the K_mean clustering. We observe that only a few data are labeled as zero. If we observe the data from those countries, we can see that they are high yield countries. So let's label: 0=high Yield Country, 1=normal Yield

```
High_Yield=Df_FAO[Df_FAO.Yield==0]
High_Yield
```

	country	Wheat and products	Rice (Milled Equivalent)	Barley and products	Maize and products	Rye and products	Oats	Millet and products	Sorghum and products	Cereals, Other	Yield
34	China, mainland	99340	113501	696	107132	779	716	1982	2925	1363	0
73	India	66993	80696	1182	11233	0	12	9638	6756	0	0
165	United States of America	30739	2108	2351	140556	189	3279	204	5663	133	0

Problem 5

1. It would seem that the data is skewed by a few "high yield" countries. This is not a fair comparison, and without using land mass it would be difficult to make it fair, so let's remove the "high yield" countries and repeat steps 2 through 4 from Problem 4 above.

2. Cluster the remaining data into two groups with these outliers removed. Label the 2 new clusters "normal yield" and "low yield"

3. Now that the data is labeled, partition it into a training and test set. Hold back 30% of the data for testing.

Solution: Remove the High Yield Countries:

```
Df_FAO_t =Df_FAO[Df_FAO.Yield!= 0]
Df_FAO_t
```

	country	Wheat and products	Rice (Milled Equivalent)	Barley and products	Maize and products	Rye and products	Oats	Millet and products	Sorghum and products	Cereals, Other	Yield
0	Afghanistan	3332	400	257	279	0	0	17	0	0	1
1	Albania	526	38	5	252	2	16	0	0	0	1
2	Algeria	6431	68	1065	1739	2	58	0	2	4	1
3	Angola	488	98	0	674	0	0	66	26	0	1
4	Antigua and Barbuda	4	1	0	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...
169	Venezuela (Bolivarian Republic of)	1282	586	4	2432	1	40	0	370	17	1
170	Viet Nam	1026	14076	0	3443	0	0	1	0	1	1
171	Yemen	2225	262	32	298	0	0	70	356	2	1
172	Zambia	141	26	0	1434	0	0	19	19	0	1
173	Zimbabwe	308	43	6	1552	0	0	48	69	3	1

171 rows × 11 columns

Normalized the new data:

```
#normalize the data
#normalize the data mean=0, std=1 over countries
import pandas as pd
from sklearn import preprocessing
x1=Df_FAO_t[cereals]
#normalized over the column (over wheat, Rice....etc for all country)
x_scaled = preprocessing.scale(x1)
CerealNorm_revise =  pd.DataFrame(x_scaled)
```

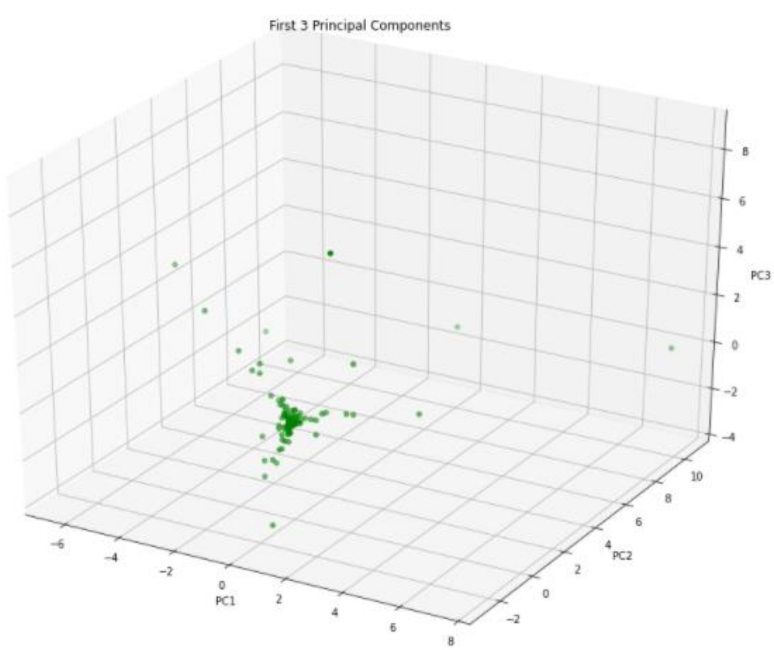
Principle Component Analysis:

```
#Use Principal Component Analysis for Scatter Plot
from sklearn.decomposition import PCA
pca = PCA(svd_solver='full')
pca.fit(CerealNorm_revise)
X1 = pca.transform(CerealNorm_revise)
p=pca.explained_variance_ratio_
Src1 = pd.DataFrame(data=X1)
#tempdata = [['pos']].join(Src)
print(Src1)
```

```
            0         1         2  ...         6         7         8
0   -0.306108 -0.312109 -0.041650  ... -0.193212 -0.341259 -0.020134
1   -0.684407 -0.407678  0.157507  ...  0.050514 -0.088702  0.004012
2    0.431189 -0.160993 -0.330198  ... -0.324685 -0.339829 -0.040717
3   -0.682977 -0.253618  0.192826  ...  0.126556 -0.034435  0.029164
4   -0.779935 -0.450039  0.215370  ...  0.036199 -0.030807  0.033645
..        ...       ...       ...  ...       ...       ...       ...
166 -0.421587  0.208363 -0.075514  ...  0.082171 -0.138862  0.000011
167 -0.431526  1.198457 -2.698681  ... -0.222335  0.130507 -0.008039
168 -0.463951 -0.014518  0.227488  ... -0.230035 -0.355475 -0.000861
169 -0.691450 -0.217903  0.084332  ...  0.226659  0.044313  0.057338
170 -0.656068 -0.131407  0.113562  ...  0.221492  0.031864  0.054994

[171 rows x 9 columns]
```

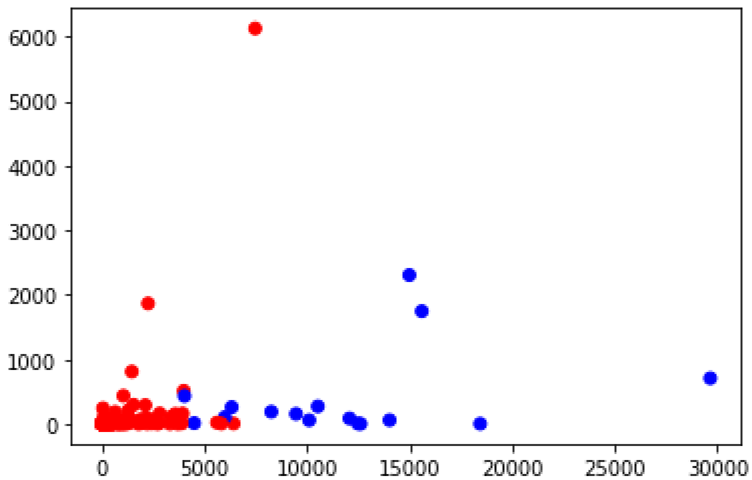
Plot:



K_mean Clustering:

```
#k mean clustering when k=2
#Implementation of K-Means Clustering for new data
sse = []
Xk1=Df_FAO_t[cereals].values
#Xk1=CerealNorm_revise
model1 = KMeans(n_clusters = 2)
model1.fit(Xk1)
model1.labels_
colormap = np.array(['Red', 'Blue'])
plt.scatter(Xk1[:, 0], Xk1[:, 8], c = colormap[model1.labels_])
```

```
<matplotlib.collections.PathCollection at 0x7ff348b04588>
```



Cluster the data into Normal and Low Yield:

If label=1, Normal Yield, If label=0, low Yield

Split data into training and Testing data: Where 30% is the testing dataset:

```
#split the data into features and target
Y=Df_FAO_t.iloc[:,[-1]]
X=Df_FAO_t.drop(Y.columns,axis = 1)
Y
```

```
#Partition the dataset into training and Testing
#While holding 30% data for testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
```

Problem 6

Perform classification on the labeled data with 3 different classifiers below. Calculate the training error (in-sample error) and testing error (estimate of out-of-sample error)

1. k-nearest neighbor. Try different values of k to find the lowest error.

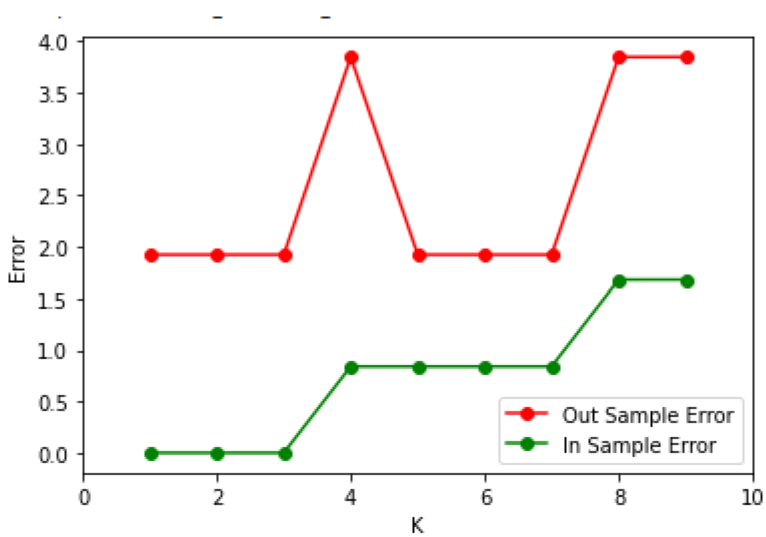
2. decision trees. Prune the tree to different levels (e.g. 3, 4, etc..) to find the lowest error. Visualize the decision tree classifier and comment on the results.

3. naive Bayes. Try different distributions to find the lowest error.

Solution:

KNN:

```
n=10 #number of K
knn_error_out=np.zeros(n-1)
knn_error_in=np.zeros(n-1)
for k in range(1,n):
  KNN_model = KNeighborsClassifier(n_neighbors=k)
  KNN_model.fit(X_train1, y_train)
  pred_out = KNN_model.predict(X_test1)
  pred_in=KNN_model.predict(X_train1)
  eout=1-accuracy_score(y_test,pred_out)
  ein=1-accuracy_score(y_train,pred_in)
  knn_error_out[k-1]=eout*100
  knn_error_in[k-1]=ein*100
```



```
print("the minimum out sample error is:",np.min(knn_error_out))
num_K=np.argmin(knn_error_out)
print("at the K of:",num_K+1)
```

```
the minimum out sample error is: 1.9230769230769273
at the K of: 1
```

Naïve Bias:

1. Multinomial Naïve Bias:

```
#train the data using multinomial Naive Bias
import numpy as np
from sklearn.naive_bayes import MultinomialNB
mn = MultinomialNB()
mn.fit(X_train1,y_train)
```

```
#classification loss (out Sample)
from sklearn.metrics import accuracy_score
y_pred_mn=mn.predict(X_test1)
errRateNB1=1-accuracy_score(y_test,y_pred_mn)
print(errRateNB1*100)
```

```
46.15384615384615
```

```
#classification loss (in Sample)
from sklearn.metrics import accuracy_score
y_pred_mn_in=mn.predict(X_train1)
errRateNB1_in=1-accuracy_score(y_train,y_pred_mn_in)
print(errRateNB1_in*100)
```

```
43.69747899159664
```

2. Gaussian Naïve Bias:

```
#Gaussian Naive bias where distribution is assumed to be normal
from sklearn.naive_bayes import GaussianNB
normal = GaussianNB()
normal.fit(X_train1,y_train)
```

```
#Classification loss (out Sample)
y_pred_normal=normal.predict(X_test1)
errRateNB2=1-accuracy_score(y_test,y_pred_normal)
print(errRateNB2*100)
```

```
7.692307692307687
```

```
#Classification loss (in Sample)
y_pred_normal_in=normal.predict(X_train1)
errRateNB2_in=1-accuracy_score(y_train,y_pred_normal_in)
print(errRateNB2_in*100)
```

```
3.361344537815125
```

3. Bernoulli Naïve Bias

```
#Barnouli Naive Bias
from sklearn.naive_bayes import BernoulliNB
bnb = BernoulliNB()
bnb.fit(X_train1,y_train)
```

```
#Classification loss (out sample)
y_pred_bnb=bnb.predict(X_test1)
errRateNB3=1-accuracy_score(y_test,y_pred_bnb)
print(errRateNB3*100)
```

```
11.538461538461542
```

```
#Classification loss (in sample)
y_pred_bnb_in=bnb.predict(X_train1)
errRateNB3_in=1-accuracy_score(y_train,y_pred_bnb_in)
print(errRateNB3_in*100)
```

```
14.28571428571429
```

4. Complement Naïve Bias

```
#Complemental Naive Bias
from sklearn.naive_bayes import ComplementNB
Cmp = ComplementNB()
Cmp.fit(X_train1,y_train)
```

```
#Classification loss (out Sample)
y_pred_Cmp=Cmp.predict(X_test1)
errRateNB4=1-accuracy_score(y_test,y_pred_Cmp)
print(errRateNB4*100)
```

```
50.0
```

```
#Classification loss (in sample)
y_pred_Cmp_in=Cmp.predict(X_train1)
errRateNB4_in=1-accuracy_score(y_train,y_pred_Cmp_in)
print(errRateNB4_in*100)
```

```
49.57983193277311
```

5. Categorical Naïve Bias:

```
#Categorical Naive Bias
from sklearn.naive_bayes import CategoricalNB
ct = CategoricalNB()
ct.fit(X_train1,y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/v
  y = column_or_1d(y, warn=True)
CategoricalNB(alpha=1.0, class_prior=None, fit_prior=T
```

```
#Classification loss (out Sample)
y_pred_ct=ct.predict(X_train1)
errRateNB5=1-accuracy_score(y_train,y_pred_ct)
print(errRateNB5*100)
```

```
3.361344537815125
```

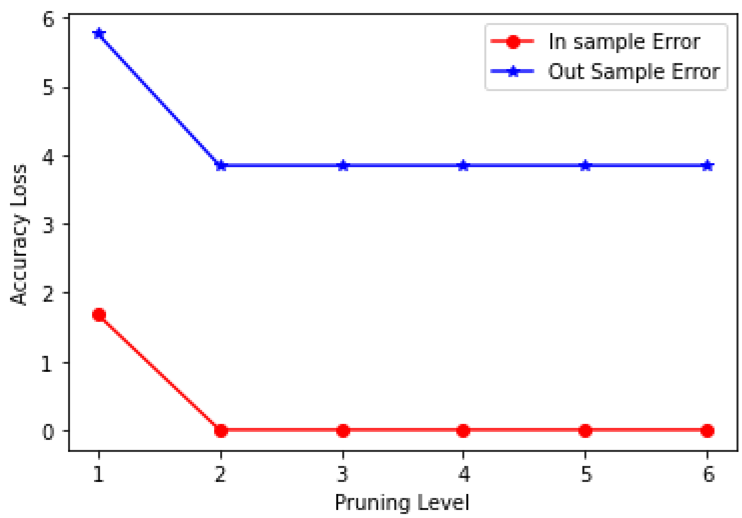
Comment: For me, Gaussian Naïve Bias (considering normal distribution) gives the lowest error value in both in-sample and out sample error. As the number of samples are very low, the other model accuracy considering different distribution is not that impressive, but I think the accuracy of the model can be improved by adding more samples.

Decision Tree:

Prune decision tree on different level:

```
#pruning over 7 level
from sklearn.metrics import accuracy_score
level=7
error_final_in=np.zeros(level-1)
error_final_out=np.zeros(level-1)
for i in range(1,level):
  prune = tree.DecisionTreeClassifier(max_depth=i,random_state=42)
  prune = prune.fit(X_train1, y_train)
  error1,error2=error_calc(X_train1,y_train,X_test1,y_test)
  error_final_out[i-1]=error1
  error_final_in[i-1]=error2
```

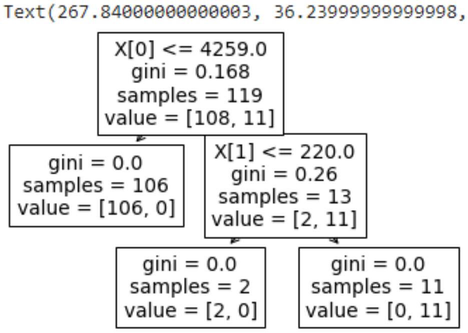
```
<matplotlib.legend.Legend at 0x7fb271a8c908>
```

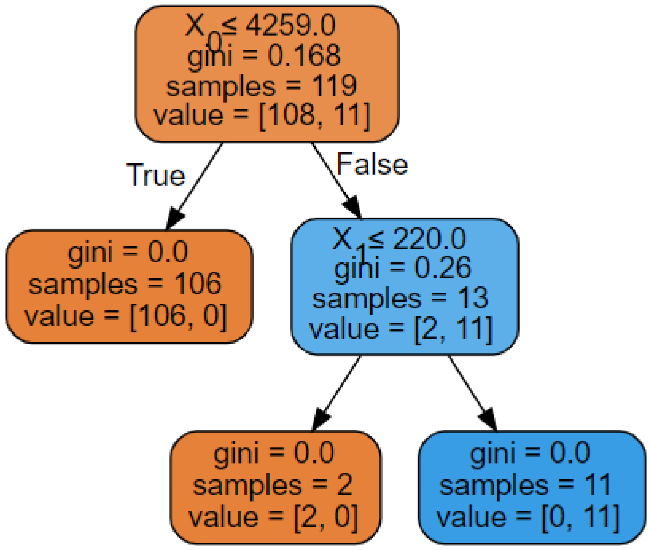


Visualize the Decision Tree classifier:

```
mdtree = tree.DecisionTreeClassifier(max_depth=2,random_state=42)
mdtree = mdtree.fit(X_train1, y_train)
tree.plot_tree(mdtree)
```

```
[Text(133.92000000000002, 181.2, 'X[0] <= 4259.0\ngini = 0.168\nsamp
 Text(66.96000000000001, 108.72, 'gini = 0.0\nsamples = 106\nvalue =
 Text(200.88000000000002, 108.72, 'X[1] <= 220.0\ngini = 0.26\nsampl
 Text(133.92000000000002, 36.23999999999998, 'gini = 0.0\nsamples =
 Text(267.84000000000003, 36.23999999999998, 'gini = 0.0\nsamples =
```





comment: The lowest error I have found is when number of level=2. This result is not always consistent. As the training and testing data choice would be depend on random state, so this result may vary. For example, for a different random state (random=10), the lowest error was for on level 1.