Assignment 01 - CSCI 31022 - Machine Learning and Pattern Recognition

Student ID : CS/2020/007

GitHub Link - https://github.com/AminduBhashana/ML-Exercises/tree/06693734e5609867cd5c5a3fd931c6b3c1aa8138/Assignment1%20-%20ML%20course%20module

Data Set - https://archive.ics.uci.edu/dataset/109/wine

import the necessary libraries

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

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Load the Data Set and identify the data set

In [2]:
```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data"

column_names = [
 "Class", "Alcohol", "Malic_acid", "Ash", "Alcalinity_of_ash", "Magnesium",
 "Total_phenols", "Flavanoids", "Nonflavanoid_phenols", "Proanthocyanins",
 "Color_intensity", "Hue", "0D280_0D315_of_diluted_wines", "Proline"
]

df = pd.read_csv(url, header=None, names=column_names)
```

In [3]:
```
# Number of data rows and number of feature columns that data set have
df.shape
```

Out[3]:
```
(178, 14)
```

In [4]:
```
# # print the first 10 rows of the data set
df.head(10)
```

Out[4]:

	Class	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonf
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	
5	1	14.20	1.76	2.45	15.2	112	3.27	3.39	
6	1	14.39	1.87	2.45	14.6	96	2.50	2.52	
7	1	14.06	2.15	2.61	17.6	121	2.60	2.51	
8	1	14.83	1.64	2.17	14.0	97	2.80	2.98	
9	1	13.86	1.35	2.27	16.0	98	2.98	3.15	

Information about the dataset

In [5]:
```
# using describe function to get some statistics about dataset

df.describe()
```

Out[5]:

	Class	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phe
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880

In [6]:
```
# using info function we can obtain the metadata of each feature
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Class                         178 non-null    int64  
 1   Alcohol                       178 non-null    float64
 2   Malic_acid                    178 non-null    float64
 3   Ash                           178 non-null    float64
 4   Alcalinity_of_ash             178 non-null    float64
 5   Magnesium                     178 non-null    int64  
 6   Total_phenols                 178 non-null    float64
 7   Flavanoids                    178 non-null    float64
 8   Nonflavanoid_phenols          178 non-null    float64
 9   Proanthocyanins               178 non-null    float64
 10  Color_intensity               178 non-null    float64
 11  Hue                           178 non-null    float64
 12  0D280_0D315_of_diluted_wines  178 non-null    float64
 13  Proline                       178 non-null    int64  
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

In [7]:
```
# available feature(column) names

df.columns
```

Out[7]:
```
Index(['Class', 'Alcohol', 'Malic_acid', 'Ash', 'Alcalinity_of_ash',
       'Magnesium', 'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols',
       'Proanthocyanins', 'Color_intensity', 'Hue',
       '0D280_0D315_of_diluted_wines', 'Proline'],
      dtype='object')
```

Checking for the null values

In [8]:
```
# check the null values in dataset

df.isnull().sum()
```

Out[8]:
```
Class                           0
Alcohol                         0
Malic_acid                      0
Ash                             0
Alcalinity_of_ash               0
Magnesium                       0
Total_phenols                   0
Flavanoids                      0
Nonflavanoid_phenols            0
Proanthocyanins                 0
Color_intensity                 0
Hue                             0
0D280_0D315_of_diluted_wines    0
Proline                         0
dtype: int64
```

Checking for duplicates

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

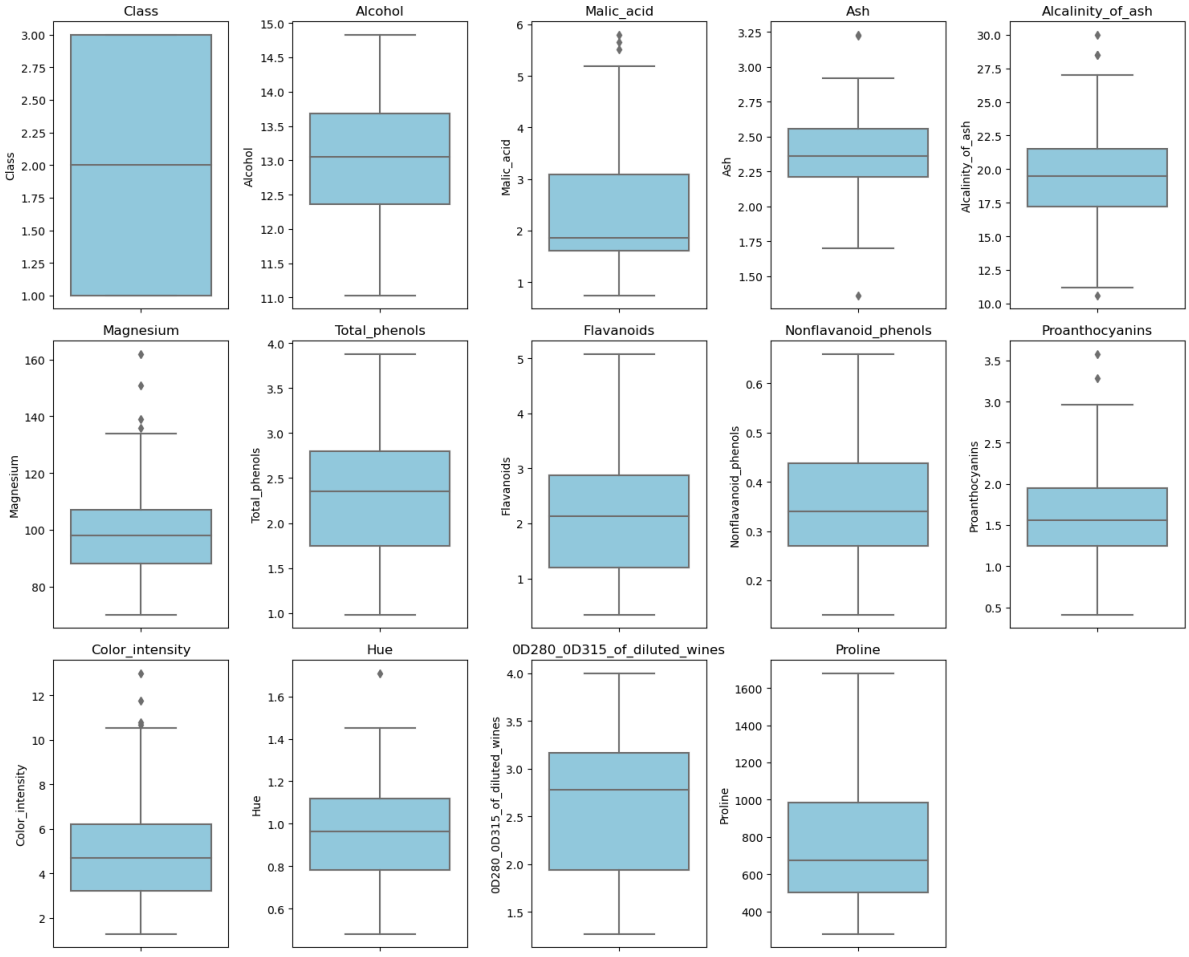
Counting value count in each class

In [10]:
```
df['Class'].value_counts()
```

Out[10]:
```
Class
2    71
1    59
3    48
Name: count, dtype: int64
```

Boxplot Visualization

In [11]:
```
plt.figure(figsize=(15,20))
for i,column in enumerate(df.columns,1):
    plt.subplot(5,5,i)
    sns.boxplot(y=df[column],color="skyblue")
    plt.title(column)
plt.tight_layout()
plt.show()
```



*As this is a small data set we don't remove outliers here.

Selecting feature variables

In [12]:
```
X = df.drop(['Class'], axis="columns")
X.head()
```

Out[12]:

	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavano
0	14.23	1.71	2.43	15.6	127	2.80	3.06	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	

Selecting target variables

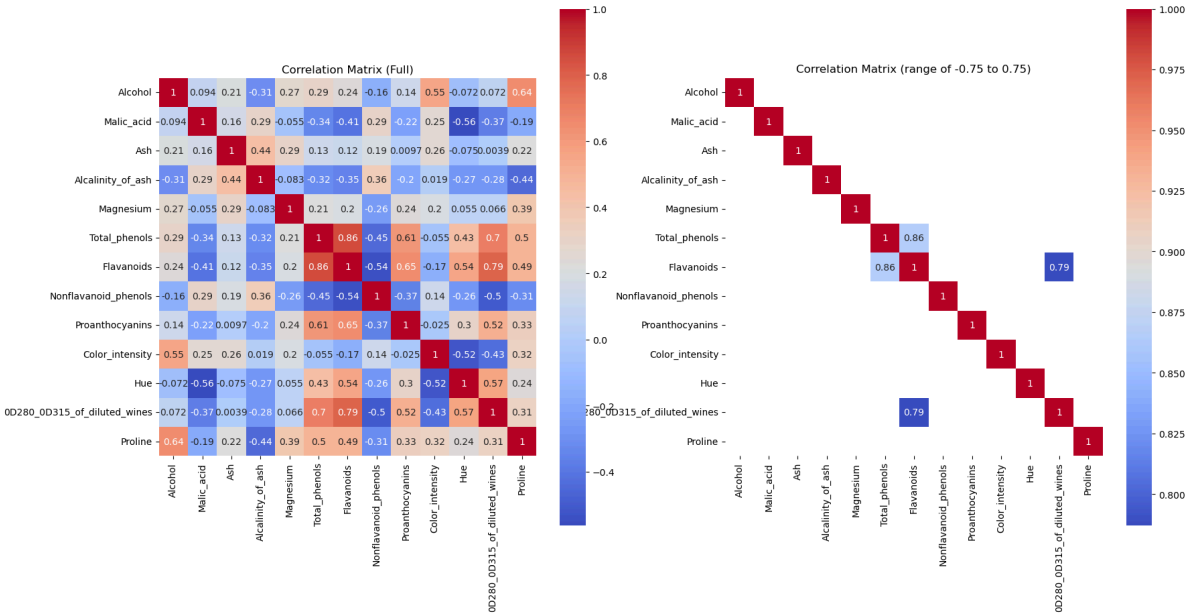
In [13]:
```
y = df['Class']
y.head()
```

Out[13]:
```
0    1
1    1
2    1
3    1
4    1
Name: Class, dtype: int64
```

Check for multicollinearity

In [14]:
```
#show the correlation matrix and the correlation matrix with the range of -0.75 to

correlation_matrix_1 = X.corr()
correlation_matrix_2 = correlation_matrix_1[(correlation_matrix_1>0.75) | (correlat

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 10))
axes = axes.flatten()

sns.heatmap(correlation_matrix_1, annot=True, cmap='coolwarm', square=True , ax=axe
axes[0].set_title('Correlation Matrix (Full)')

sns.heatmap(correlation_matrix_2, annot=True, cmap='coolwarm', square=True , ax=axe
axes[1].set_title('Correlation Matrix (range of -0.75 to 0.75)')
```

Out[14]:
```
Text(0.5, 1.0, 'Correlation Matrix (range of -0.75 to 0.75)')
```



By considering above heat map we can see some of the features are correlated with each other. So let's remove one of the feature column in higherly correlated pair.

Let's set the threshold to 0.75 remove `Total_phenols` and `0D280_0D315_of_diluted_wines` feature columns to avoid multicolinerity.

Dropping correlated variable

In [15]:
```
X = X.drop(['0D280_0D315_of_diluted_wines','Total_phenols'], axis=1)
X.head()
```

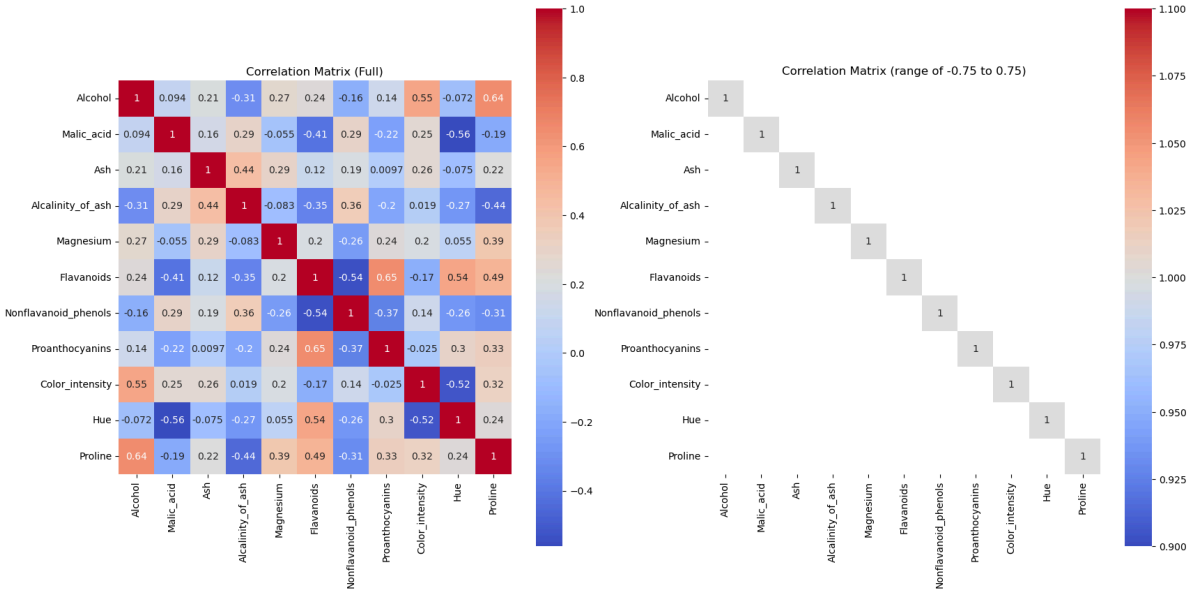
Out[15]:

	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Flavanoids	Nonflavanoid_phenols	Pro
0	14.23	1.71	2.43	15.6	127	3.06	0.28	
1	13.20	1.78	2.14	11.2	100	2.76	0.26	
2	13.16	2.36	2.67	18.6	101	3.24	0.30	
3	14.37	1.95	2.50	16.8	113	3.49	0.24	
4	13.24	2.59	2.87	21.0	118	2.69	0.39	

Check the Confusion Matrix after dropping correlated features

In [16]:
```
correlation_matrix_1 = X.corr()
correlation_matrix_2 = correlation_matrix_1[(correlation_matrix_1>0.75) | (correlat

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 10))
axes = axes.flatten()

sns.heatmap(correlation_matrix_1, annot=True, cmap='coolwarm', square=True , ax=axe
axes[0].set_title('Correlation Matrix (Full)')

sns.heatmap(correlation_matrix_2, annot=True, cmap='coolwarm', square=True , ax=axe
axes[1].set_title('Correlation Matrix (range of -0.75 to 0.75)')
```

Out[16]:
```
Text(0.5, 1.0, 'Correlation Matrix (range of -0.75 to 0.75)')
```



Standardizing data using StandardScaler

In [17]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Splitting training data and testing data

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, rar
```

In [19]:
```
print("Number of data that is used to train : ",len(X_train))
print("Number of data that is used to test : ",len(X_test))
```

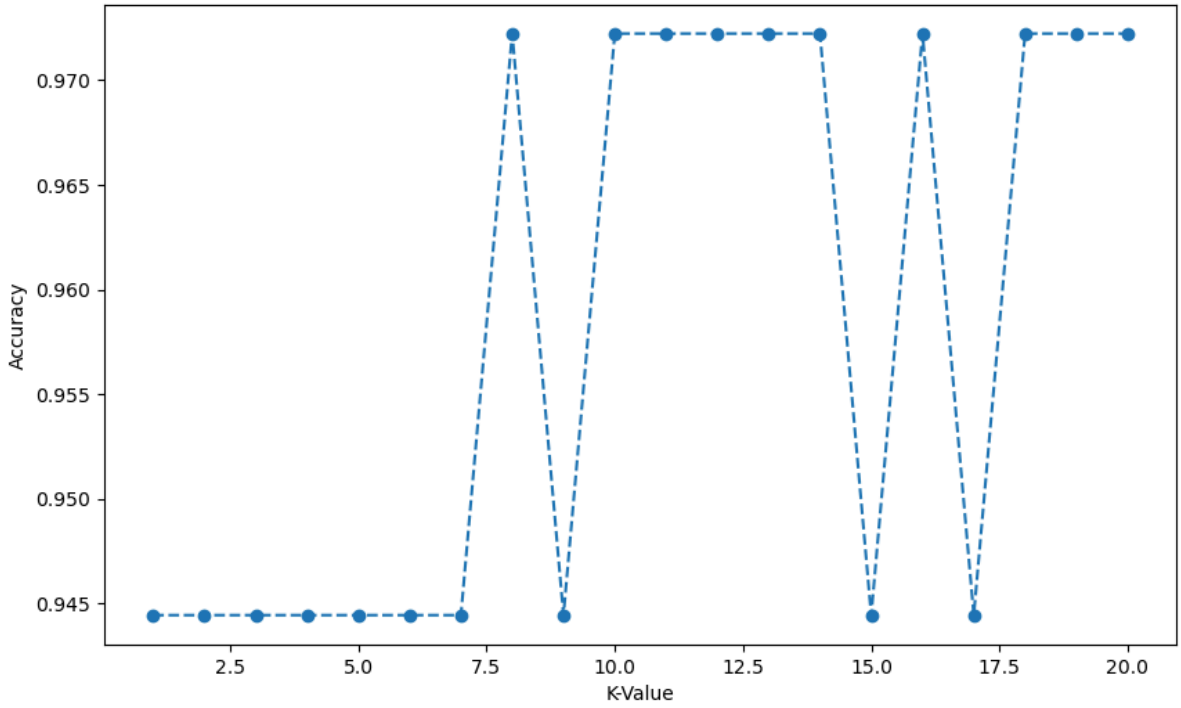
```
Number of data that is used to train :  142
Number of data that is used to test :  36
```

display and store each accuracy value for corresponding k value

In [20]:
```
# train KNN for different K values 0 - 20
k_values = range(1,21)

# store each accuracy value for corresponding k value
accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("accuracy score at k = ",k," : ",accuracy)
    accuracies.append(accuracy)
```

```
accuracy score at k =  1  :  0.9444444444444444
accuracy score at k =  2  :  0.9444444444444444
accuracy score at k =  3  :  0.9444444444444444
accuracy score at k =  4  :  0.9444444444444444
accuracy score at k =  5  :  0.9444444444444444
accuracy score at k =  6  :  0.9444444444444444
accuracy score at k =  7  :  0.9444444444444444
accuracy score at k =  8  :  0.9722222222222222
accuracy score at k =  9  :  0.9444444444444444
accuracy score at k =  10  :  0.9722222222222222
accuracy score at k =  11  :  0.9722222222222222
accuracy score at k =  12  :  0.9722222222222222
accuracy score at k =  13  :  0.9722222222222222
accuracy score at k =  14  :  0.9722222222222222
accuracy score at k =  15  :  0.9444444444444444
accuracy score at k =  16  :  0.9722222222222222
accuracy score at k =  17  :  0.9444444444444444
accuracy score at k =  18  :  0.9722222222222222
accuracy score at k =  19  :  0.9722222222222222
accuracy score at k =  20  :  0.9722222222222222
```

In [21]:
```
plt.figure(figsize=(10,6))
plt.plot(k_values, accuracies, marker='o', linestyle='--')
plt.xlabel('K-Value')
plt.ylabel('Accuracy')
plt.show()
```



Find the best K value

In [22]:
```
best_k_val = k_values[accuracies.index(max(accuracies))]
best_accuracy = max(accuracies)
print(f"Best value for k: {best_k_val} with accuracy: {best_accuracy:.2f}")
```

```
Best value for k: 8 with accuracy: 0.97
```

Initialize the K-NN model with best k value

In [23]:
```
knn_model = KNeighborsClassifier(n_neighbors=best_k_val)

knn_model.fit(X_train, y_train)
```

Out[23]:
```
KNeighborsClassifier(n_neighbors=8)
```

In [24]:
```
y_pred = knn_model.predict(X_test)
```

Classification Report

In [25]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.93      1.00      0.97        14
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00         8

    accuracy                           0.97        36
   macro avg       0.98      0.98      0.98        36
weighted avg       0.97      0.97      0.97        36
```

Confusion matrix

In [26]:
```
conf_matrix = confusion_matrix(y_test, y_pred)

display = ConfusionMatrixDisplay(conf_matrix, display_labels=knn.classes_)
display.plot(cmap='Blues', values_format='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```