1) Importing modules & dataframe

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import softmax_regression as sr
import guassian_naive_bayes as gnb
import utils

from sklearn.preprocessing import LabelEncoder
```

In [2]:
```
df = pd.read_excel('./Dry_Bean_Dataset.xlsx')
```

2) Preprocessing

2.1) Checking out our data

In [3]:
```
df.head()
```

Out[3]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiamete
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.14109
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.27275
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.41090
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.46706
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.89650

We have to predict Class given the rest of the features. And because Class is a discrete variable, this is a classification problem.

In [4]:
```
df.describe()
```

Out[4]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	Convex
count	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.00
mean	53048.284549	855.283459	320.141867	202.270714	1.583242	0.750895	53768.20
std	29324.095717	214.289696	85.694186	44.970091	0.246678	0.092002	29774.91
min	20420.000000	524.736000	183.601165	122.512653	1.024868	0.218951	20684.00
25%	36328.000000	703.523500	253.303633	175.848170	1.432307	0.715928	36714.50
50%	44652.000000	794.941000	296.883367	192.431733	1.551124	0.764441	45178.00
75%	61332.000000	977.213000	376.495012	217.031741	1.707109	0.810466	62294.00
max	254616.000000	1985.370000	738.860153	460.198497	2.430306	0.911423	263261.00

2.2) Dealing with missing values

In [5]:
```
df.isna().sum()
```

Out[5]:
```
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    0
AspectRation       0
Eccentricity       0
ConvexArea         0
EquivDiameter      0
Extent             0
Solidity           0
roundness          0
Compactness        0
ShapeFactor1       0
ShapeFactor2       0
ShapeFactor3       0
ShapeFactor4       0
Class              0
dtype: int64
```

In [6]:
```
(df == "?").sum()
```

Out[6]:
```
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    0
AspectRation       0
Eccentricity       0
ConvexArea         0
EquivDiameter      0
Extent             0
Solidity           0
roundness          0
Compactness        0
ShapeFactor1       0
ShapeFactor2       0
ShapeFactor3       0
ShapeFactor4       0
Class              0
dtype: int64
```

We don't have any missing values. So we are good to go!

2.3) Dealing with categorical and non-numeric data

In [7]:
```
# Checking for categorical and non-numeric data
df.head()
```

Out[7]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiamete
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.14109
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.27275
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.41090
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.46706
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.89650

We have only one categorical column here - Class (the target variable). We will do label encoding for it.

In [8]:
```
df['Class'] = LabelEncoder().fit_transform(df['Class'])
```

2.4) Dropping duplicate rows

In [9]:
```
# Checking for duplicate rows
df.duplicated().sum()
```

Out[9]:
```
68
```

We have 68 duplicate rows. Let's drop them.

In [10]:
```
df.drop(axis='rows', labels=df.index[df.duplicated()], inplace=True)
```

In [11]:
```
df.duplicated().sum()
```

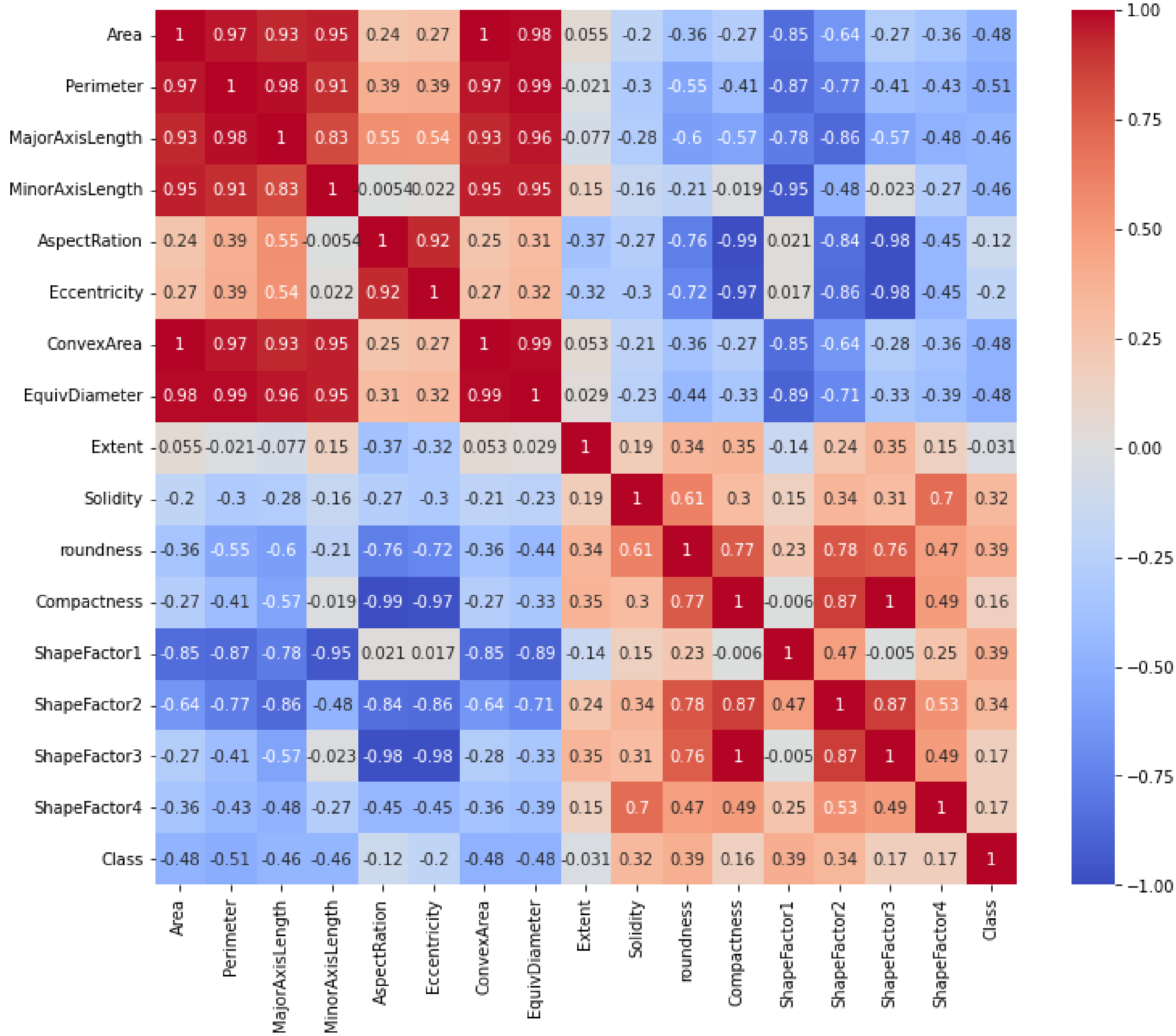
Out[11]:
```
0
```

We have no duplicate rows now!

3) EDA

3.1) Correlation

In [12]:
```
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(), vmin=-1, cmap="coolwarm", annot=True)
plt.show()
```



Area, Perimeter, ConvexArea, EquivDiameter, ShapeFactor1 --> Dependent on MajorAxisLength and MinorAxisLength.
AspectRation, ShapeFactor3, Compactness --> Dependent on Eccentricity.

Thus we can drop the dependent columns.

In [13]:
```
df.drop(['Area', 'Perimeter', 'ConvexArea', 'EquivDiameter', 'ShapeFactor1', 'AspectRati
```

In [14]:
```
df.head()
```

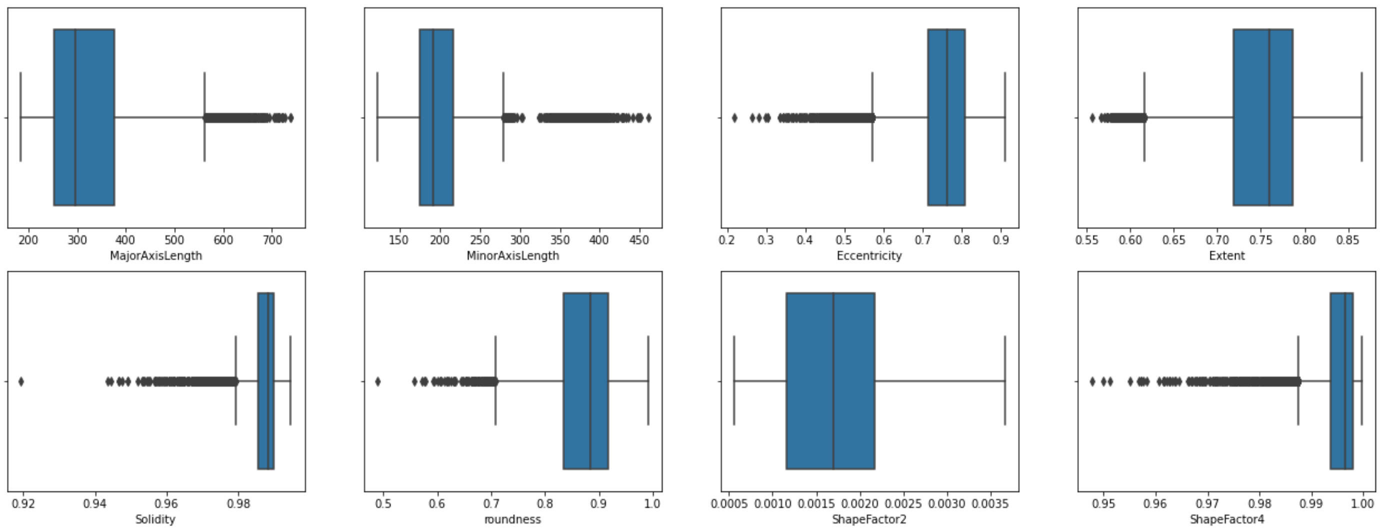
Out[14]:

	MajorAxisLength	MinorAxisLength	Eccentricity	Extent	Solidity	roundness	ShapeFactor2	ShapeFactor4
0	208.178117	173.888747	0.549812	0.763923	0.988856	0.958027	0.003147	0.998724
1	200.524796	182.734419	0.411785	0.783968	0.984986	0.887034	0.003564	0.998430
2	212.826130	175.931143	0.562727	0.778113	0.989559	0.947849	0.003048	0.999066
3	210.557999	182.516516	0.498616	0.782681	0.976696	0.903936	0.003215	0.994199
4	201.847882	190.279279	0.333680	0.773098	0.990893	0.984877	0.003665	0.999166

Columns MajorAxisLength & MinorAxisLength have a correlation of 0.83. Thus, Mutivariate Guassian model would be more suitable here rather than Naive Bayes model.

3.2) Outlier detection and removal

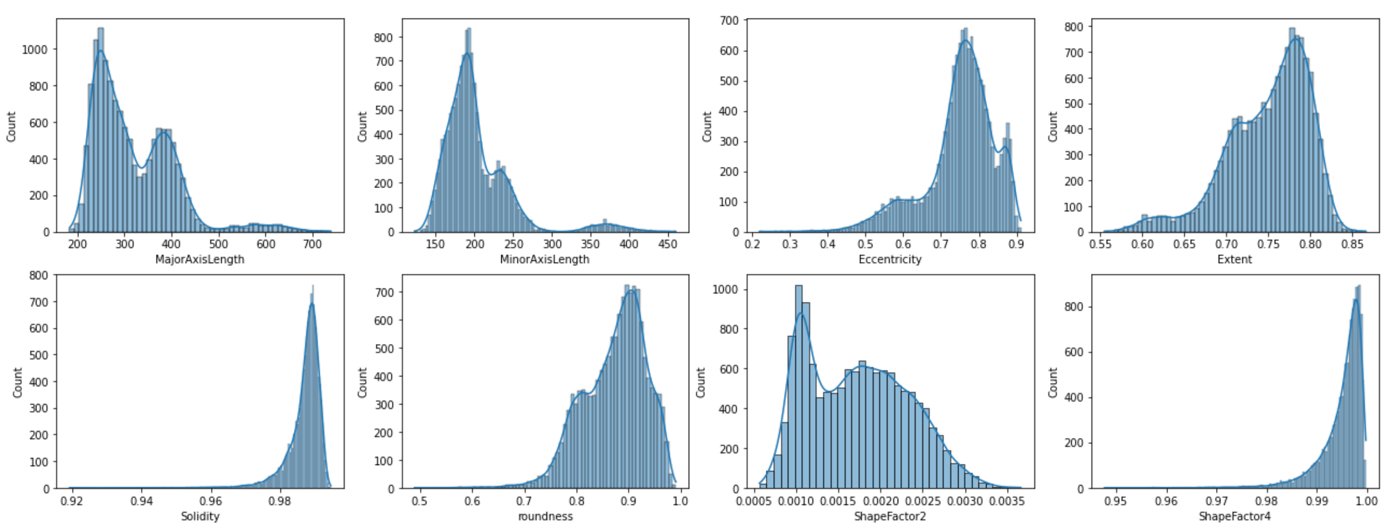
In [15]:
```
features = ['MajorAxisLength', 'MinorAxisLength', 'Eccentricity', 'Extent', 'Solidity',
fig, ax = plt.subplots(2, 4, figsize=(22, 8))
f = 0
for i in range(2):
  for j in range(4):
    sns.boxplot(ax=ax[i, j], x=df[features[f]])
    f+=1
plt.show()
```



I tried removing the outliers and what happened was that all of them belonged to a particular class. So, maybe the outliers represent some important information regarding that class. Thus, we will not remove the outliers.

3.3) Skew

In [16]:
```
features = ['MajorAxisLength', 'MinorAxisLength', 'Eccentricity', 'Extent', 'Solidity',
fig, ax = plt.subplots(2, 4, figsize=(22, 8))
f = 0
for i in range(2):
  for j in range(4):
    sns.histplot(ax=ax[i, j], data=df[features[f]], kde=True)
    ax[i, j].set_xlabel(features[f])
    f+=1
plt.show()
```



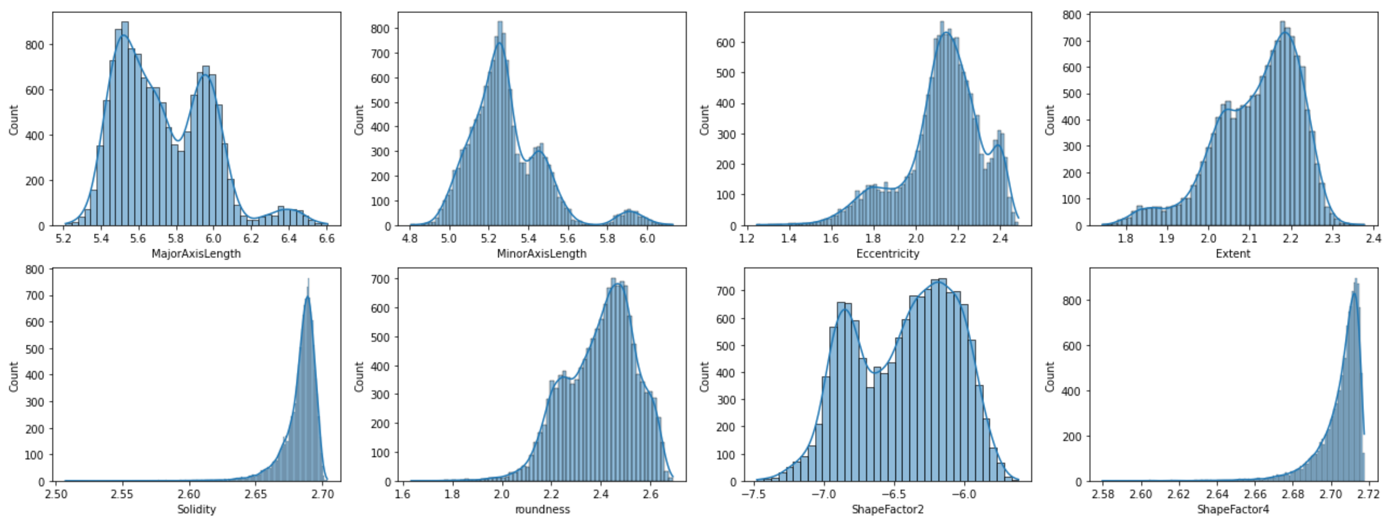
MajorAxisLength, MinorAxisLength, ShapeFactor2 are right skewed.

Eccentricity, Extent, Solidity, roundness, ShapeFactor4 are left skewed.

In [17]:
```
# Removing right skew
right_skewed = ['MajorAxisLength', 'MinorAxisLength', 'ShapeFactor2']
for i in right_skewed:
  df[i] = np.log(df[i])

# Removing left skew
left_skewed = ['Eccentricity', 'Extent', 'Solidity', 'roundness', 'ShapeFactor4']
for i in left_skewed:
  df[i] = np.exp(df[i])
```

In [18]:
```
features = ['MajorAxisLength', 'MinorAxisLength', 'Eccentricity', 'Extent', 'Solidity',
fig, ax = plt.subplots(2, 4, figsize=(22, 8))
f = 0
for i in range(2):
  for j in range(4):
    sns.histplot(ax=ax[i, j], data=df[features[f]], kde=True)
    ax[i, j].set_xlabel(features[f])
    f+=1
plt.show()
```



3.4) Normalization and standardization

In [19]:
```
features = ['MajorAxisLength', 'MinorAxisLength', 'Eccentricity', 'Extent', 'Solidity',
for i in features:
  df[i] = utils.normalize(df[i])
```

4) Softmax Regression

4.1) Feature-target split

In [20]:
```
X = df.drop(axis='columns', labels='Class').to_numpy().astype(np.float64)

# adding a column of ones to data matrix
n, m = X.shape
X =  np.c_[ np.ones(n), X]

y = df['Class'].to_numpy().astype(np.float64)
```

4.2) Train-test split

In [21]:
```
X_train, X_test, y_train, y_test = utils.train_test_split(X, y, train_size=0.75)
```

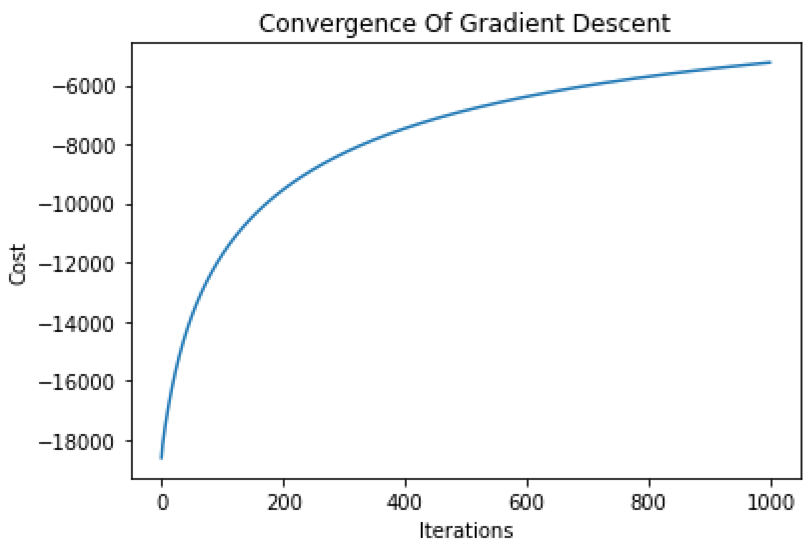
4.3) Training

In [22]:
```
iters = 1000
W, cost = sr.fit(X_train, y_train, 0.0001, iters)

yhat_train = sr.predict(X_train, W)
print('Metrics for the training data:')
print('Accuracy score', utils.accuracy_score(y_train, yhat_train))
print('f1 score', utils.f1_score(y_train, yhat_train))
```

```
Metrics for the training data:
Accuracy score 0.9001673722555873
f1 score 0.9117532923884589
```

In [23]:
```
# Convergence of gradient descent
plt.title('Convergence Of Gradient Descent')
plt.ylabel('Cost')
plt.xlabel('Iterations')
plt.plot(range(iters), cost)
plt.show()
```



4.4) Testing

In [24]:
```
yhat_test = sr.predict(X_test, W)
print('Metrics for test data:')
print('Accuracy score', utils.accuracy_score(y_test, yhat_test))
print('f1 score', utils.f1_score(y_test, yhat_test))
```

```
Metrics for test data:
Accuracy score 0.8966331955109273
f1 score 0.907970432766956
```

5) Gaussian Naive Bayes

5.1) Feature-target split

In [25]:
```
X = df.drop(axis='columns', labels='Class').to_numpy().astype(np.float64)
```

```
y = df['Class'].to_numpy().astype(np.float64)
```

5.2) Train-test split

In [26]:
```
X_train, X_test, y_train, y_test = utils.train_test_split(X, y, train_size=0.75)
```

5.3) Training

In [27]:
```
theta = gnb.fit(X_train, y_train)

yhat_train = gnb.predict(X_train, theta)
print('Metrics for the training data:')
print('Accuracy score', utils.accuracy_score(y_train, yhat_train))
print('f1 score', utils.f1_score(y_train, yhat_train))
```

```
Metrics for the training data:
Accuracy score 0.9029240917593778
f1 score 0.9157299289058302
```

5.4) Testing

In [28]:
```
yhat_test = gnb.predict(X_test, theta)
print('Metrics for test data:')
print('Accuracy score', utils.accuracy_score(y_test, yhat_test))
print('f1 score', utils.f1_score(y_test, yhat_test))
```

```
Metrics for test data:
Accuracy score 0.8998818665091554
f1 score 0.9113481685838162
```