Wine Recognition Dataset Classification

Classification

- Covers classification based model
- Built various classification models based on different algorithm
- All models are based on Wine recognition dataset

Imports and Settings

Here are all the settings and imports used in the project.

```
import numpy as np
import pandas as pd
# Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets # for using built-in datasets
from sklearn import metrics # for checking the model accuracy
#To plot the graph embedded in the notebook
import matplotlib.pyplot as plt
from pandas.plotting import parallel_coordinates
from sklearn.model_selection import train_test_split
# importing the necessary package to use the classification algorithm
from sklearn import svm #for Support Vector Machine (SVM) Algorithm
# importing the necessary package to use the classification algorithm
from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm
# importing the necessary package to use the classification algorithm
from sklearn.neighbors import KNeighborsClassifier # for K nearest neighbours
# importing the necessary package to use the classification algorithm
from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
# importing the necessary package to use the classification algorithm
from sklearn.naive_bayes import GaussianNB
```

Load Dataset: Wine recognition

- We will load the wine recognition dataset from the scikit-learn library.
- The objective is to predict the class of wine using the given features.

Input:

```
from sklearn.datasets import load_wine
wineData = load_wine()
```

Explore the Dataset

- sklearn returns Dictionary-like object.

Input:

```
print(wineData.keys())
```

Output:

dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])

Input:

```
# Let's print the description of wine dataset
print(wineData.DESCR)
```

Output:

.. _wine_dataset:

Wine recognition dataset

Data Set Characteristics:

:Number of Instances: 178 (50 in each of three classes)

:Number of Attributes: 13 numeric, predictive attributes and the class

:Attribute Information:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines

- Proline

- class:
    - class_0
    - class_1
    - class_2

:Summary Statistics:

	Min	Max	Mean	SD
Alcohol:	11.0	14.8	13.0	0.8
Malic Acid:	0.74	5.80	2.34	1.12
Ash:	1.36	3.23	2.36	0.27
Alcalinity of Ash:	10.6	30.0	19.5	3.3
Magnesium:	70.0	162.0	99.7	14.3
Total Phenols:	0.98	3.88	2.29	0.63
Flavanoids:	0.34	5.08	2.03	1.00
Nonflavanoid Phenols:	0.13	0.66	0.36	0.12
Proanthocyanins:	0.41	3.58	1.59	0.57
Colour Intensity:	1.3	13.0	5.1	2.3
Hue:	0.48	1.71	0.96	0.23
OD280/OD315 of diluted wines:	1.27	4.00	2.61	0.71
Proline:	278	1680	746	315

:Missing Attribute Values: None

:Class Distribution: class_0 (59), class_1 (71), class_2 (48)

:Creator: R.A. Fisher

:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

:Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

The data is the results of a chemical analysis of wines grown in the same region in Italy by three different cultivators. There are thirteen different measurements taken for different constituents found in the three types of wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

(1) S. Aeberhard, D. Coomans and O. de Vel,
Comparison of Classifiers in High Dimensional Settings,
Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Technometrics).

The data was used with many others for comparing various
classifiers. The classes are separable, though only RDA
has achieved 100% correct classification.
(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
(All results using the leave-one-out technique)

(2) S. Aeberhard, D. Coomans and O. de Vel,
"THE CLASSIFICATION PERFORMANCE OF RDA"
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Journal of Chemometrics).

- As we can see from the dataset description that there is no missing values.
- And, it contains equal number of samples for each of the class of wine.

Input:

```
#Let's print the data (features matrix) of wine dataset
print(wineData.data)
```

Output:

[[1.423e+01 1.710e+00 2.430e+00 ... 1.040e+00 3.920e+00 1.065e+03]

 [1.320e+01 1.780e+00 2.140e+00 ... 1.050e+00 3.400e+00 1.050e+03]

 [1.316e+01 2.360e+00 2.670e+00 ... 1.030e+00 3.170e+00 1.185e+03]

 ...

 [1.327e+01 4.280e+00 2.260e+00 ... 5.900e-01 1.560e+00 8.350e+02]

 [1.317e+01 2.590e+00 2.370e+00 ... 6.000e-01 1.620e+00 8.400e+02]

 [1.413e+01 4.100e+00 2.740e+00 ... 6.100e-01 1.600e+00 5.600e+02]]

Input:

```
# Let's check the shape of features matrix
print(wineData.data.shape)
```

Output:

(178, 13)

Input:

```python

# Let's print the feature names
print(wineData.feature_names)

```

Output:

['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']

Input:

```python

# Let's print the target vector
print(wineData.target)

```

Output:

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

Input:

```python

# Let's check the shape of target
print(wineData.target.shape)

```

Output:

(178,)

Input:

```python

#Let's print the target class/species names
print(wineData.target_names)

```

Output:

['class_0' 'class_1' 'class_2']

Input:

```
# Let's load the data (features matrix) into pandas DataFrame
wine_df = pd.DataFrame(wineData.data, columns=wineData.feature_names)
print(wine_df)
```

Output:

```
     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  ...  proanthocyanins  color_intensity
hue  od280/od315_of_diluted_wines  proline
0      14.23        1.71  2.43               15.6      127.0  ...             2.29             5.64  1.04
3.92   1065.0
1      13.20        1.78  2.14               11.2      100.0  ...             1.28             4.38  1.05
3.40   1050.0
2      13.16        2.36  2.67               18.6      101.0  ...             2.81             5.68  1.03
3.17   1185.0
3      14.37        1.95  2.50               16.8      113.0  ...             2.18             7.80  0.86
3.45   1480.0
4      13.24        2.59  2.87               21.0      118.0  ...             1.82             4.32  1.04
2.93    735.0
..       ...         ...   ...                ...        ...  ...              ...              ...   ...
...      ...
173    13.71        5.65  2.45               20.5       95.0  ...             1.06             7.70  0.64
1.74    740.0
174    13.40        3.91  2.48               23.0      102.0  ...             1.41             7.30  0.70
1.56    750.0
175    13.27        4.28  2.26               20.0      120.0  ...             1.35            10.20  0.59
1.56    835.0
176    13.17        2.59  2.37               20.0      120.0  ...             1.46             9.30  0.60
1.62    840.0
```

177   14.13   4.10 2.74   24.5   96.0 ...   1.35   9.20 0.61

1.60   560.0

[178 rows x 13 columns]

- We can see that the target label recognition is missing in the DataFrame (that is alright).
- Now, create a new column for target label and add it to the dataframe.

Input:

```

# Let's add target label into pandas DataFrame
wine_df['recog'] = wineData.target
print(wine_df)

```

Output:

alcohol malic_acid   ash alcalinity_of_ash magnesium total_phenols ... proanthocyanins

color_intensity   hue od280/od315_of_diluted_wines proline recog

0   14.23   1.71 2.43   15.6   127.0   2.80 ...   2.29   5.64 1.04

3.92   1065.0

0

1   13.20   1.78 2.14   11.2   100.0   2.65 ...   1.28   4.38 1.05

3.40   1050.0

0

2   13.16   2.36 2.67   18.6   101.0   2.80 ...   2.81   5.68 1.03

3.17   1185.0

0

3   14.37   1.95 2.50   16.8   113.0   3.85 ...   2.18   7.80 0.86

3.45   1480.0

0

4   13.24   2.59 2.87   21.0   118.0   2.80 ...   1.82   4.32 1.04

2.93   735.0

0

.. ..
... ...

173 13.71 5.65 2.45 20.5 95.0 1.68 ... 1.06 7.70 0.64
1.74 740.0

2

174 13.40 3.91 2.48 23.0 102.0 1.80 ... 1.41 7.30 0.70
1.56 750.0

2

175 13.27 4.28 2.26 20.0 120.0 1.59 ... 1.35 10.20 0.59
1.56 835.0

2

176 13.17 2.59 2.37 20.0 120.0 1.65 ... 1.46 9.30 0.60
1.62 840.0

2

177 14.13 4.10 2.74 24.5 96.0 2.05 ... 1.35 9.20 0.61
1.60 560.0

2

[178 rows x 14 columns]

Target species names:

- 0 = 'class_0'
- 1 = 'class_1'
- 2 = 'class_2'

Input:

```python
# replace the target values with class names
wine_df['recog'] = wine_df['recog'].replace([0, 1, 2], ['class_0', 'class_1', 'class_2'])
print(wine_df)
```

Output:

```
    alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  flavanoids
nonflavanoid_phenols  proanthocyanins  color_intensity   hue  od280/od315_of_diluted_wines
proline    recog

0      14.23        1.71  2.43               15.6      127.0           2.80        3.06                  0.28
2.29             5.64  1.04                    3.92   1065.0  class_0

1      13.20        1.78  2.14               11.2      100.0           2.65        2.76                  0.26
1.28             4.38  1.05                    3.40   1050.0  class_0

2      13.16        2.36  2.67               18.6      101.0           2.80        3.24                  0.30
2.81             5.68  1.03                    3.17   1185.0  class_0

3      14.37        1.95  2.50               16.8      113.0           3.85        3.49                  0.24
2.18             7.80  0.86                    3.45   1480.0  class_0

4      13.24        2.59  2.87               21.0      118.0           2.80        2.69                  0.39
1.82             4.32  1.04                    2.93    735.0  class_0

..      ...         ...   ...                ...        ...           ...         ...                   ...              ...              ...   ...
...     ...      ...

173    13.71        5.65  2.45               20.5       95.0           1.68        0.61                  0.52
1.06             7.70  0.64                    1.74    740.0  class_2

174    13.40        3.91  2.48               23.0      102.0           1.80        0.75                  0.43
1.41             7.30  0.70                    1.56    750.0  class_2

175    13.27        4.28  2.26               20.0      120.0           1.59        0.69                  0.43
1.35            10.20  0.59                    1.56    835.0  class_2

176    13.17        2.59  2.37               20.0      120.0           1.65        0.68                  0.53
1.46             9.30  0.60                    1.62    840.0  class_2

177    14.13        4.10  2.74               24.5       96.0           2.05        0.76                  0.56
1.35             9.20  0.61                    1.60    560.0  class_2
```

[178 rows x 14 columns]

Exploratory Data Analysis

Input:

```
# Return numerical summary of each attribute of wine
print(wine_df.describe())
```

Output:

alcohol malic_acid ash alcalinity_of_ash magnesium total_phenols ... nonflavanoid_phenols proanthocyanins color_intensity hue od280/od315_of_diluted_wines proline

count 178.000000 178.000000 178.000000 178.000000 178.000000 178.000000 ... 178.000000 178.000000 178.000000 178.000000 178.000000 178.000000

mean 13.000618 2.336348 2.366517 19.494944 99.741573 2.295112 ... 0.361854 1.590899 5.058090 0.957449 2.611685 746.893258

std 0.811827 1.117146 0.274344 3.339564 14.282484 0.625851 ... 0.124453 0.572359 2.318286 0.228572 0.709990 314.907474

min 11.030000 0.740000 1.360000 10.600000 70.000000 0.980000 ... 0.130000 0.410000 1.280000 0.480000 1.270000 278.000000

25% 12.362500 1.602500 2.210000 17.200000 88.000000 1.742500 ... 0.270000 1.250000 3.220000 0.782500 1.937500 500.500000

50% 13.050000 1.865000 2.360000 19.500000 98.000000 2.355000 ... 0.340000 1.555000 4.690000 0.965000 2.780000 673.500000

75% 13.677500 3.082500 2.557500 21.500000 107.000000 2.800000 ... 0.437500 1.950000 6.200000 1.120000 3.170000 985.000000

max 14.830000 5.800000 3.230000 30.000000 162.000000 3.880000 ... 0.660000 3.580000 13.000000 1.710000 4.000000 1680.000000

[8 rows x 13 columns]

Input:

```python

# let's check number of samples for each class of wine
print(wine_df.groupby('recog').size())

```

Output:

recog

class_0   59

class_1   71

class_2   48

dtype: int64

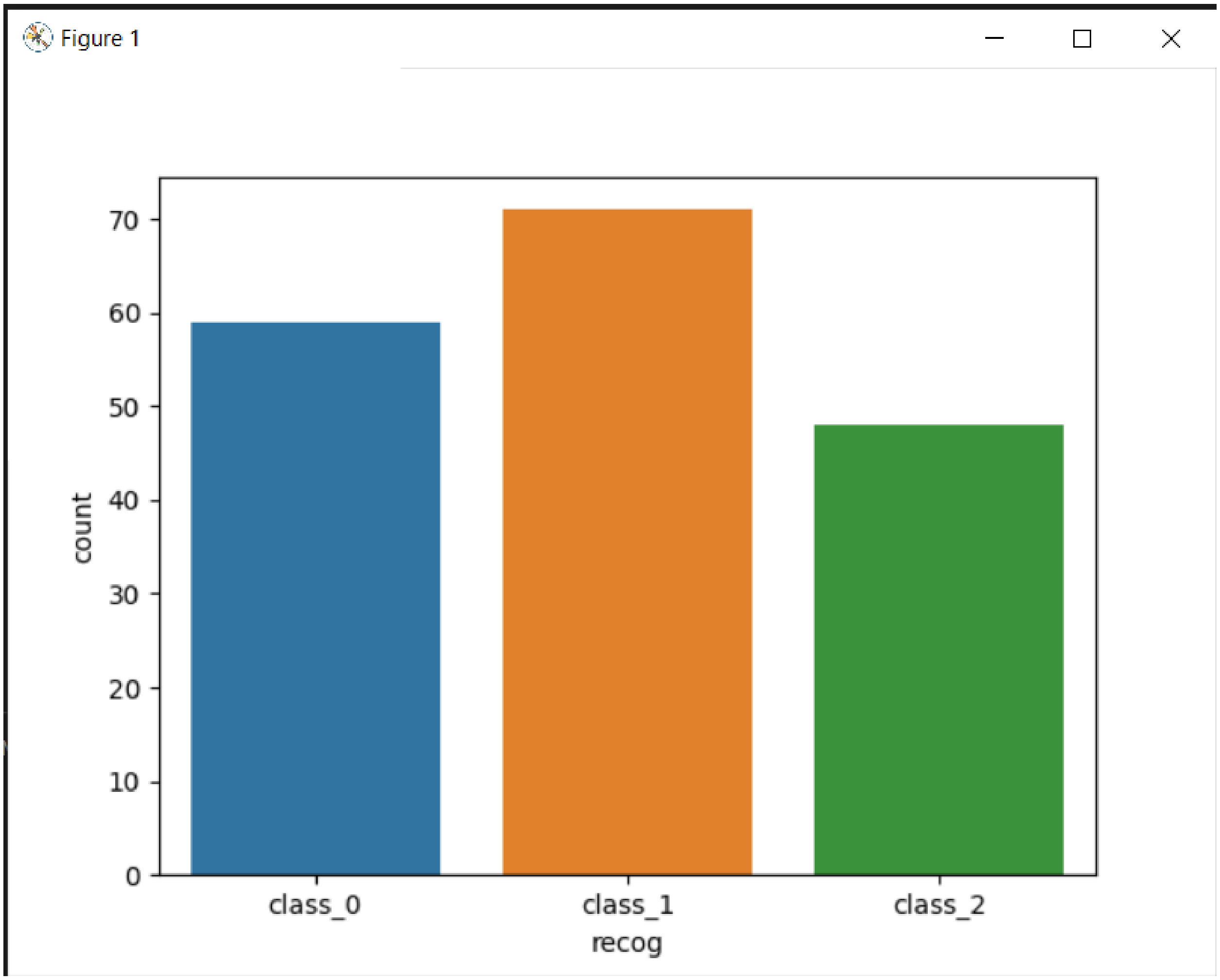
Input:

```python

# let's visualise the number of samples for each class with count plot
sns.countplot(x='recog', data=wine_df)
plt.show()

```

Output:

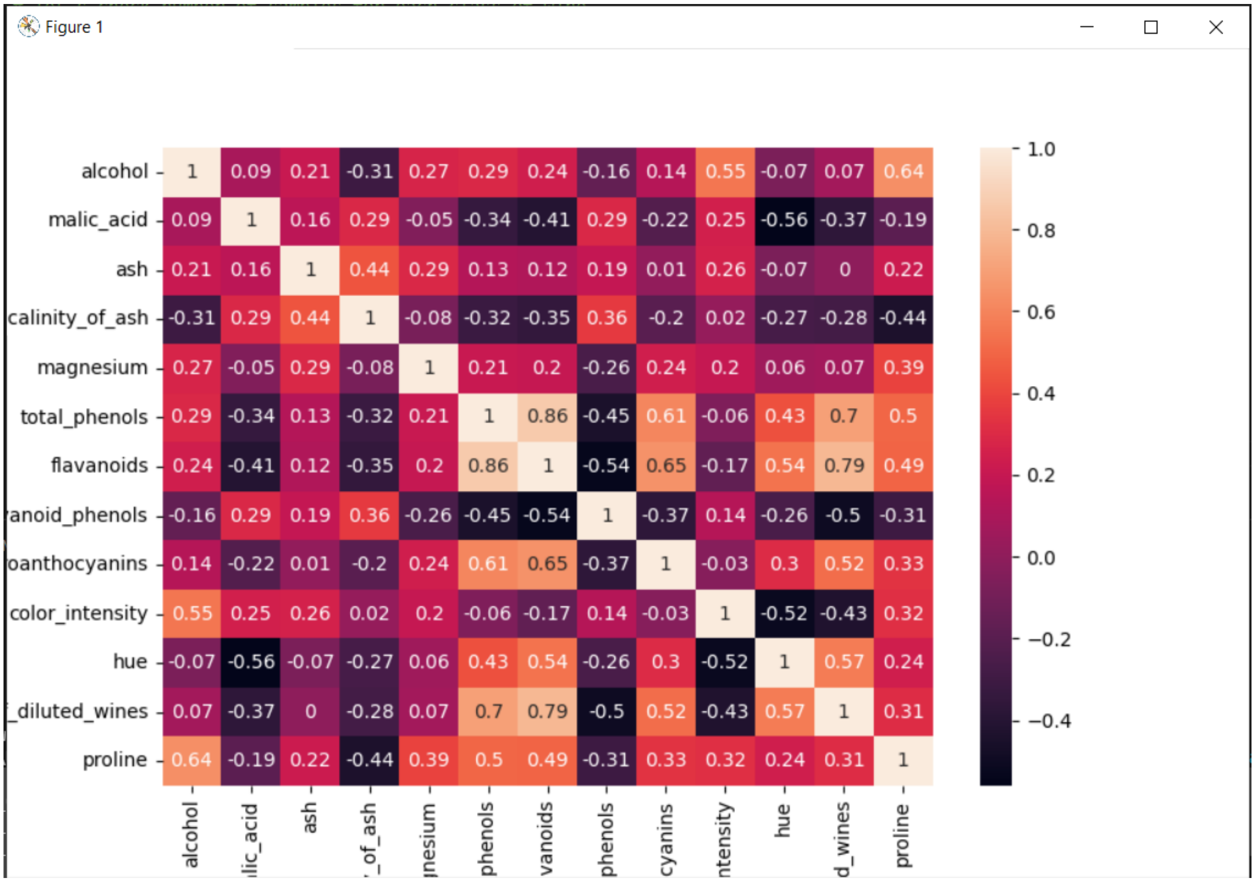


- Next, let's make a correlation matrix to quantitatively examine the relationship between variables.
- If there are features and many of the features are highly correlated, then training an algorithm with all the features will reduce the accuracy. Thus features selection should be done carefully. This dataset has less features but still we will see the correlation.
- The correlation matrix can be formed by using the corr function from the pandas library.
- The correlation coefficient ranges from -1 to 1 . If the value is close to 1 , it means that there is a strong positive correlation between the two variables. When it is close to -1 , the variables have a strong negative correlation.
- Then, we will use the heatmap function from the seaborn library to plot the correlation matrix.

Input:

```
# changing the figure size
plt.figure(figsize = (9, 6))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
plt.show()
```

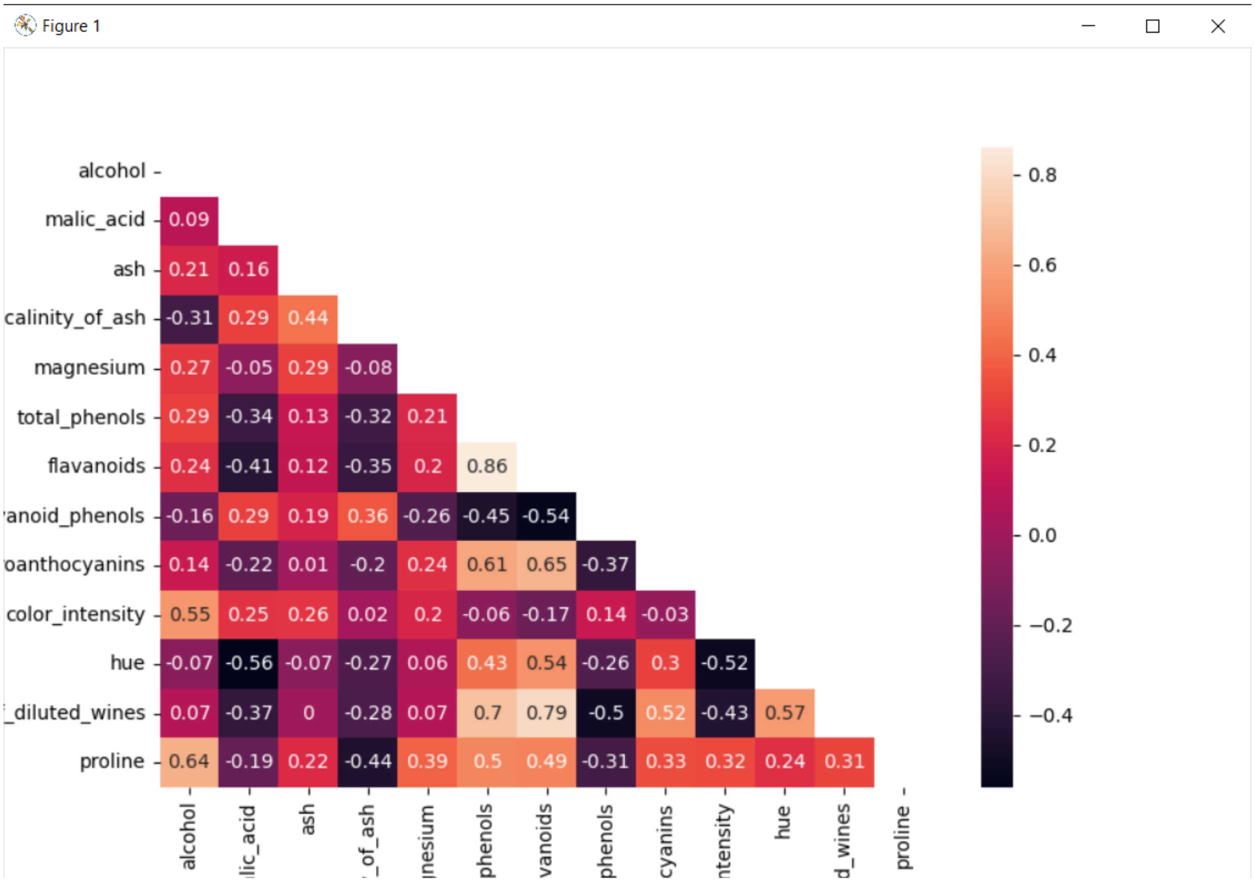
Output:



Input:

```
# Steps to remove redundant values
# Return a array filled with zeros
mask = np.zeros_like(correlation_matrix)
# Return the indices for the upper-triangle of array
mask[np.triu_indices_from(mask)] = True
# changing the figure size
plt.figure(figsize = (9, 6))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True, mask=mask)
plt.show()
```

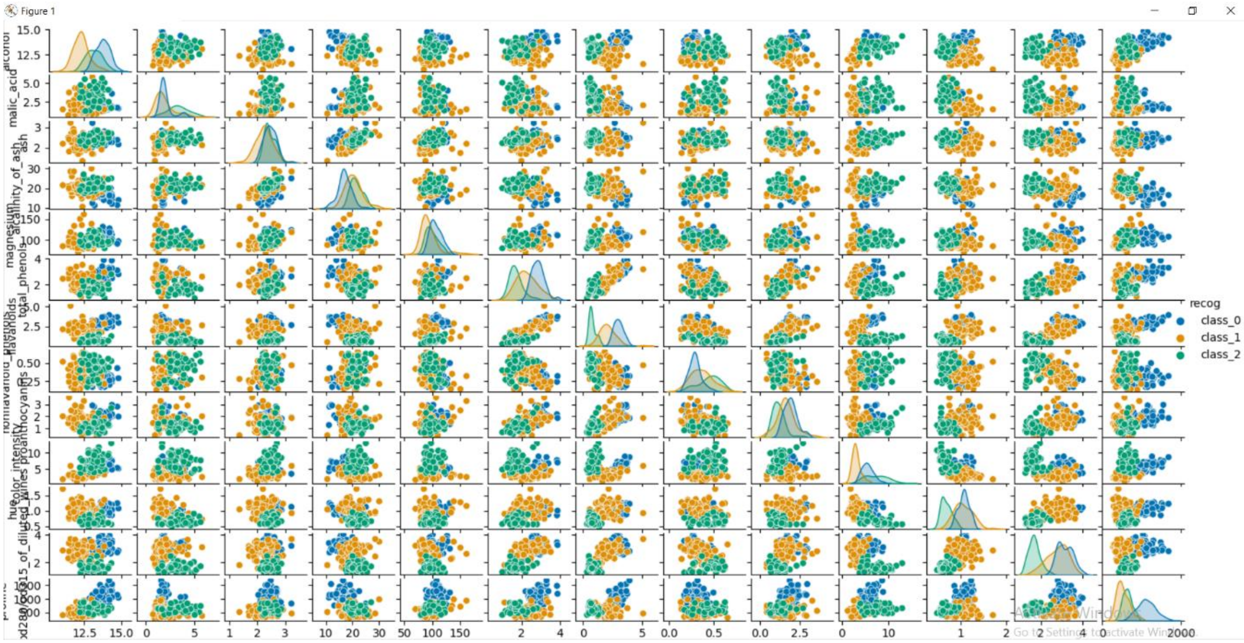
Output:



Input:

```
# let's create pairplot to visualise the data for each pair of attributes
sns.pairplot(wine_df, hue="recog", height = 2, palette = 'colorblind')
plt.show()
```

Output:

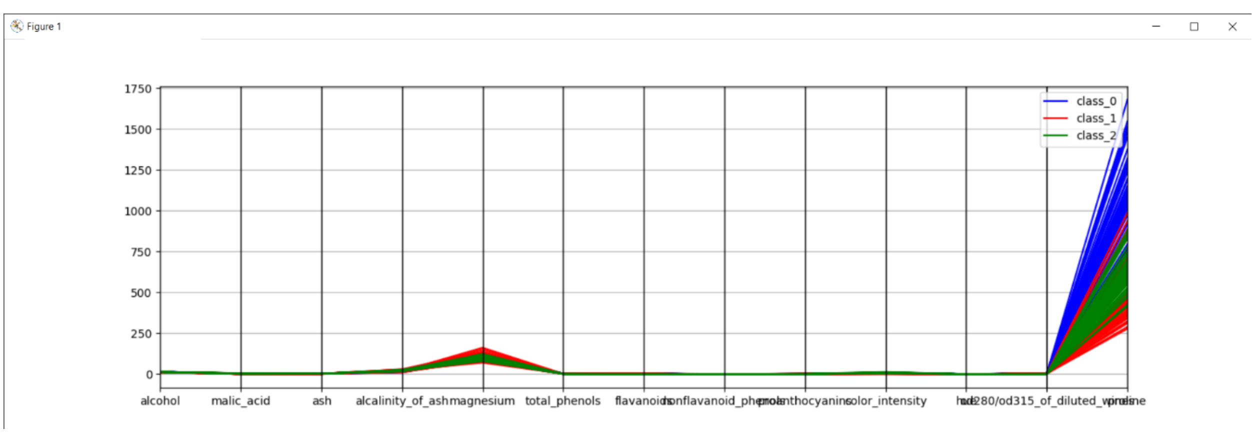


- As we can see from the above figure, wine classes are pretty close to each other.
- For this dataset, another useful visualization plot is parallel coordinate, which represents each row as a line.
- A parallel plot allows to compare the feature of several individual observations (series) on a set of numeric variables. Interestingly, Pandas is probably the best way to plot a parallel coordinate plot with python.

Input:

```
parallel_coordinates(wine_df, "recog", color = ['blue', 'red', 'green'])
plt.show()
```

Output:



Create Features Matrix & Target Variable

Input:

```
X = wine_df[['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',
'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']]
print(X)
```

Output:

```
     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  flavanoids
nonflavanoid_phenols  proanthocyanins  color_intensity   hue  od280/od315_of_diluted_wines
proline

0      14.23        1.71  2.43               15.6      127.0           2.80        3.06                  0.28
2.29             5.64  1.04                          3.92   1065.0

1      13.20        1.78  2.14               11.2      100.0           2.65        2.76                  0.26
1.28             4.38  1.05                          3.40   1050.0

2      13.16        2.36  2.67               18.6      101.0           2.80        3.24                  0.30
2.81             5.68  1.03                          3.17   1185.0

3      14.37        1.95  2.50               16.8      113.0           3.85        3.49                  0.24
2.18             7.80  0.86                          3.45   1480.0

4      13.24        2.59  2.87               21.0      118.0           2.80        2.69                  0.39
1.82             4.32  1.04                          2.93    735.0

..       ...         ...   ...                ...        ...            ...         ...                   ...              ...              ...   ...
...       ...

173    13.71        5.65  2.45               20.5       95.0           1.68        0.61                  0.52
1.06             7.70  0.64                          1.74    740.0

174    13.40        3.91  2.48               23.0      102.0           1.80        0.75                  0.43
1.41             7.30  0.70                          1.56    750.0

175    13.27        4.28  2.26               20.0      120.0           1.59        0.69                  0.43
1.35            10.20  0.59                          1.56    835.0

176    13.17        2.59  2.37               20.0      120.0           1.65        0.68                  0.53
1.46             9.30  0.60                          1.62    840.0
```

177 14.13 4.10 2.74 24.5 96.0 2.05 0.76 0.56
1.35 9.20 0.61 1.60 560.0

[178 rows x 13 columns]

Input:

```
y = wine_df['recog']
print(y)
```

Output:

```
0      class_0
1      class_0
2      class_0
3      class_0
4      class_0
        ...
173    class_2
174    class_2
175    class_2
176    class_2
177    class_2
Name: recog, Length: 178, dtype: object
```

Split the dataset

Input:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 16)
print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
print("y_train shape: ", y_train.shape)
print("y_test shape: ", y_test.shape)
```

Output:

X_train shape: (124, 13)

X_test shape: (54, 13)

y_train shape: (124,)

y_test shape: (54,)

Create Model: Support Vector Machine (SVM)

Input:

```
model_svm = svm.SVC() #select the algorithm
model_svm.fit(X_train, y_train) #train the model with the training dataset
y_prediction_svm = model_svm.predict(X_test) # pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)
print("----------------------------------")
print('The accuracy of the SVM is: {}'.format(score_svm))
print("----------------------------------")
# save the accuracy score
score = set()
score.add(('SVM', score_svm))
```

Output:

The accuracy of the SVM is: 0.6111

Create Model: Decision Tree

Input:

```
model_dt = DecisionTreeClassifier(random_state=4)
model_dt.fit(X_train, y_train) #train the model with the training dataset
y_prediction_dt = model_dt.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
print("---------------------------------")
print('The accuracy of the DT is: {}'.format(score_dt))
print("---------------------------------")
# save the accuracy score
score = set()
score.add(('DT', score_dt))
```

Output:

The accuracy of the DT is: 0.9444

Create Model: K Nearest Neighbours (KNN)

Input:

```
from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
model_knn = KNeighborsClassifier(n_neighbors=3) # 3 neighbours for putting the new data into a class
model_knn.fit(X_train, y_train) #train the model with the training dataset
y_prediction_knn = model_knn.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)
print("---------------------------------")
print('The accuracy of the KNN is: {}'.format(score_knn))
print("---------------------------------")
# save the accuracy score
score = set()
score.add(('KNN', score_knn))
```

Output:

The accuracy of the KNN is: 0.7037

Create Model: Logistic Regression

Input:

```
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train) #train the model with the training dataset
y_prediction_lr = model_lr.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_lr = metrics.accuracy_score(y_prediction_lr, y_test).round(4)
print("--------------------------------")
print('The accuracy of the LR is: {}'.format(score_lr))
print("--------------------------------")
# save the accuracy score
score = set()
score.add(('LR', score_lr))
```

Output:

The accuracy of the LR is: 0.9444

Input:

```python

model_nb = GaussianNB()
model_nb.fit(X_train, y_train) #train the model with the training dataset
y_prediction_nb = model_nb.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_nb = metrics.accuracy_score(y_prediction_nb, y_test).round(4)
print("--------------------------------")
print('The accuracy of the NB is: {}'.format(score_nb))
print("--------------------------------")
# save the accuracy score
score = set()
score.add(('NB', score_nb))

```

Output:

The accuracy of the NB is: 1.0

- Best accuracy

Compare Accuracy Score of Different Models

The accuracy scores of different Models:

('NB', 1.0)

('LR', 0.9444)

('SVM', 0.6111)

('KNN', 0.7037)

('DT', 0.9444)

- Here we can see that all the models accuracy score are not good.
- Only accuracy score above 0.9 is good.
- NB model gave the prefect accuracy