```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Use dataset of Breast Cancer patients with Malignant and Benign tumor https://www.kaggle.com/uciml/breast-cancerwisconsin-data. Apply Logistic Regression to predict whether the given patient is having Malignant or Benign tumor based on the attributes in the given dataset.

```
data = pd.read_csv('/content/data.csv')
print (data)
```

```
565    926682         M        20.13         28.25          131.20      1261.0
566    926954         M        16.60         28.08          108.30       858.1
567    927241         M        20.60         29.33          140.10      1265.0
568     92751         B         7.76         24.54           47.92       181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
1            0.08474           0.07864         0.08690              0.07017
2            0.10960           0.15990         0.19740              0.12790
3            0.14250           0.28390         0.24140              0.10520
4            0.10030           0.13280         0.19800              0.10430
..               ...               ...             ...                  ...
564          0.11100           0.11590         0.24390              0.13890
565          0.09780           0.10340         0.14400              0.09791
566          0.08455           0.10230         0.09251              0.05302
567          0.11780           0.27700         0.35140              0.15200
568          0.05263           0.04362         0.00000              0.00000

     ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0    ...          17.33           184.60      2019.0           0.16220
1    ...          23.41           158.80      1956.0           0.12380
2    ...          25.53           152.50      1709.0           0.14440
3    ...          26.50            98.87       567.7           0.20980
4    ...          16.67           152.20      1575.0           0.13740
..   ...            ...              ...         ...               ...
564  ...          26.40           166.10      2027.0           0.14100
565  ...          38.25           155.00      1731.0           0.11660
566  ...          34.12           126.70      1124.0           0.11390
567  ...          39.42           184.60      1821.0           0.16500
568  ...          30.37            59.16       268.6           0.08996

     compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0              0.66560           0.7119                0.2654          0.4601
1              0.18660           0.2416                0.1860          0.2750
2              0.42450           0.4504                0.2430          0.3613
3              0.86630           0.6869                0.2575          0.6638
4              0.20500           0.4000                0.1625          0.2364
..                 ...              ...                   ...             ...
```

```
1                    0.08902         NaN
2                    0.08758         NaN
3                    0.17300         NaN
4                    0.07678         NaN
..                       ...         ...
564                  0.07115         NaN
565                  0.06637         NaN
566                  0.07820         NaN
567                  0.12400         NaN
568                  0.07039         NaN

[569 rows x 33 columns]
```

```
is_null = data.isnull().sum()
print (is_null)
```

```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

```
data.shape
```

```
(569, 33)
```

```
data.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactnes
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.

8 rows × 32 columns

1.Apply Logistic Regression to predict whether the given patient is having Malignant or Benign tumor based on the attributes in the given dataset.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']
print(X)
print(y)
```

```
           id  radius_mean  texture_mean  perimeter_mean  area_mean  \
0      842302        17.99         10.38          122.80     1001.0
1      842517        20.57         17.77          132.90     1326.0
2    84300903        19.69         21.25          130.00     1203.0
3    84348301        11.42         20.38           77.58      386.1
4    84358402        20.29         14.34          135.10     1297.0
..        ...          ...           ...             ...        ...
564    926424        21.56         22.39          142.00     1479.0
565    926682        20.13         28.25          131.20     1261.0
566    926954        16.60         28.08          108.30      858.1
567    927241        20.60         29.33          140.10     1265.0
568     92751         7.76         24.54           47.92      181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
1            0.08474           0.07864         0.08690              0.07017
2            0.10960           0.15990         0.19740              0.12790
3            0.14250           0.28390         0.24140              0.10520
4            0.10030           0.13280         0.19800              0.10430
..               ...               ...             ...                  ...
564          0.11100           0.11590         0.24390              0.13890
565          0.09780           0.10340         0.14400              0.09791
566          0.08455           0.10230         0.09251              0.05302
567          0.11780           0.27700         0.35140              0.15200
568          0.05263           0.04362         0.00000              0.00000

     symmetry_mean  ...  texture_worst  perimeter_worst  area_worst  \
```

```
0          0.2419  ...          17.33          184.60      2019.0
1          0.1812  ...          23.41          158.80      1956.0
2          0.2069  ...          25.53          152.50      1709.0
3          0.2597  ...          26.50           98.87       567.7
4          0.1809  ...          16.67          152.20      1575.0
..            ...  ...            ...             ...         ...
564        0.1726  ...          26.40          166.10      2027.0
565        0.1752  ...          38.25          155.00      1731.0
566        0.1590  ...          34.12          126.70      1124.0
567        0.2397  ...          39.42          184.60      1821.0
568        0.1587  ...          30.37           59.16       268.6

     smoothness_worst  compactness_worst  concavity_worst  \
0             0.16220            0.66560           0.7119
1             0.12380            0.18660           0.2416
2             0.14440            0.42450           0.4504
3             0.20980            0.86630           0.6869
4             0.13740            0.20500           0.4000
..                ...                ...              ...
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     concave points_worst  symmetry_worst  fractal_dimension_worst  \
0                  0.2654          0.4601                  0.11890
1                  0.1860          0.2750                  0.08902
2                  0.2430          0.3613                  0.08758
3                  0.2575          0.6638                  0.17300
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
print(X_train)
print(X_test)
print(y_train)
print(y_test)
```

```
89          NaN
199         NaN
411         NaN
18          NaN
390         NaN

[143 rows x 32 columns]
287    B
512    M
402    B
446    M
210    M
      ..
71     B
106    B
270    B
435    M
102    B
Name: diagnosis, Length: 426, dtype: object
204    B
70     M
131    M
431    B
540    B
      ..
89     B
199    M
411    B
18     M
390    B
Name: diagnosis, Length: 143, dtype: object
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
```

```
imputer = SimpleImputer(strategy='mean')
print(imputer)
```

```
SimpleImputer()
```

```
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)
print(X_train)
print(X_test)
```

```
[[8.913000e+03 1.289000e+01 1.312000e+01 ... 5.366000e-02 2.309000e-01
  6.915000e-02]
 [9.156910e+05 1.340000e+01 2.052000e+01 ... 2.051000e-01 3.585000e-01
  1.109000e-01]
 [9.046890e+05 1.296000e+01 1.829000e+01 ... 6.608000e-02 3.207000e-01
  7.247000e-02]
 ...
 [8.910721e+06 1.429000e+01 1.682000e+01 ... 3.333000e-02 2.458000e-01
  6.120000e-02]
 [9.084890e+05 1.398000e+01 1.962000e+01 ... 1.827000e-01 3.179000e-01
  1.055000e-01]
 [8.629650e+05 1.218000e+01 2.052000e+01 ... 7.431000e-02 2.694000e-01
```

```
  6.878000e-02]]
[[8.7930000e+04 1.2470000e+01 1.8600000e+01 ... 1.0150000e-01
  3.0140000e-01 8.7500000e-02]
 [8.5957500e+05 1.8940000e+01 2.1310000e+01 ... 1.7890000e-01
  2.5510000e-01 6.5890000e-02]
 [8.6700000e+03 1.5460000e+01 1.9480000e+01 ... 1.5140000e-01
  2.8370000e-01 8.0190000e-02]
 ...
 [9.0552000e+05 1.1040000e+01 1.6830000e+01 ... 7.4310000e-02
  2.9980000e-01 7.8810000e-02]
 [8.4901400e+05 1.9810000e+01 2.2150000e+01 ... 2.3880000e-01
  2.7680000e-01 7.6150000e-02]
 [9.0317302e+07 1.0260000e+01 1.2220000e+01 ... 6.6960000e-02
  2.9370000e-01 7.7220000e-02]]
/usr/local/lib/python3.10/dist-packages/sklearn/impute/_base.py:598: UserWarning: Skipping features wit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/impute/_base.py:598: UserWarning: Skipping features wit
  warnings.warn(
```

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
print(model)
```

```
LogisticRegression(max_iter=1000)
```

```
y_pred = model.predict(X_test)
print(y_pred)
```

```
['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M'
 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B']
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.951048951048951
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```

Classification Report:
              precision    recall  f1-score   support

           B       0.95      0.98      0.96        89
           M       0.96      0.91      0.93        54

    accuracy                           0.95       143
   macro avg       0.95      0.94      0.95       143
weighted avg       0.95      0.95      0.95       143
```

```
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[87  2]
 [ 5 49]]
```

2. Implement a Classifier using Random Forest Classifier for the pima Indians dataset. Evaluate the performance of the classifier using Accuracy Score, Confusion Matrix, Precision & Recall.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)  # You can adjust n_estimators
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
print(y_pred_rf)
```

```
['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M'
 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B']
```

```
accuracy_rf = accuracy_score(y_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf, average='weighted')
recall_rf = recall_score(y_test, y_pred_rf, average='weighted')

print(f"Random Forest Accuracy: {accuracy_rf}")
print(f"Random Forest Confusion Matrix:\n{conf_matrix_rf}")
print(f"Random Forest Precision: {precision_rf}")
print(f"Random Forest Recall: {recall_rf}")
```

```
Random Forest Accuracy: 0.972027972027972
Random Forest Confusion Matrix:
[[88  1]
 [ 3 51]]
Random Forest Precision: 0.972220087604703
Random Forest Recall: 0.972027972027972
```

3. Select a random dataset suitable for classification, and develop an ML model using Decision Tree Classifier. Evaluate the performance of the classifier using Accuracy Score, Confusion Matrix, Precision & Recall.

```
from sklearn.tree import DecisionTreeClassifier


dt_model = DecisionTreeClassifier(random_state=42)  # You can adjust parameters
dt_model.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier ⓘ ?
DecisionTreeClassifier(random_state=42)
```

```
y_pred_dt = dt_model.predict(X_test)
print(y_pred_dt)
```

```
['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'M'
 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'M' 'B']
```

```
accuracy_dt = accuracy_score(y_test, y_pred_dt)
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt, average='weighted')
recall_dt = recall_score(y_test, y_pred_dt, average='weighted')

print(f"Decision Tree Accuracy: {accuracy_dt}")
print(f"Decision Tree Confusion Matrix:\n{conf_matrix_dt}")
print(f"Decision Tree Precision: {precision_dt}")
print(f"Decision Tree Recall: {recall_dt}")
```

```
Decision Tree Accuracy: 0.951048951048951
Decision Tree Confusion Matrix:
[[84  5]
 [ 2 52]]
Decision Tree Precision: 0.9524013318382963
Decision Tree Recall: 0.951048951048951
```

4. Develop an SVM Classifier (SVC) for the Wine recognition dataset in sklearn. Evaluate the performance of the classifier using Accuracy Score, Confusion Matrix, Precision & Recall.

```
from sklearn.svm import SVC
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
```

```
wine = load_wine()
X = wine.data
y = wine.target
print(X)
print(y)
```

```
[[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]]
```

```
[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]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
print(X_train)
print(X_test)
print(y_train)
print(y_test)
```

```
[[1.316e+01 2.360e+00 2.670e+00 ... 1.030e+00 3.170e+00 1.185e+03]
 [1.208e+01 2.080e+00 1.700e+00 ... 1.270e+00 2.960e+00 7.100e+02]
 [1.242e+01 4.430e+00 2.730e+00 ... 9.200e-01 3.120e+00 3.650e+02]
 ...
 [1.438e+01 1.870e+00 2.380e+00 ... 1.200e+00 3.000e+00 1.547e+03]
 [1.269e+01 1.530e+00 2.260e+00 ... 9.600e-01 2.060e+00 4.950e+02]
 [1.234e+01 2.450e+00 2.460e+00 ... 8.000e-01 3.380e+00 4.380e+02]]
[[1.364000e+01 3.100000e+00 2.560000e+00 1.520000e+01 1.160000e+02
  2.700000e+00 3.030000e+00 1.700000e-01 1.660000e+00 5.100000e+00
  9.600000e-01 3.360000e+00 8.450000e+02]
 [1.421000e+01 4.040000e+00 2.440000e+00 1.890000e+01 1.110000e+02
  2.850000e+00 2.650000e+00 3.000000e-01 1.250000e+00 5.240000e+00
  8.700000e-01 3.330000e+00 1.080000e+03]
 [1.293000e+01 2.810000e+00 2.700000e+00 2.100000e+01 9.600000e+01
  1.540000e+00 5.000000e-01 5.300000e-01 7.500000e-01 4.600000e+00
  7.700000e-01 2.310000e+00 6.000000e+02]
 [1.373000e+01 1.500000e+00 2.700000e+00 2.250000e+01 1.010000e+02
  3.000000e+00 3.250000e+00 2.900000e-01 2.380000e+00 5.700000e+00
  1.190000e+00 2.710000e+00 1.285000e+03]
 [1.237000e+01 1.170000e+00 1.920000e+00 1.960000e+01 7.800000e+01
  2.110000e+00 2.000000e+00 2.700000e-01 1.040000e+00 4.680000e+00
  1.120000e+00 3.480000e+00 5.100000e+02]
 [1.430000e+01 1.920000e+00 2.720000e+00 2.000000e+01 1.200000e+02
  2.800000e+00 3.140000e+00 3.300000e-01 1.970000e+00 6.200000e+00
  1.070000e+00 2.650000e+00 1.280000e+03]
 [1.200000e+01 3.430000e+00 2.000000e+00 1.900000e+01 8.700000e+01
  2.000000e+00 1.640000e+00 3.700000e-01 1.870000e+00 1.280000e+00
  9.300000e-01 3.050000e+00 5.640000e+02]
 [1.340000e+01 3.910000e+00 2.480000e+00 2.300000e+01 1.020000e+02
  1.800000e+00 7.500000e-01 4.300000e-01 1.410000e+00 7.300000e+00
  7.000000e-01 1.560000e+00 7.500000e+02]
 [1.161000e+01 1.350000e+00 2.700000e+00 2.000000e+01 9.400000e+01
  2.740000e+00 2.920000e+00 2.900000e-01 2.490000e+00 2.650000e+00
  9.600000e-01 3.260000e+00 6.800000e+02]
 [1.336000e+01 2.560000e+00 2.350000e+00 2.000000e+01 8.900000e+01
  1.400000e+00 5.000000e-01 3.700000e-01 6.400000e-01 5.600000e+00
  7.000000e-01 2.470000e+00 7.800000e+02]
 [1.350000e+01 1.810000e+00 2.610000e+00 2.000000e+01 9.600000e+01
  2.530000e+00 2.610000e+00 2.800000e-01 1.660000e+00 3.520000e+00
  1.120000e+00 3.820000e+00 8.450000e+02]
 [1.350000e+01 3.120000e+00 2.620000e+00 2.400000e+01 1.230000e+02
  1.400000e+00 1.570000e+00 2.200000e-01 1.250000e+00 8.600000e+00
  5.900000e-01 1.300000e+00 5.000000e+02]
 [1.341000e+01 3.840000e+00 2.120000e+00 1.880000e+01 9.000000e+01
  2.450000e+00 2.680000e+00 2.700000e-01 1.480000e+00 4.280000e+00
  9.100000e-01 3.000000e+00 1.035000e+03]
 [1.277000e+01 3.430000e+00 1.980000e+00 1.600000e+01 8.000000e+01
  1.630000e+00 1.250000e+00 4.300000e-01 8.300000e-01 3.400000e+00
  7.000000e-01 2.120000e+00 3.720000e+02]
 [1.363000e+01 1.810000e+00 2.700000e+00 1.720000e+01 1.120000e+02
  2.850000e+00 2.910000e+00 3.000000e-01 1.460000e+00 7.300000e+00
  1.280000e+00 2.880000e+00 1.310000e+03]
```

```
 [1.252000e+01 2.430000e+00 2.170000e+00 2.100000e+01 8.800000e+01
  2.550000e+00 2.270000e+00 2.600000e-01 1.220000e+00 2.000000e+00
  9.000000e-01 2.780000e+00 3.250000e+02]
 [1.141000e+01 7.400000e-01 2.500000e+00 2.100000e+01 8.800000e+01
  2.480000e+00 2.010000e+00 4.200000e-01 1.440000e+00 3.080000e+00
  1.100000e+00 2.310000e+00 4.340000e+02]
```

```
svm_model = SVC(kernel='linear', random_state=42)  # You can experiment with different kernels
svm_model.fit(X_train, y_train)
print(svm_model)
```

```
SVC(kernel='linear', random_state=42)
```

```
y_pred_svm = svm_model.predict(X_test)
print(y_pred_svm)
```

```
[0 0 2 0 1 0 1 2 1 2 0 2 0 1 0 1 1 1 0 1 0 1 1 2 2 2 1 1 1 0 0 1 2 0 0 0 2
 2 1 2 0 1 1 2 2]
```

```
accuracy_svm = accuracy_score(y_test, y_pred_svm)
conf_matrix_svm = confusion_matrix(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm, average='weighted')
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')

print(f"SVM Accuracy: {accuracy_svm}")
print(f"SVM Confusion Matrix:\n{conf_matrix_svm}")
print(f"SVM Precision: {precision_svm}")
print(f"SVM Recall: {recall_svm}")
```

```
SVM Accuracy: 0.9777777777777777
SVM Confusion Matrix:
[[15  0  0]
 [ 0 17  1]
 [ 0  0 12]]
SVM Precision: 0.9794871794871796
SVM Recall: 0.9777777777777777
```