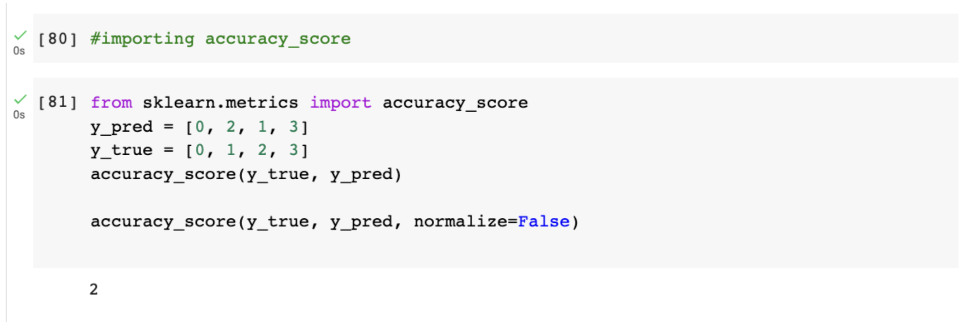
Report

- This code consists of several parts that perform different tasks and use contrasting functions and libraries from the scikit-learn package.

- The first part imports the **accuracy score** function from **sklearn.metrics** and creates two lists (**y_pred** and **y_true**) to test the function's accuracy score. It then prints two different outputs of the **accuracy_score** function, one with the **normalize** parameter set to True and the other set to False.

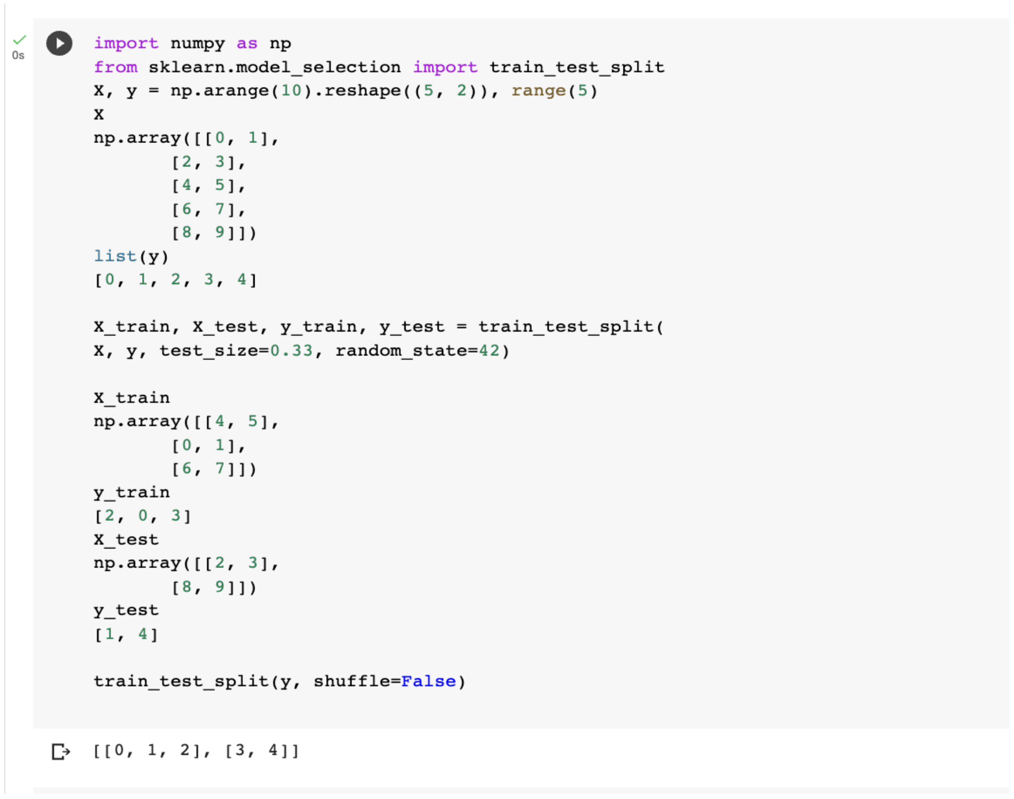


```
[80] #importing accuracy_score
```

```
[81] from sklearn.metrics import accuracy_score
     y_pred = [0, 2, 1, 3]
     y_true = [0, 1, 2, 3]
     accuracy_score(y_true, y_pred)

     accuracy_score(y_true, y_pred, normalize=False)
```

```
2
```

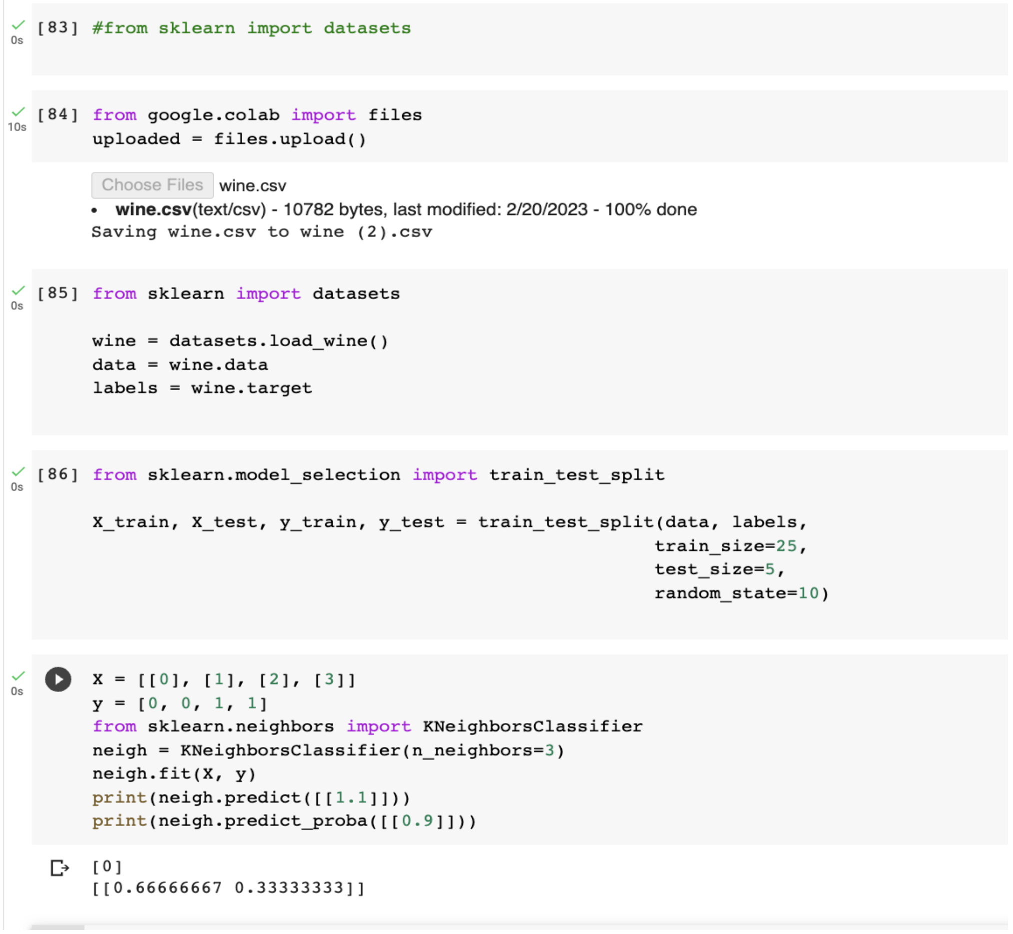
- The second part creates a numpy array **X** and a range list **y** of equal length. It then applies **train_test_split** function to the array and the list, splitting them into training and test data (X_train, X_test) and training and test labels (y_train, y_test). The function's parameters determine the size of the test data set and the random state.



```
import numpy as np
from sklearn.model_selection import train_test_split
X, y = np.arange(10).reshape((5, 2)), range(5)
X
np.array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
list(y)
[0, 1, 2, 3, 4]

X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.33, random_state=42)

X_train
np.array([[4, 5],
       [0, 1],
       [6, 7]])
y_train
[2, 0, 3]
X_test
np.array([[2, 3],
       [8, 9]])
y_test
[1, 4]

train_test_split(y, shuffle=False)
```

```
[[0, 1, 2], [3, 4]]
```

- The third part imports a wine dataset from the **sklearn.datasets** module and creates variables **data** and **labels**. It then uses **train_test_split** function to create training and test data sets for these variables with specific sizes and random state.



```
[83] #from sklearn import datasets
```

```
[84] from google.colab import files
     uploaded = files.upload()
```

Choose Files wine.csv

- **wine.csv**(text/csv) - 10782 bytes, last modified: 2/20/2023 - 100% done

```
Saving wine.csv to wine (2).csv
```

```
[85] from sklearn import datasets

     wine = datasets.load_wine()
     data = wine.data
     labels = wine.target
```

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

     X_train, X_test, y_train, y_test = train_test_split(data, labels,
                                                         train_size=25,
                                                         test_size=5,
                                                         random_state=10)
```

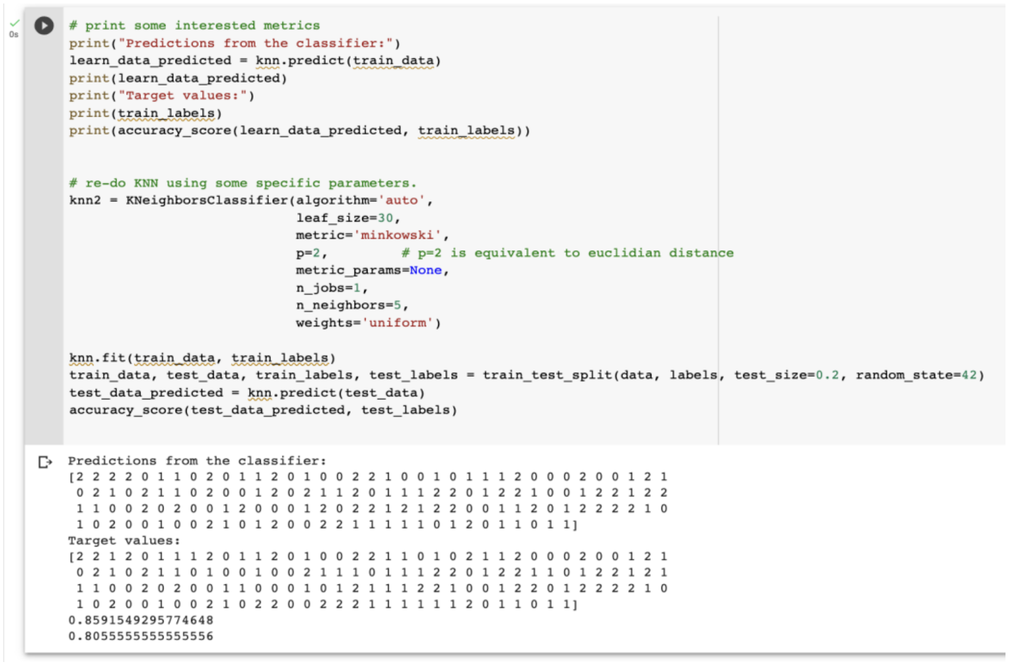
```
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
print(neigh.predict([[1.1]]))
print(neigh.predict_proba([[0.9]]))
```

```
[0]
[[0.66666667 0.33333333]]
```

- The fourth part creates a new set of data and labels and trains a K-Nearest Neighbors classifier on the data using the **KNeighborsClassifier** function from **sklearn.neighbors**. The classifier is then used to make predictions on new data points and print the results.

- The fifth part prints the predictions from the classifier and compares them to the target values. It also prints the accuracy score of the classifier on the training data.

- The final part re-creates the KNN classifier with some specific parameters and applies it to the wine dataset to predict the labels of the test data. It then calculates and prints the accuracy score of the classifier on the test data.



```python
# print some interested metrics
print("Predictions from the classifier:")
learn_data_predicted = knn.predict(train_data)
print(learn_data_predicted)
print("Target values:")
print(train_labels)
print(accuracy_score(learn_data_predicted, train_labels))


# re-do KNN using some specific parameters.
knn2 = KNeighborsClassifier(algorithm='auto',
                            leaf_size=30,
                            metric='minkowski',
                            p=2,         # p=2 is equivalent to euclidian distance
                            metric_params=None,
                            n_jobs=1,
                            n_neighbors=5,
                            weights='uniform')

knn.fit(train_data, train_labels)
train_data, test_data, train_labels, test_labels = train_test_split(data, labels, test_size=0.2, random_state=42)
test_data_predicted = knn.predict(test_data)
accuracy_score(test_data_predicted, test_labels)
```

```
Predictions from the classifier:
[2 2 2 2 0 1 1 0 2 0 1 1 2 0 1 0 0 2 2 1 0 0 1 0 1 1 1 2 0 0 0 2 0 0 1 2 1
 0 2 1 0 2 1 1 0 2 0 0 1 2 0 2 1 1 2 0 1 1 1 2 2 0 1 2 2 1 0 0 1 2 2 1 2 2
 1 1 0 0 2 0 2 0 0 1 2 0 0 0 1 2 0 2 2 1 2 1 2 2 0 0 1 1 2 0 1 2 2 2 2 1 0
 1 0 2 0 0 1 0 0 2 1 0 1 2 0 0 2 2 1 1 1 1 1 0 1 2 0 1 1 0 1 1]
Target values:
[2 2 1 2 0 1 1 1 2 0 1 1 2 0 1 0 0 2 2 1 1 0 1 0 2 1 1 2 0 0 0 2 0 0 1 2 1
 0 2 1 0 2 1 1 0 1 0 0 1 0 0 2 1 1 1 0 1 1 1 2 2 0 1 2 2 1 1 0 1 2 2 1 2 1
 1 1 0 0 2 0 2 0 0 1 1 0 0 0 1 0 1 2 1 1 1 2 2 1 0 0 1 2 2 0 1 2 2 2 2 1 0
 1 0 2 0 0 1 0 0 2 1 0 2 2 0 0 2 2 2 1 1 1 1 1 1 2 0 1 1 0 1 1]
0.8591549295774648
0.8055555555555556
```

- Overall, this code demonstrates the use of several key functions and tools in scikit-learn for data splitting, classification, and accuracy scoring.