10-Wine Classification Solution

October 17, 2021

1 Classification Challenge

Wine experts can identify wines from specific vineyards through smell and taste, but the factors that give different wines their individual charateristics are actually based on their chemical composition.

In this challenge, you must train a classification model to analyze the chemical and visual features of wine samples and classify them based on their cultivar (grape variety).

> **Citation**: The data used in this exercise was originally collected by 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.
>
> It can be downloaded from the UCI dataset repository (Dua, D. and Graff, C. (2019). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science).

1.1 Explore the data

Run the following cell to load a CSV file of wine data, which consists of 12 numeric features and a classification label with the following classes:

- **0** (*variety A*)
- **1** (*variety B*)
- **2** (*variety C*)

[ ]:
```
import pandas as pd

# load the training dataset
data = pd.read_csv('data/wine.csv')
data.sample(10)
```

[ ]:
```
     Alcohol  Malic_acid   Ash  Alcalinity  Magnesium  Phenols  Flavanoids  \
35     13.48        1.81  2.41        20.5        100     2.70        2.98
27     13.30        1.72  2.14        17.0         94     2.40        2.19
1      13.20        1.78  2.14        11.2        100     2.65        2.76
103    11.82        1.72  1.88        19.5         86     2.50        1.64
126    12.43        1.53  2.29        21.5         86     2.74        3.15
100    12.08        2.08  1.70        17.5         97     2.23        2.17
37     13.05        1.65  2.55        18.0         98     2.45        2.43
```

```
97     12.29        1.41  1.98        16.0         85     2.55          2.50
170    12.20        3.03  2.32        19.0         96     1.25          0.49
10     14.10        2.16  2.30        18.0        105     2.95          3.32

     Nonflavanoids  Proanthocyanins  Color_intensity   Hue  \
35            0.26             1.86             5.10  1.04
27            0.27             1.35             3.95  1.02
1             0.26             1.28             4.38  1.05
103           0.37             1.42             2.06  0.94
126           0.39             1.77             3.94  0.69
100           0.26             1.40             3.30  1.27
37            0.29             1.44             4.25  1.12
97            0.29             1.77             2.90  1.23
170           0.40             0.73             5.50  0.66
10            0.22             2.38             5.75  1.25

     OD280_315_of_diluted_wines  Proline  WineVariety
35                         3.47      920            0
27                         2.77     1285            0
1                          3.40     1050            0
103                        2.44      415            1
126                        2.84      352            1
100                        2.96      710            1
37                         2.51     1105            0
97                         2.74      428            1
170                        1.83      510            2
10                         3.17     1510            0
```

Your challenge is to explore the data and train a classification model that achieves an overall *Recall* metric of over 0.95 (95%).

1.1.1 Separate features and label

[ ]:

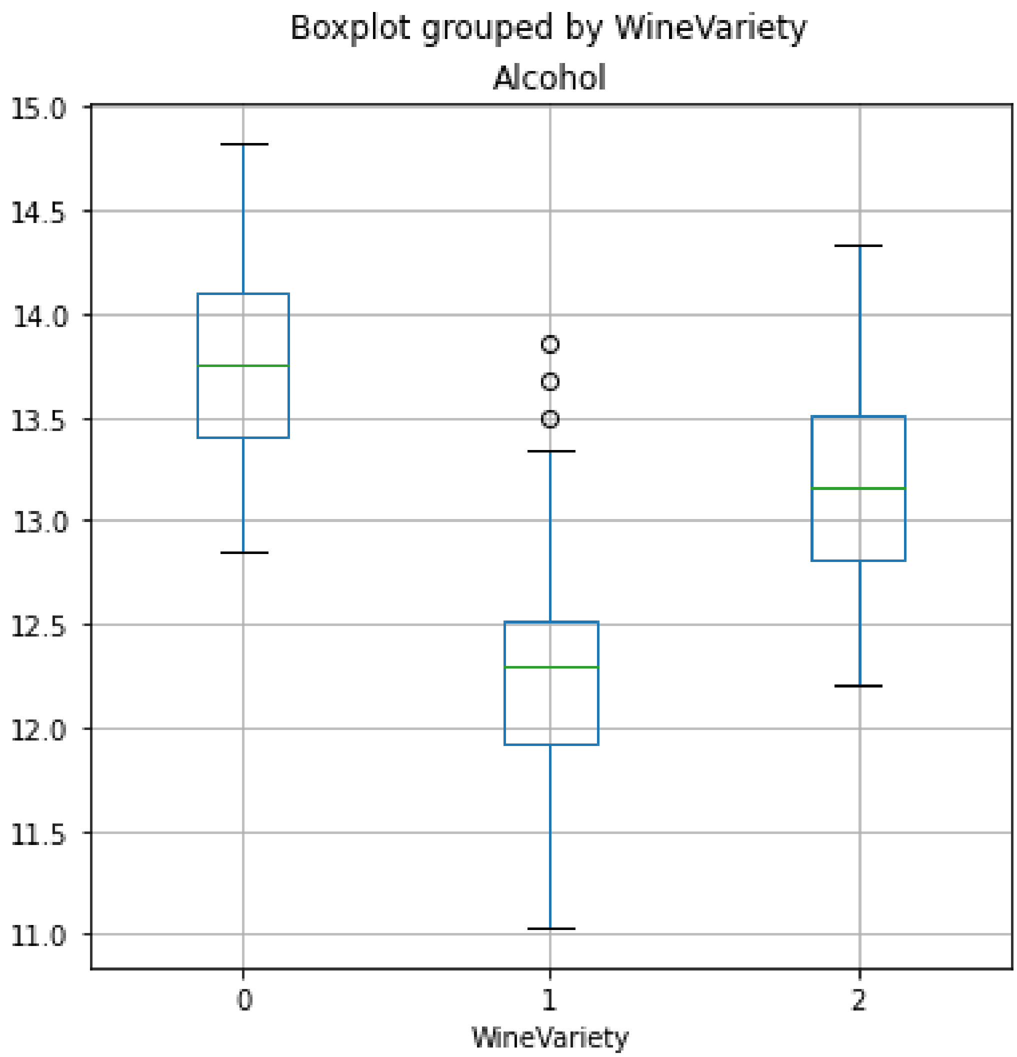
```
# Separate features and labels
features =␣
 ↪['Alcohol','Malic_acid','Ash','Alcalinity','Magnesium','Phenols','Flavanoids','Nonflavanoid
label = 'WineVariety'
X, y = data[features].values, data[label].values

for n in range(0,4):
    print("Wine", str(n+1), "\n  Features:",list(X[n]), "\n  Label:", y[n])
```

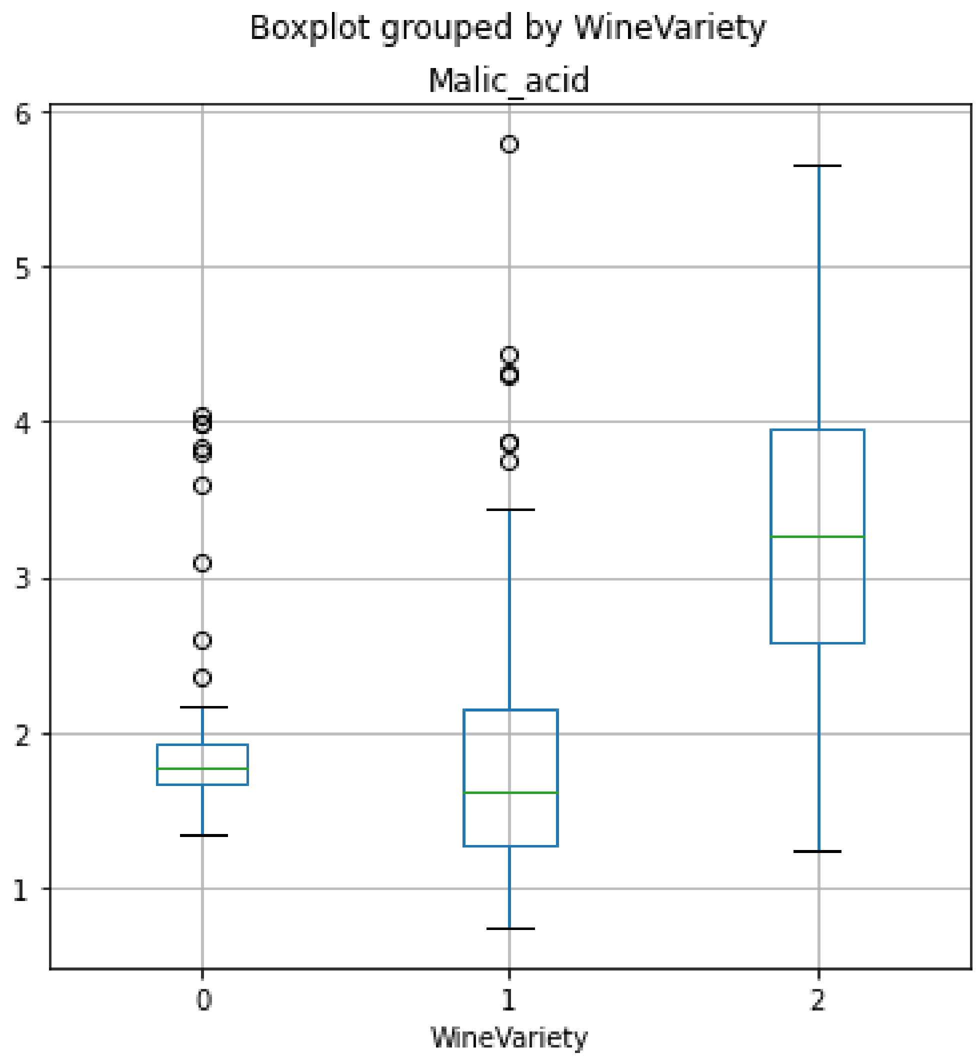
```
Wine 1
  Features: [14.23, 1.71, 2.43, 15.6, 127.0, 2.8, 3.06, 0.28, 2.29, 5.64, 1.04,
3.92, 1065.0]
  Label: 0
Wine 2
  Features: [13.2, 1.78, 2.14, 11.2, 100.0, 2.65, 2.76, 0.26, 1.28, 4.38, 1.05,
```

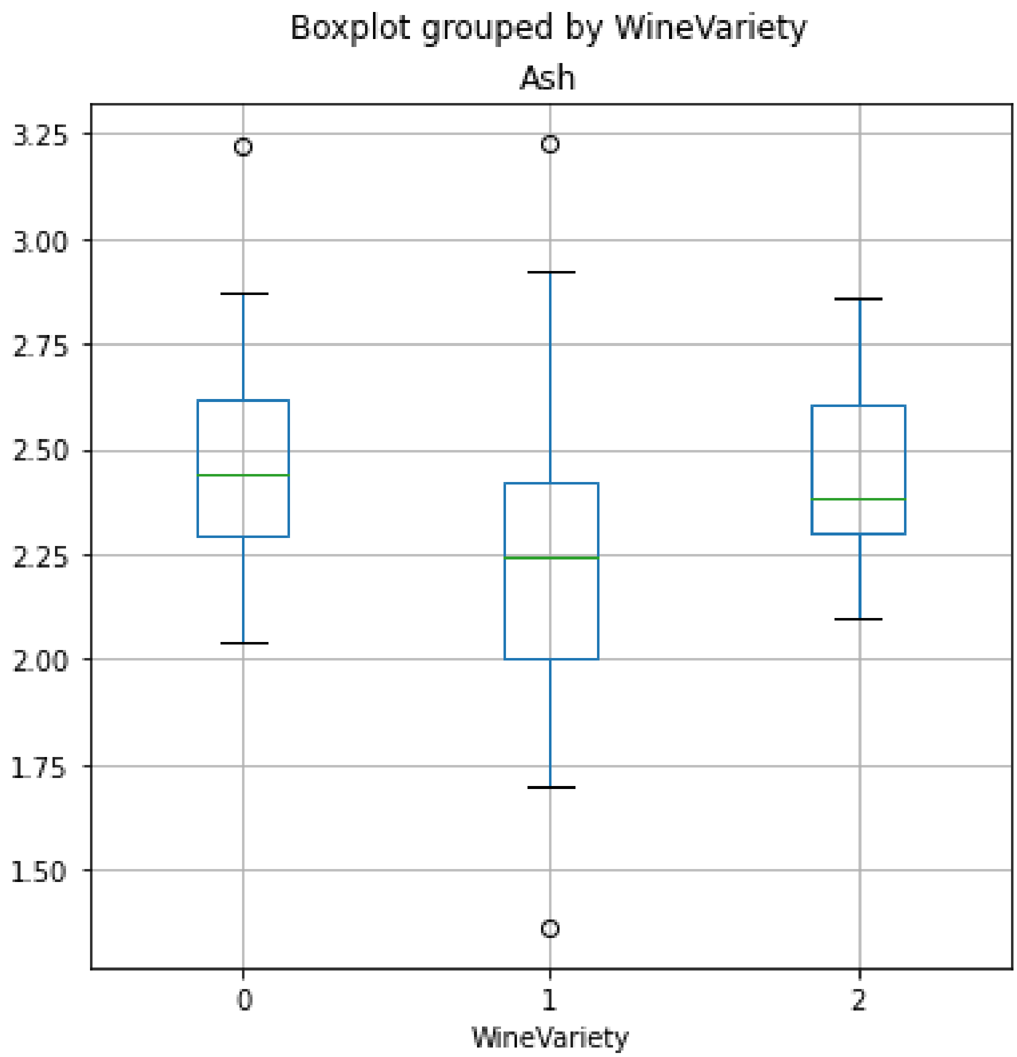
```
3.4, 1050.0]
  Label: 0
Wine 3
  Features: [13.16, 2.36, 2.67, 18.6, 101.0, 2.8, 3.24, 0.3, 2.81, 5.68, 1.03,
3.17, 1185.0]
  Label: 0
Wine 4
  Features: [14.37, 1.95, 2.5, 16.8, 113.0, 3.85, 3.49, 0.24, 2.18, 7.8, 0.86,
3.45, 1480.0]
  Label: 0
```

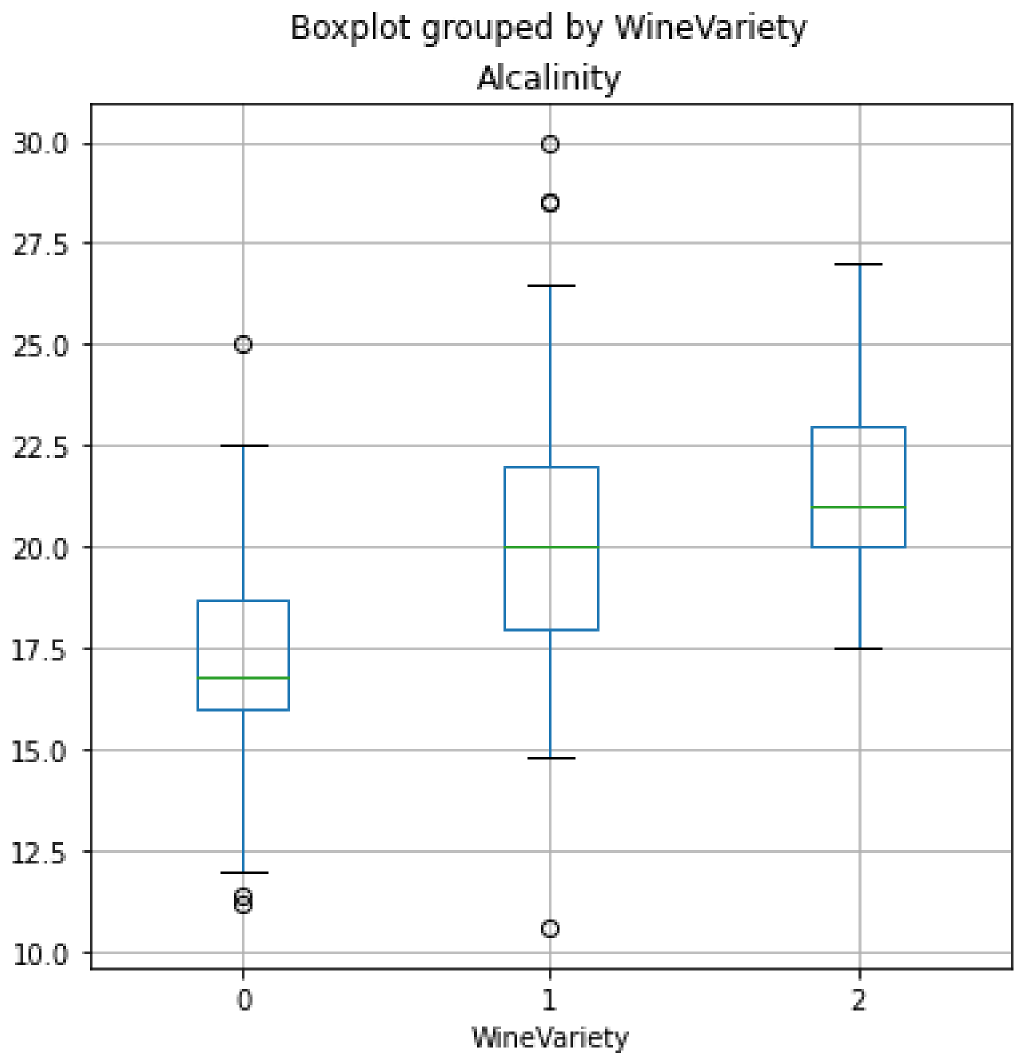
1.1.2 Compare feature distributions

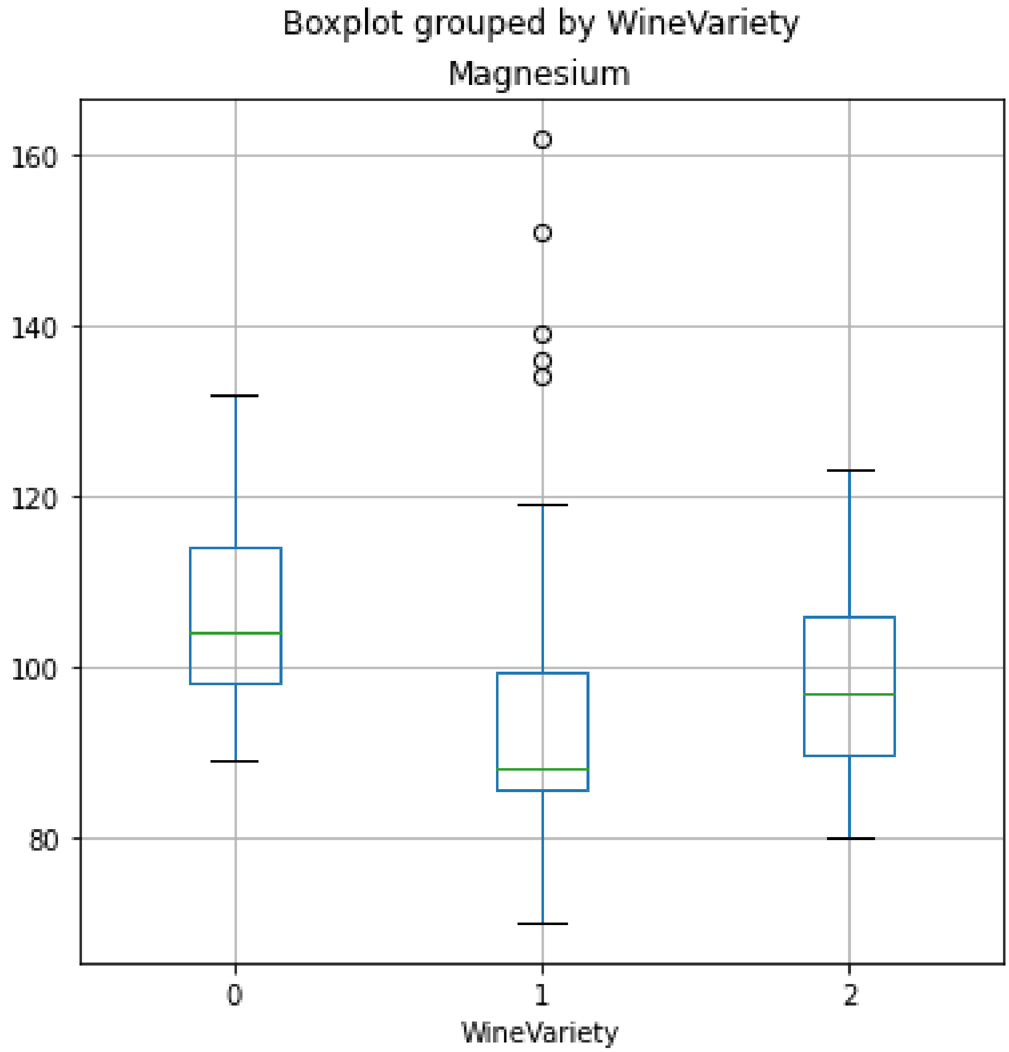
[ ]:
```python
from matplotlib import pyplot as plt
%matplotlib inline

for col in features:
    data.boxplot(column=col, by=label, figsize=(6,6))
    plt.title(col)
plt.show()
```

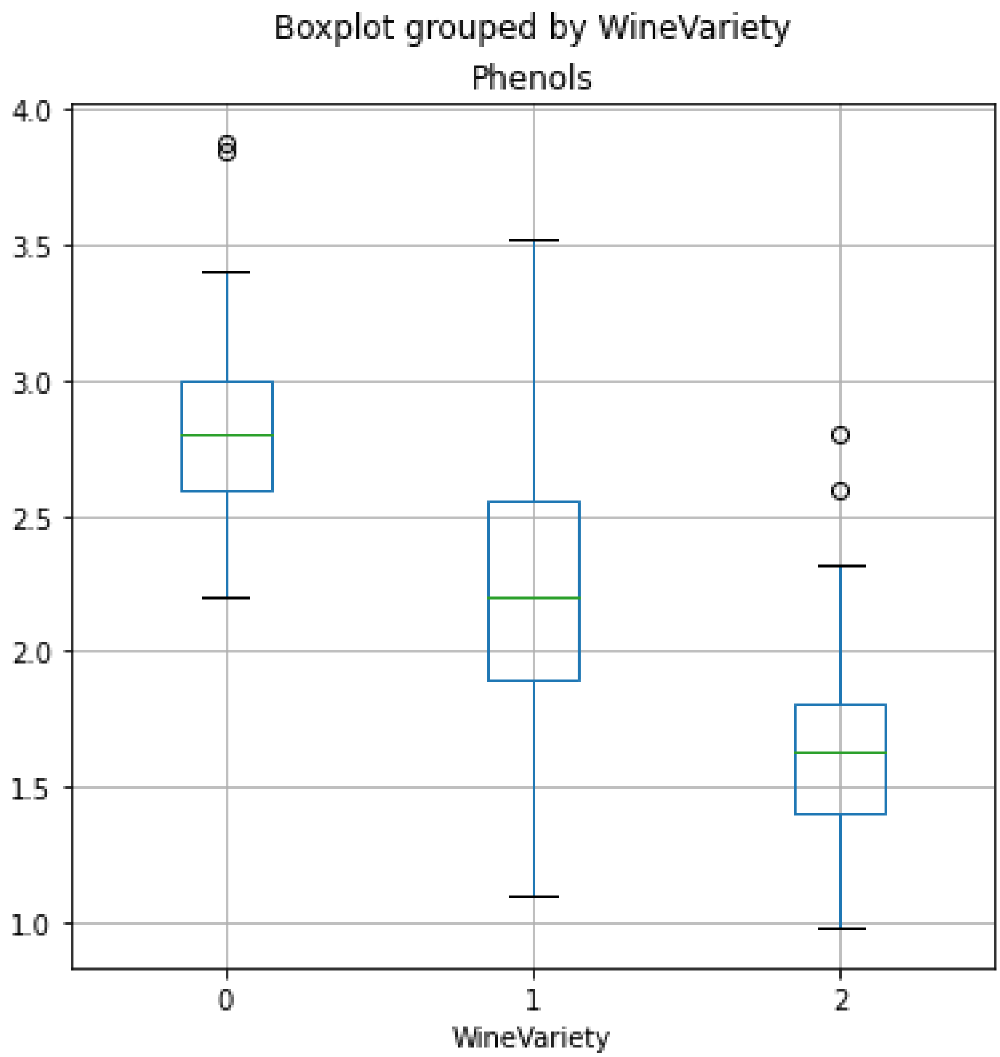

Boxplot grouped by WineVariety
Alcohol
15.0
14.5
14.0
13.5
13.0
12.5
12.0
11.5
11.0
0
1
2
WineVariety

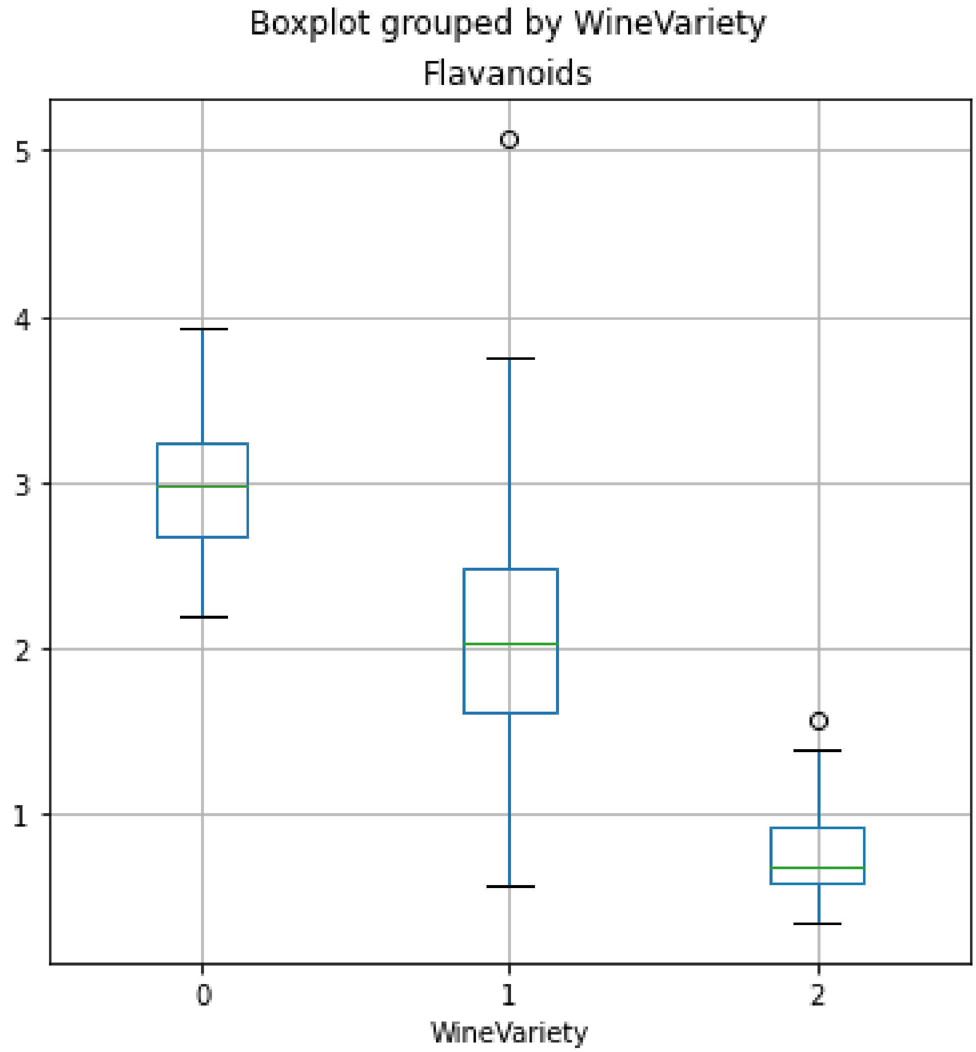

Boxplot grouped by WineVariety
Malic_acid
6
5
4
3
2
1
0
1
2
WineVariety

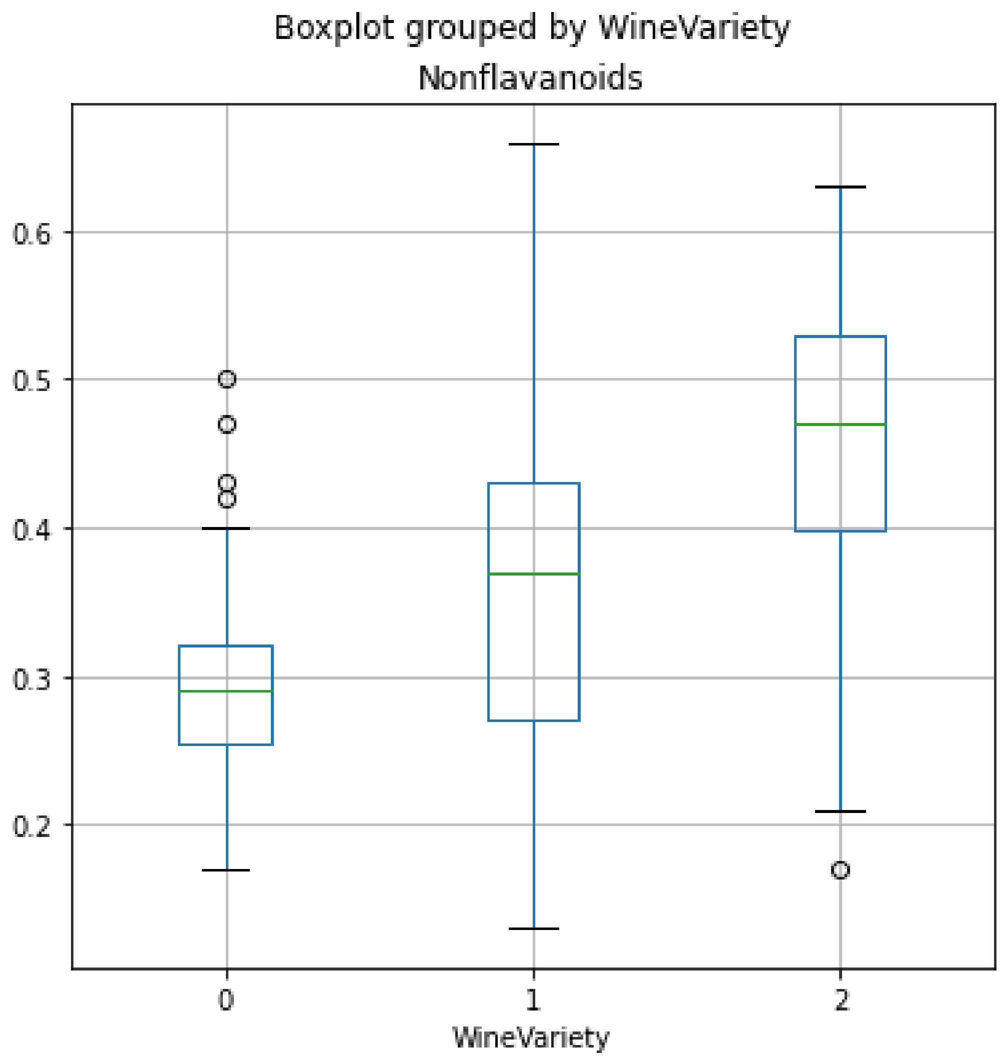

Boxplot grouped by WineVariety
Ash
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
0
1
2
WineVariety


Boxplot grouped by WineVariety
Alcalinity
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
0
1
2
WineVariety

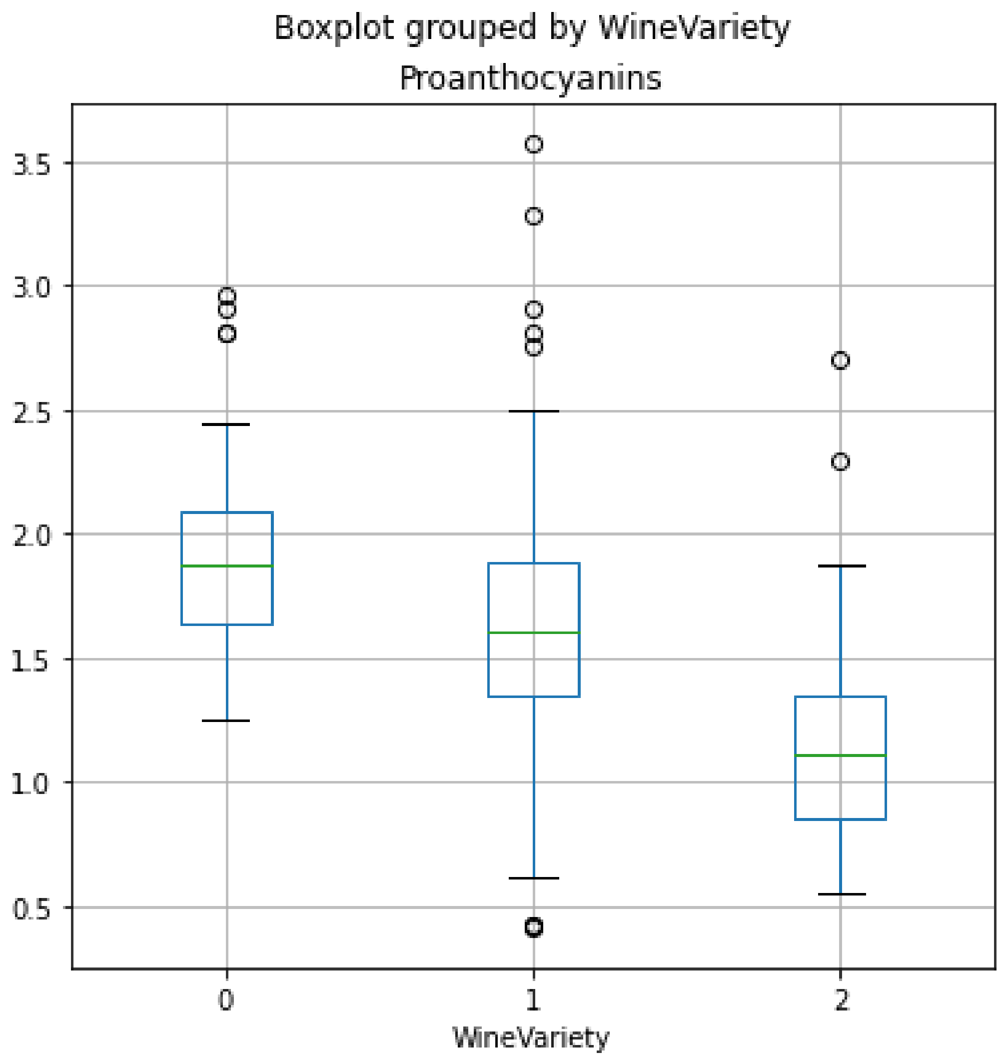

Boxplot grouped by WineVariety
Magnesium
160
140
120
100
80
0
1
2
WineVariety

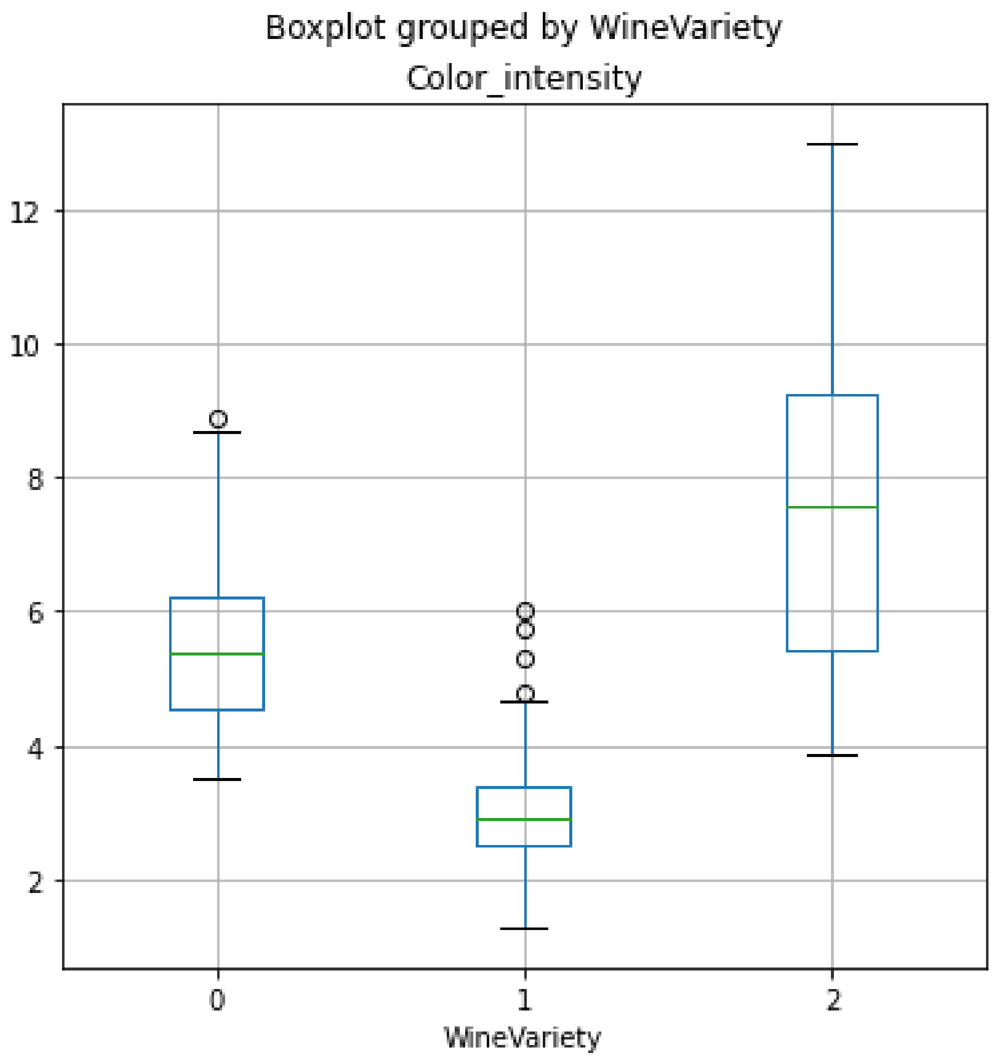

Boxplot grouped by WineVariety
Phenols
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
2
WineVariety

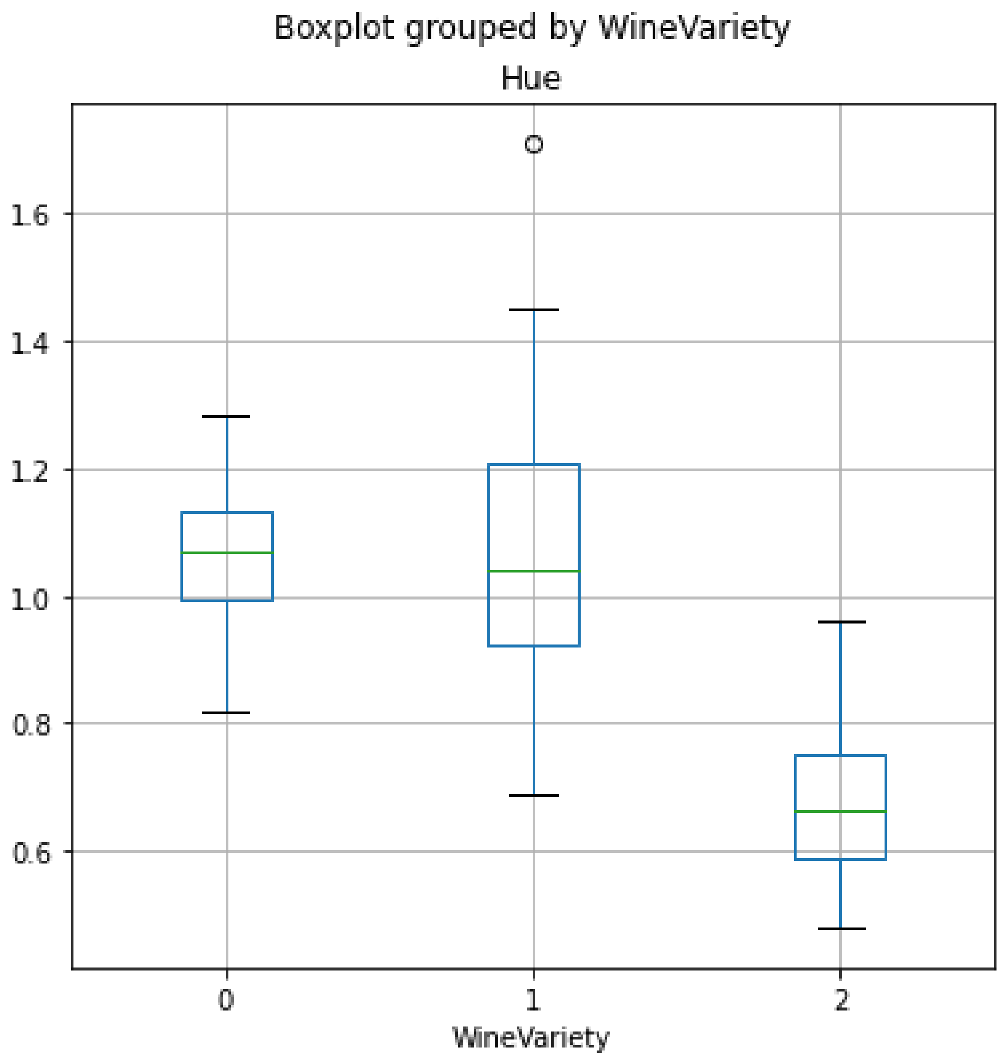

Boxplot grouped by WineVariety
Flavanoids
5
4
3
2
1
0
1
2
WineVariety

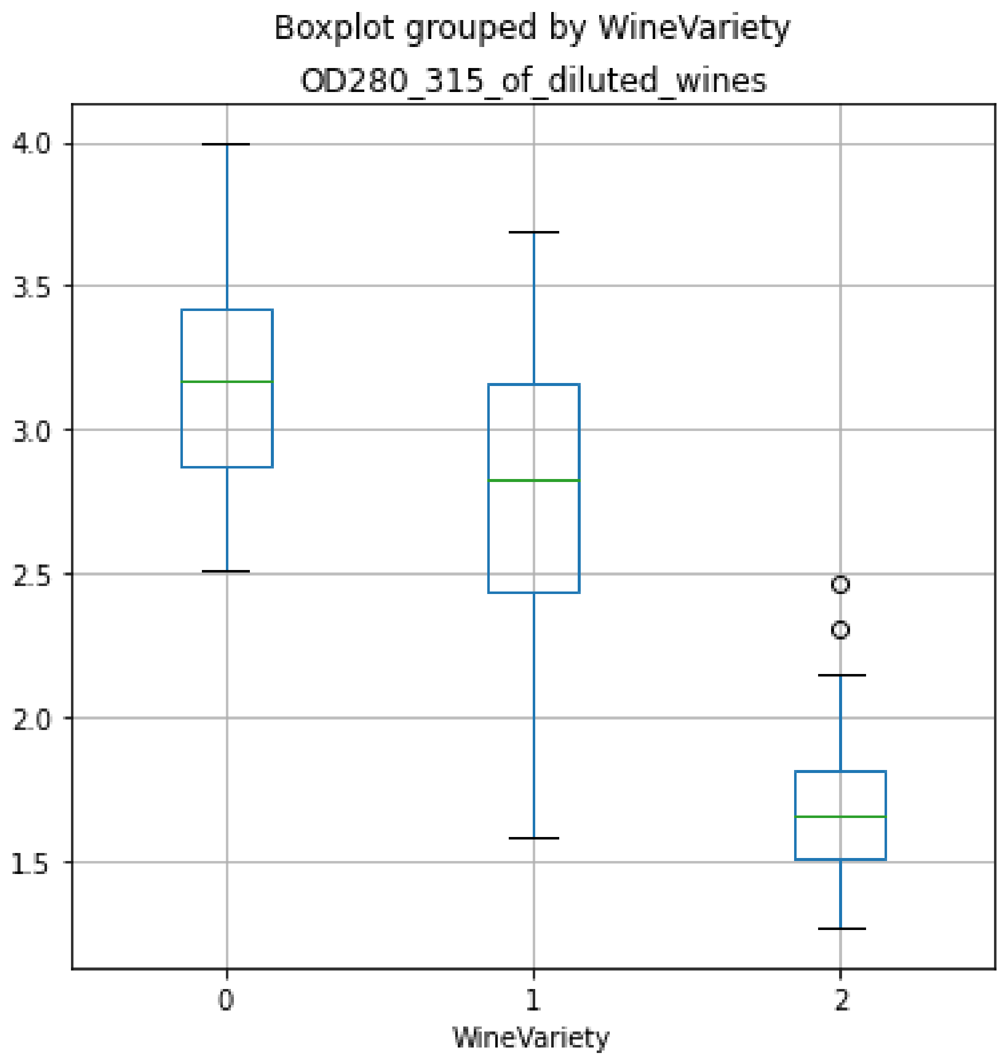


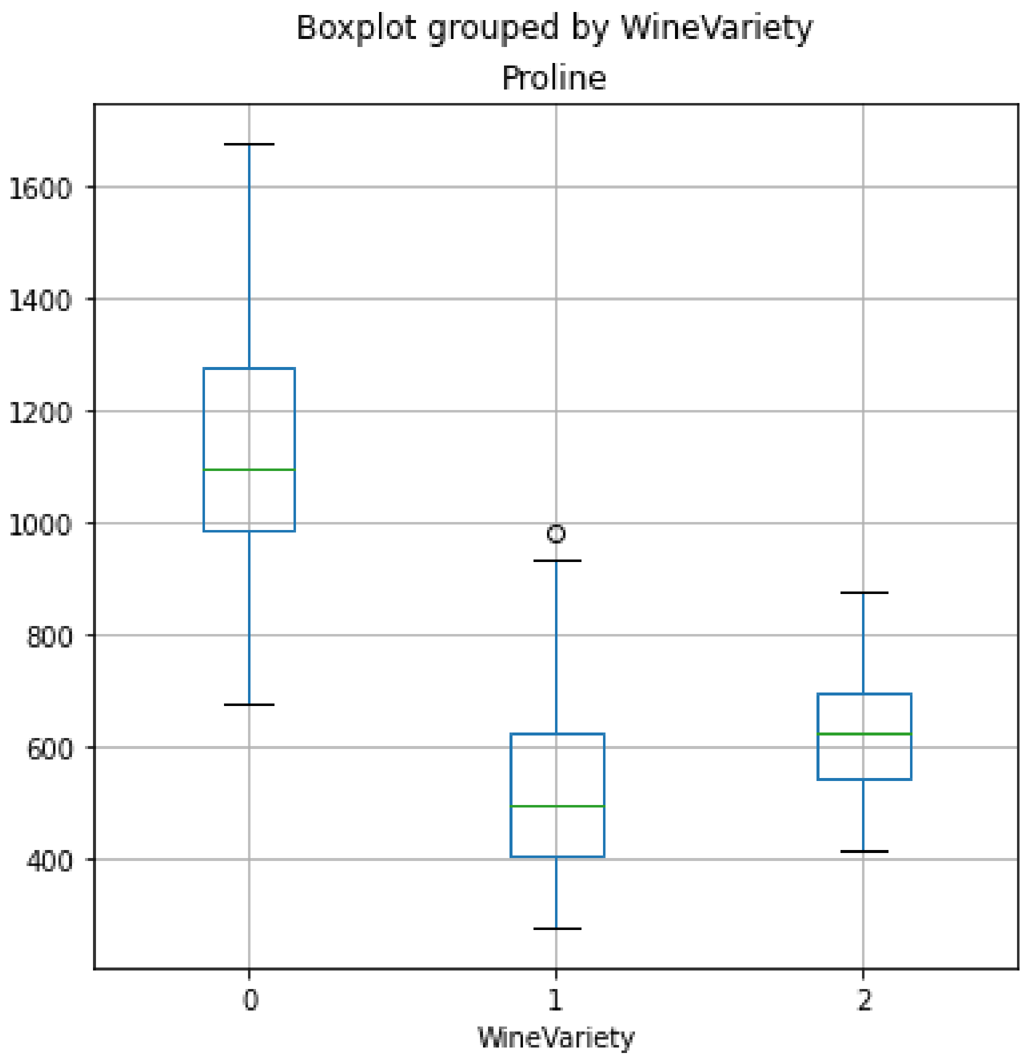
Boxplot grouped by WineVariety
Nonflavanoids
0.6
0.5
0.4
0.3
0.2
0
1
2
WineVariety


Boxplot grouped by WineVariety
Proanthocyanins
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
1
2
WineVariety


Boxplot grouped by WineVariety
Color_intensity
12
10
8
6
4
2
0
1
2
WineVariety


Boxplot grouped by WineVariety
Hue
1.6
1.4
1.2
1.0
0.8
0.6
0
1
2
WineVariety


Boxplot grouped by WineVariety
OD280_315_of_diluted_wines
4.0
3.5
3.0
2.5
2.0
1.5
0
1
2
WineVariety



1.1.3 Split the data for training and validation

```
[ ]: from sklearn.model_selection import train_test_split

# Split data 70%-30% into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state=0)

print ('Training cases: %d\nTest cases: %d' % (X_train.shape[0], X_test.
 ↪shape[0]))
```

```
Training cases: 124
Test cases: 54
```

1.1.4 Normalize features and train model

[ ]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

# Define preprocessing for numeric columns (scale them)
feature_columns = [0,1,2,3,4,5,6]
feature_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
    ])

# Create preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('preprocess', feature_transformer, feature_columns)])

# Create training pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('regressor', LogisticRegression(solver='lbfgs',␣
 ↪multi_class='auto'))])


# fit the pipeline to train a linear regression model on the training set
model = pipeline.fit(X_train, y_train)
print (model)
```

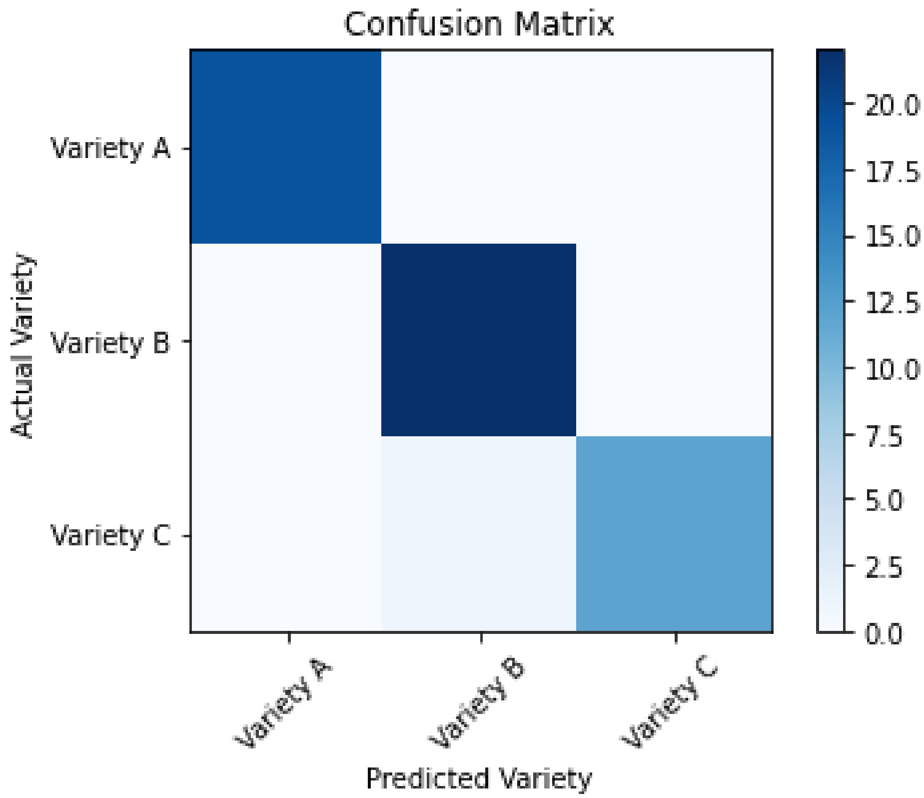
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('preprocess',
                                                  Pipeline(memory=None,
                                                           steps=[('scaler',
StandardScaler(copy=True,
  with_mean=True,
  with_std=True))],
                                                           verbose=False),
                                                  [0, 1, 2, 3, 4, 5, 6])],
                                   verbose=False)),
                ('regressor',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l2', random_state=None,
```

```
                                        solver='lbfgs', tol=0.0001, verbose=0,
                                        warm_start=False))],
         verbose=False)
```

1.1.5 Evaluate model

```
[ ]: from sklearn. metrics import accuracy_score, precision_score, recall_score,␣
      ↪confusion_matrix
     import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline

     # Get predictions from test data
     predictions = model.predict(X_test)

     # Get metrics
     print("Overall Accuracy:",accuracy_score(y_test, predictions))
     print("Overall Precision:",precision_score(y_test, predictions,␣
      ↪average='macro'))
     print("Overall Recall:",recall_score(y_test, predictions, average='macro'))

     # Plot confusion matrix
     cm = confusion_matrix(y_test, predictions)
     classes = ['Variety A','Variety B','Variety C']
     plt.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
     plt.colorbar()
     tick_marks = np.arange(len(classes))
     plt.xticks(tick_marks, classes, rotation=45)
     plt.yticks(tick_marks, classes)
     plt.title('Confusion Matrix')
     plt.xlabel("Predicted Variety")
     plt.ylabel("Actual Variety")
     plt.show()
```

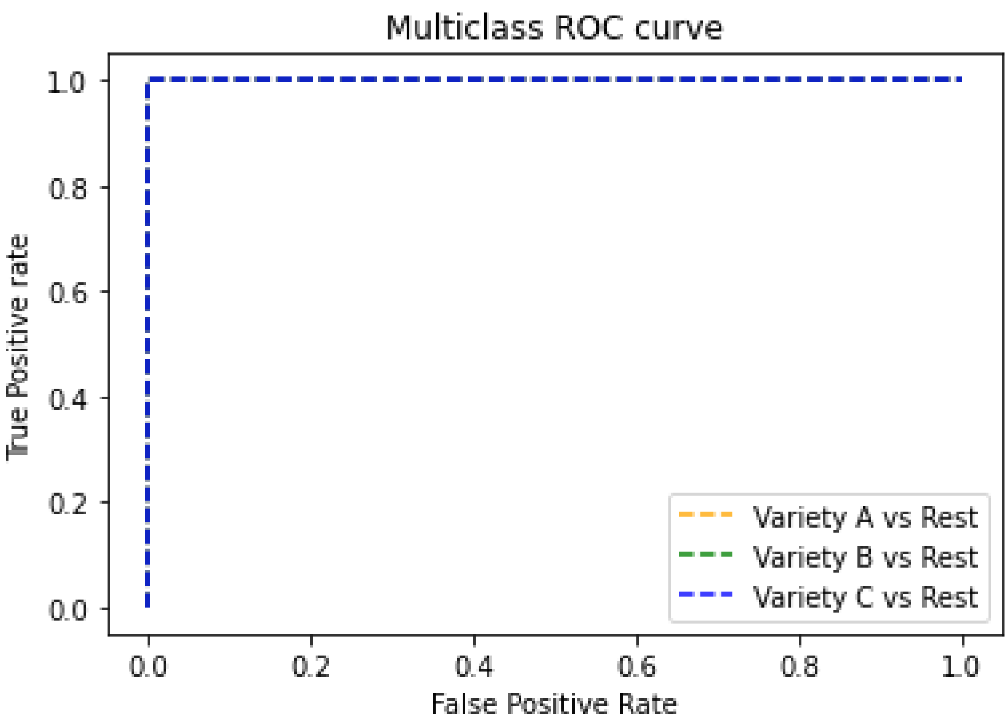
```
Overall Accuracy: 0.9814814814814815
Overall Precision: 0.9855072463768115
Overall Recall: 0.9743589743589745
```



[ ]:
```
from sklearn.metrics import roc_curve, roc_auc_score

# Get class probability scores
probabilities = model.predict_proba(X_test)

auc = roc_auc_score(y_test,probabilities, multi_class='ovr')
print('Average AUC:', auc)

# Get ROC metrics for each class
fpr = {}
tpr = {}
thresh ={}
for i in range(len(classes)):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, probabilities[:,i],␣
 ↪pos_label=i)

# Plot the ROC chart
plt.plot(fpr[0], tpr[0], linestyle='--',color='orange', label=classes[0] + ' vs␣
 ↪Rest')
plt.plot(fpr[1], tpr[1], linestyle='--',color='green', label=classes[1] + ' vs␣
 ↪Rest')
```

```
plt.plot(fpr[2], tpr[2], linestyle='--',color='blue', label=classes[2] + ' vs␣
↪Rest')
plt.title('Multiclass ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```

```
Average AUC: 1.0
```



1.2 Use the model with new data observation

When you're happy with your model's predictive performance, save it and then use it to predict classes for the following two new wine samples:

- [13.72,1.43,2.5,16.7,108,3.4,3.67,0.19,2.04,6.8,0.89,2.87,1285]
- [12.37,0.94,1.36,10.6,88,1.98,0.57,0.28,0.42,1.95,1.05,1.82,520]

[ ]:
```
import joblib

# Save the model as a pickle file
filename = './wine_classifer.pkl'
joblib.dump(model, filename)
```

```python
# Load the saved model
model = joblib.load(filename)

# Get predictions for two new wine samples
x_new = np.array([[13.72,1.43,2.5,16.7,108,3.4,3.67,0.19,2.04,6.8,0.89,2.
 ↪87,1285],
                  [12.37,0.94,1.36,10.6,88,1.98,0.57,0.28,0.42,1.95,1.05,1.
 ↪82,520]])

# Call the web service, passing the input data
predictions = model.predict(x_new)

# Get the predicted classes.
for prediction in predictions:
    print(prediction, '(' + classes[prediction] +')')
```

```
0 (Variety A)
1 (Variety B)
```