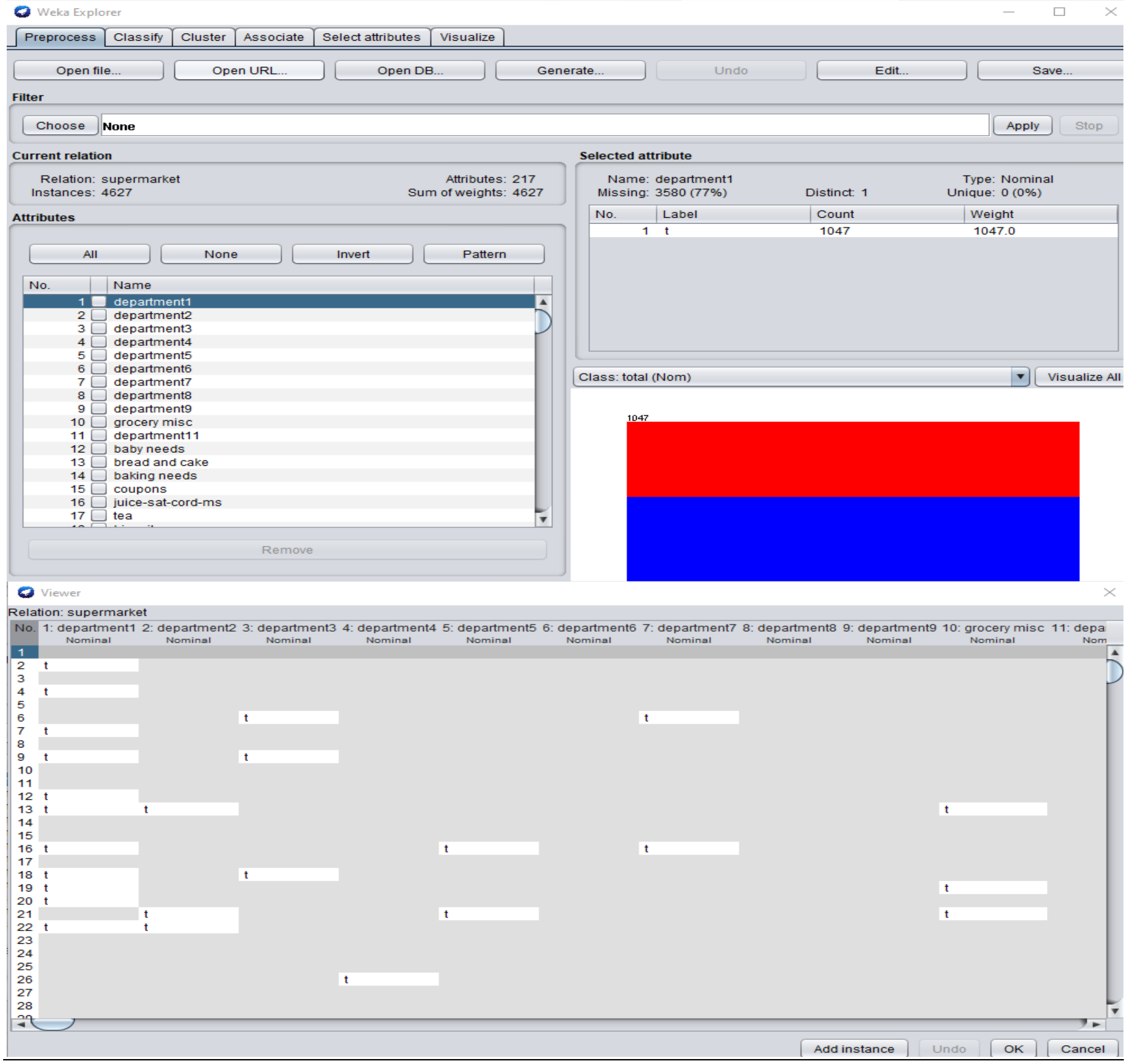
PRACTICAL – 4

Aim: Perform following task as per given instructions.
a. Simulation of Apriori algorithm using Weka tool.

Description:

Apriori algorithm is a sequence of steps to be followed to find the most frequent itemset in the given database.



Filter

None

Current relation

Relation: supermarket
Instances: 4627
Attributes: 217
Sum of weights: 4627

Attributes

No.		Name
1	□	department1
2	□	department2
3	□	department3
4	□	department4
5	□	department5
6	□	department6
7	□	department7
8	□	department8
9	□	department9
10	□	grocery misc
11	□	department11
12	□	baby needs
13	□	bread and cake
14	□	baking needs
15	□	coupons
16	□	juice-sat-cord-ms
17	□	tea

Selected attribute

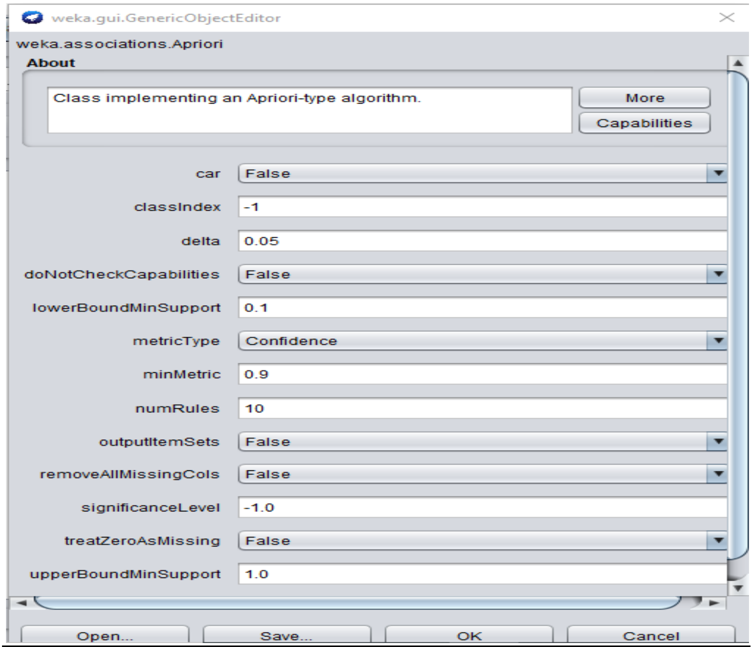
Name: department1
Missing: 3580 (77%)
Distinct: 1
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	t	1047	1047.0

Class: total (Nom)

Relation: supermarket

No.	1: department1 Nominal	2: department2 Nominal	3: department3 Nominal	4: department4 Nominal	5: department5 Nominal	6: department6 Nominal	7: department7 Nominal	8: department8 Nominal	9: department9 Nominal	10: grocery misc Nominal	11: depa Nom
1											
2	t										
3											
4	t										
5											
6			t				t				
7	t										
8											
9	t		t								
10											
11											
12	t										
13	t	t								t	
14											
15											
16	t				t		t				
17											
18	t		t								
19	t									t	
20	t										
21		t			t					t	
22	t	t									
23											
24											
25											
26				t							
27											
28											



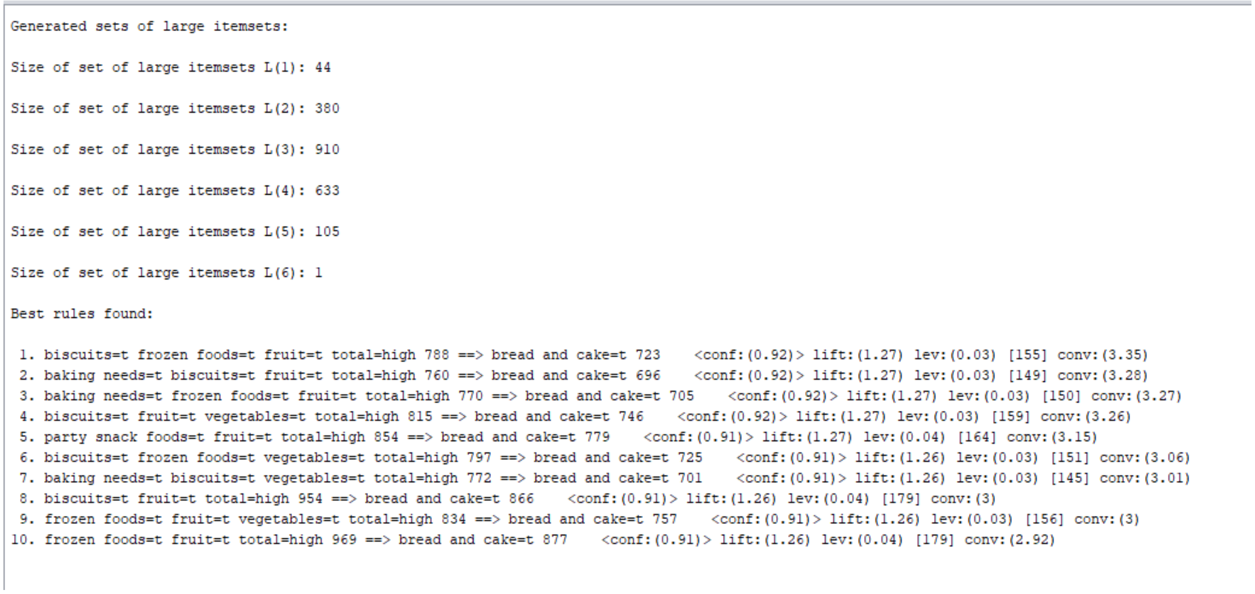
weka.gui.GenericObjectEditor

weka.associations.Apriori

About

Class implementing an Apriori-type algorithm.

```
                    car  False
             classIndex  -1
                  delta  0.05
 doNotCheckCapabilities  False
   lowerBoundMinSupport  0.1
             metricType  Confidence
              minMetric  0.9
               numRules  10
         outputItemSets  False
    removeAllMissingCols  False
      significanceLevel  -1.0
      treatZeroAsMissing  False
   upperBoundMinSupport  1.0
```



```
Generated sets of large itemsets:

Size of set of large itemsets L(1): 44

Size of set of large itemsets L(2): 380

Size of set of large itemsets L(3): 910

Size of set of large itemsets L(4): 633

Size of set of large itemsets L(5): 105

Size of set of large itemsets L(6): 1

Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

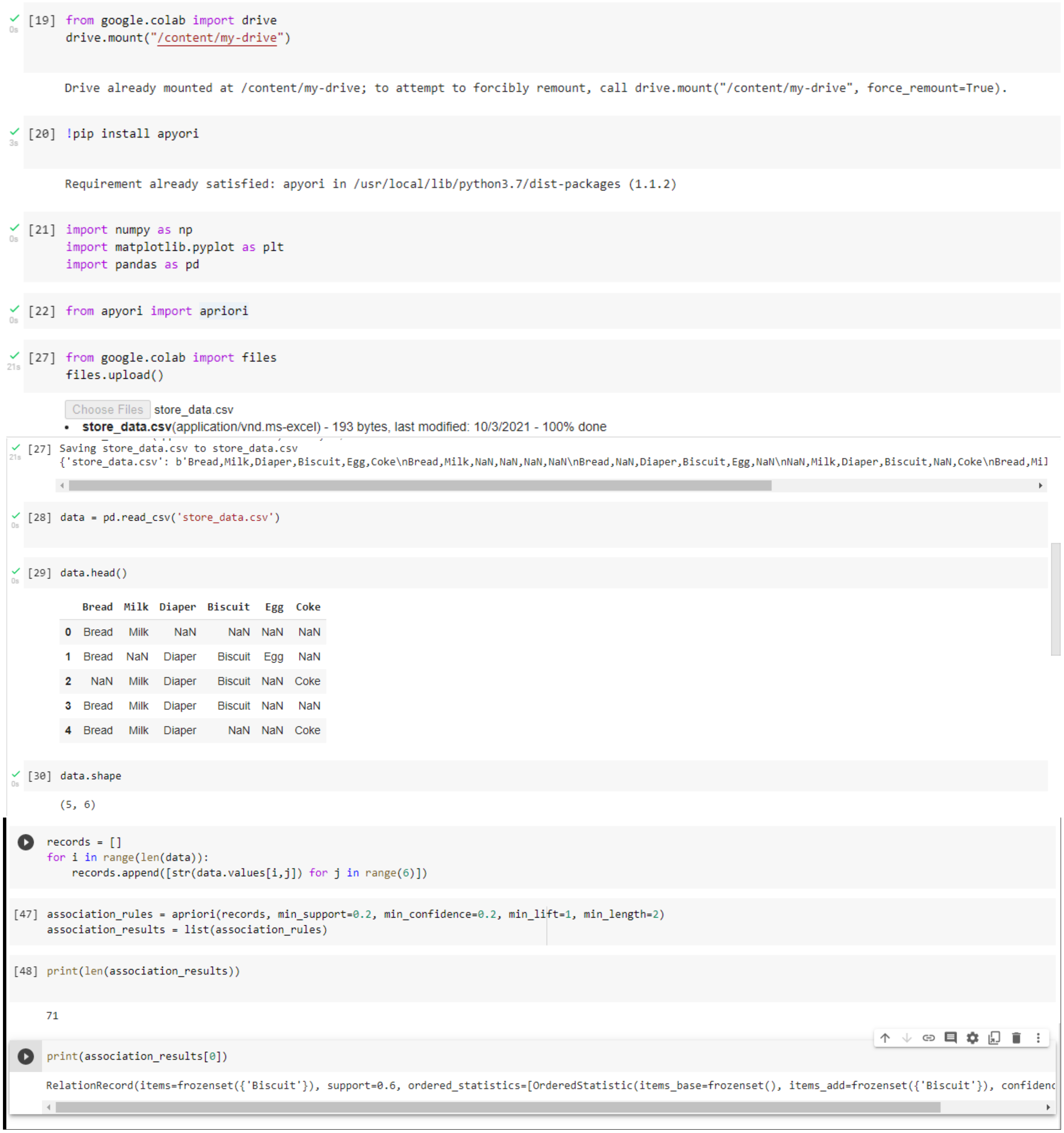
PRACTICAL – 5

Aim: Implement the Apriori algorithm for frequent itemset mining.

Program:

```
from google.colab import drive
drive.mount("/content/my-drive")

!pip install apyori

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from apyori import apriori

from google.colab import files
files.upload()

data = pd.read_csv('store_data.csv')

data.head()

data.shape

records = []
for i in range(len(data)):
    records.append([str(data.values[i,j]) for j in range(6)])

association_rules = apriori(records, min_support=0.2, min_confidence=0.2, min_lift=1, min_len
gth=2)
association_results = list(association_rules)

print(len(association_results))

print(association_results[0])
```

Output:



```
[19] from google.colab import drive
     drive.mount("/content/my-drive")

     Drive already mounted at /content/my-drive; to attempt to forcibly remount, call drive.mount("/content/my-drive", force_remount=True).

[20] !pip install apyori

     Requirement already satisfied: apyori in /usr/local/lib/python3.7/dist-packages (1.1.2)

[21] import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd

[22] from apyori import apriori

[27] from google.colab import files
     files.upload()
```

store_data.csv

- **store_data.csv**(application/vnd.ms-excel) - 193 bytes, last modified: 10/3/2021 - 100% done

```
[27] Saving store_data.csv to store_data.csv
     {'store_data.csv': b'Bread,Milk,Diaper,Biscuit,Egg,Coke\nBread,Milk,NaN,NaN,NaN,NaN\nBread,NaN,Diaper,Biscuit,Egg,NaN\nNaN,Milk,Diaper,Biscuit,NaN,Coke\nBread,Mil

[28] data = pd.read_csv('store_data.csv')

[29] data.head()
```

	Bread	Milk	Diaper	Biscuit	Egg	Coke
0	Bread	Milk	NaN	NaN	NaN	NaN
1	Bread	NaN	Diaper	Biscuit	Egg	NaN
2	NaN	Milk	Diaper	Biscuit	NaN	Coke
3	Bread	Milk	Diaper	Biscuit	NaN	NaN
4	Bread	Milk	Diaper	NaN	NaN	Coke

```
[30] data.shape

     (5, 6)

     records = []
     for i in range(len(data)):
         records.append([str(data.values[i,j]) for j in range(6)])

[47] association_rules = apriori(records, min_support=0.2, min_confidence=0.2, min_lift=1, min_length=2)
     association_results = list(association_rules)

[48] print(len(association_results))

     71

     print(association_results[0])

     RelationRecord(items=frozenset({'Biscuit'}), support=0.6, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Biscuit'}), confidenc
```

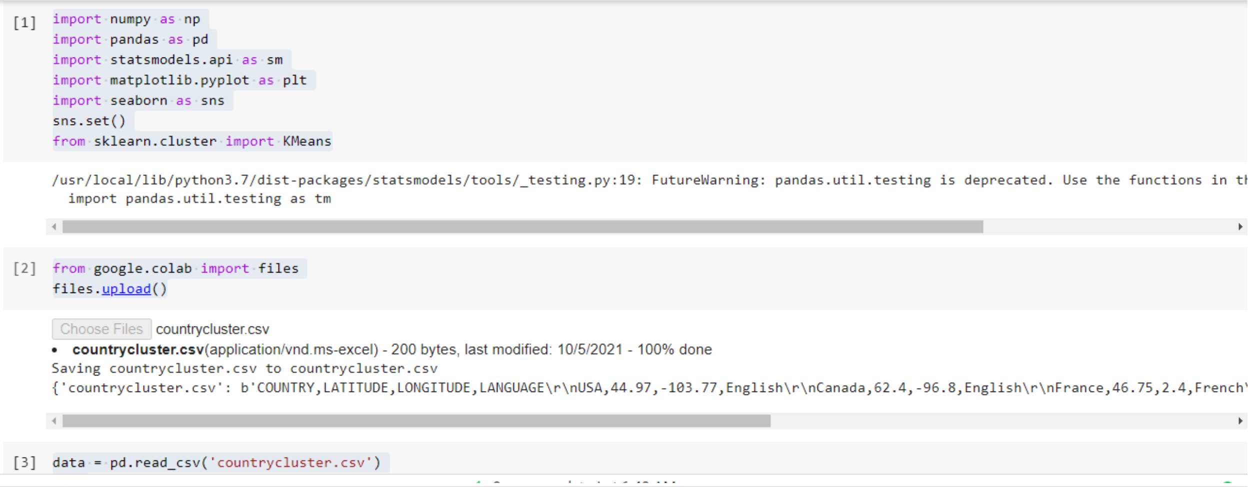
PRACTICAL – 6

Aim: Implement the k-means clustering algorithm.

Program:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans

from google.colab import files
files.upload()

data = pd.read_csv('countrycluster.csv')
data

plt.scatter(data['LONGITUDE'],data['LATITUDE'])
plt.xlim(-180,180)
plt.ylim(-90,90)
plt.show()

x = data.iloc[:,1:3] # 1t for rows and second for columns
x

kmeans = KMeans(3)

identified_clusters = kmeans.fit_predict(x)
identified_clusters

data_with_clusters = data.copy()
data_with_clusters['Clusters'] = identified_clusters
plt.scatter(data_with_clusters['LONGITUDE'],data_with_clusters['LATITUDE'],c=data_with_clusters['Clusters'],cmap='rainbow')
```

Output:



```
[1] import numpy as np
    import pandas as pd
    import statsmodels.api as sm
    import matplotlib.pyplot as plt
    import seaborn as sns
    sns.set()
    from sklearn.cluster import KMeans
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in th
  import pandas.util.testing as tm
```

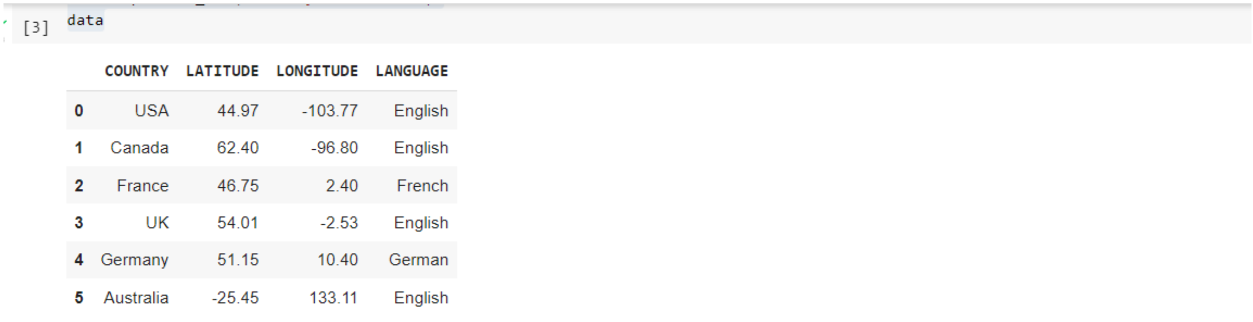
```
[2] from google.colab import files
    files.upload()
```

countrycluster.csv

- **countrycluster.csv**(application/vnd.ms-excel) - 200 bytes, last modified: 10/5/2021 - 100% done

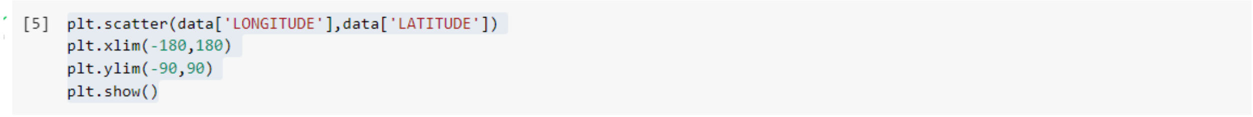
```
Saving countrycluster.csv to countrycluster.csv
{'countrycluster.csv': b'COUNTRY,LATITUDE,LONGITUDE,LANGUAGE\r\nUSA,44.97,-103.77,English\r\nCanada,62.4,-96.8,English\r\nFrance,46.75,2.4,French\
```

```
[3] data = pd.read_csv('countrycluster.csv')
```

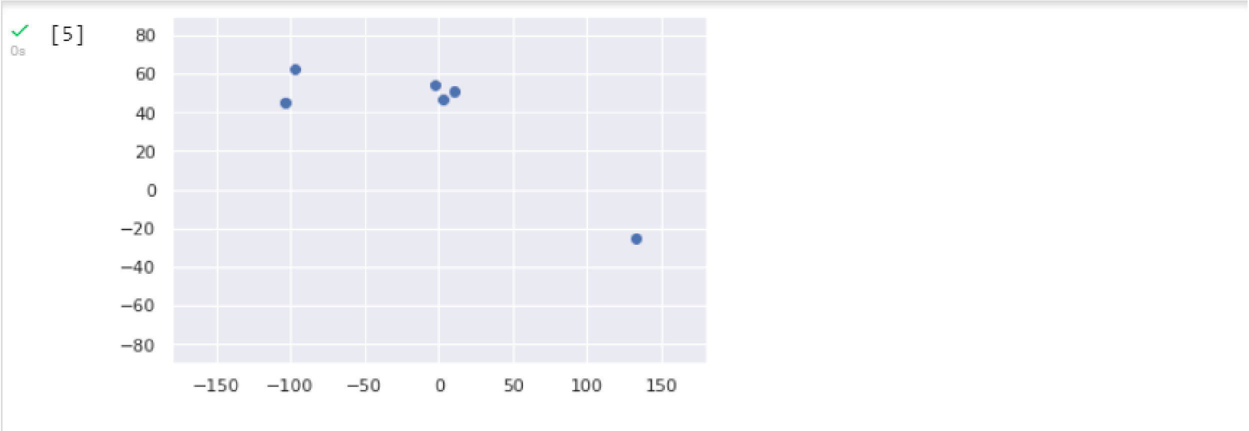


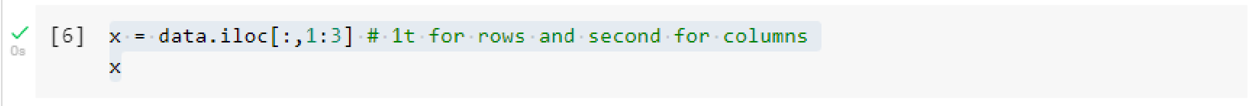
```
[3] data
```

	COUNTRY	LATITUDE	LONGITUDE	LANGUAGE
0	USA	44.97	-103.77	English
1	Canada	62.40	-96.80	English
2	France	46.75	2.40	French
3	UK	54.01	-2.53	English
4	Germany	51.15	10.40	German
5	Australia	-25.45	133.11	English

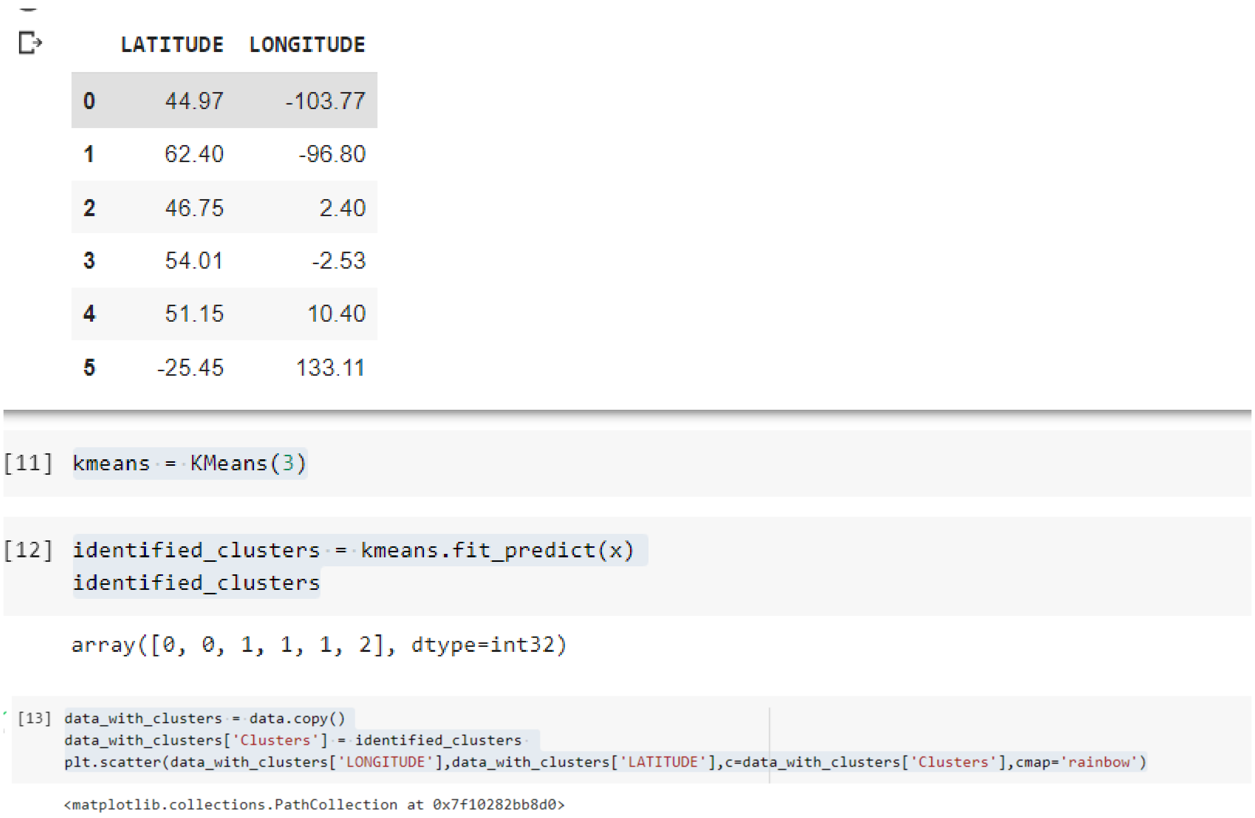


```
[5] plt.scatter(data['LONGITUDE'],data['LATITUDE'])
    plt.xlim(-180,180)
    plt.ylim(-90,90)
    plt.show()
```





```
[6] x = data.iloc[:,1:3] # 1t for rows and second for columns
    x
```



	LATITUDE	LONGITUDE
0	44.97	-103.77
1	62.40	-96.80
2	46.75	2.40
3	54.01	-2.53
4	51.15	10.40
5	-25.45	133.11

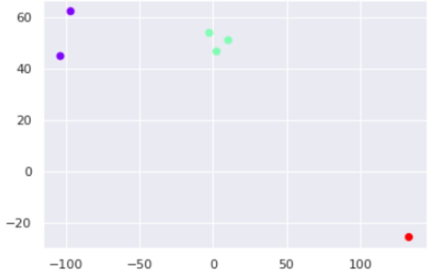
```
[11] kmeans = KMeans(3)
```

```
[12] identified_clusters = kmeans.fit_predict(x)
     identified_clusters
```

```
array([0, 0, 1, 1, 1, 2], dtype=int32)
```

```
[13] data_with_clusters = data.copy()
     data_with_clusters['Clusters'] = identified_clusters
     plt.scatter(data_with_clusters['LONGITUDE'],data_with_clusters['LATITUDE'],c=data_with_clusters['Clusters'],cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7f10282bb8d0>
```



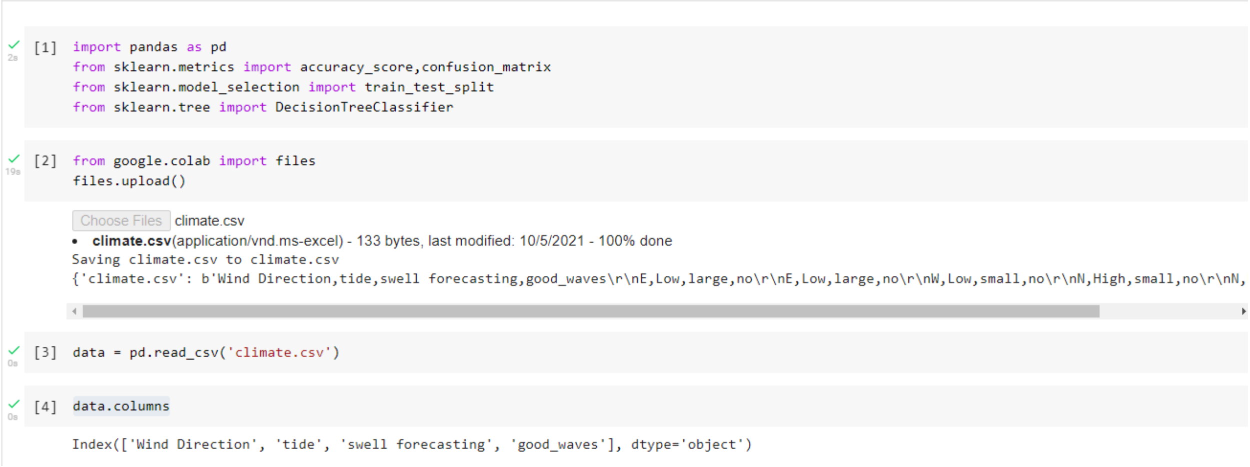
PRACTICAL – 7

Aim: Create an ID3 based classification model for the given dataset.

Program:

```
import pandas as pd
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

from google.colab import files
files.upload()

data = pd.read_csv('climate.csv')
data.head()

data.columns

feature_columns = ['Wind Direction','swell forecasting',]
X = data[feature_columns] # Features
y = data.good_waves # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% trai
ning and 30% test

clf = DecisionTreeClassifier()

clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output:



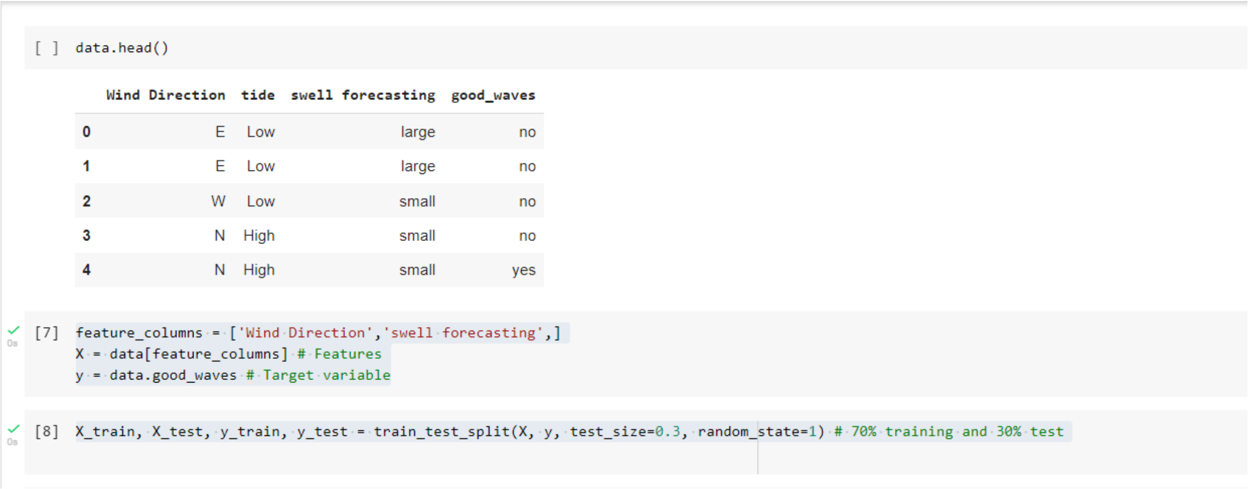
```
[1]  import pandas as pd
     from sklearn.metrics import accuracy_score,confusion_matrix
     from sklearn.model_selection import train_test_split
     from sklearn.tree import DecisionTreeClassifier
```

```
[2]  from google.colab import files
     files.upload()
```

```
Choose Files  climate.csv
• climate.csv(application/vnd.ms-excel) - 133 bytes, last modified: 10/5/2021 - 100% done
Saving climate.csv to climate.csv
{'climate.csv': b'Wind Direction,tide,swell forecasting,good_waves\r\nE,Low,large,no\r\nE,Low,large,no\r\nW,Low,small,no\r\nN,High,small,no\r\nN,
```

```
[3]  data = pd.read_csv('climate.csv')
```

```
[4]  data.columns

     Index(['Wind Direction', 'tide', 'swell forecasting', 'good_waves'], dtype='object')
```

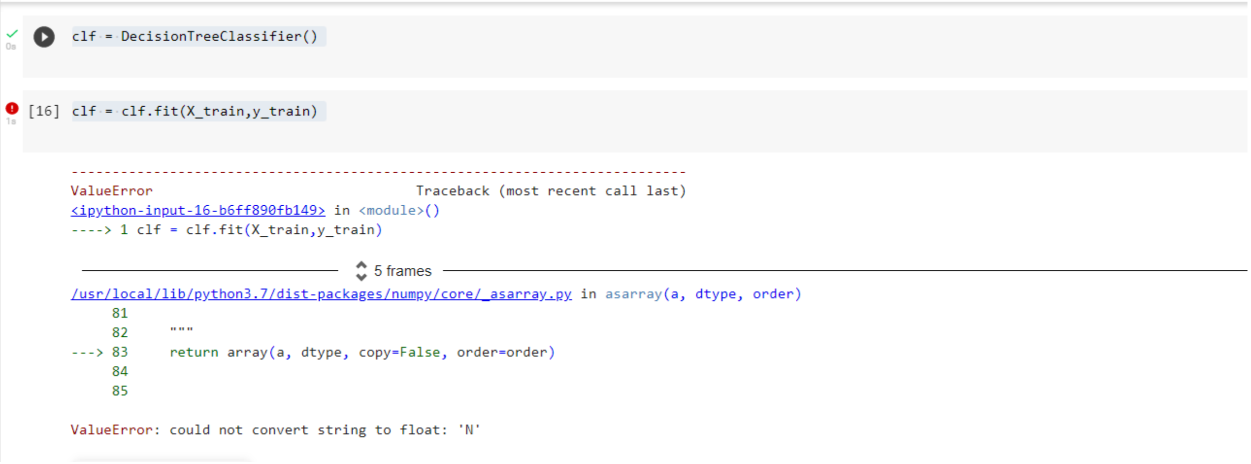


```
[ ]  data.head()
```

	Wind Direction	tide	swell forecasting	good_waves
0	E	Low	large	no
1	E	Low	large	no
2	W	Low	small	no
3	N	High	small	no
4	N	High	small	yes

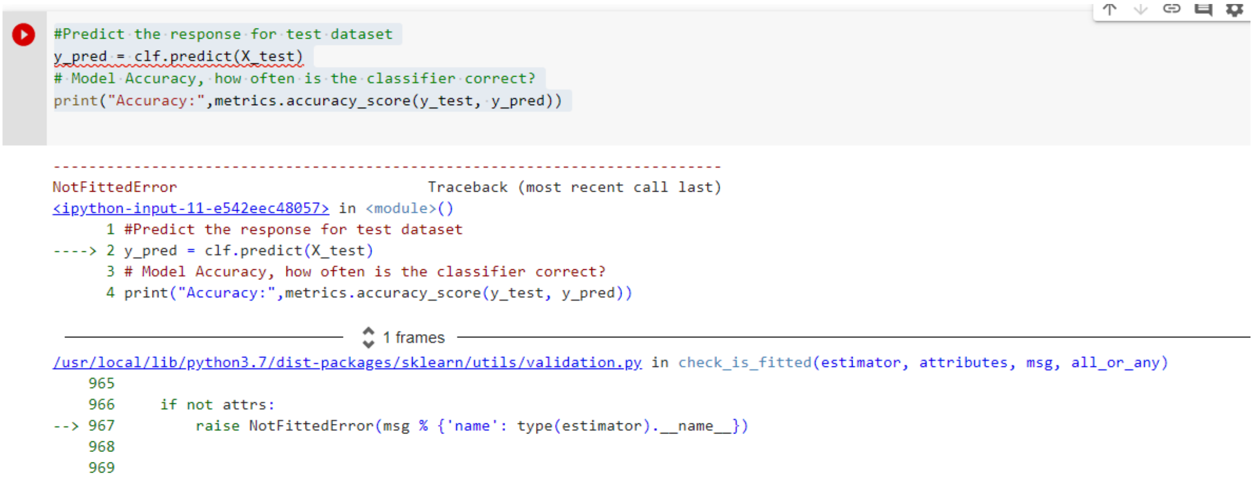
```
[7]  feature_columns = ['Wind Direction','swell forecasting',]
     X = data[feature_columns] # Features
     y = data.good_waves # Target variable
```

```
[8]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
```



```
     clf = DecisionTreeClassifier()
```

```
[16] clf = clf.fit(X_train,y_train)

     ---------------------------------------------------------------------------
     ValueError                                Traceback (most recent call last)
     <ipython-input-16-b6ff890fb149> in <module>()
     ----> 1 clf = clf.fit(X_train,y_train)

      ────────────────────────────── ⇕ 5 frames ──────────────────────────────
     /usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py in asarray(a, dtype, order)
          81
          82     """
     ---> 83     return array(a, dtype, copy=False, order=order)
          84
          85

     ValueError: could not convert string to float: 'N'
```



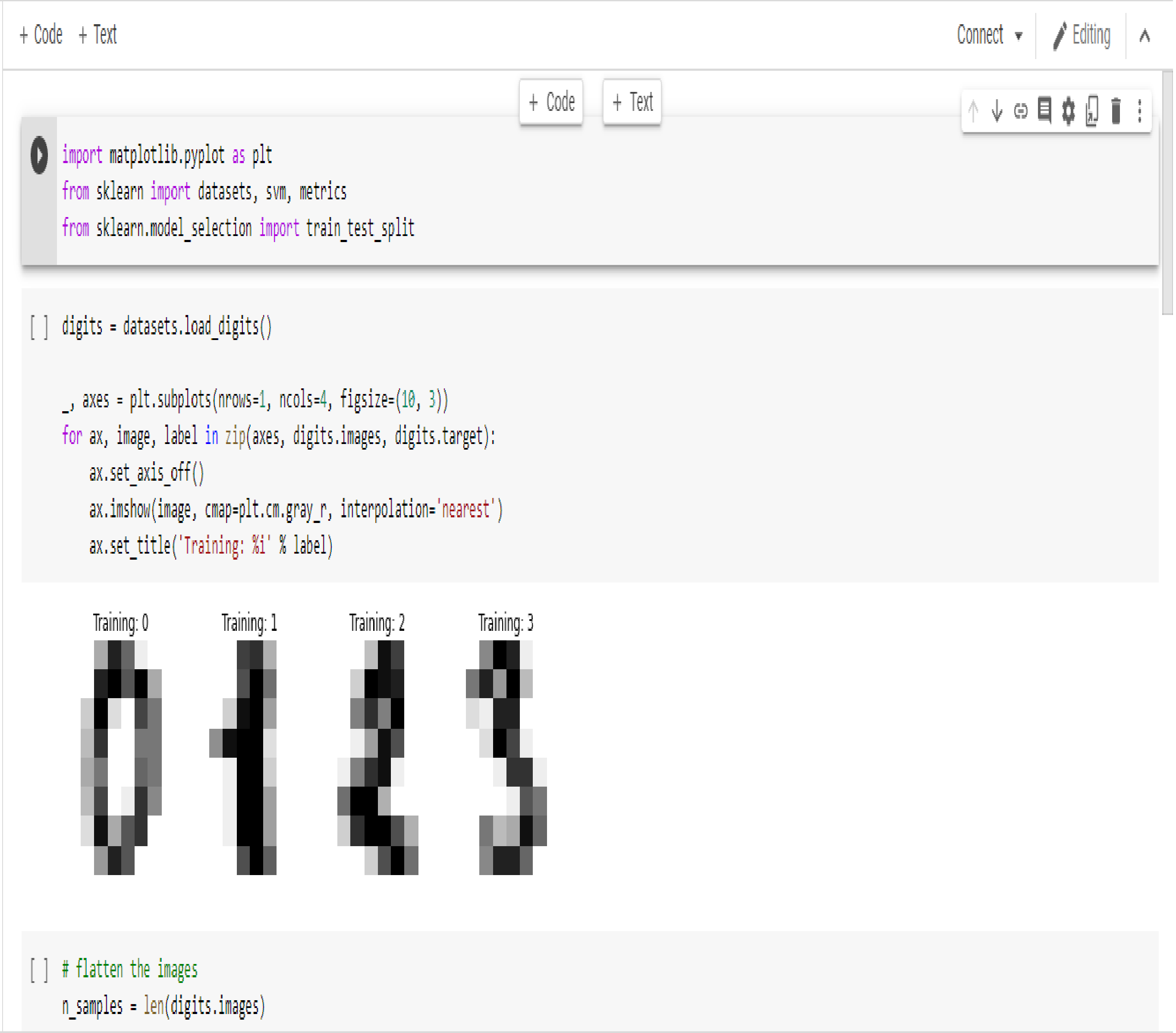
```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
---------------------------------------------------------------------------
NotFittedError                            Traceback (most recent call last)
<ipython-input-11-e542eec48057> in <module>()
      1 #Predict the response for test dataset
----> 2 y_pred = clf.predict(X_test)
      3 # Model Accuracy, how often is the classifier correct?
      4 print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

                                1 frames
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py in check_is_fitted(estimator, attributes, msg, all_or_any)
    965 
    966     if not attrs:
--> 967         raise NotFittedError(msg % {'name': type(estimator).__name__})
    968 
    969 
```

PRACTICAL – 8

Aim: Write a program for classification of handwritten digits using Scikit- learn.

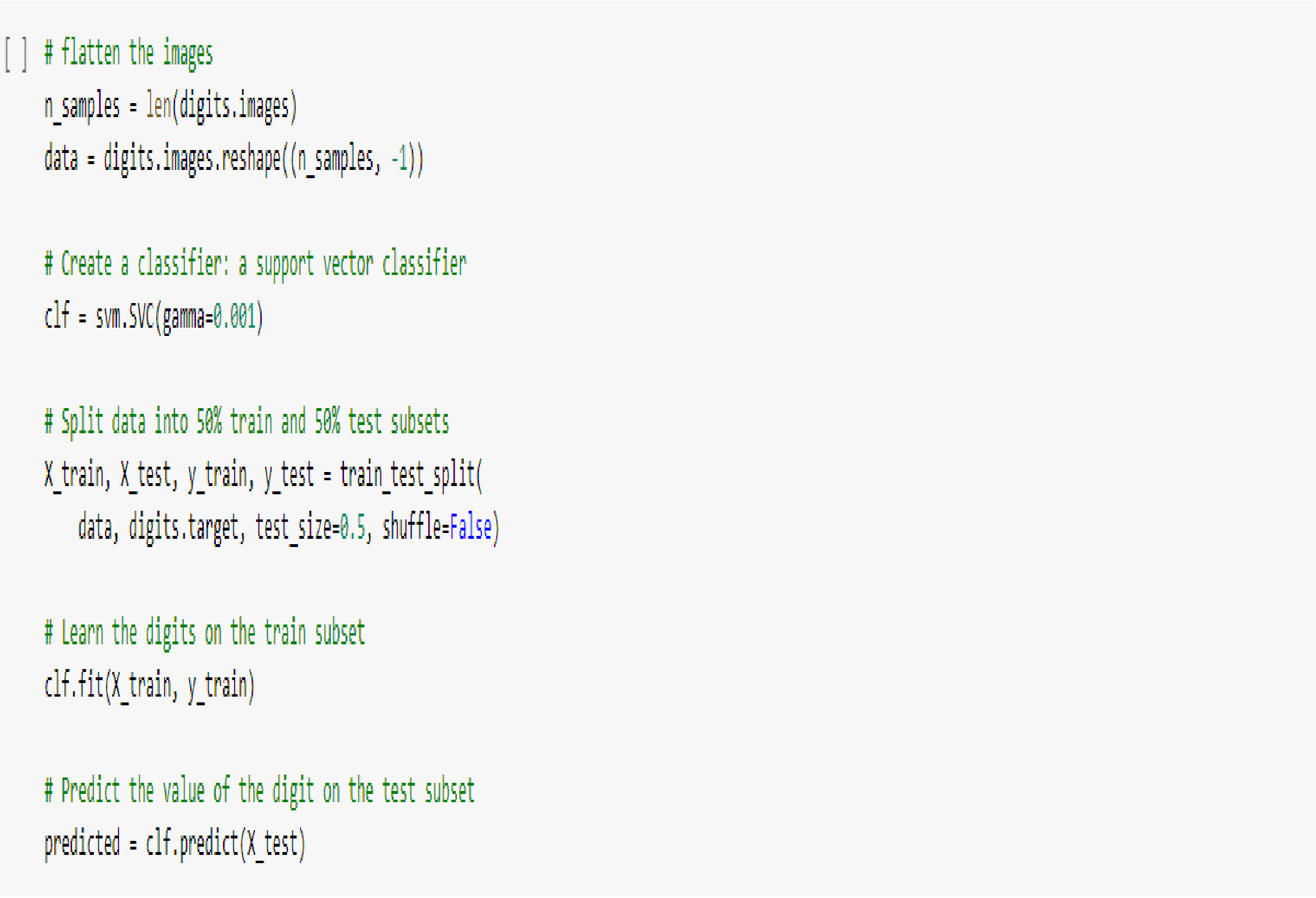
Program:



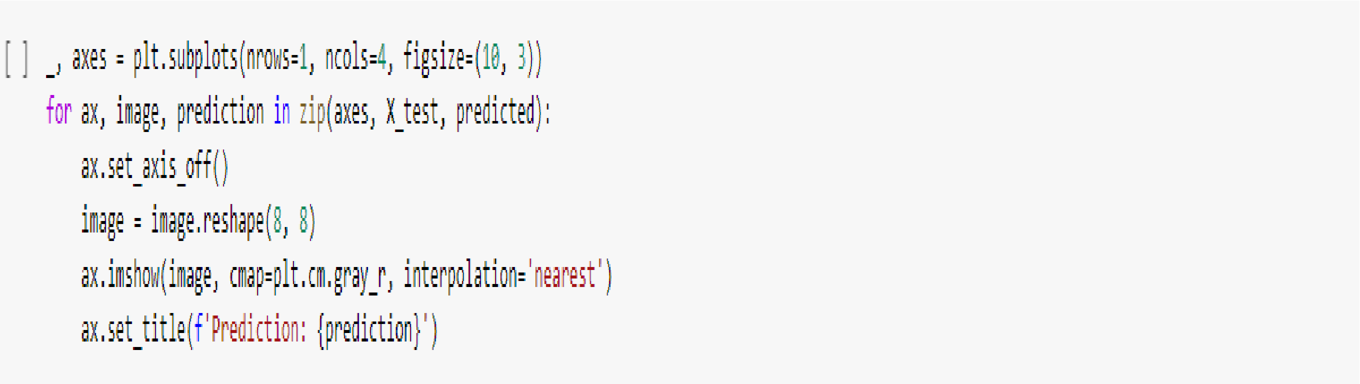
```
import matplotlib.pyplot as plt
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split
```

```
[ ] digits = datasets.load_digits()

    _, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
    for ax, image, label in zip(axes, digits.images, digits.target):
        ax.set_axis_off()
        ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
        ax.set_title('Training: %i' % label)
```

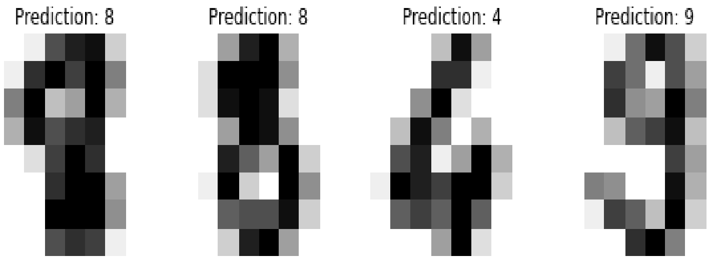
```
[ ] # flatten the images
    n_samples = len(digits.images)
```

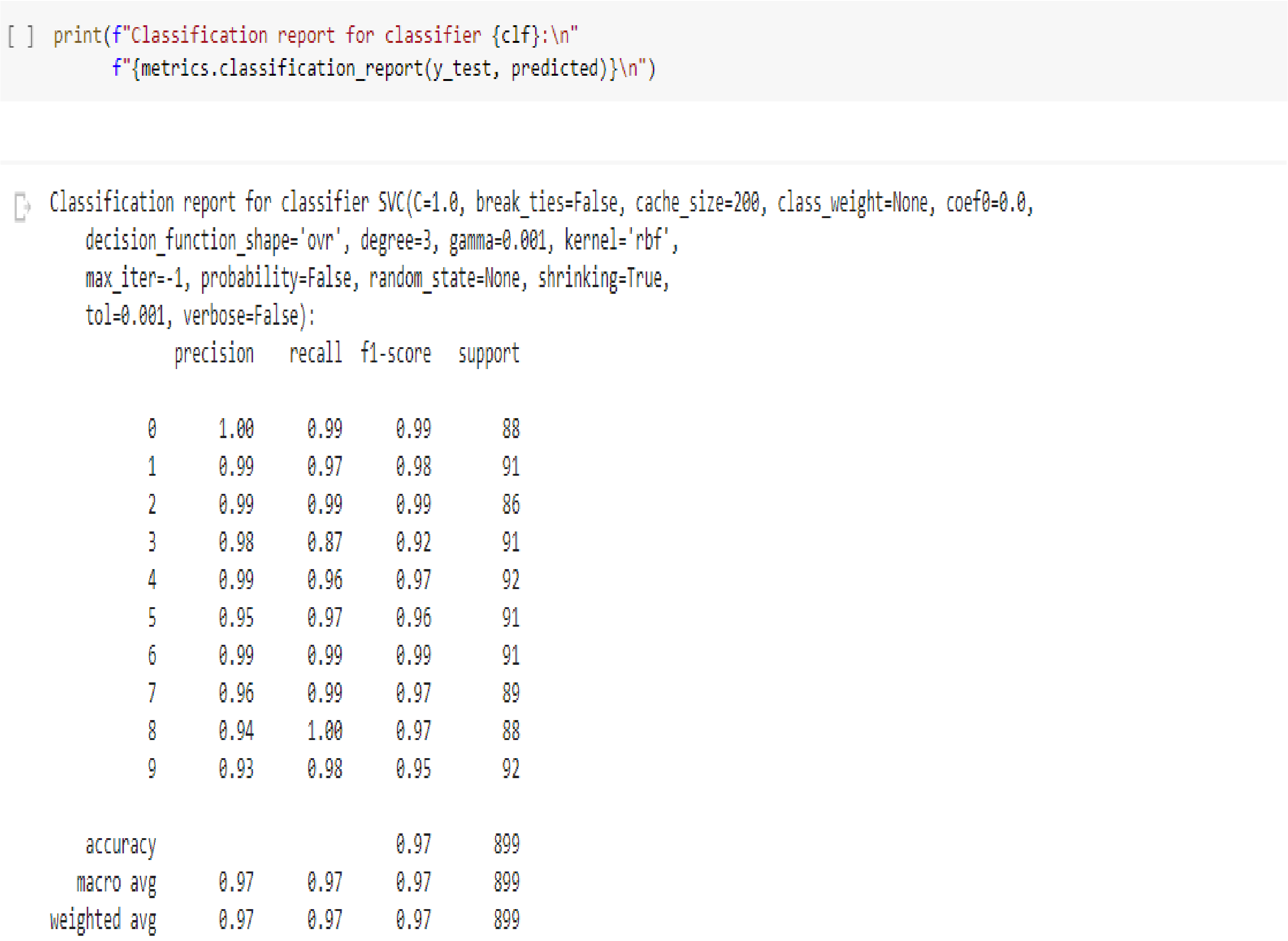


```
[ ] # flatten the images
    n_samples = len(digits.images)
    data = digits.images.reshape((n_samples, -1))

    # Create a classifier: a support vector classifier
    clf = svm.SVC(gamma=0.001)

    # Split data into 50% train and 50% test subsets
    X_train, X_test, y_train, y_test = train_test_split(
        data, digits.target, test_size=0.5, shuffle=False)

    # Learn the digits on the train subset
    clf.fit(X_train, y_train)

    # Predict the value of the digit on the test subset
    predicted = clf.predict(X_test)
```



```
[ ] _, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
    for ax, image, prediction in zip(axes, X_test, predicted):
        ax.set_axis_off()
        image = image.reshape(8, 8)
        ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
        ax.set_title(f'Prediction: {prediction}')
```





```
[ ] print(f"Classification report for classifier {clf}:\n"
          f"{metrics.classification_report(y_test, predicted)}\n")
```

```
Classification report for classifier SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False):
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        88
           1       0.99      0.97      0.98        91
           2       0.99      0.99      0.99        86
           3       0.98      0.87      0.92        91
           4       0.99      0.96      0.97        92
           5       0.95      0.97      0.96        91
           6       0.99      0.99      0.99        91
           7       0.96      0.99      0.97        89
           8       0.94      1.00      0.97        88
           9       0.93      0.98      0.95        92

    accuracy                           0.97       899
   macro avg       0.97      0.97      0.97       899
weighted avg       0.97      0.97      0.97       899
```