In [1]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
```

In [2]:

```
#print short Description
data = load_breast_cancer()
print(data.DESCR[:760])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline appr
```

In [3]:

```
print(f"Types of cancer (targets) are {data.target_names}")
```

```
Types of cancer (targets) are ['malignant' 'benign']
```

In [4]:

```
X = data.data  # features
y = data.target  # labels
print(f"Shape of features is {X.shape}, and shape of target is {y.shape}")
print("Targets are: ", y)
```

```
Shape of features is (569, 30), and shape of target is (569,)
Targets are:  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 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 0 0 0 0 0 0 1]
```

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=200, random_state=42, stratify=y)
y_train[:10]
```

Out[5]:

```
array([1, 1, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [6]:

```
classifier = svm.SVC(kernel='linear', probability=True, verbose=True)
```

In [7]:

```
classifier.fit(X_train, y_train)
```

```
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
```

In [8]:

```
y_preds = classifier.predict(X_test)
y_proba = classifier.predict_proba(X_test)
print("Prediction as class - malignanat or benign ", y_preds[:5])
print("Test Data are: ", y_test[:5])
print("Probability of malignant & probability of benign. ", y_proba[:5])
```

```
Prediction as class - malignanat or benign  [1 0 1 1 1]
Test Data are:  [1 0 1 1 1]
Probability of malignant & probability of benign.  [[6.43922108e-06 9.99993561e-01]
 [9.45265463e-01 5.47345371e-02]
 [1.62229852e-06 9.99998378e-01]
 [2.20009919e-02 9.77999008e-01]
 [1.31519564e-07 9.99999868e-01]]
```

In [9]:

```
y_proba = y_proba[:,1].reshape((y_proba.shape[0],))
print("2D to 1D reshaped Probability of benign. ", y_proba[:5])
```

```
2D to 1D reshaped Probability of benign.  [0.99999356 0.05473454 0.99999838 0.97799901 0.99999987]
```

In [14]:

```
TN, FP, FN, TP = metrics.confusion_matrix(list(y_test), list(y_preds), labels=[0, 1]).ravel() #0,1 is default label of sklearn
print("True Negatives", TN)
print("True Positives", TP)
print("False Positives", FP)
print("False Negatives", FN)

sklearnconf = metrics.confusion_matrix(y_test, y_preds)
print("\nsklearn Confusion Matrix is \n", sklearnconf)

conf=metrics.confusion_matrix(y_test, y_preds,labels=[1,0]) #Note to change the labels from the default 0,1 to 1,0
print("Confusion Matrix we want is: \n", conf)
```

```
True Negatives 68
True Positives 123
False Positives 7
False Negatives 2

sklearn Confusion Matrix is
 [[ 68   7]
 [  2 123]]
Confusion Matrix we want is:
 [[123   2]
 [  7  68]]
```

In [17]:

```
results={} #To Store all the metrics Result Values
```

In [18]:

```
#Finding Accuracy
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}") #Note the Formatting with f"{metric} and rounding to 3 decimal points with .3f
```

```
ACC is  0.955
```

In [19]:

```
#Finding True Negative Rate
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.907
```

In [20]:

```
#Finding Precision Predictive Value
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.946
```

In [27]:

```
#Finding Negative Predictive Value
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.971
```

In [25]:

```
#Finding True Postive Rate
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.984
```

In [26]:

```
#Finding F1-Score
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.965
```

In [28]:

```
#Finding MCC Value
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.904
```

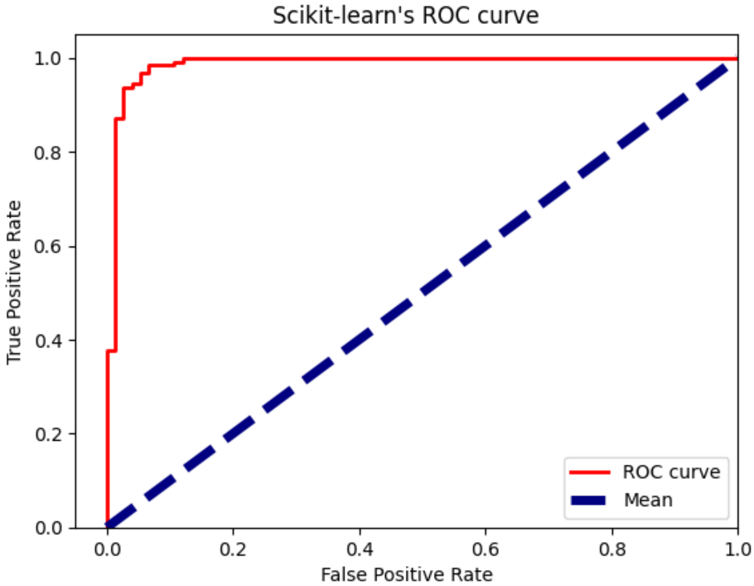
In [30]:

```
ring Our Values to Scikit-Learning Values
f"Calculated and scikit-learn Accuracy:                       {results['ACC']: .3f}, {metrics.accuracy_score(y_test, y_preds
f"Calculated and scikit-learn Precision score:                {results['PPV']: .3f}, {metrics.precision_score(y_test, y_preds
f"Calculated and scikit-learn Recall score:                   {results['TPR']: .3f}, {metrics.recall_score(y_test, y_preds):
f"Calculated and scikit-learn F1 score:                       {results['F1']: .3f}, {metrics.f1_score(y_test, y_preds): .3f}
f"Calculated and scikit-learn Matthew's correlation coefficient: {results['MCC']: .3f}, {metrics.matthews_corrcoef(y_test, y_pre
```

```
Calculated and scikit-learn Accuracy:                          0.955,  0.955
Calculated and scikit-learn Precision score:                   0.946,  0.946
Calculated and scikit-learn Recall score:                      0.984,  0.984
Calculated and scikit-learn F1 score:                          0.965,  0.965
Calculated and scikit-learn Matthew's correlation coefficient: 0.904,  0.904
```

In [35]:

```
#Plotting AUC and ROC Curve
FPRs, TPRs, _ = metrics.roc_curve(y_test, y_proba)

# Plot the ROC curve
plt.plot(FPRs, TPRs, color='red', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label="Mean")  #lw  = linewidth
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("Scikit-learn's ROC curve")
plt.legend(loc="lower right")
plt.show()
```



In [36]:

```
auc_score = metrics.roc_auc_score(y_test, y_proba)
print(f"Scikit's ROC-AUC score of SVC model is {auc_score: .4f}")
```

```
Scikit's ROC-AUC score of SVC model is  0.9872
```

In [ ]:

In [2]:
```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd
```

In [3]:
```
df = pd.read_csv('iris.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	PetalLength	SepalWidth	SepalLength	PetalWidth	Species
0	1.4	3.5	5.1	0.2	Iris-setosa
1	1.4	3.0	4.9	0.2	Iris-setosa
2	1.3	3.2	4.7	0.2	Iris-setosa
3	1.5	3.1	4.6	0.2	Iris-setosa
4	1.4	3.6	5.0	0.2	Iris-setosa

In [6]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df = df.apply(encoder.fit_transform)
df
```

Out[6]:

	PetalLength	SepalWidth	SepalLength	PetalWidth	Species
0	4	14	8	1	0
1	4	9	6	1	0
2	3	11	4	1	0
3	5	10	3	1	0
4	4	15	7	1	0
...	...	...	...	...	...
145	28	9	24	19	2
146	26	4	20	15	2
147	28	9	22	16	2
148	30	13	19	19	2
149	27	9	16	14	2

150 rows × 5 columns

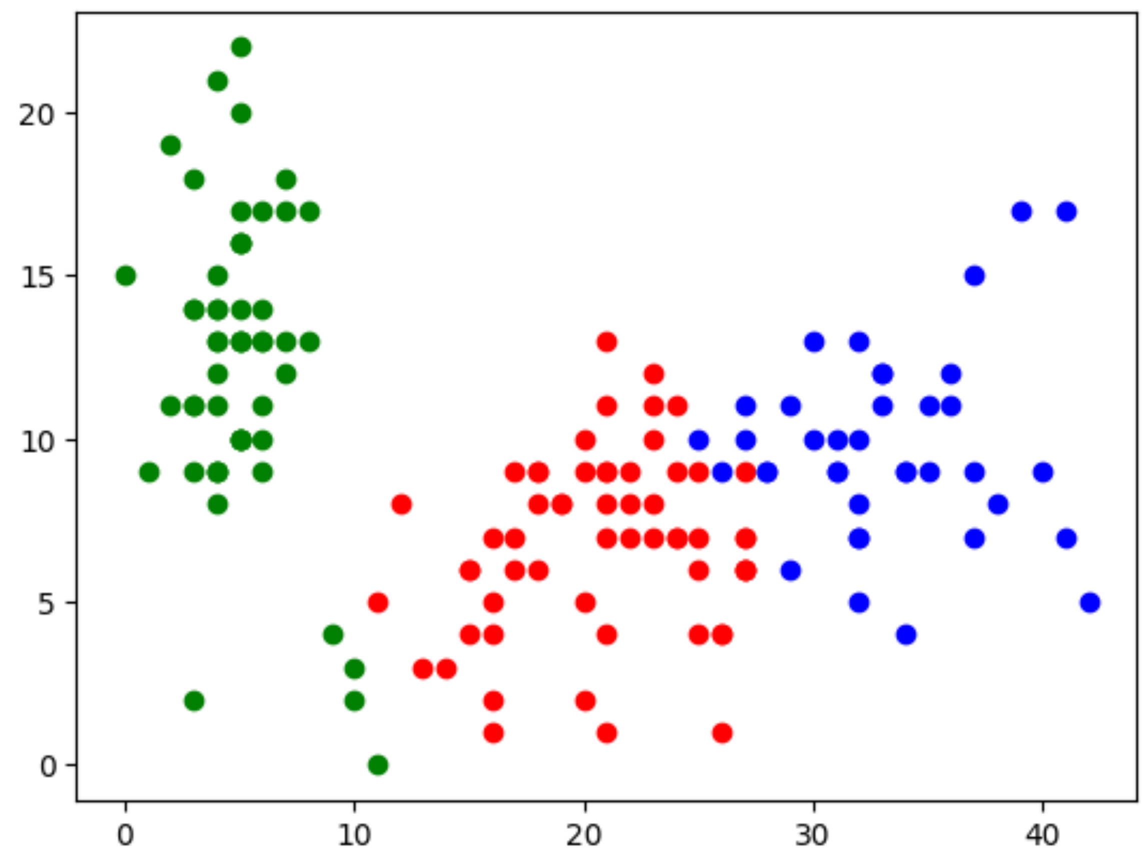
In [7]:
```
X = df.iloc[:,:].values
print(X[:5])
```

```
[[ 4 14  8  1  0]
 [ 4  9  6  1  0]
 [ 3 11  4  1  0]
 [ 5 10  3  1  0]
 [ 4 15  7  1  0]]
```

In [38]:

```
import random
import numpy as np

class KMeans1:
    def __init__(self,n_clusters=3,max_iter=100):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.centroids = None

    def fit_predict(self,X):

        random_index = random.sample(range(0,X.shape[0]),self.n_clu
        self.centroids = X[random_index]

        for i in range(self.max_iter):
            # assign clusters
            cluster_group = self.assign_clusters(X)
            old_centroids = self.centroids
            # move centroids
            self.centroids = self.move_centroids(X,cluster_group)
            # check finish
            if (old_centroids == self.centroids).all():
                break

        return cluster_group

    def assign_clusters(self,X):
        cluster_group = []
        distances = []

        for row in X:
            for centroid in self.centroids:
                distances.append(np.sqrt(np.dot(row-centroid,row-ce
            min_distance = min(distances)
            index_pos = distances.index(min_distance)
            cluster_group.append(index_pos)
            distances.clear()

        return np.array(cluster_group)

    def move_centroids(self,X,cluster_group):
        new_centroids = []

        cluster_type = np.unique(cluster_group)

        for type in cluster_type:
            new_centroids.append(X[cluster_group == type].mean(axis

        return np.array(new_centroids)
```

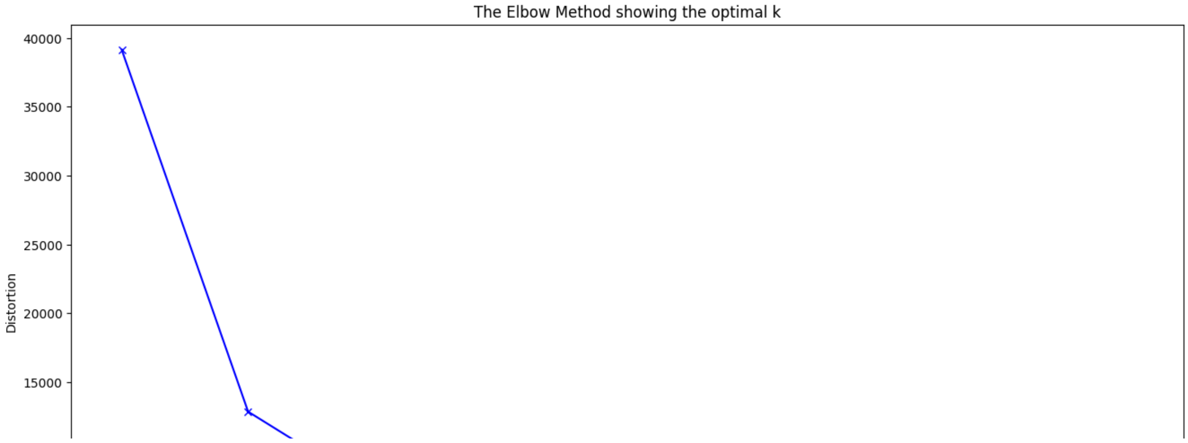
In [45]:
```
km = KMeans1(n_clusters=3,max_iter=1000)
y_means = km.fit_predict(X)

plt.scatter(X[y_means == 0,0],X[y_means == 0,1],color='red')
plt.scatter(X[y_means == 1,0],X[y_means == 1,1],color='blue')
plt.scatter(X[y_means == 2,0],X[y_means == 2,1],color='green')
# plt.scatter(X[y_means == 3,0],X[y_means == 3,1],color='yellow')
plt.show()
```

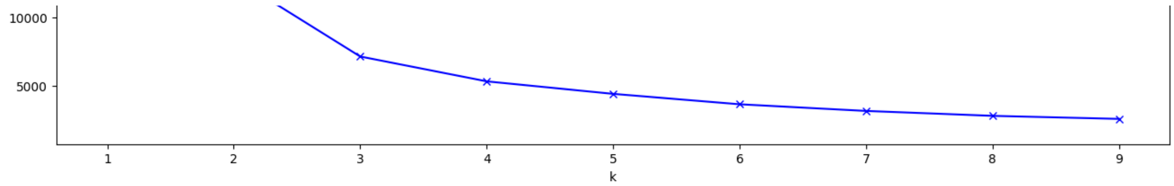


In [42]:
```
from sklearn.cluster import KMeans
distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(df)
    distortions.append(kmeanModel.inertia_)

plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
```

```
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/
site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The d
efault value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```





In [ ]:

In [25]:
```
import numpy as np
```

In [26]:
```
class KNNClassifier:

    def __init__(self, n_neighbours='auto', p=2):
        self.n_neighbours = n_neighbours
        self.p = p


    def fit(self, X, y):

        self.X = X
        self.y = y

        if self.n_neighbours == 'auto':
            self.n_neighbours = int(np.sqrt(len(self.X)))
            if self.n_neighbours % 2 != 0:
                self.n_neighbours += 1

        return self


    def predict(self, X):
#         dim_check([X], [2], ['X'])
        predictions = []
        self.confidence = []
        for pred_row in X:
            euclidean_distances = []
            for X_row in self.X:
                distance = np.linalg.norm(X_row - pred_row, ord=sel
                euclidean_distances.append(distance)

            neighbours = self.y[np.argsort(euclidean_distances)[:se
            neighbours_bc = np.bincount(neighbours)
            prediction = np.argmax(neighbours_bc)
            self.confidence.append(neighbours_bc[prediction]/len(ne
            predictions.append(prediction)

        predictions = np.array(predictions)
        return predictions
```

In [34]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

In [35]:
```
X,y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

In [36]:
```
knn = KNNClassifier()
knn.fit(X_train,y_train)
```

Out[36]:
```
<__main__.KNNClassifier at 0x145be4c40>
```

In [37]:
```
y_pred=knn.predict(X_test)
```

In [32]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

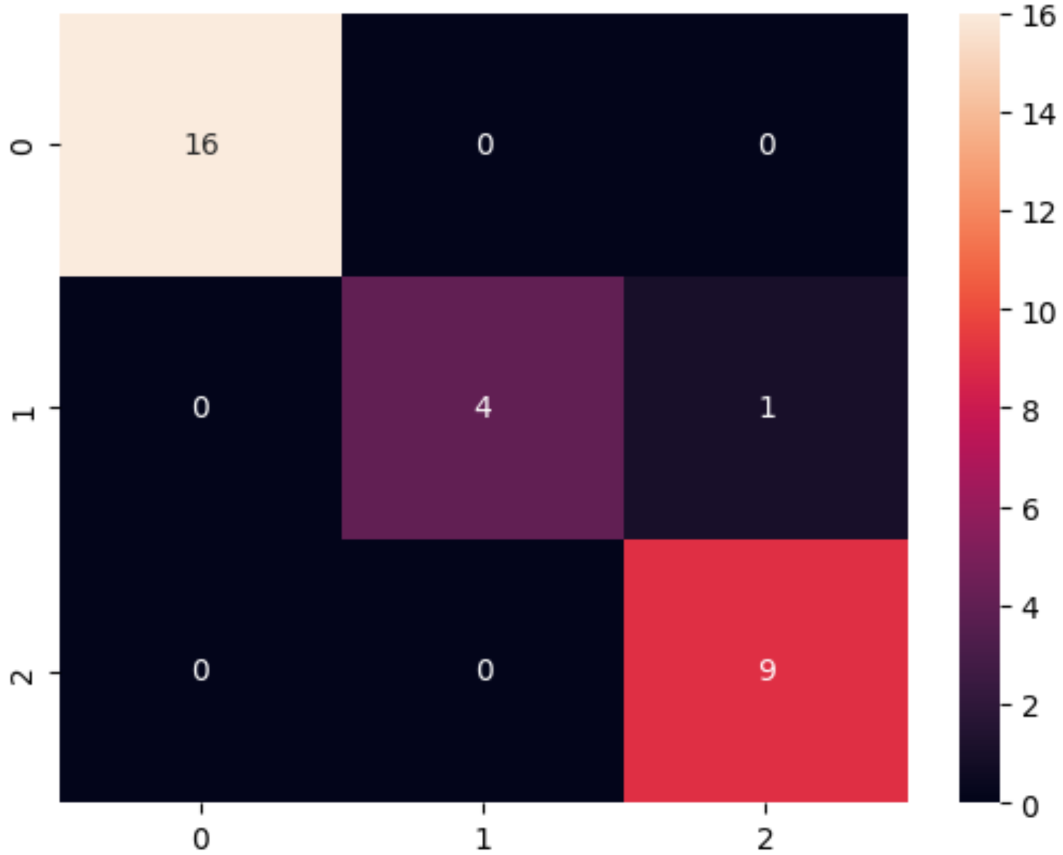
In [38]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[16  0  0]
 [ 0  4  1]
 [ 0  0  9]]
```

Out[38]:
```
0.9666666666666667
```

In [22]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(y_test, y_pred),annot = True)
```

Out[22]:
```
<AxesSubplot: >
```



In [ ]:

In [23]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
from sklearn.linear_model import LinearRegression
```

In [24]:

```
data = load_iris()
print(data.DESCR[:500])
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
```

In [25]:

```
print(data.target_names)
```

```
['setosa' 'versicolor' 'virginica']
```

In [26]:

```
X = data.data  # features
y = data.target  # labels
print(f"Shape of features is {X.shape}, and shape of target is {y.shape}")
print("Targets are: ", y)
```

```
Shape of features is (150, 4), and shape of target is (150,)
Targets are:  [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 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 2
 2 2]
```

In [27]:

```
print(data.data)
```

```
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
```

In [28]:

```
print(data.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

In [29]:

```
print(data.target)
```

```
[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 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
 2 2]
```

In [32]:

```
iris_df = pd.DataFrame(data= data.data, columns= data.feature_names)
target_df = pd.DataFrame(data= data.target, columns= ['species'])
def converter(specie):
    if specie == 0:
        return 'setosa'
    elif specie == 1:
        return 'versicolor'
    else:
        return 'virginica'
target_df['species'] = target_df['species'].apply(converter)
# Concatenate the DataFrames
iris_df = pd.concat([iris_df, target_df], axis= 1)
```

In [16]:

```
target_df = pd.DataFrame(columns= ['species'], data= data.target)
iris_df = pd.concat([iris_df, target_df], axis= 1)
```

In [78]:

```
# Variables
X= iris_df.drop(labels = ['sepal length (cm)', 'petal length (cm)', 'petal width (cm)', 'species'], axis= 1)
y= iris_df['sepal length (cm)']
print(X)
print(y)

# Splitting the Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.33, random_state= 101)
```

```
     sepal width (cm)
0                 3.5
1                 3.0
2                 3.2
3                 3.1
4                 3.6
..                ...
145               3.0
146               2.5
147               3.0
148               3.4
149               3.0

[150 rows x 1 columns]
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: sepal length (cm), Length: 150, dtype: float64
```

In [47]:

```
# Instantiating LinearRegression() Model
lr = LinearRegression()

# Training/Fitting the Model
lr.fit(X_train, y_train)
```

Out[47]:

```
▾ LinearRegression
LinearRegression()
```

In [93]:

```
print(lr.score(X_test, y_test))
# The coefficients
print('Coefficients: \n', lr.coef_)
```

```
0.006469855487622134
Coefficients:
 [-0.22561002]
```

In [50]:

```
# Making Predictions
lr.predict(X_test)
pred = lr.predict(X_test)
print(X_test[:5])
print(pred[:5])
print(y_test[:5])
```

```
     sepal width (cm)
33                4.2
16                3.9
43                3.5
129               3.0
50                3.2
[5.61492654 5.68260954 5.77285355 5.88565856 5.84053656]
33     5.5
16     5.4
43     5.0
129    7.2
50     7.0
Name: sepal length (cm), dtype: float64

/var/folders/kb/2qtwss7n3y3091dclgrcn9hm0000gn/T/ipykernel_97907/2836971499.py:6: FutureWarning: The behavior of
`series[i:j]` with an integer-dtype index is deprecated. In a future version, this will be treated as *label-base
d* indexing, consistent with e.g. `series[i]` lookups. To retain the old behavior, use `series.iloc[i:j]`. To get
the future behavior, use `series.loc[i:j]`.
  print(y_test[:5])
```

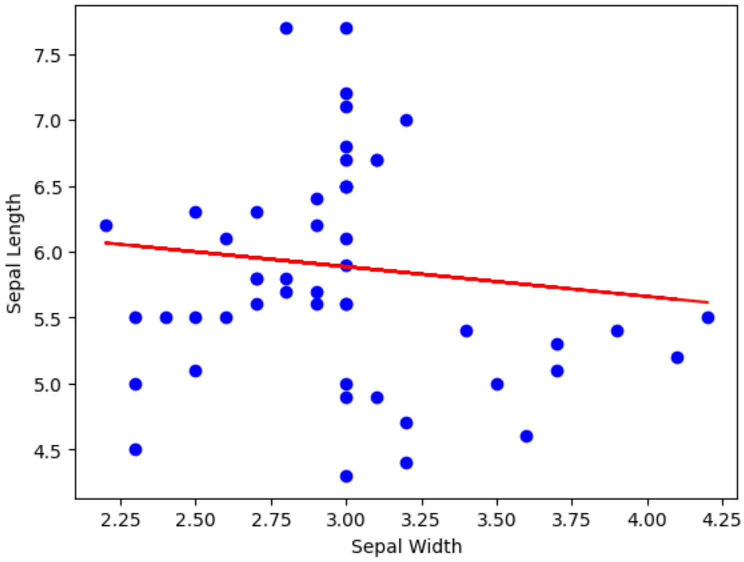
In [51]:

```
# Evaluating Model's Performance
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, pred))
print('Mean Root Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
Mean Absolute Error: 0.6655951111688988
Mean Squared Error: 0.6598987478645434
Mean Root Squared Error: 0.8123415216917964
```

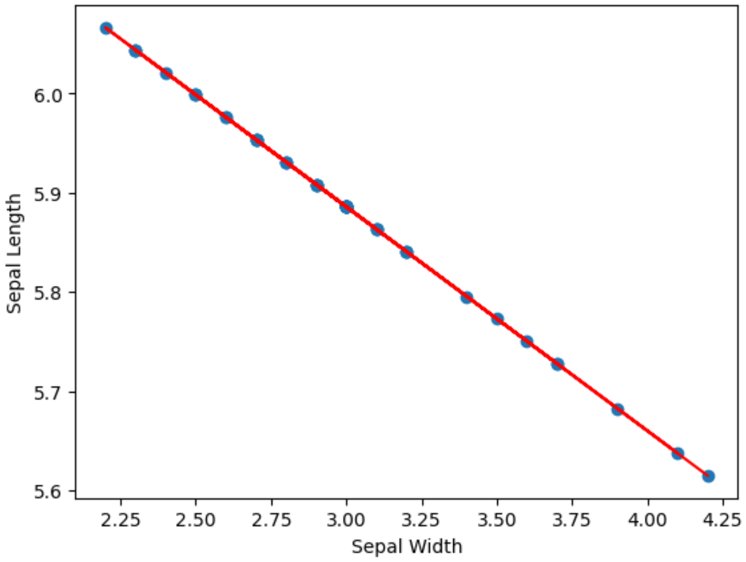
In [94]:

```
plt.scatter(X_test,y_test, color = 'b')
plt.plot(X_test,lr.predict(X_test),color = 'r')
plt.xlabel("Sepal Width")
plt.ylabel("Sepal Length")
plt.show()
```



In [104]:

```
plt.scatter(X_test,pred)
plt.plot(X_test,lr.predict(X_test),color = 'r')
plt.xlabel("Sepal Width")
plt.ylabel("Sepal Length")
plt.show()
```



In [90]:

```
#predicting values
d = {
    'sepal width (cm)' : [5.3]
}
testing = pd.DataFrame(data = d)
y_predicted_value = lr.predict(testing)
print(y_predicted_value)
```

```
[5.36675552]
```

In [105]:

```
#Coefficient of determination
r_squared = lr.score(X,y)
print(r_squared)

#slope
slope = lr.coef_
print(slope)

#intercept
intercept = lr.intercept_
print(intercept)

#SSR(sum of squared residuals)
residuals = y_test - pred

SSR = np.sum(residuals**2)
print(SSR)
```

```
0.012553070673583133
[-0.22561002]
6.562488623065922
32.99493739322717
```

In [ ]:

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.model_selection import train_test_split
data = pd.read_csv("advertising.csv")
data.head()
```

Out[2]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

In [3]:

```
data.tail()
```

Out[3]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
195	196	38.2	3.7	13.8	7.6
196	197	94.2	4.9	8.1	9.7
197	198	177.0	9.3	6.4	12.8
198	199	283.6	42.0	66.2	25.5
199	200	232.1	8.6	8.7	13.4

In [4]:

```
data.shape
```

Out[4]:

```
(200, 5)
```

In [6]:

```
s = data.drop(['Sales', 'Unnamed: 0'], axis=1)
 = data['Sales'].values.reshape(-1,1)
x_train, x_test, y_train, y_test = train_test_split(Xs,y,test_size = 0.3) #Train is 70% and Test is 30%
_train, x_test, y_train, y_test = train_test_split(Xs,y,random_state=1) # default split is 75% for training and 25% for testing
 200 records : 75% train = 150 and 25% test = 50
int(x_train.shape)
int(x_test.shape)
int(y_train.shape)
int(y_test.shape)
eg = LinearRegression()
eg.fit(x_train, y_train)
```

```
(150, 3)
(50, 3)
(150, 1)
(50, 1)
```

Out[6]:

```
▾ LinearRegression
LinearRegression()
```

In [7]:

```
print("Slope: ",reg.coef_)
print("Intercept: ",reg.intercept_)
```

```
Slope:  [[0.04656457 0.17915812 0.00345046]]
Intercept:  [2.87696662]
```

In [8]:

```
print("The linear model is: Y = {:.5} + {:.5}*TV + {:.5}*radio + {:.5}*newspaper".format(reg.intercept_[0], reg.coef_[0][0], re
```

```
The linear model is: Y = 2.877 + 0.046565*TV + 0.17916*radio + 0.0034505*newspaper
```

In [10]:

```
# make predictions on the testing set

y_pred = reg.predict(x_test)
print(y_pred)
```

```
[[21.70910292]
 [16.41055243]
 [ 7.60955058]
 [17.80769552]
 [18.6146359 ]
 [23.83573998]
 [16.32488681]
 [13.43225536]
 [ 9.17173403]
 [17.333853  ]
 [14.44479482]
 [ 9.83511973]
 [17.18797614]
 [16.73086831]
 [15.05529391]
 [15.61434433]
 [12.42541574]
 [17.17716376]
 [11.08827566]
 [18.00537501]
 [ 9.28438889]
 [12.98458458]
 [ 8.79950614]
 [10.42382499]
 [11.3846456 ]
 [14.98082512]
 [ 9.78853268]
 [19.39643187]
 [18.18099936]
 [17.12807566]
 [21.54670213]
 [14.69809481]
 [16.24641438]
 [12.32114579]
 [19.92422501]
 [15.32498602]
 [13.88726522]
 [10.03162255]
 [20.93105915]
 [ 7.44936831]
 [ 3.64695761]
 [ 7.22020178]
 [ 5.9962782 ]
 [18.43381853]
 [ 8.39408045]
 [14.08371047]
 [15.02195699]
 [20.35836418]
 [20.57036347]
 [19.60636679]]
```

In [11]:

```
def myfunc(TV,radio,newspaper):
  Y = 2.877 + 0.046565*TV + 0.17916*radio + 0.0034505*newspaper
  return Y
predictedsales = myfunc(39.5,41.1,10.8)
print("Predicted Sales is ", predictedsales)
```

```
Predicted Sales is  12.117058900000002
```

In [12]:

```
def myfunc(TV,radio):
  Y = Y = 2.927 + 0.0466*TV + 0.1811*radio
  return Y
predictedsales = myfunc(39.5,41.1)
print("Predicted Sales is ", predictedsales)
```

```
Predicted Sales is  12.21091
```

In [13]:

```
from sklearn.metrics import mean_absolute_error
predictions = reg.predict(x_test)
mae = mean_absolute_error(y_test,predictions)
print("Mean Absolute Error = ",mae)
```

```
Mean Absolute Error =  1.066891708259521
```

In [14]:

```
from sklearn.metrics import mean_squared_error
predictions = reg.predict(x_test)
mse = mean_squared_error(y_test,predictions)
print("Mean Squared Error = ",mse)
```

```
Mean Squared Error =  1.9730456202283355
```

In [15]:

```
from sklearn.linear_model import LinearRegression
predictions = reg.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test,predictions))
print("Root Mean Squared Error = ",rmse)
```

```
Root Mean Squared Error =  1.4046514230328946
```

In [16]:

```
reg.score(Xs, y)
```

Out[16]:

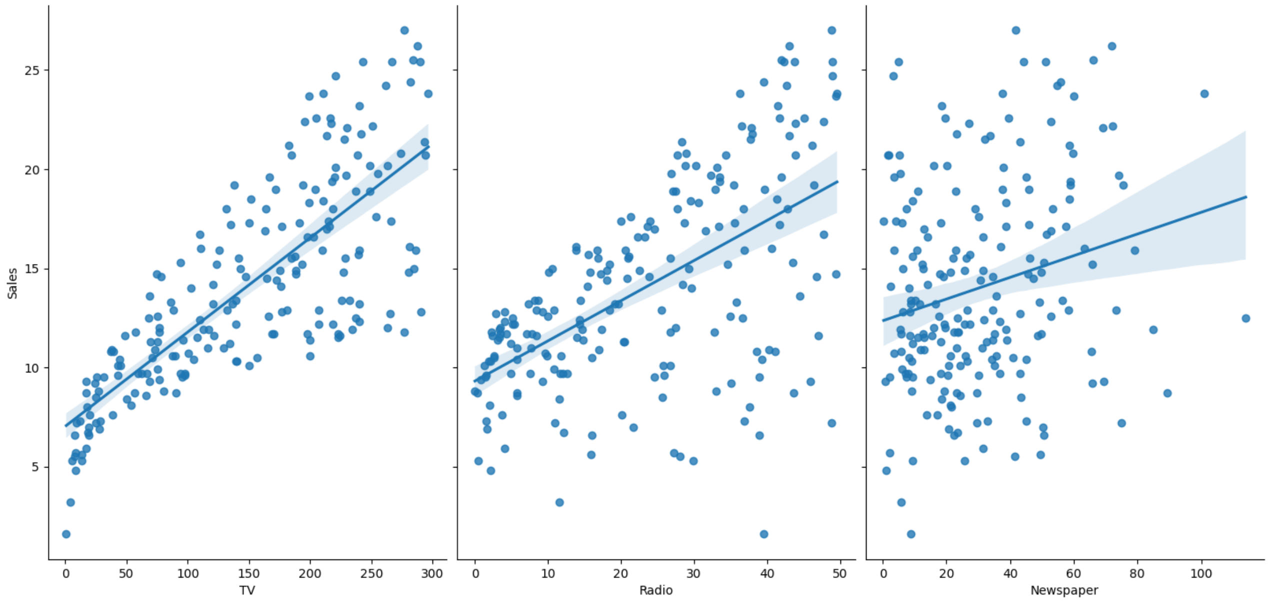
```
0.8963161233045729
```

In [22]:

```
# conventional way to import seaborn
import seaborn as sns

# allow plots to appear within the notebook
%matplotlib inline

# visualize the relationship between the features and the response using scatterplots
# this produces pairs of scatterplot as shown
# use aspect= to control the size of the graphs
# use kind='reg' to plot linear regression on the graph
sns.pairplot(data, x_vars=['TV','Radio','Newspaper'], y_vars='Sales', height=7, aspect=0.7, kind='reg')
```

Out[22]:

```
<seaborn.axisgrid.PairGrid at 0x13f761720>
```



In [ ]:

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
df = pd.read_csv("online_shoppers_intention.csv")
```

In [3]:

```
df.head()
```

Out[3]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageVal
0	0	0.0	0	0.0	1	0.000000	0.20	0.20	
1	0	0.0	0	0.0	2	64.000000	0.00	0.10	
2	0	0.0	0	0.0	1	0.000000	0.20	0.20	
3	0	0.0	0	0.0	2	2.666667	0.05	0.14	
4	0	0.0	0	0.0	10	627.500000	0.02	0.05	

In [4]:

```
df.isnull().any()
```

Out[4]:

```
Administrative             False
Administrative_Duration    False
Informational              False
Informational_Duration     False
ProductRelated             False
ProductRelated_Duration    False
BounceRates                False
ExitRates                  False
PageValues                 False
SpecialDay                 False
Month                      False
OperatingSystems           False
Browser                    False
Region                     False
TrafficType                False
VisitorType                False
Weekend                    False
Revenue                    False
dtype: bool
```

In [5]:

```
revenue_df=pd.get_dummies(df['Revenue'],drop_first=True)
revenue_df
```

Out[5]:

	True
0	0
1	0
2	0
3	0
4	0
...	...
12325	0
12326	0
12327	0
12328	0
12329	0

12330 rows × 1 columns

In [6]:

```
print(revenue_df)
```

```
       True
0         0
1         0
2         0
3         0
4         0
...     ...
12325     0
12326     0
12327     0
12328     0
12329     0

[12330 rows x 1 columns]
```

In [7]:

```
df.drop(['Revenue'],axis=1,inplace=True)
df
```

Out[7]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	Pag
0	0	0.0	0	0.0	1	0.000000	0.200000	0.200000	C
1	0	0.0	0	0.0	2	64.000000	0.000000	0.100000	C
2	0	0.0	0	0.0	1	0.000000	0.200000	0.200000	C
3	0	0.0	0	0.0	2	2.666667	0.050000	0.140000	C
4	0	0.0	0	0.0	10	627.500000	0.020000	0.050000	C
...	...	...	...	...	...	...	...	...	
12325	3	145.0	0	0.0	53	1783.791667	0.007143	0.029031	12
12326	0	0.0	0	0.0	5	465.750000	0.000000	0.021333	C
12327	0	0.0	0	0.0	6	184.250000	0.083333	0.086667	C
12328	4	75.0	0	0.0	15	346.000000	0.000000	0.021053	C
12329	0	0.0	0	0.0	3	21.250000	0.000000	0.066667	C

12330 rows × 17 columns

In [8]:

```
df=pd.concat([df,revenue_df],axis=1)
df
```

Out[8]:

ductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month	OperatingSystems	Browser	Region	TrafficType	VisitorType	Weekend
0.000000	0.200000	0.200000	0.000000	0.0	Feb	1	1	1	1	Returning_Visitor	False
64.000000	0.000000	0.100000	0.000000	0.0	Feb	2	2	1	2	Returning_Visitor	False
0.000000	0.200000	0.200000	0.000000	0.0	Feb	4	1	9	3	Returning_Visitor	False
2.666667	0.050000	0.140000	0.000000	0.0	Feb	3	2	2	4	Returning_Visitor	False
627.500000	0.020000	0.050000	0.000000	0.0	Feb	3	3	1	4	Returning_Visitor	True
...	...	...	...	...	...	...	...	...	...	...	...
1783.791667	0.007143	0.029031	12.241717	0.0	Dec	4	6	1	1	Returning_Visitor	True
465.750000	0.000000	0.021333	0.000000	0.0	Nov	3	2	1	8	Returning_Visitor	True
184.250000	0.083333	0.086667	0.000000	0.0	Nov	3	2	1	13	Returning_Visitor	True
346.000000	0.000000	0.021053	0.000000	0.0	Nov	2	2	3	11	Returning_Visitor	False
21.250000	0.000000	0.066667	0.000000	0.0	Nov	3	2	1	2	New_Visitor	True

In [9]:

```
X = df.iloc[:, [5, 6,7]].values
X
```

Out[9]:

```
array([[0.0000000e+00, 2.0000000e-01, 2.0000000e-01],
       [6.4000000e+01, 0.0000000e+00, 1.0000000e-01],
       [0.0000000e+00, 2.0000000e-01, 2.0000000e-01],
       ...,
       [1.8425000e+02, 8.3333333e-02, 8.6666667e-02],
       [3.4600000e+02, 0.0000000e+00, 2.1052632e-02],
       [2.1250000e+01, 0.0000000e+00, 6.6666667e-02]])
```

In [10]:

```
y = df.iloc[:, -1].values
y
```

Out[10]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=uint8)
```

In [11]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

In [12]:

```
# Training the Naive Bayes model on the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[12]:

```
▾ GaussianNB
GaussianNB()
```

In [16]:

```
y_pred = classifier.predict(X_test)
print('Predicted Value')
print(y_pred[:5])
print('Actual Value')
print(y_test[:5])
```

```
Predicted Value
[0 1 0 0 0]
Actual Value
[0 0 0 0 0]
```

In [14]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
ac = accuracy_score(y_test,y_pred)
ac
```

Out[14]:

```
0.8203568532035685
```

In [21]:

```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[21]:

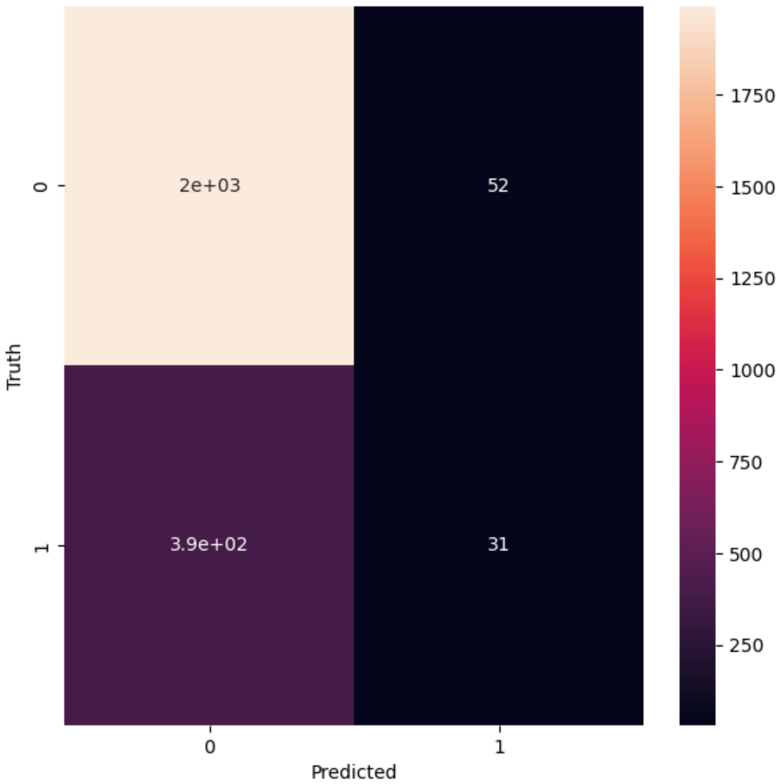
```
array([[1992,   52],
       [ 391,   31]])
```

In [22]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(7,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[22]:

```
Text(58.222222222222214, 0.5, 'Truth')
```



In [ ]:

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
```

In [66]:

```
iris = load_iris()
print(iris.DESCR[:760])
# print(iris.data)
print(iris.target_names)
print(iris.feature_names)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:
```

In [67]:

```
X = iris.data
y = iris.target
```

In [68]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
```

In [69]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

Out[69]:

```
▾ GaussianNB
GaussianNB()
```

In [60]:

```
y_pred = gnb.predict(X_test)
```

In [70]:

```
print('Predicted Values')
print(y_pred[:5])
print('Actual Values')
print(y_test[:5])
```

```
Predicted Values
[0 1 1 0 2]
Actual Values
[0 1 1 0 2]
```

In [71]:

```
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)*100)
```

```
Gaussian Naive Bayes model accuracy(in %): 95.0
```

In [72]:

```
print('Accuracy : ' , metrics.accuracy_score(y_test, y_pred))
print('Precison  : ' , metrics.precision_score(y_test, y_pred, average="weighted"))
print('Recall Score : ' , metrics.recall_score(y_test, y_pred, average="weighted"))
print('F1 Score : ' , metrics.f1_score(y_test, y_pred, average="weighted"))
print('MCC : ' , metrics.matthews_corrcoef(y_test, y_pred))
```

```
Accuracy :  0.95
Precison  :  0.9507539682539683
Recall Score :  0.95
F1 Score :  0.95
MCC :  0.9253544620517098
```

In [78]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.95      0.90      0.93        21
           2       0.90      0.95      0.93        20

    accuracy                           0.95        60
   macro avg       0.95      0.95      0.95        60
weighted avg       0.95      0.95      0.95        60
```

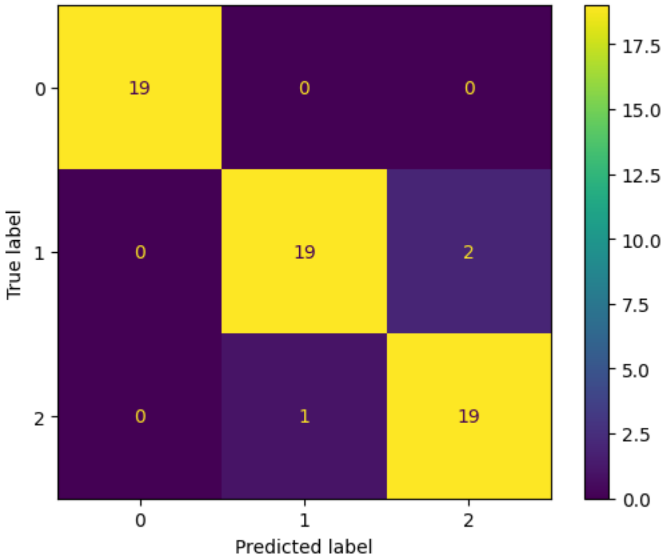
In [95]:

```
sklearnconf = metrics.confusion_matrix(y_test, y_pred)
print("\nsklearn Confusion Matrix is \n", sklearnconf)
```

```

sklearn Confusion Matrix is 
 [[19  0  0]
 [ 0 19  2]
 [ 0  1 19]]
```

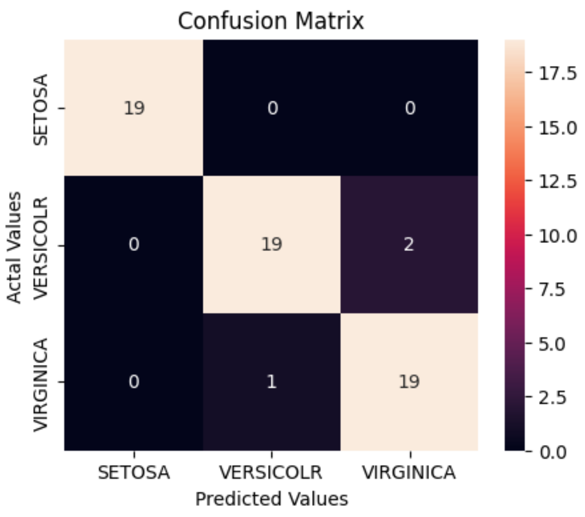
In [86]:

```
labels = [1,0]
cm = metrics.confusion_matrix(y_test, y_pred)
disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot();
plt.show();
```



In [94]:

```
cm = metrics.confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm,
                     index = ['SETOSA','VERSICOLR','VIRGINICA'],
                     columns = ['SETOSA','VERSICOLR','VIRGINICA'])
#Plotting the confusion matrix
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



In [ ]:

```
# sensitivity1 = cm1[0,0]/(cm1[0,0]+cm1[0,1])
# print('Sensitivity : ', sensitivity1 )

# specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])
# print('Specificity : ', specificity1)
```

In [1]:
```
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from sklearn import tree
import numpy
```

In [2]:
```
df = pd.read_csv("playtennis.csv")
df
```

Out[2]:

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

In [3]:
```
encoder = LabelEncoder()
df = df.apply(encoder.fit_transform)
df
```

Out[3]:

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1
5	1	0	1	0	0
6	0	0	1	0	1
7	2	2	0	1	0
8	2	0	1	1	1
9	1	2	1	1	1
10	2	2	1	0	1
11	0	2	0	0	1
12	0	1	1	1	1
13	1	2	0	0	0

In [4]:
```
X = df.iloc[:,:-1].to_numpy()
Y = df.iloc[:,-1].to_numpy()
```

In [5]:
```
model_cart = DecisionTreeClassifier(criterion = 'gini',max_depth =
model_id3 = DecisionTreeClassifier(criterion = 'entropy',max_depth
```

In [6]:
```
model_cart.fit(X,Y)
model_id3.fit(X,Y)
```

Out[6]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [7]:
```
pred_cart = model_cart.predict(X)
pred_id3 = model_id3.predict(X)
```

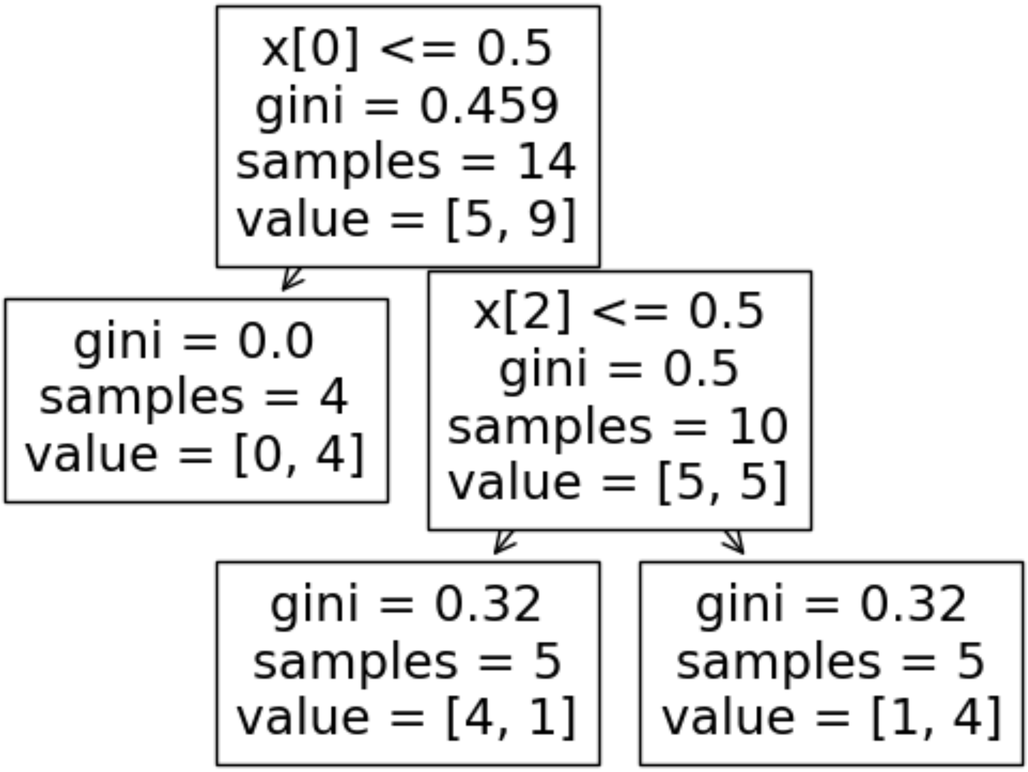
In [8]:
```
cm_cart = confusion_matrix(Y,pred_cart)
cm_id3 = confusion_matrix(Y,pred_id3)
```

In [9]:
```
print(cm_cart)
print(cm_id3)
```

```
[[4 1]
 [1 8]]
[[4 1]
 [1 8]]
```

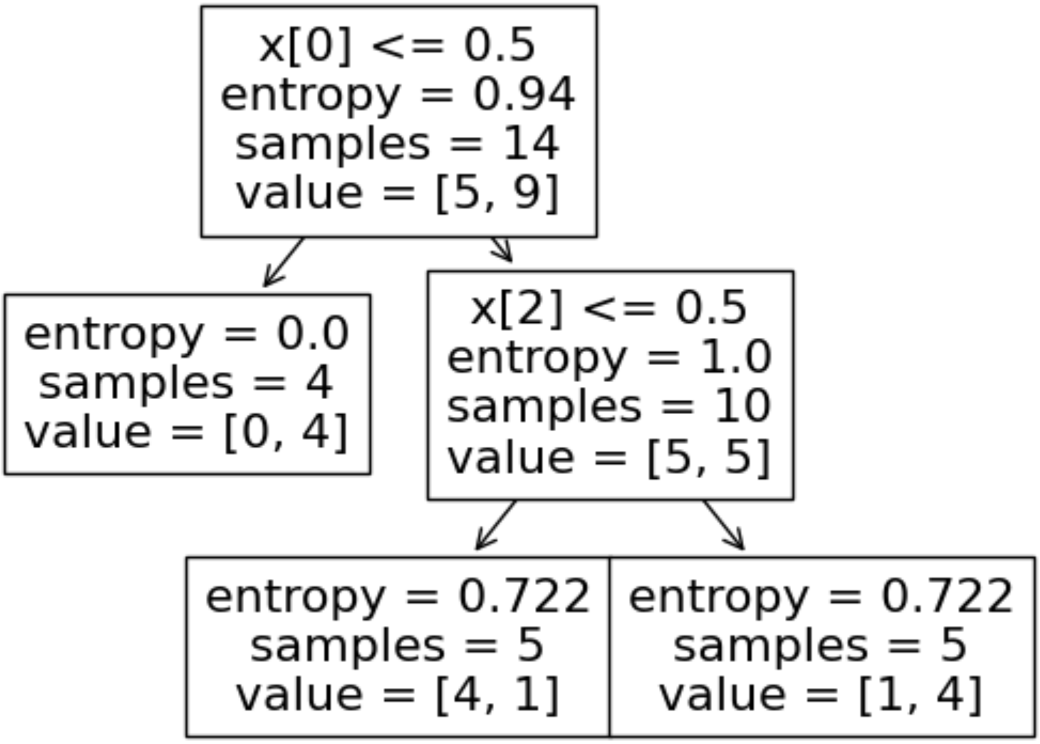
In [10]:
```
tree.plot_tree(model_cart)
```

Out[10]:
```
[Text(0.4, 0.8333333333333334, 'x[0] <= 0.5\ngini = 0.459\nsamples
= 14\nvalue = [5, 9]'),
 Text(0.2, 0.5, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.6, 0.5, 'x[2] <= 0.5\ngini = 0.5\nsamples = 10\nvalue = [5
, 5]'),
 Text(0.4, 0.16666666666666666, 'gini = 0.32\nsamples = 5\nvalue =
[4, 1]'),
 Text(0.8, 0.16666666666666666, 'gini = 0.32\nsamples = 5\nvalue =
[1, 4]')]
```



In [11]:
```
tree.plot_tree(model_id3)
```

Out[11]:
```
[Text(0.4, 0.8333333333333334, 'x[0] <= 0.5\nentropy = 0.94\nsampl
es = 14\nvalue = [5, 9]'),
 Text(0.2, 0.5, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.6, 0.5, 'x[2] <= 0.5\nentropy = 1.0\nsamples = 10\nvalue =
[5, 5]'),
 Text(0.4, 0.16666666666666666, 'entropy = 0.722\nsamples = 5\nval
ue = [4, 1]'),
 Text(0.8, 0.16666666666666666, 'entropy = 0.722\nsamples = 5\nval
ue = [1, 4]')]
```



In [ ]:

In [15]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.feature_selection import RFE
```

In [6]:

```
data = pd.read_csv('social_network.csv')
data.head()
```

Out[6]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [7]:

```
data.isnull().any()
```

Out[7]:

```
User ID            False
Gender             False
Age                False
EstimatedSalary    False
Purchased          False
dtype: bool
```

In [8]:

```
X = data.iloc[:, [2]].values
X
```

Out[8]:

```
array([[19],
       [35],
       [26],
       [27],
       [19],
       [27],
       [27],
       [32],
       [25],
       [35],
       [26],
       [26],
       [20],
       [32],
       [18],
       [29],
       [47],
       [45],
```

In [10]:

```
y = data.iloc[:, -1].values
y
```

Out[10]:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1])
```

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

In [18]:

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

Out[18]:

```
▾ LogisticRegression
LogisticRegression()
```

In [21]:

```
print('Predicted Value')
y_pred = logreg.predict(X_test)
print(y_pred[:10])
print('Actual Value')
print(y_test[:10])
```

```
Predicted Value
[0 0 0 0 0 0 0 0 0 1]
Actual Value
[0 0 0 0 0 0 0 1 0 0]
```

In [20]:

```
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))
```

```
Accuracy of logistic regression classifier on test set: 0.94
```

In [22]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[57  1]
 [ 4 18]]
```

In [23]:

```
TN, FP, FN, TP = metrics.confusion_matrix(list(y_test), list(y_pred), labels=[0, 1]).ravel() #0,1 is default label of sklearn
print("True Negatives", TN)
print("True Positives", TP)
print("False Positives", FP)
print("False Negatives", FN)
```

```
True Negatives 57
True Positives 18
False Positives 1
False Negatives 4
```

In [24]:

```
results={} #To Store all the metrics Result Values
```

In [25]:

```
#Finding Accuracy
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}") #Note the Formatting with f"{metric} and rounding to 3 decimal points with .3f
```

```
ACC is  0.938
```

In [26]:

```
#Finding True Negative Rate
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.983
```

In [27]:

```
#Finding Precision Predictive Value
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.947
```

In [28]:

```
#Finding Negative Predictive Value
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.934
```

In [29]:

```
#Finding True Postive Rate
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.818
```

In [30]:

```
#Finding F1-Score
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.878
```

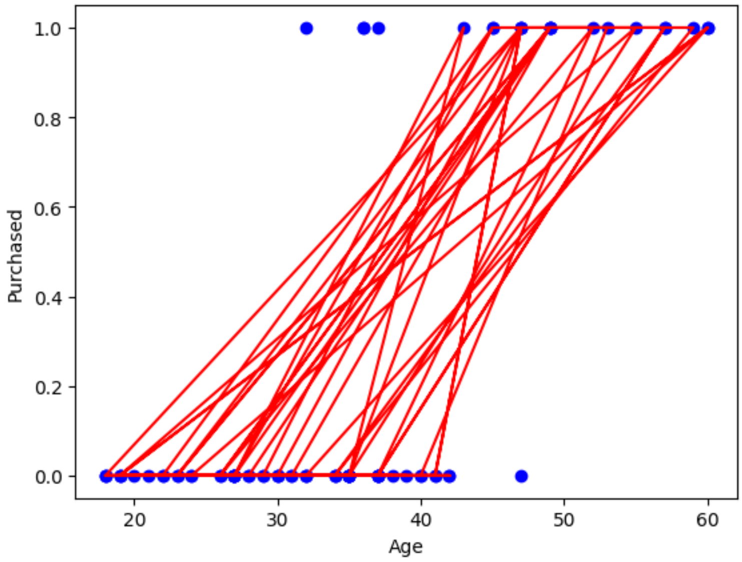
In [31]:

```
#Finding MCC Value
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.840
```

In [44]:

```
plt.scatter(X_test,y_test, color = 'b')
plt.plot(X_test,logreg.predict(X_test),color = 'r')
plt.xlabel("Age")
plt.ylabel("Purchased")
plt.show()
```



In [47]:

```
y_proba = logreg.predict_proba(X_test)
print("Probability of not purchased & probability of purchased. ", y_proba[:5])
y_proba = y_proba[:,1].reshape((y_proba.shape[0],))
print("2D to 1D reshaped Probability of Purchased. ", y_proba[:5])
```

```
Probability of not purchased & probability of purchased.  [[0.89588762 0.10411238]
 [0.68568636 0.31431364]
 [0.78493273 0.21506727]
 [0.89588762 0.10411238]
 [0.78493273 0.21506727]]
2D to 1D reshaped Probability of Purchased.  [0.10411238 0.31431364 0.21506727 0.10411238 0.21506727]
```

In [ ]:

Un-Supervised Learning Algorithms - Hierarchical Clustering: Using any dataset implement Hierarchical Clustering (AGNES and DIANA). Plot the Dendrogram for Hierarchical Clustering and analyze your result. Plot the clustering output for the same dataset using these two hierarchical techniques. Compare the results. Write the inference.

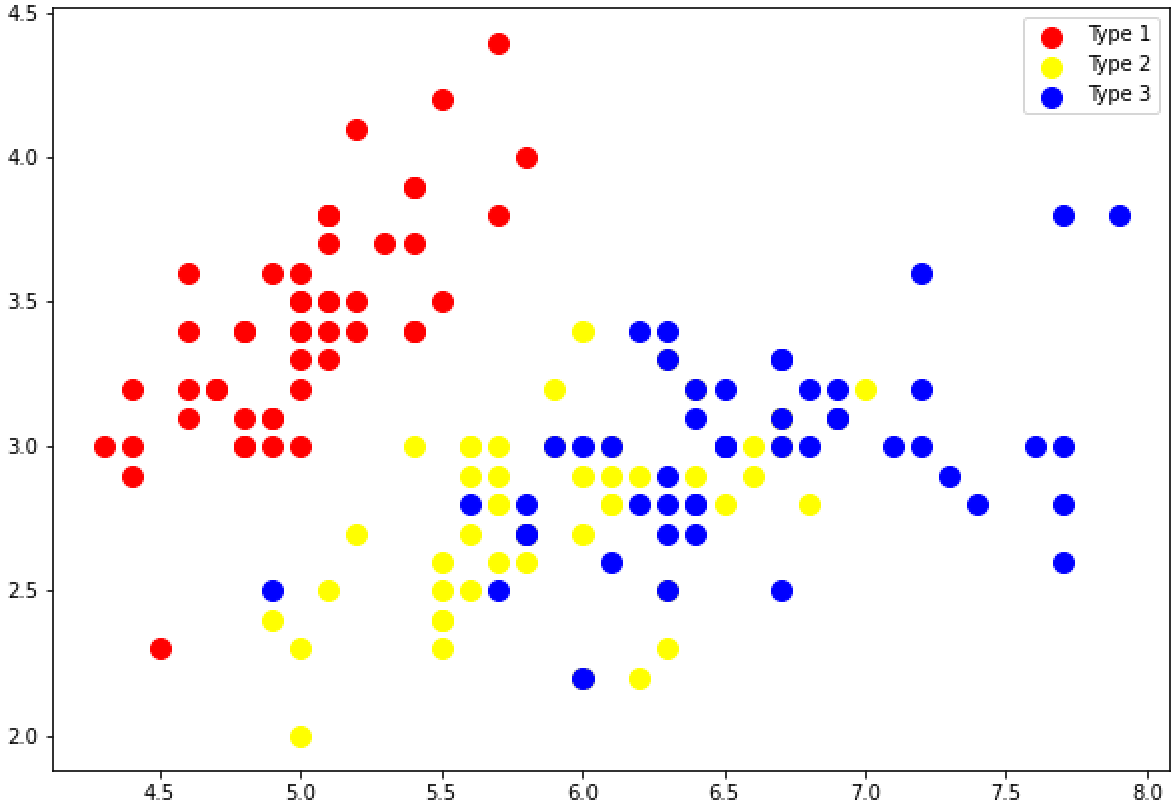
In [13]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering

# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

# Import iris data
iris = datasets.load_iris()

iris_data = pd.DataFrame(iris.data)
iris_data.columns = iris.feature_names
iris_data['flower_type'] = iris.target
iris_data.head()
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower_type
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

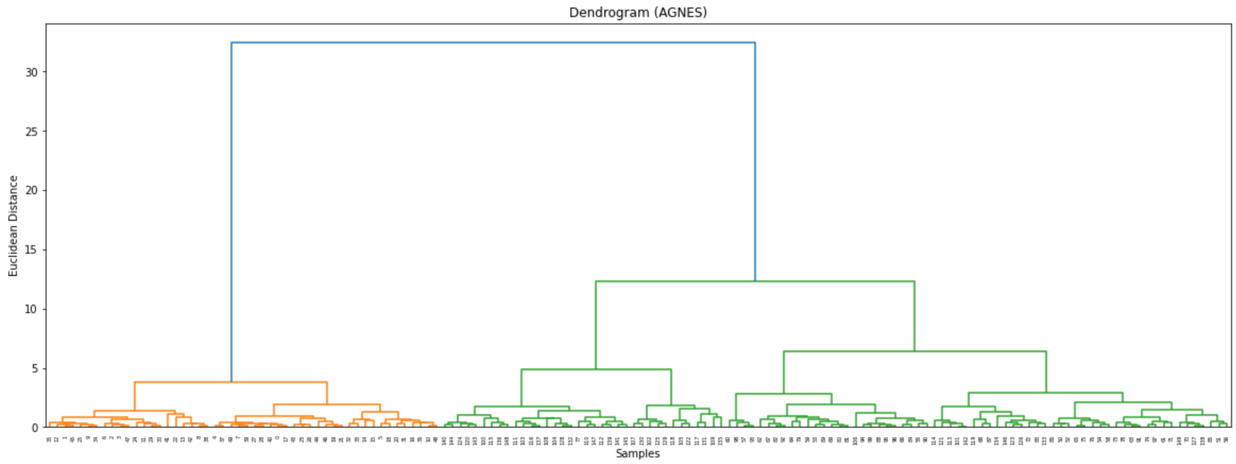
In [ ]:
```
#Visualise the classes
```

In [15]:
```
iris_X = iris_data.iloc[:, [0, 1, 2,3]].values
iris_Y = iris_data.iloc[:,4].values

import matplotlib.pyplot as plt
plt.figure(figsize=(10, 7))
plt.scatter(iris_X[iris_Y == 0, 0], iris_X[iris_Y == 0, 1], s=100, c='red', label='Typ
plt.scatter(iris_X[iris_Y == 1, 0], iris_X[iris_Y == 1, 1], s=100, c='yellow', label='
plt.scatter(iris_X[iris_Y == 2, 0], iris_X[iris_Y == 2, 1], s=100, c='blue', label='Ty
plt.legend()
plt.show()
```

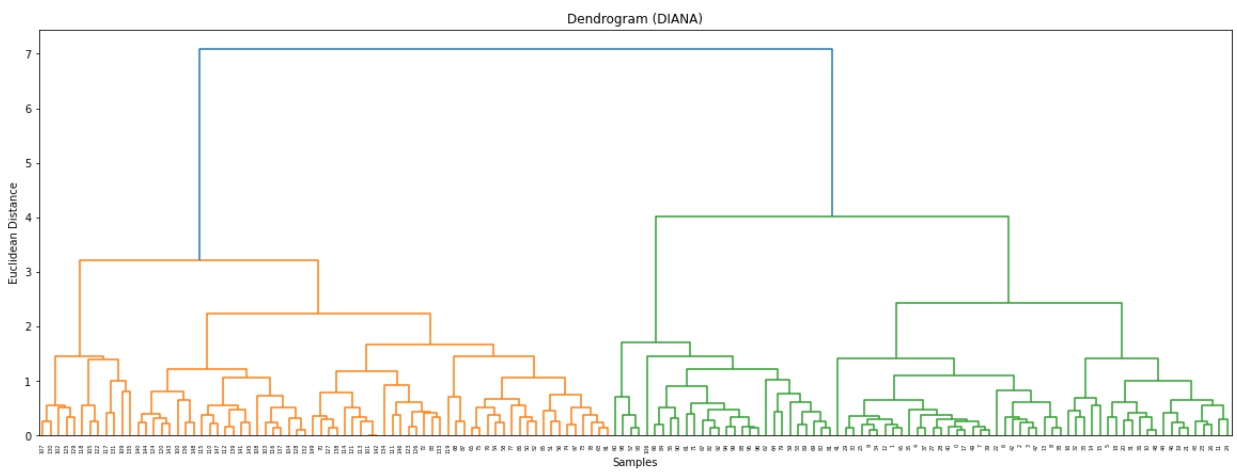


In [ ]:
```
"""
Create a dendrogram
We start by importing the library that will help to create dendrograms.
Dendrogram helps to give a rough idea of the number of clusters.
"""
```

In [34]:
```
import scipy.cluster.hierarchy as sc

# Plot dendrogram
plt.figure(figsize=(20, 7))
plt.title("Dendrograms")

# Create dendrogram for AGNES
sc.dendrogram(sc.linkage(iris_X, method='ward'))

# Plot the Dendrogram
plt.title('Dendrogram (AGNES)')
plt.xlabel('Samples')
plt.ylabel('Euclidean Distance')
plt.show()
```



In [35]:
```
# Plot dendrogram
plt.figure(figsize=(20, 7))
plt.title("Dendrograms")

# Create dendrogram for DIANA
sc.dendrogram(sc.linkage(iris_X, method='complete'))

# Plot the Dendrogram
plt.title('Dendrogram (DIANA)')
plt.xlabel('Samples')
plt.ylabel('Euclidean Distance')
plt.show()
```



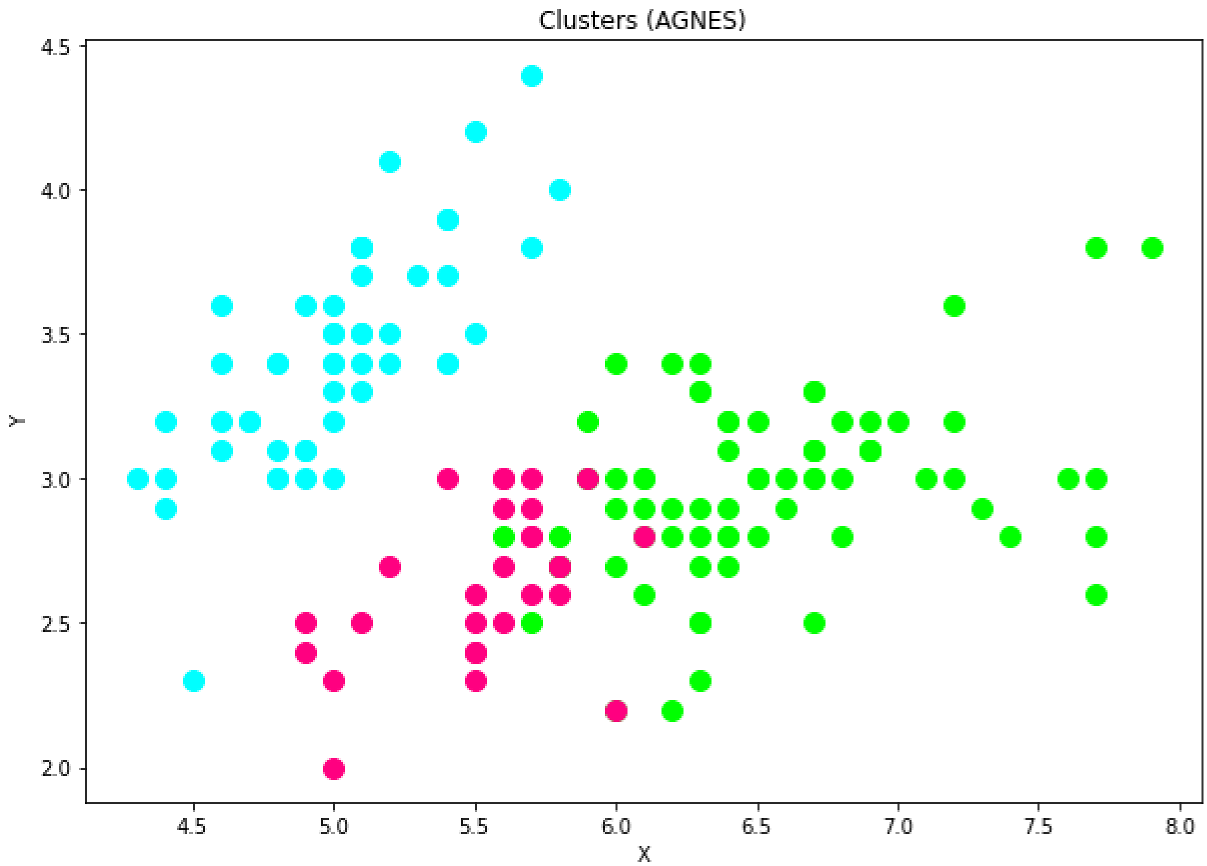
In [ ]:
```
"""
Fit the model
We instantiate the AgglomerativeClustering.
Pass euclidean distance as the measure of the distance between points and ward and com
proximity.
Then we fit the model on our data points.
Finally, we return an array of integers where the values correspond to the distinct ca
"""
```

In [36]:
```
#AGNES
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(
    n_clusters=3, affinity='euclidean', linkage='ward')
```

```
cluster.fit(iris_X)
labels = cluster.labels_
labels
```

Out[36]:
```
array([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, 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, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,
       2, 0, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int64)
```

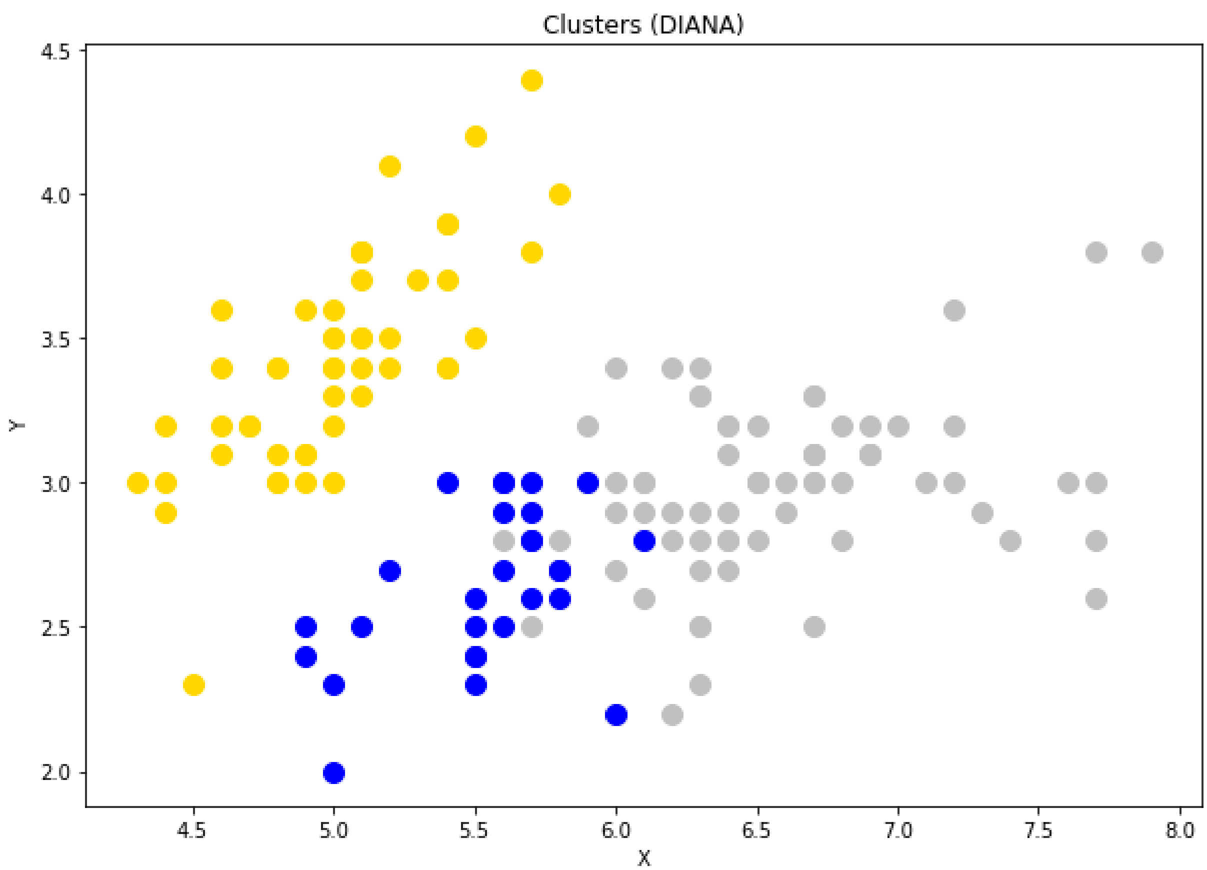
In [49]:
```
plt.figure(figsize=(10, 7))
plt.scatter(iris_X[labels == 0, 0], iris_X[labels == 0, 1], s = 100, c = '#00ff00', la
plt.scatter(iris_X[labels == 1, 0], iris_X[labels == 1, 1], s = 100, c = '#00ffff', la
plt.scatter(iris_X[labels == 2, 0], iris_X[labels == 2, 1], s = 100, c = '#ff0080', la
plt.title('Clusters (AGNES)')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



In [41]:
```
#DIANA
cluster = AgglomerativeClustering(
    n_clusters=3, affinity='euclidean', linkage='complete')

cluster.fit(iris_X)
labels = cluster.labels_
labels
```

Out[41]:
```
array([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, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 2, 2, 2, 0, 2, 0,
       2, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 2, 0, 0, 0,
       2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 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], dtype=int64)
```

In [50]:
```
plt.figure(figsize=(10, 7))
plt.scatter(iris_X[labels == 0, 0], iris_X[labels == 0, 1], s = 100, c = 'silver', lat
plt.scatter(iris_X[labels == 1, 0], iris_X[labels == 1, 1], s = 100, c = 'gold', label
plt.scatter(iris_X[labels == 2, 0], iris_X[labels == 2, 1], s = 100, c = '#0000ff', la
plt.title('Clusters (DIANA)')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



The Dendrogram shows the hierarchical relationships between the data points, where the height of the vertical lines represents the distance between the clusters. The dendrogram shows that the iris dataset can be divided into 3 clusters. Both AGNES and DIANA clustering also divide the dataset into 3 clusters. However, the clustering outputs look slightly different as AGNES creates more compact clusters while DIANA creates more dispersed clusters. Overall, both methods are effective in clustering the iris dataset.