```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler


from collections import Counter
from sklearn import preprocessing, svm
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
import sklearn.metrics as metrics
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from bioinfokit.visuz import cluster
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.stats import randint, loguniform
```

```
df=pd.read_csv("/content/data.csv")
df.head()
```

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity
0	28395	610.291	208.178117	173.888747	1.197191	0.549812
1	28734	638.018	200.524796	182.734419	1.097356	0.411785
2	29380	624.110	212.826130	175.931143	1.209713	0.562727
3	30008	645.884	210.557999	182.516516	1.153638	0.498616
4	30140	620.134	201.847882	190.279279	1.060798	0.333680

```
df.shape
```

```
(13611, 17)
```

```
df.nunique()
```

```
Area               12011
Perimeter          13351
MajorAxisLength    13543
MinorAxisLength    13543
AspectRation       13543
Eccentricity       13543
ConvexArea         12066
EquivDiameter      12011
Extent             13535
Solidity           13522
roundness          13540
Compactness        13543
ShapeFactor1       13521
ShapeFactor2       13506
ShapeFactor3       13543
ShapeFactor4       13532
Class                  7
dtype: int64
```

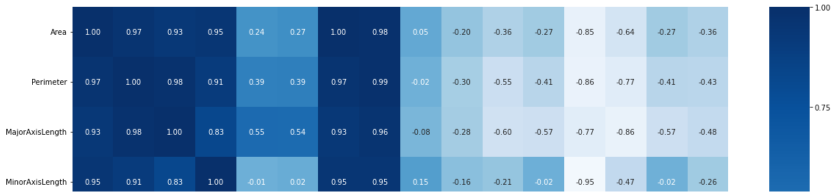
```
df["Class"].value_counts()
```

```
DERMASON    3546
SIRA        2636
SEKER       2027
HOROZ       1928
CALI        1630
BARBUNYA    1322
BOMBAY       522
Name: Class, dtype: int64
```

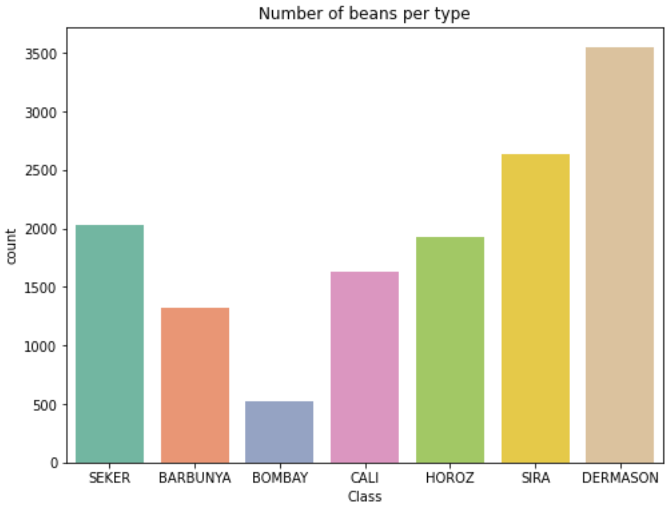
```
#check null value
df.isnull().sum()
```

```
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
```

```
#check correlation between features
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True,cmap="Blues",fmt=".2f",vmin=-1.00,vmax=1.00)
plt.show()
```



```
#display beans per type
plt.figure(figsize=(8,6))
sns.countplot(x=df["Class"],palette="Set2")
plt.title("Number of beans per type")
plt.show()
```

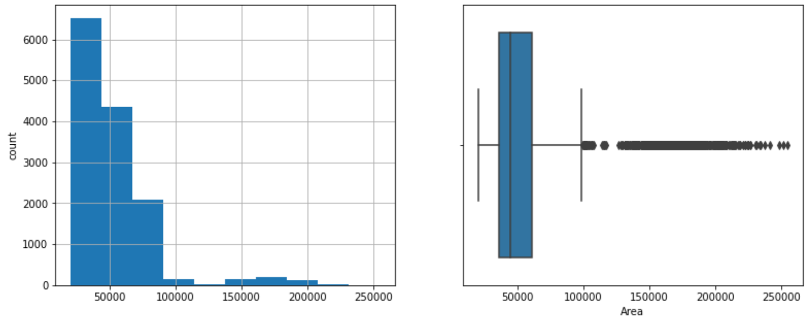



```
#display distibutions and outliet
numerical_features=df.select_dtypes(include=[np.number]).columns
print("numerical features:",numerical_features)
numerical_features=numerical_features.tolist()
```

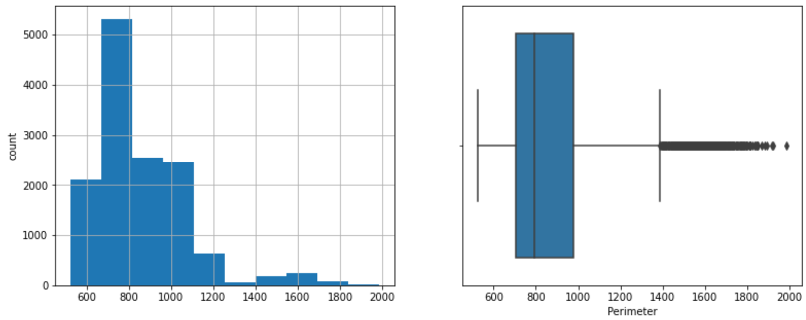
```
numerical features: Index(['Area', 'Perimeter', 'MajorAxisLength', 'MinorAxisLength',
       'AspectRation', 'Eccentricity', 'ConvexArea', 'EquivDiameter', 'Extent',
       'Solidity', 'roundness', 'Compactness', 'ShapeFactor1', 'ShapeFactor2',
       'ShapeFactor3', 'ShapeFactor4'],
      dtype='object')
```

```
for i in numerical_features:
  print(i)
  print('Skewness : ' , round(df[i].skew(),3))
  plt.figure(figsize = (13,5))
  plt.subplot(1,2,1)
  df[i].hist()
  plt.ylabel('count')
  plt.subplot(1,2,2)
  sns.boxplot(x = df[i])
  plt.show()
```

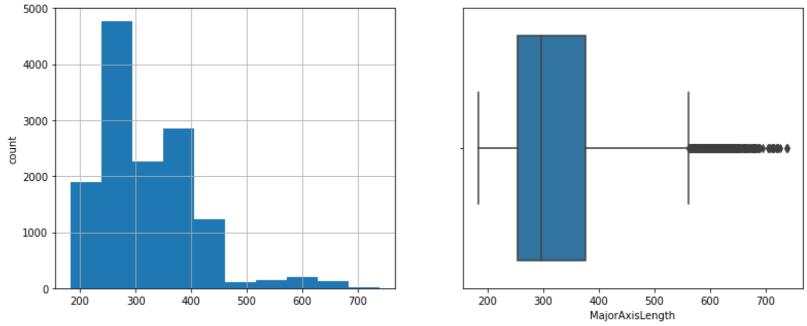
```
Area
Skewness :  2.953
```



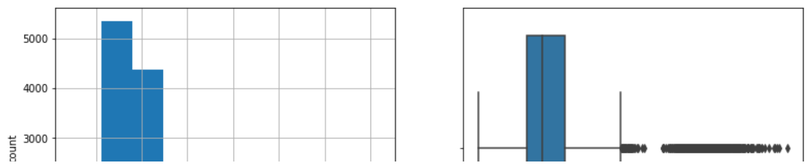

```
Perimeter
Skewness :  1.626
```




```
MajorAxisLength
Skewness :  1.358
```




```
MinorAxisLength
Skewness :  2.238
```



```
X = df.drop("Class", axis=1)
Y = df['Class']
```

```
print(X.shape)
print(Y.shape)
```

```
(13611, 16)
(13611,)
```

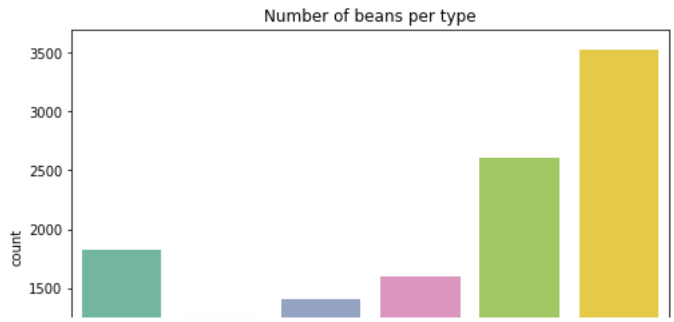
```
# Detect  outliers in the dataset
```

```
def detect_outliers(df, features):
    outlier_indices = []

    for c in features:
        Q1 = np.percentile(df[c], 25)
        Q3 = np.percentile(df[c], 75)
        IQR = Q3 - Q1
        outlier_step = IQR * 1.5
        outlier_list_col = df[(df[c] < Q1 - outlier_step) | (df[c] > Q3 + outlier_step)].index
        outlier_indices.extend(outlier_list_col)

    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(i for i, v in outlier_indices.items() if v > 1)

    return multiple_outliers
```

```
data = df.drop(detect_outliers(df,['Area', 'Perimeter', 'MajorAxisLength', 'MinorAxisLength', 'AspectRation', 'Eccentricity',
                                       'ConvexArea', 'EquivDiameter', 'Extent', 'Solidity', 'roundness', 'Compactness', 'ShapeFactor1',
                                       'ShapeFactor2', 'ShapeFactor3', 'ShapeFactor4']), axis=0).reset_index(drop=True)
print('Number of of samples in the dataset after removing outliers: %d' % len(data))
```

```
Number of of samples in the dataset after removing outliers: 12218
```

```
# Bar Chart to visualize the labels in the output variable
plt.figure(figsize=(8,6))
sns.countplot(x=data["Class"],palette="Set2")
plt.title("Number of beans per type")
plt.show()
```




```
# Convert Class String labels into Integers
from sklearn.preprocessing import LabelEncoder
lab_enc = LabelEncoder()
label_Y = lab_enc.fit_transform(Y)
```

```
# Normalize the input features of the dataset
from sklearn.preprocessing import StandardScaler
normalizer = StandardScaler()
norm_X = normalizer.fit_transform(X)
```

```
############## Feature Extraction ##################
####################################################
from sklearn.decomposition import PCA

# Visualizing the Principal Components in the feature space
pca = PCA()
pca.fit(norm_X)
loadings = pca.components_
num_pc = pca.n_features_
pc_list = ["PC" + str(i) for i in list(range(1, num_pc + 1))]
loadings_df = pd.DataFrame.from_dict(dict(zip(pc_list, loadings)))
loadings_df['variable'] = X.columns.values
loadings_df = loadings_df.set_index('variable')
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/deprecation.py:101: FutureWarning: Attribute `n_features_` was deprecated in v
  warnings.warn(msg, category=FutureWarning)
```

```
# Screeplot of Principal Components
from bioinfokit.visuz import cluster
cluster.screeplot(obj=[pc_list, pca.explained_variance_ratio_])
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
```

```
# 2D Bi-plot of Principal Components
pca_scores = PCA().fit_transform(norm_X)
cluster.biplot(cscore=pca_scores, loadings=loadings, labels=X.columns.values, var1=round(pca.explained_variance_ratio_[0]*100, 2), var2=r
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
```

```
# Cumulative Explained Variance Plot
plt.plot(np.cumsum(pca.explained_variance_ratio_)); plt.title('CUMULATIVE EXPLAINED VARIANCE OF THE PRINCIPAL COMPONENTS')
plt.xlabel('Number of Components'); plt.ylabel('Cumulative Explained Variance')
plt.show()
```

```
#machine learning
def training_model_metrics(model, X, Y):
    train_x, test_x, train_y, test_y = train_test_split(X, Y, test_size=0.2, random_state=12, shuffle=True)
    model.fit(train_x, train_y)
    y_pred = model.predict(test_x)
    model_acc = metrics.accuracy_score(test_y, y_pred)
    f1_measure = metrics.f1_score(test_y, y_pred, average='macro')
    model_precision = metrics.precision_score(test_y, y_pred, average='macro')
    model_recall = metrics.recall_score(test_y, y_pred, average='macro')
    print('Accuracy: %.3f, f1 measure: %.3f, precision: %.3f, recall: %.3f' % (model_acc, f1_measure, model_precision, model_recall))
    metrics.plot_confusion_matrix(model, test_x, test_y);plt.show()


def optimize_param(model, param, X_optim, Y_optim):
    rf_grid = RandomizedSearchCV(estimator=model, n_iter=30, param_distributions=param, scoring='f1_macro', n_jobs=-1,
                                 cv=5, verbose=2, random_state=12)
    print('Performance Metrics for ML Model of Dataset using optimized hyper-parameters')
    print('----------------------------------------------------------------------')
    training_model_metrics(rf_grid, X_optim, Y_optim)
    print('The hyper-parameters with the best f1_macro performance:')
    print('----------------------------------------------------------')
    print(rf_grid.best_params_)



def evaluate_PC(model, user_input, user_output):
    train_x, test_x, train_y, test_y = train_test_split(user_input, user_output, test_size=0.2, random_state=12, shuffle=True)
    acc, comp = list(), list()

    for n in range(1, 16):
        pca = PCA(n_components=n)
        pca.fit(train_x)
        pca_transform = pca.fit_transform(train_x)
        cv = KFold(n_splits=5, shuffle=True, random_state=12)
        scores = cross_val_score(model, pca_transform, train_y, scoring='f1_macro', cv=cv, n_jobs=-1)
        acc.append(np.mean(scores))
        comp.append(n)
        print('> No of Components=%d, Accuracy=%.3f' % (n, np.mean(scores)))

    return acc, comp



def display_perf_plot(acc, comp):
    plt.plot(comp, acc)
    plt.title('PRINCIPAL COMPONENT ANALYSIS PERFORMANCE PLOT USING CROSS-VALIDATION')
    plt.axhline(y=max(acc), color='r', linestyle='--')
    plt.xlabel('NUMBER OF COMPONENTS')
    plt.ylabel('F1-MEASURE')
    plt.show()



def KFold_evaluation(model, X, Y):
    means, mins, maxs = list(), list(), list()
    folds = range(2, 13)
    for k in folds:
        cv = KFold(n_splits=k, shuffle=True, random_state=12)
        scores = cross_val_score(model, X, Y, scoring='f1_macro', cv=cv, n_jobs=-1)
        means.append(np.mean(scores))
```