CLEANING

```
import pandas as pd
import seaborn as sns
sns.set(color_codes=True)
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn import linear_model
from IPython import get_ipython
from sklearn.cluster import KMeans
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
```

```
data=pd.read_csv("/content/Wholesale customers data 2.csv")
data
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Bread	Target
0	two	3	12669	9656	7561	214	2674	1338	1517
1	two	3	7057	9810	9568	1762	3293	1776	2978
2	two	three	6353	8808	7684	2405	3516	7844	2954
3	one	three	13265	1196	4221	6404	507	1788	3263
4	two	three	22615	5410	7198	3915	1777	5185	5856

```
data= data.dropna(axis=1,how='all')
data
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Bread	Tar
0	two	3	12669	9656	7561	214	2674	1338	1
1	two	3	7057	9810	9568	1762	3293	1776	2
2	two	three	6353	8808	7684	2405	3516	7844	2
3	one	three	13265	1196	4221	6404	507	1788	3
4	two	three	22615	5410	7198	3915	1777	5185	5
...	...	...	...	...	...	...	...	...	
435	1	3	29703	12051	16027	13135	182	2204	2
436	1	3	39228	1431	764	4510	93	2346	2
437	2	3	14531	15488	30243	437	14841	1867	8
438	1	3	10290	1981	2232	1038	168	2125	
439	1	3	2787	1698	2510	65	477	52	6

```
data = data.fillna(data.mean())
```

```
data
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Bread	Tar
0	two	3	12669	9656	7561	214	2674	1338	1
1	two	3	7057	9810	9568	1762	3293	1776	29
2	two	three	6353	8808	7684	2405	3516	7844	29
3	one	three	13265	1196	4221	6404	507	1788	32
4	two	three	22615	5410	7198	3915	1777	5185	58
...	...	...	...	...	...	...	...	...	
435	1	3	29703	12051	16027	13135	182	2204	26
436	1	3	39228	1431	764	4510	93	2346	23
437	2	3	14531	15488	30243	437	14841	1867	87
438	1	3	10290	1981	2232	1038	168	2125	
439	1	3	2787	1698	2510	65	477	52	68

```
data["Channel"]=data['Channel'].str.replace("one","1")
data["Channel"]=data['Channel'].str.replace("two","2")
data["Region"]=data['Channel'].str.replace("three","3")
data
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Bread	Target
0	2	2	12669	9656	7561	214	2674	1338	1517
1	2	2	7057	9810	9568	1762	3293	1776	2978
2	2	2	6353	8808	7684	2405	3516	7844	2954
3	1	1	13265	1196	4221	6404	507	1788	3263
4	2	2	22615	5410	7198	3915	1777	5185	5856
...	...	...	...	...	...	...	...	...	...
435	1	1	29703	12051	16027	13135	182	2204	2646

```
data.to_csv ('/content/Wholesale customers data2.csv', index = False, header=True) # index=true will write row names and header=true will write column na
```

```
data
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Bread	Tar

TRANSFORMATION

2	2	2	6353	8808	7684	2405	3516	7844	2

```
data.drop([0,439]) #dropping row
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Bread	Tar
1	2	2	7057	9810	9568	1762	3293	1776	2
2	2	2	6353	8808	7684	2405	3516	7844	2
3	1	1	13265	1196	4221	6404	507	1788	32
4	2	2	22615	5410	7198	3915	1777	5185	58
5	2	2	9413	8259	5126	666	1795	1451	1
...	...	...	...	...	...	...	...	...	
434	1	1	16731	3922	7994	688	2371	838	93
435	1	1	29703	12051	16027	13135	182	2204	26
436	1	1	39228	1431	764	4510	93	2346	23
437	2	2	14531	15488	30243	437	14841	1867	87
438	1	1	10290	1981	2232	1038	168	2125	

```
data.drop(["Bread"], axis = 1) #dropping column
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Target
0	2	2	12669	9656	7561	214	2674	1517
1	2	2	7057	9810	9568	1762	3293	2978
2	2	2	6353	8808	7684	2405	3516	2954
3	1	1	13265	1196	4221	6404	507	3263
4	2	2	22615	5410	7198	3915	1777	5856
...	...	...	...	...	...	...	...	...
435	1	1	29703	12051	16027	13135	182	2646
436	1	1	39228	1431	764	4510	93	2382
437	2	2	14531	15488	30243	437	14841	8783

```
data.sort_values(by = ["Region"], ascending=[True])
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Bread	Target
219	1	1	4155	367	1390	2306	86	130	8566

```
data.sort_values(by = ["Channel"], ascending=[True])
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Bread	Target
219	1	1	4155	367	1390	2306	86	130	8566
275	1	1	680	1610	223	862	96	379	4569
274	1	1	894	1703	1841	744	759	1153	9118
273	1	1	36817	3045	1493	4802	210	1824	337
272	1	1	514	8323	6869	529	93	1040	4312
...	...	...	...	...	...	...	...	...	...
335	2	2	27082	6817	10790	1365	4111	2139	5329
163	2	2	5531	15726	26870	2367	13726	446	5063
77	2	2	12205	12697	28540	869	12034	1009	8440
81	2	2	219	9540	14403	283	7818	156	1383
0	2	2	12669	9656	7561	214	2674	1338	1517

440 rows × 9 columns

```
# reading the data and looking at the first five rows of the data
# data=pd.read_csv(r"/content/Wholesale customers data.csv")
# data.head()
d1=pd.DataFrame(data)
print(d1)
print(data.describe())
#-----------------------------------------#
print("#-----------------------------------------#")
```

```
print("\nFresh :-\n")
print("#----------------------------------------------#")
print("Sum: ",data["Fresh"].sum())
print("Standard Deviation: ",data["Fresh"].std())
print("Mean: ",data["Fresh"].mean())
print("Maximum: ",data["Fresh"].max())
print("Minimum: ",data["Fresh"].min())
#----------------------------------------------#
print("#----------------------------------------------#")
print("\nMilk :-\n")
print("#----------------------------------------------#")
print("Sum: ",data["Milk"].sum())
print("Standard Deviation: ",data["Milk"].std())
print("Mean: ",data["Milk"].mean())
print("Maximum: ",data["Milk"].max())
print("Minimum: ",data["Milk"].min())
print("Standard Deviation: ",data["Milk"].std())
#----------------------------------------------#
print("#----------------------------------------------#")
print("\nGrocery :-\n")
print("#----------------------------------------------#")
print("Sum: ",data["Grocery"].sum())
print("Standard Deviation: ",data["Grocery"].std())
print("Mean: ",data["Grocery"].mean())
print("Maximum: ",data["Grocery"].max())
print("Minimum: ",data["Grocery"].min())

#----------------------------------------------#
print("#----------------------------------------------#")
print("\nFrozen :-\n")
print("#----------------------------------------------#")
print("Sum: ",data["Frozen"].sum())
print("Standard Deviation: ",data["Frozen"].std())
print("Mean: ",data["Frozen"].mean())
print("Maximum: ",data["Frozen"].max())
print("Minimum: ",data["Frozen"].min())
#----------------------------------------------#
print("#----------------------------------------------#")
```

```
print("\nDetergents_Paper :-\n")
print("#----------------------------------------------#")
print("Sum: ",data["Detergents_Paper"].sum())
print("Standard Deviation: ",data["Detergents_Paper"].std())
print("Mean: ",data["Detergents_Paper"].mean())
print("Maximum: ",data["Detergents_Paper"].max())
print("Minimum: ",data["Detergents_Paper"].min())

#----------------------------------------------#
print("#----------------------------------------------#")
print("\nBread :-\n")
print("#----------------------------------------------#")
print("Sum: ",data["Bread"].sum())
print("Standard Deviation: ",data["Bread"].std())
print("Mean: ",data["Bread"].mean())
print("Maximum: ",data["Bread"].max())
print("Minimum: ",data["Bread"].min())
#----------------------------------------------#

print("#----------------------------------------------#")
print("Region Frequency\n",data["Region"].value_counts())
#----------------------------------------------#
print("#----------------------------------------------#")
print("Channel Frequency\n",data["Channel"].value_counts())
```

```
     Channel Region  Fresh   Milk  Grocery  Frozen  Detergents_Paper  Bread  \
0          2      2  12669   9656     7561     214             2674   1338
1          2      2   7057   9810     9568    1762             3293   1776
2          2      2   6353   8808     7684    2405             3516   7844
3          1      1  13265   1196     4221    6404              507   1788
4          2      2  22615   5410     7198    3915             1777   5185
..       ...    ...    ...    ...      ...     ...              ...    ...
435        1      1  29703  12051    16027   13135              182   2204
436        1      1  39228   1431      764    4510               93   2346
437        2      2  14531  15488    30243     437            14841   1867
438        1      1  10290   1981     2232    1038              168   2125
439        1      1   2787   1698     2510      65              477     52
```

```
    Target
0     1517
1     2978
2     2954
3     3263
4     5856
..     ...
435   2646
436   2382
437   8783
438     18
439   6865

[440 rows x 9 columns]
              Fresh          Milk       Grocery        Frozen  \
count    440.000000    440.000000    440.000000    440.000000
mean   12281.947727   5796.265909   7951.277273   3071.931818
std    12521.969395   7380.377175   9503.162829   4854.673333
min        3.000000     55.000000      3.000000     25.000000
25%     3294.000000   1533.000000   2153.000000    742.250000
50%     8829.000000   3627.000000   4755.500000   1526.000000
75%    18044.000000   7190.250000  10655.750000   3554.250000
max   112151.000000  73498.000000  92780.000000  60869.000000

      Detergents_Paper         Bread       Target
count       440.000000    440.000000   440.000000
mean       2881.493182   1524.870455  4956.820455
std        4767.854448   2820.105937  2955.281414
min           3.000000      3.000000    16.000000
25%         256.750000    408.250000  2384.250000
50%         816.500000    965.500000  4915.000000
75%        3922.000000   1820.250000  7455.750000
max       40827.000000  47943.000000  9990.000000
#---------------------------------------------#

Fresh :-

#---------------------------------------------#
Sum:  5404057
Standard Deviation:  12521.969395038903
Mean:  12281.947727272727
```

```
Maximum:  112151
Minimum:  3
#-----------------------------------------------#
```

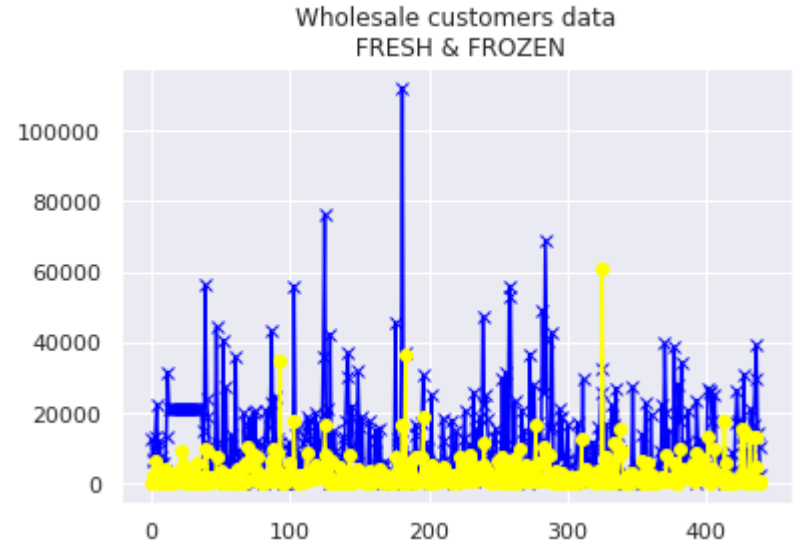
VISUALIZATION

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Channel          440 non-null    object
 1   Region           440 non-null    object
 2   Fresh            440 non-null    int64
 3   Milk             440 non-null    int64
 4   Grocery          440 non-null    int64
 5   Frozen           440 non-null    int64
 6   Detergents_Paper  440 non-null    int64
 7   Bread            440 non-null    int64
 8   Target           440 non-null    int64
dtypes: int64(7), object(2)
memory usage: 31.1+ KB
```

```
#-----------------------------------------------#

plt.plot(data['Milk'] ,color="red",marker='x')
plt.plot(data['Bread'], color="green",marker='o')

plt.title('Wholesale customers data\n MILK & BREAD')
plt.show()
#-----------------------------------------------#
```


```
#-----------------------------------------#

plt.plot(data['Fresh'] ,color="blue",marker='x')
plt.plot(data['Frozen'], color="yellow",marker='o')

plt.title('Wholesale customers data\n FRESH & FROZEN')
plt.show()
#-----------------------------------------#
```



```
#-------------------------------------------#
print("#-------------------------------------------#")

plt.hist(data['Fresh'])
plt.hist(data['Frozen'])

plt.hist(data['Grocery'])
plt.hist(data['Detergents_Paper'])

plt.hist(data['Milk'])
plt.hist(data['Bread'])

plt.title('Wholesale customers data')
plt.show()

#-------------------------------------------#
print("#-------------------------------------------#")
```

```
#-------------------------------------------#
```


```
#-------------------------------------------#
```

DECISION TREE CLASSIFIER

```
# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


# Function to split the dataset
def splitdataset(balance_data):

    # Separating the target variable
    X = data.values[:, 1:5]
    Y = data.values[:, 0]

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size = 0.3, random_state = 100)

    return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):

    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)
```

```
    return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
    classification_report(y_test, y_pred))

# Driver code
def main():
```

```
    # Building Phase
    # data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

    # Operational Phase
    print("Results Using Gini Index:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    print("Results Using Entropy:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__=="__main__":
    main()
```

```
Results Using Gini Index:
Predicted values:
['1' '2' '1' '1' '1' '1' '1' '2' '2' '1' '1' '1' '1' '1' '2' '1' '1' '1'
 '1' '1' '1' '1' '2' '2' '2' '1' '1' '1' '1' '2' '1' '1' '1' '1' '1' '1'
 '1' '1' '2' '1' '2' '1' '1' '1' '2' '2' '2' '1' '1' '1' '1' '1' '2' '2'
 '2' '1' '1' '1' '2' '1' '1' '1' '1' '1' '1' '2' '2' '2' '2' '2' '1' '1'
 '1' '2' '1' '1' '1' '2' '1' '1' '1' '1' '1' '1' '1' '1' '1' '2' '1' '1'
 '2' '1' '1' '1' '2' '1' '2' '2' '1' '1' '1' '1' '2' '1' '2' '1' '2' '1'
 '1' '1' '2' '1' '1' '1' '2' '2' '1' '1' '1' '1' '2' '1' '2' '1' '2' '1'
 '1' '1' '1' '1' '1' '1']
Confusion Matrix:  [[94  0]
 [ 0 38]]
Accuracy :  100.0
Report :              precision    recall  f1-score   support
```

```
              1       1.00      1.00      1.00        94
              2       1.00      1.00      1.00        38

       accuracy                           1.00       132
      macro avg       1.00      1.00      1.00       132
   weighted avg       1.00      1.00      1.00       132

   Results Using Entropy:
   Predicted values:
   ['1' '2' '1' '1' '1' '1' '1' '2' '2' '1' '1' '1' '1' '1' '2' '1' '1' '1'
    '1' '1' '1' '1' '2' '2' '2' '1' '1' '1' '1' '2' '1' '1' '1' '1' '1' '1'
    '1' '1' '2' '1' '2' '1' '1' '1' '2' '2' '2' '1' '1' '1' '1' '1' '2' '2'
    '2' '1' '1' '1' '2' '1' '1' '1' '1' '1' '1' '2' '2' '2' '2' '2' '1' '1'
    '1' '2' '1' '1' '1' '2' '1' '1' '1' '1' '1' '1' '1' '1' '1' '2' '1' '1'
    '2' '1' '1' '1' '2' '1' '2' '2' '1' '1' '1' '1' '2' '1' '2' '1' '2' '1'
    '1' '1' '2' '1' '1' '1' '2' '2' '1' '1' '1' '1' '2' '1' '2' '1' '2' '1'
    '1' '1' '1' '1' '1' '1']
   Confusion Matrix:  [[94  0]
    [ 0 38]]
   Accuracy :  100.0
   Report :               precision    recall  f1-score   support

              1       1.00      1.00      1.00        94
              2       1.00      1.00      1.00        38

       accuracy                           1.00       132
      macro avg       1.00      1.00      1.00       132
   weighted avg       1.00      1.00      1.00       132
```

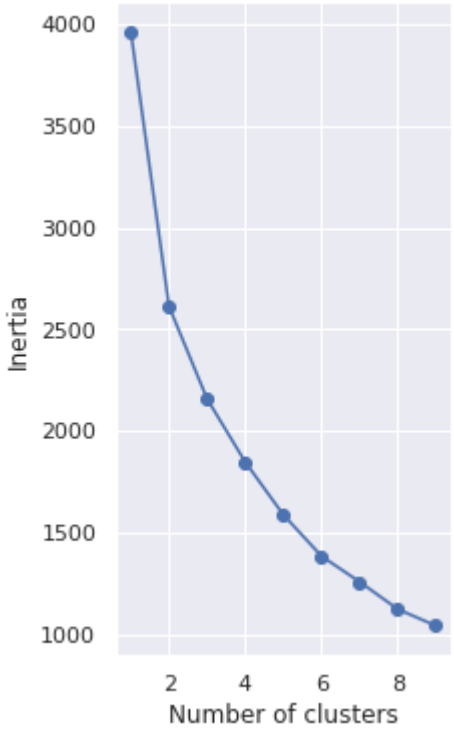
K-Means

```
get_ipython().run_line_magic('matplotlib', 'inline')
data.head()
data.describe()

#-----------------------------------------#
```

```
# print("#-----------------------------------------------#")

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
pd.DataFrame(data_scaled)
# statistics of scaled data
pd.DataFrame(data_scaled).describe()

#-----------------------------------------------#
# print("#-----------------------------------------------#")


kmeans = KMeans(n_clusters=2, init='k-means++')

# fitting the k means algorithm on scaled data
kmeans.fit(data_scaled)

#-----------------------------------------------#
# print("#-----------------------------------------------#")

kmeans.inertia_


#-----------------------------------------------#
# print("#-----------------------------------------------#")

SSE = []
for cluster in range(1,10):
    kmeans = KMeans(n_clusters = cluster, init='k-means++')
    kmeans.fit(data_scaled)
    SSE.append(kmeans.inertia_)


#-----------------------------------------------#
# print("#-----------------------------------------------#")

frame = pd.DataFrame({'Cluster':range(1,10), 'SSE':SSE})
plt.figure(figsize=(3,6))
```

```
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')

kmeans = KMeans(n_clusters = 3, init='k-means++')
kmeans.fit(data_scaled)
pred = kmeans.predict(data_scaled)

frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

```
2    296
1    131
0     13
Name: cluster, dtype: int64
```



Linear Regression

```
#Linear Regression

X = data[['Fresh', 'Milk']]
Y = data['Region']

regr = linear_model.LinearRegression()
regr.fit(X, Y)
predicted = regr.predict([[90, 12]])
print(predicted)
print(regr.coef_)

#-----------------------------------------------#
print("#-----------------------------------------------#")
```

```
[1.24225532]
[-7.89028982e-06  3.05431938e-05]
#-----------------------------------------------#
```

```
# separating x and y and standardizing..

from sklearn.preprocessing import StandardScaler

features = ['Fresh','Milk', 'Grocery']

# Separating out the features
x = data.loc[:, features].values

# Separating out the target
y = data.loc[:,['Target']].values
```

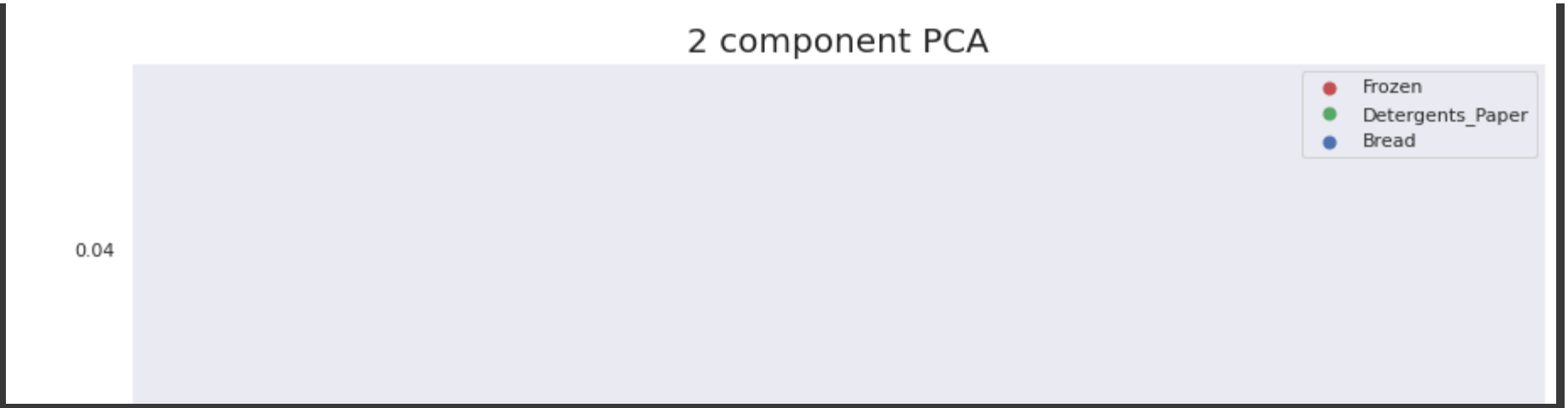
```
# Standardizing the features
x = StandardScaler().fit_transform(x)
```

```
# PCA projection to 2D

from sklearn.decomposition import PCA

pca = PCA(n_components=2)

principalComponents = pca.fit_transform(x)

principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
```

```
# Concatenating DataFrame along axis = 1. finalDf is the final DataFrame before plotting the data.
finalDf = pd.concat([principalDf, data[['Target']]], axis = 1)
```

```
# Visualize 2D projection..

import matplotlib.pyplot as plt

fig = plt.figure(figsize = (15,15))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

targets = ['Frozen', 'Detergents_Paper', 'Bread']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
```

```
ax.legend(targets)
ax.grid()
```



Mean square error and R2 score comparision on training and testing

```
X = pd.DataFrame(np.c_[data['Frozen'], data['Fresh']], columns = ['Frozen','Fresh'])
Y = data['Target']
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

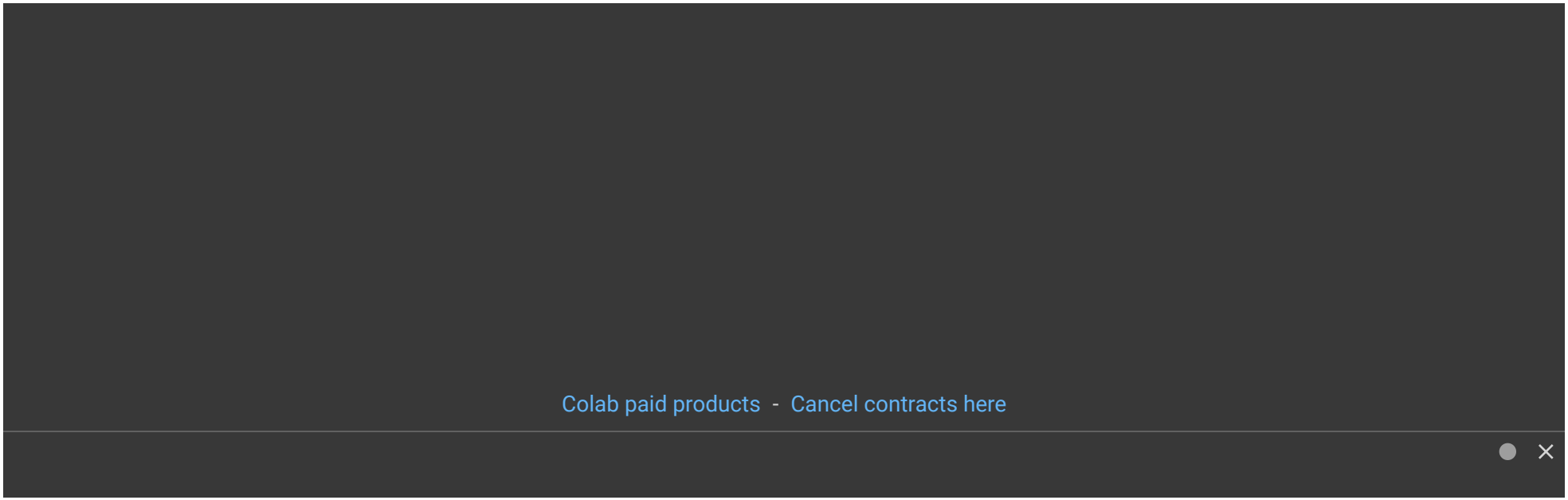
```
(352, 2)
(88, 2)
(352,)
(88,)
```

```
from sklearn.linear_model import LinearRegression

lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

```
LinearRegression()
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)
# training set
print('Training set:\n\nResultant MSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))

# testing set
y_test_predict = lin_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)

print('\n\nTesting set:\n\nResultant MSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
Training set:

Resultant MSE is 2931.499933475999
R2 score is 0.002574087527131952


Testing set:

Resultant MSE is 3032.0422508990127
R2 score is -0.03049678338781514
```


Colab paid products - Cancel contracts here