Question 1. Classify the iris dataset using a decision tree classifier. Divide the dataset into training and testing in the ratio 80:20. Use the functions from the sklearn package. Display the final decision tree

In [1]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot as plt
from sklearn import tree
```

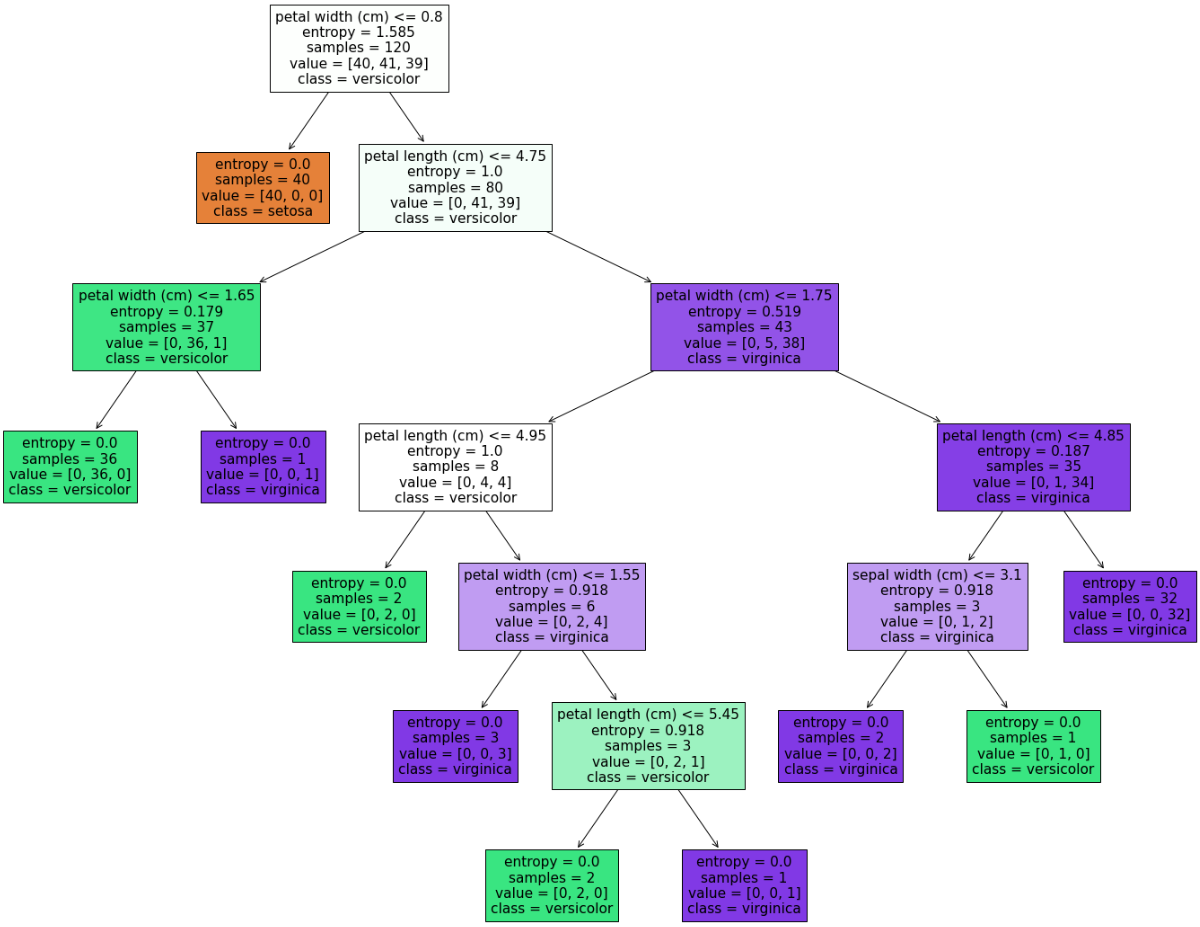
In [2]:
```
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=42)
```

In [3]:
```
clf=DecisionTreeClassifier(criterion='entropy',max_depth=None,random_state=0)
clf.fit(X_train,y_train)
```

Out[3]:
```
DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [4]:
```
fig=plt.figure(figsize=(25,20))
tree.plot_tree(clf,feature_names=iris.feature_names,class_names=iris.target_names,filled=True)
```

Out[4]:
```
[Text(0.3076923076923077, 0.9285714285714286, 'petal width (cm) <= 0.8\nentropy = 1.585\nsamples = 120\nvalue = [40, 41, 39]
\nclass = versicolor'),
 Text(0.23076923076923078, 0.7857142857142857, 'entropy = 0.0\nsamples = 40\nvalue = [40, 0, 0]\nclass = setosa'),
 Text(0.38461538461538464, 0.7857142857142857, 'petal length (cm) <= 4.75\nentropy = 1.0\nsamples = 80\nvalue = [0, 41, 39]\n
class = versicolor'),
 Text(0.15384615384615385, 0.6428571428571429, 'petal width (cm) <= 1.65\nentropy = 0.179\nsamples = 37\nvalue = [0, 36, 1]\n
class = versicolor'),
 Text(0.07692307692307693, 0.5, 'entropy = 0.0\nsamples = 36\nvalue = [0, 36, 0]\nclass = versicolor'),
 Text(0.23076923076923078, 0.5, 'entropy = 0.0\nsamples = 1\nvalue = [0, 0, 1]\nclass = virginica'),
 Text(0.6153846153846154, 0.6428571428571429, 'petal width (cm) <= 1.75\nentropy = 0.519\nsamples = 43\nvalue = [0, 5, 38]\nc
lass = virginica'),
 Text(0.38461538461538464, 0.5, 'petal length (cm) <= 4.95\nentropy = 1.0\nsamples = 8\nvalue = [0, 4, 4]\nclass = versicolo
r'),
 Text(0.3076923076923077, 0.35714285714285715, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2, 0]\nclass = versicolor'),
 Text(0.46153846153846156, 0.35714285714285715, 'petal width (cm) <= 1.55\nentropy = 0.918\nsamples = 6\nvalue = [0, 2, 4]\nc
lass = virginica'),
 Text(0.38461538461538464, 0.21428571428571427, 'entropy = 0.0\nsamples = 3\nvalue = [0, 0, 3]\nclass = virginica'),
 Text(0.5384615384615384, 0.21428571428571427, 'petal length (cm) <= 5.45\nentropy = 0.918\nsamples = 3\nvalue = [0, 2, 1]\nc
lass = versicolor'),
 Text(0.46153846153846156, 0.07142857142857142, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2, 0]\nclass = versicolor'),
 Text(0.6153846153846154, 0.07142857142857142, 'entropy = 0.0\nsamples = 1\nvalue = [0, 0, 1]\nclass = virginica'),
 Text(0.8461538461538461, 0.5, 'petal length (cm) <= 4.85\nentropy = 0.187\nsamples = 35\nvalue = [0, 1, 34]\nclass = virgini
ca'),
 Text(0.7692307692307693, 0.35714285714285715, 'sepal width (cm) <= 3.1\nentropy = 0.918\nsamples = 3\nvalue = [0, 1, 2]\ncla
ss = virginica'),
 Text(0.6923076923076923, 0.21428571428571427, 'entropy = 0.0\nsamples = 2\nvalue = [0, 0, 2]\nclass = virginica'),
 Text(0.8461538461538461, 0.21428571428571427, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]\nclass = versicolor'),
 Text(0.9230769230769231, 0.35714285714285715, 'entropy = 0.0\nsamples = 32\nvalue = [0, 0, 32]\nclass = virginica')]
```

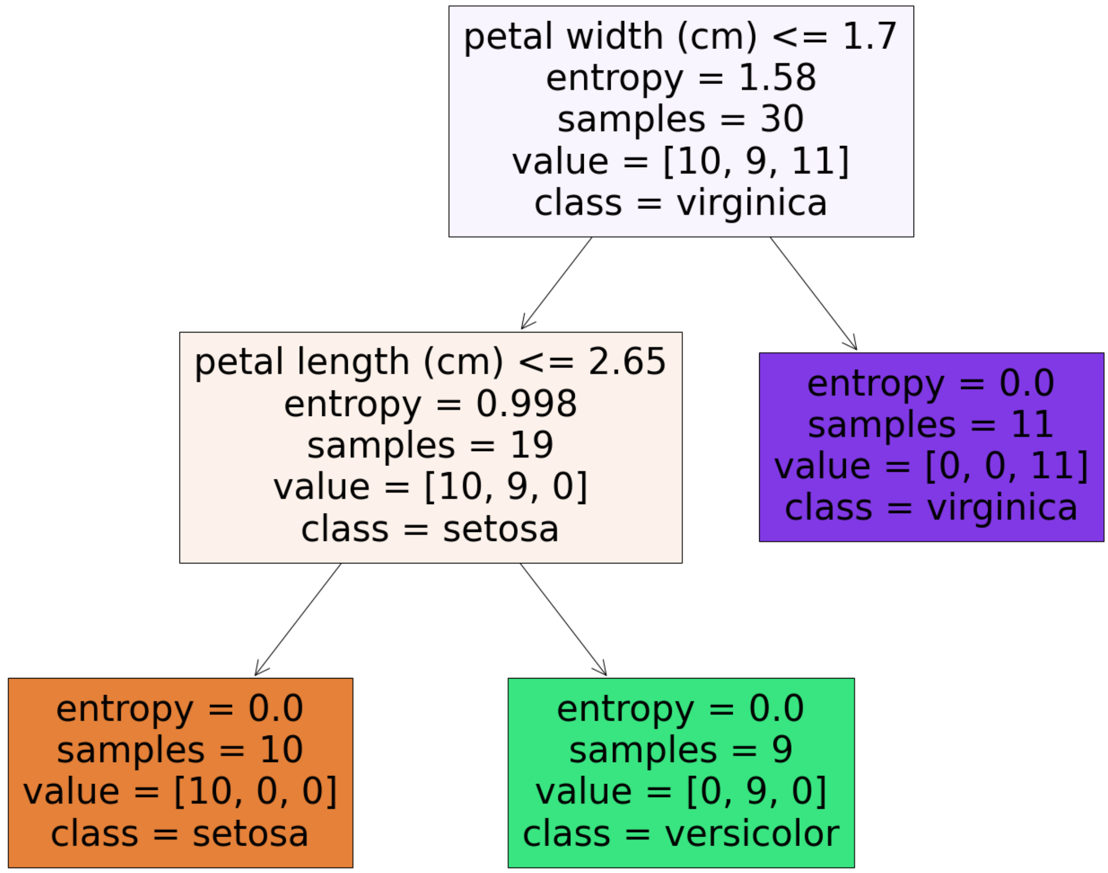


In [5]:
```
clf.fit(X_test,y_test)
```

Out[5]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [6]:
```
fig=plt.figure(figsize=(25,20))
tree.plot_tree(clf,feature_names=iris.feature_names,class_names=iris.target_names,filled=True)
```

Out[6]:
```
[Text(0.6, 0.8333333333333334, 'petal width (cm) <= 1.7\nentropy = 1.58\nsamples = 30\nvalue = [10, 9, 11]\nclass = virginic
a'),
 Text(0.4, 0.5, 'petal length (cm) <= 2.65\nentropy = 0.998\nsamples = 19\nvalue = [10, 9, 0]\nclass = setosa'),
 Text(0.2, 0.16666666666666666, 'entropy = 0.0\nsamples = 10\nvalue = [10, 0, 0]\nclass = setosa'),
 Text(0.6, 0.16666666666666666, 'entropy = 0.0\nsamples = 9\nvalue = [0, 9, 0]\nclass = versicolor'),
 Text(0.8, 0.5, 'entropy = 0.0\nsamples = 11\nvalue = [0, 0, 11]\nclass = virginica')]
```


petal width (cm) <= 1.7
entropy = 1.58
samples = 30
value = [10, 9, 11]
class = virginica
petal length (cm) <= 2.65
entropy = 0.998
samples = 19
value = [10, 9, 0]
class = setosa
entropy = 0.0
samples = 11
value = [0, 0, 11]
class = virginica
entropy = 0.0
samples = 10
value = [10, 0, 0]
class = setosa
entropy = 0.0
samples = 9
value = [0, 9, 0]
class = versicolor

In [ ]:

Question 2. Classify the iris dataset using a Bayes classifier. Divide the dataset into training and testing in the ratio 80:20. Use the functions from the sklearn package. Assume the data follows a gaussian distribution. Display the training and testing accuracy, confusion matrix.

In [1]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
```

In [2]:
```
iris_data=load_iris()
```

In [3]:
```
X_train,X_test,y_train,y_test=train_test_split(iris_data.data,iris_data.target,test_size=0.2,random_state=42)
```

In [4]:
```
clf=GaussianNB()
clf.fit(X_train,y_train)
```

Out[4]:
```
▾ GaussianNB
GaussianNB()
```

In [5]:
```
y_pred=clf.predict(X_test)
y_pred
```

Out[5]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

In [6]:
```
print('Training Accuracy: ', clf.score(X_train,y_train))
print('Testing Accuracy: ', clf.score(X_test,y_test))
```

```
Training Accuracy:  0.95
Testing Accuracy:  1.0
```

In [7]:
```
print('Confusion Matrix:')
print(confusion_matrix(y_test,y_pred))
```

```
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

In [ ]:

Question 3. Classify the iris dataset using the KNN classifier. Divide the dataset into training, validation, and testing in the ratio 70:15:15. Use the functions from the sklearn package. Find the best value for k. Normalize the dataset before applying the model. Display the training, validation, and testing accuracy, confusion matrix

In [1]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target
```

In [3]:
```
# Split the dataset into training, validation, and testing sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

In [4]:
```
# Normalize the dataset
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

In [5]:
```
# Find the best value for k using the validation set
best_k = 0
best_score = 0
for k in range(1, 21):
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X_train, y_train)
    score = accuracy_score(y_val, clf.predict(X_val))
    if score > best_score:
        best_k = k
        best_score = score
```

In [6]:
```
# Train the model using the best value for k
clf = KNeighborsClassifier(n_neighbors=best_k)
clf.fit(X_train, y_train)
```

Out[6]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

In [7]:
```
# Make predictions on the training, validation, and testing sets
y_train_pred = clf.predict(X_train)
y_val_pred = clf.predict(X_val)
y_test_pred = clf.predict(X_test)
```

In [8]:
```
# Calculate the training, validation, and testing accuracy
train_acc = accuracy_score(y_train, y_train_pred)
val_acc = accuracy_score(y_val, y_val_pred)
test_acc = accuracy_score(y_test, y_test_pred)
```

In [9]:
```
# Calculate the confusion matrix for the testing set
confusion_mat = confusion_matrix(y_test, y_test_pred)
```

In [10]:
```
# Display the results
print("Best value for k:", best_k)
print("Training Accuracy:", train_acc)
print("Validation Accuracy:", val_acc)
print("Testing Accuracy:", test_acc)
print("Confusion Matrix:\n", confusion_mat)
```

```
Best value for k: 1
Training Accuracy: 1.0
Validation Accuracy: 1.0
Testing Accuracy: 0.9565217391304348
Confusion Matrix:
 [[6 0 0]
 [0 9 1]
 [0 0 7]]
```

In [ ]:

Question 4. Create a linear regression model using ordinary least squares estimation. Find the best fit line for the dataset 'salary.csv' using the above model. Display the training and testing dataset in the scatter plot and draw the best fit line in the same. Also find the MSE and R2 for the testing dataset.

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

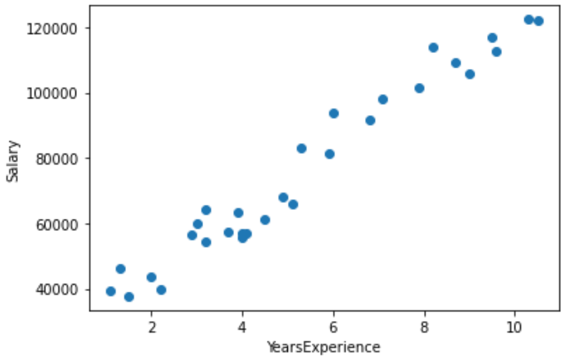
In [4]:
```
data=pd.read_csv('Salary_data.csv')
data
```

Out[4]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

In [5]:
```
X=data['YearsExperience']
y=data['Salary']
```

In [6]:
```
plt.scatter(X,y)
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.show()
```



In [7]:
```
class OLS:
    def __init__(self):
        self.coeff=[0,0]

    def train(self, X, y):
        diff_x=X-np.mean(X)
        diff_y=y-np.mean(y)
        self.coeff[1] = sum(diff_x*diff_y)/sum(diff_x*diff_x)
        self.coeff[0] = np.mean(y)-self.coeff[1]*np.mean(X)

    def predict(self,X):
        y=self.coeff[0]+self.coeff[1]*X
        return y

    def RSS(self,y,y_pred):
        error=y-y_pred
        rss=sum(error*error)
        return rss

    def TSS(self,y):
        error=y-y.mean()
        tss=sum(error*error)
        return tss

    def R2(self,rss,tss):
        r2= 1-(rss/tss)
        return r2
```

In [8]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

In [9]:
```
model=OLS()
model.train(X_train,y_train)
```

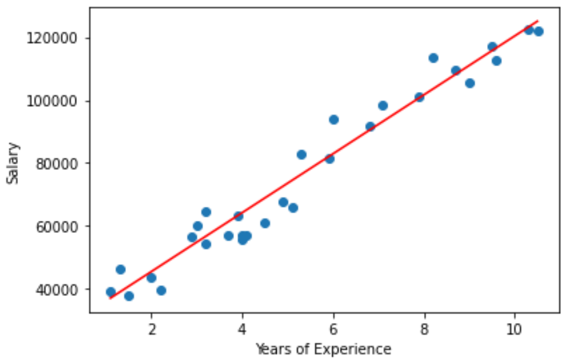
In [10]:
```
print('Value of coefficient b:', model.coeff[0])
```

```
Value of coefficient b: 26777.39134119761
```

In [11]:
```
print('Value of coefficient m: ',model.coeff[1])
```

```
Value of coefficient m:  9360.261286193658
```

In [12]:
```
plt.scatter(X, y)
plt.plot(X, model.coeff[0] + (model.coeff[1] * X), 'r-')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```



In [13]:
```
y_pred=model.predict(X_test)
y_pred
```

Out[13]:
```
2      40817.783270
28    123188.082589
13     65154.462615
10     63282.410357
26    115699.873560
24    108211.664531
27    116635.899689
11     64218.436486
17     76386.776158
Name: YearsExperience, dtype: float64
```

In [14]:
```
#calculating R2 Score for testing data

rss=model.RSS(y_test,y_pred)
tss=model.TSS(y_test)
r2=model.R2(rss,tss)
print('R2 Score: ', r2)
```

```
R2 Score:  0.9740993407213511
```

In [15]:
```
#calculating MSE for testing data

mse=rss/len(y_test)
print('MSE: ',mse)
```

```
MSE:  23370078.800832942
```

In [ ]:

Question 5 Consider the dataset california_housing from sklearn . Find the correlation b/w the different attributes of this dataset. Using the least square estimation method from sklearn, find the best fit line. Also find the error.

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
```

In [2]:
```
california_housing = fetch_california_housing(as_frame=True).frame
```

In [3]:
```
correlation = california_housing.corr()
print(correlation)
```

```
               MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  \
MedInc       1.000000 -0.119034  0.326895  -0.062040    0.004834  0.018766
HouseAge    -0.119034  1.000000 -0.153277  -0.077747   -0.296244  0.013191
AveRooms     0.326895 -0.153277  1.000000   0.847621   -0.072213 -0.004852
AveBedrms   -0.062040 -0.077747  0.847621   1.000000   -0.066197 -0.006181
Population   0.004834 -0.296244 -0.072213  -0.066197    1.000000  0.069863
AveOccup     0.018766  0.013191 -0.004852  -0.006181    0.069863  1.000000
Latitude    -0.079809  0.011173  0.106389   0.069721   -0.108785  0.002366
Longitude   -0.015176 -0.108197 -0.027540   0.013344    0.099773  0.002476
MedHouseVal  0.688075  0.105623  0.151948  -0.046701   -0.024650 -0.023737

             Latitude  Longitude  MedHouseVal
MedInc      -0.079809  -0.015176     0.688075
HouseAge     0.011173  -0.108197     0.105623
AveRooms     0.106389  -0.027540     0.151948
AveBedrms    0.069721   0.013344    -0.046701
Population  -0.108785   0.099773    -0.024650
AveOccup     0.002366   0.002476    -0.023737
Latitude     1.000000  -0.924664    -0.144160
Longitude   -0.924664   1.000000    -0.045967
MedHouseVal -0.144160  -0.045967     1.000000
```

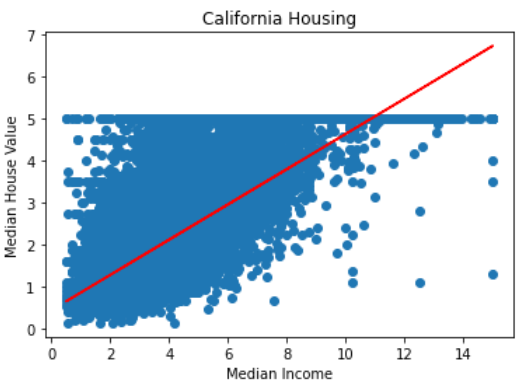
In [4]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
```

In [5]:
```
# prepare the data
X = california_housing['MedInc'].values.reshape(-1, 1)
y = california_housing['MedHouseVal'].values
```

In [6]:
```
# fit the linear regression model
model = LinearRegression()
model.fit(X, y)
```

Out[6]:
```
▾ LinearRegression
LinearRegression()
```

In [7]:
```
import matplotlib.pyplot as plt

# plot the data
plt.scatter(X, y)

# plot the best fit line
plt.plot(X, model.predict(X), color='red')

# set the axis labels and title
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
plt.title('California Housing')

# show the plot
plt.show()
```



In [8]:
```
# compute the predicted values
y_pred = model.predict(X)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [9]:
```
mse = np.mean((y - y_pred)**2)
rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
Root Mean Squared Error: 0.8373357452616917
```

In [ ]:

Question 6 Consider the dataset 'Adveristing.csv'. Find the correlation coefficient between the input attributes TV, Radio, Newspaper and Output Attribute Sales. Use least square estimation method to find the line of regression b/w

1. TV and Sales
2. Radio and Sales
3. Newspaper and Sales For all of the above options, also draw a scatter plot and line of regression. Also find the error in each of the above.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [2]:
```
data=pd.read_csv('Advertising.csv')
data
```

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
...	...	...	...	...	...
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

200 rows × 5 columns

In [3]:
```
data.drop(['Unnamed: 0'], axis=1,inplace=True)
```

In [4]:
```
data
```

Out[4]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

In [5]:
```
corr_Matrix=data.corr()
corr_Matrix
```

Out[5]:

	TV	Radio	Newspaper	Sales
TV	1.000000	0.054809	0.056648	0.782224
Radio	0.054809	1.000000	0.354104	0.576223
Newspaper	0.056648	0.354104	1.000000	0.228299
Sales	0.782224	0.576223	0.228299	1.000000

In [6]:
```
class OLS:
    def __init__(self):
        self.coeff=[0,0]

    def train(self, X, y):
        diff_x=X-np.mean(X)
```

```
        diff_y=y-np.mean(y)
        self.coeff[1] = sum(diff_x*diff_y)/sum(diff_x*diff_x)
        self.coeff[0] = np.mean(y)-self.coeff[1]*np.mean(X)

    def predict(self,X):
        y=self.coeff[0]+self.coeff[1]*X
        return y

    def RSS(self,y,y_pred):
        error=y-y_pred
        rss=sum(error*error)
        return rss
```

1) TV and Sales

In [7]:
```
X=data['TV']
y=data['Sales']
```

In [8]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

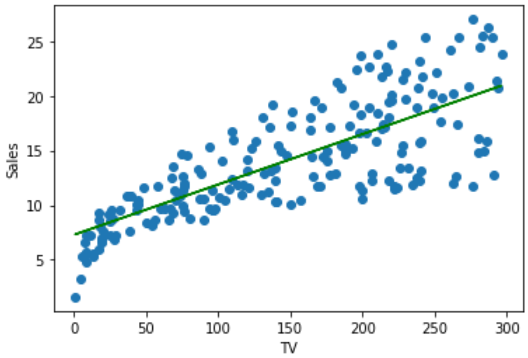
In [9]:
```
model=OLS()
model.train(X_train,y_train)
```

In [10]:
```
y_pred=model.predict(y_test)
```

In [11]:
```
m=model.coeff[1]
b=model.coeff[0]
m
```

Out[11]:
```
0.04600778960301719
```

In [12]:
```
plt.scatter(X,y)
plt.plot(X, b + (m*X),'g-')
plt.xlabel('TV')
plt.ylabel('Sales')
plt.show()
```



In [13]:
```
#calculating error
rss=model.RSS(y_test,y_pred)
rmse=rss/len(y_test)
print('Error: ', rmse)
```

```
Error:  57.16396652853107
```

2) Radio and Sales

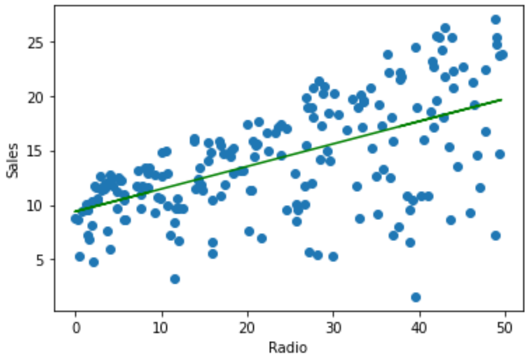
In [14]:
```
X2=data['Radio']
```

In [15]:
```
X2_train,X2_test,y_train,y_test=train_test_split(X2,y,test_size=0.2,random_state=0)
model2=OLS()
model2.train(X2_train,y_train)
y_pred2=model2.predict(y_test)
```

In [16]:
```
m2=model2.coeff[1]
b2=model2.coeff[0]
m2
```

Out[16]:
```
0.20651176537911176
```

In [24]:
```
plt.scatter(X2,y)
plt.plot(X2, b2 + (m2*X2),'g-')
plt.xlabel('Radio')
plt.ylabel('Sales')
plt.show()
```



In [18]:
```
#calculating error
rss2=model2.RSS(y_test,y_pred2)
rmse2=rss2/len(y_test)
print('Error:  ', rmse2)
```

```
Error:   21.068731505518777
```

3) Newspaper and Sales

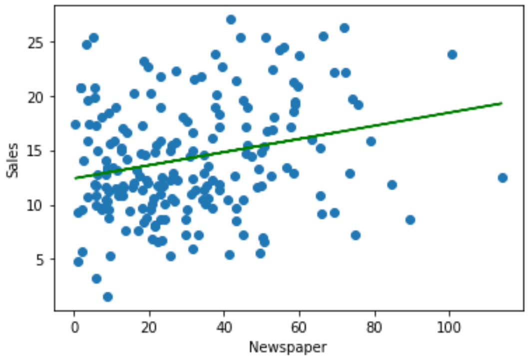
In [19]:
```
X3=data['Newspaper']
```

In [20]:
```
X3_train,X3_test,y_train,y_test=train_test_split(X3,y,test_size=0.2,random_state=0)
model3=OLS()
model3.train(X3_train,y_train)
y_pred3=model3.predict(y_test)
```

In [21]:
```
m3=model3.coeff[1]
b3=model3.coeff[0]
m3
```

Out[21]:
```
0.060303781082760924
```

In [22]:
```
plt.scatter(X3,y)
plt.plot(X3, b3 +(m3*X3), 'g-' )
plt.xlabel('Newspaper')
plt.ylabel('Sales')
plt.show()
```



In [23]:
```
#calculating error
rss3=model3.RSS(y_test,y_pred3)
rmse3=rss3/len(y_test)
print('Error:  ', rmse3)
```

```
Error:   27.790289949887047
```

In [ ]:

Question 7 Consider the dataset 'Adveristing.csv'. Find the best fit regression line between the input attributes TV, Radio, Newspaper and Output Attribute Sales using gradient descent method. Also find R2 .

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

In [2]:
```
data=pd.read_csv('Advertising.csv')
data
```

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
...	...	...	...	...	...
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

200 rows × 5 columns

In [3]:
```
data.drop(['Unnamed: 0'],axis=1,inplace=True)
data
```

Out[3]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

In [4]:
```
X=data[['TV','Radio','Newspaper']]
y=data['Sales']
X = (X - X.mean()) / X.std()
```

In [5]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=100)
```

In [6]:
```
class GradientDescentLinearRegression:
    def __init__(self, learning_rate=0.01, iterations=1000):

        self.lr=learning_rate
        self.epochs=iterations

    def fit(self,X,y):
        self.intercept=0
        self.coef=np.ones(X.shape[1])
        self.m=X.shape[0]

        for i in range(self.epochs):
            y_hat=np.dot(X,self.coef)+ self.intercept

            intercept_slope= (1/self.m) * np.sum(y_hat-y)
            coef_slope= (1/self.m) * np.dot((y_hat-y),X)

            self.intercept= self.intercept - (self.lr * intercept_slope)
            self.coef=self.coef - (self.lr * coef_slope)
```

```python
        print('intercept: ',self.intercept,' coefficients: ',self.coef)

    def predict(self,X):
        return np.dot(X,self.coef)+ self.intercept

    def R2(self,y,y_pred):
        error1=y-y_pred
        rss=sum(error1*error1)

        error2=y-y.mean()
        tss=sum(error2*error2)

        r2=1-(rss/tss)
        print('R2 Score: ', r2)
```

In [7]:
```python
model=GradientDescentLinearRegression()
```

In [8]:
```python
model.fit(X_train,y_train)
```

```
intercept:  13.886043484504372  coefficients:  [3.89917084 2.81208551 0.10540172]
```

In [9]:
```python
y_pred=model.predict(X_test)
y_pred
```

Out[9]:
```
array([10.62076197, 19.99413942, 16.91437469, 19.17147372, 20.94053131,
       13.12457547, 11.80804945, 12.31941946, 20.5716779 , 20.94652704,
       10.78689465, 19.55365974,  6.42945198, 15.22532221,  8.97434355,
        7.89920392, 16.22442464, 12.03126809, 17.08523146, 11.26368089,
       16.96967849,  9.76165934, 20.81874647, 17.20584915, 15.13517074,
       21.95530544, 19.19957602, 10.0757902 , 19.37433909, 14.85792172,
       14.36106439,  7.5532602 ,  9.97577221, 14.76227071,  7.21068292,
       13.59317008,  7.49745873, 11.70852527, 13.46164563, 15.23116183,
       17.18511437, 13.5618064 , 14.3126682 , 13.73254354, 11.88124218,
        8.77544014, 12.1309813 , 19.19700093,  9.08805467,  5.15619638,
       16.22718512, 18.13539965, 12.94874752, 16.85424109, 17.85540113,
       12.33667992,  4.36436588, 11.24823142, 16.10984204, 13.56522939])
```

In [10]:
```python
model.R2(y_test,y_pred)
```

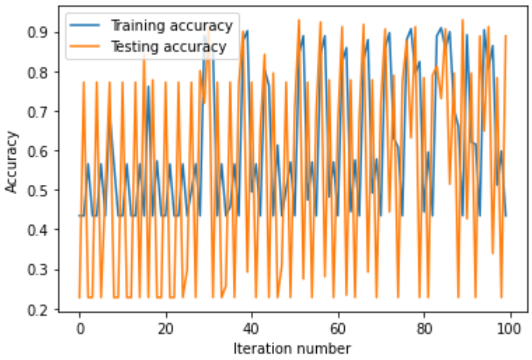
```
R2 Score:  0.9056737232402822
```

In [ ]:

Question 8 Use logistic regression to build a model to classify the breast cancer dataset Divide the dataset into training and testing in the ratio 70:30 . Print the confusion matrix, sensitivity, specificity. For each iteration of training, store the training and testing accuracy. Plot a graph showing training and testing accuracy Vs iteration no. Do not use sklearn logistic function.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer

# Load the breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Divide the dataset into training and testing sets
train_size = int(len(X) * 0.7)
X_train = X[:train_size]
y_train = y[:train_size]
X_test = X[train_size:]
y_test = y[train_size:]

# Define the logistic sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Define the logistic regression function
def logistic_regression(X, y, alpha=0.1, num_iterations=100):
    # Initialize the weights
    w = np.zeros(X.shape[1])
    # Initialize the lists to store the training and testing accuracy for each iteration
    train_acc = []
    test_acc = []
    # Iterate for the specified number of iterations
    for i in range(num_iterations):
        # Compute the predicted values
        y_pred = sigmoid(np.dot(X, w))
        # Compute the error
        error = y - y_pred
        # Update the weights
        w += alpha * np.dot(X.T, error)
        # Compute the training accuracy
        train_acc.append(np.mean((y_pred > 0.5) == y))
        # Compute the testing accuracy
        test_acc.append(np.mean((sigmoid(np.dot(X_test, w)) > 0.5) == y_test))
    # Return the weights and the lists of training and testing accuracy
    return w, train_acc, test_acc

# Add a column of ones to the feature matrices for the bias term
X_train = np.concatenate((np.ones((X_train.shape[0], 1)), X_train), axis=1)
X_test = np.concatenate((np.ones((X_test.shape[0], 1)), X_test), axis=1)

# Train the logistic regression model on the training set
w, train_acc, test_acc = logistic_regression(X_train, y_train)

# Print the confusion matrix, sensitivity, and specificity for the testing set
y_pred = sigmoid(np.dot(X_test, w))
y_pred[y_pred > 0.5] = 1
y_pred[y_pred <= 0.5] = 0
conf_matrix = np.zeros((2, 2))
for i in range(len(y_test)):
    conf_matrix[int(y_test[i]), int(y_pred[i])] += 1
tp = conf_matrix[1, 1]
tn = conf_matrix[0, 0]
fp = conf_matrix[0, 1]
fn = conf_matrix[1, 0]
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
print("Confusion matrix:")
print(conf_matrix)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)

# Plot the training and testing accuracy vs iteration number
plt.plot(train_acc, label="Training accuracy")
plt.plot(test_acc, label="Testing accuracy")
plt.xlabel("Iteration number")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_20132\404436993.py:20: RuntimeWarning: overflow encountered in exp
  return 1 / (1 + np.exp(-x))
Confusion matrix:
[[ 32.   7.]
 [ 12. 120.]]
Sensitivity: 0.9090909090909091
Specificity: 0.8205128205128205
```


Training accuracy
Testing accuracy
Accuracy
Iteration number
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100

In [ ]:

Question 9 Using logistic regression to build a model to classify the iris dataset. Divide the dataset into training and testing in the ratio 80:20 . Print the confusion matrix, sensitivity and specificity.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [2]:
```
class LogisticRegression:
    def __init__(self,learning_rate=0.01,num_iterations=1000):
        self.learning_rate=learning_rate
        self.num_iterations=num_iterations
        self.theta=None

    def sigmoid(self,z):
        return 1/(1+np.exp(-z))

    def fit(self,X,y):
        self.theta = np.zeros(X.shape[1])
        m=X.shape[0]

        for i in range(self.num_iterations):
            z=np.dot(X,self.theta)
            h=self.sigmoid(z)

            cost = (-1/m)*np.sum(y*np.log(h)+(1-y)*np.log(1-h))

            grad = (1/m)*np.dot(X.T,(h-y))

            self.theta -= self.learning_rate*grad

            if i%100==0:
                print("iteration {}: Cost={}".format(i,cost))

    def predict(self,X):
        z=np.dot(X,self.theta)
        h=self.sigmoid(z)
        y_pred=np.round(h).astype(int)
        return y_pred
```

In [3]:
```
data = load_iris()
X=data.data
y=data.target
```

In [4]:
```
X_train,X_test,y_train,y_test=train_test_split(data.data,data.target,test_size=0.2,random_state=42)
```

In [5]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [6]:
```
model = LogisticRegression(learning_rate=0.01,num_iterations=1000)
model.fit(X_train,y_train)
```

```
iteration 0: Cost=0.6931471805599453
iteration 100: Cost=-0.28592081322953067
iteration 200: Cost=-0.7717750516170782
iteration 300: Cost=-1.1568547886326463
iteration 400: Cost=-1.5078154119251628
iteration 500: Cost=-1.8434697875951882
iteration 600: Cost=-2.170888203543445
iteration 700: Cost=-2.4932756898070134
iteration 800: Cost=-2.8123041543115193
iteration 900: Cost=-3.1289458088636195
```

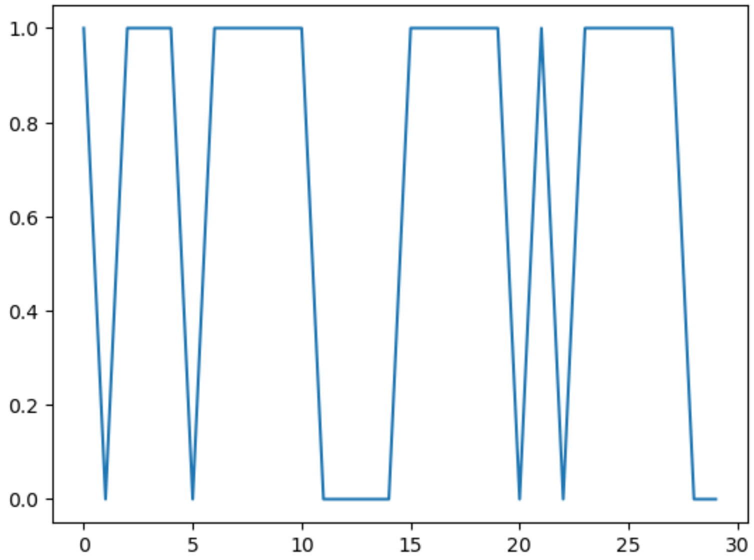
In [7]:
```
y_pred = model.predict(X_test)
```

In [8]:
```
y_pred
```

Out[8]:
```
array([1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 0, 0])
```

In [9]:
```
plt.plot(y_pred)
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x1be2d5f7940>]
```



In [10]:
```
print(confusion_matrix(y_test,y_pred))
```

```
[[10  0  0]
 [ 0  9  0]
 [ 0 11  0]]
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Question 10 Create a linear regression model using the gradient descent method. Create a class to represent the model with the following functions - init, fit and predict. Find the best fit line for the dataset Also find the MSE and R2 for the testing dataset.

In [1]:
```
import numpy as np
class LR_GD:
    def __init__(self):
        self.coeff=[0,0]

    def fit(self,x,y,epochs,lr):
        m = x.shape[0]      #no of samples
        for i in range(epochs):
            h = x*self.coeff[1] + self.coeff[0]  #hypothesis fucntion
            error = h - y
            cost = sum(error*error)/(2*m)
            temp_coeff = self.coeff[1] - (lr/m)*sum(error*x)
            temp_b = self.coeff[0] - (lr/m)*sum(error)
            self.coeff[1] = temp_coeff
            self.coeff[0] = temp_b

    def predict(self,x):
        y = self.coeff[0] + self.coeff[1]*x
        return y

    def RSS(self,y,y_pred):
        error=y-y_pred
        rss=sum(error*error)
        return rss

    def TSS(self,y):
        error=y-y.mean()
        tss=sum(error*error)

        return rss

    def R2(self,rss,tss):
        r2=1-(rss/tss)
        return r2

    def MSE(self,rss,y):
        mse=rss/len(y)
        return mse
```

In [2]:
```
X = [42,37,30,50,43,47,46]
Y = [173,149,123,201,175,188,186]
x = np.array(X)
y = np.array(Y)
model = LR_GD()
model.fit(x,y,30,0.001)
y_pred = model.predict(46)
print("Number of Passengers when temp is 46 is {}".format(y_pred))
```

```
Number of Passengers when temp is 46 is 185.77284473800063
```

In [3]:
```
print('m: ', model.coeff[1])
print('b: ',model.coeff[0])
```

```
m:  4.036421855683573
b:  0.09743937655628325
```

In [ ]:

Question 11 Consider the dataset wine from sklearn. Using PCA reduce the dimensionality of the dataset to 5. Build a classification model using gaussian naive bayes classifier. Find the training accuracy and test accuracy

In [1]:
```
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
wine=load_wine()
```

In [2]:
```
wine.feature_names
```

Out[2]:
```
['alcohol',
 'malic_acid',
 'ash',
 'alcalinity_of_ash',
 'magnesium',
 'total_phenols',
 'flavanoids',
 'nonflavanoid_phenols',
 'proanthocyanins',
 'color_intensity',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline']
```

In [3]:
```
wine.target_names
```

Out[3]:
```
array(['class_0', 'class_1', 'class_2'], dtype='<U7')
```

In [4]:
```
df=pd.DataFrame(data=wine.data,columns=wine.feature_names)
df
```

Out[4]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04
...	...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	7.30	0.70
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.20	0.59
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.30	0.60
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.20	0.61

178 rows × 13 columns

In [5]:
```
X_train,X_test,y_train,y_test=train_test_split(df,wine.target,test_size=0.2,random_state=0)
X_train.shape
```

Out[5]:
```
(142, 13)
```

In [6]:
```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(X_train,y_train)
```

Out[6]:
```
▾ GaussianNB
GaussianNB()
```

In [7]:
```
y_pred=nb.predict(X_test)
y_pred
```

Out[7]:
```
array([0, 2, 1, 0, 1, 1, 0, 2, 1, 1, 2, 2, 0, 0, 2, 1, 0, 0, 2, 0, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 2, 0, 0, 1, 0, 0, 0])
```

In [8]:
```
from sklearn import metrics
```

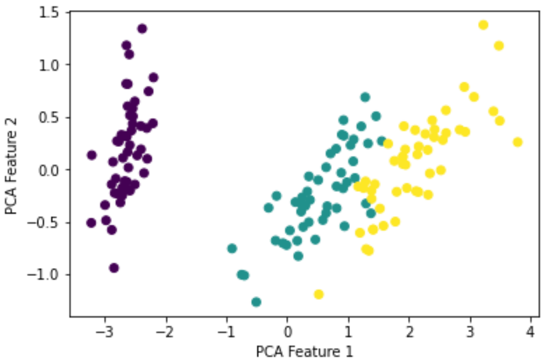
In [9]:
```
print('Accuracy: ', metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy:  0.9166666666666666
```

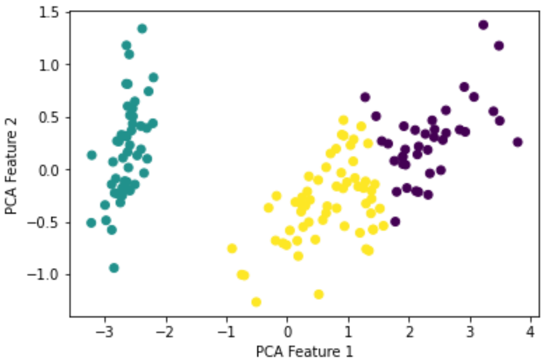
In [ ]:

Question 12 Consider the dataset iris. Apply the PCA method to select the best 2 features. Using these features plot the scatter graph. Apply k-means clustering algorithm to cluster the transformed dataset into 3 clusters.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# Load the iris dataset
data = load_iris()
X = data.data
y = data.target

# Apply PCA to select the best 2 features
pca = PCA(n_components=2)
X_transformed = pca.fit_transform(X)

# Plot the scatter graph using the transformed dataset
plt.scatter(X_transformed[:, 0], X_transformed[:, 1], c=y)
plt.xlabel("PCA Feature 1")
plt.ylabel("PCA Feature 2")
plt.show()

# Apply k-means clustering algorithm to cluster the transformed dataset into 3 clusters
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_transformed)
y_pred = kmeans.predict(X_transformed)

# Plot the scatter graph with the clusters
plt.scatter(X_transformed[:, 0], X_transformed[:, 1], c=y_pred)
plt.xlabel("PCA Feature 1")
plt.ylabel("PCA Feature 2")
plt.show()
```




```
C:\Users\admin\AppData\Local\Programs\Python\Python310\lib\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 [ ]:

Question 13 Write a program to implement a single layer perceptron model. Train this for solving a AND problem with 3 variables.

In [1]:
```python
import numpy as np

X = np.array([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1],
              [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]])


y = np.array([0, 0, 0, 0, 0, 0, 0, 1])


learning_rate = 0.1
epochs = 50


np.random.seed(0)
weights = np.random.rand(3)
bias = np.random.rand()


def threshold(z):
    return 1 if z >= 0 else 0

# Train the perceptron model on the input data and target output
for epoch in range(epochs):
    for i in range(len(X)):
        # Calculate the weighted sum of inputs and bias
        weighted_sum = np.dot(X[i], weights) + bias

        # Calculate the output using the activation function
        output = threshold(weighted_sum)

        # Calculate the error and update the weights and bias accordingly
        error = y[i] - output
        weights += learning_rate * error * X[i]
        bias += learning_rate * error

    # Print the accuracy of the perceptron model on the current epoch
    accuracy = np.sum(X.dot(weights) + bias == y) / len(y)
    print(f"Epoch {epoch + 1}/{epochs}, accuracy: {accuracy:.2f}")
```

```
Epoch 1/50, accuracy: 0.00
Epoch 2/50, accuracy: 0.00
Epoch 3/50, accuracy: 0.00
Epoch 4/50, accuracy: 0.00
Epoch 5/50, accuracy: 0.00
Epoch 6/50, accuracy: 0.00
Epoch 7/50, accuracy: 0.00
Epoch 8/50, accuracy: 0.00
Epoch 9/50, accuracy: 0.00
Epoch 10/50, accuracy: 0.00
Epoch 11/50, accuracy: 0.00
Epoch 12/50, accuracy: 0.00
Epoch 13/50, accuracy: 0.00
Epoch 14/50, accuracy: 0.00
Epoch 15/50, accuracy: 0.00
Epoch 16/50, accuracy: 0.00
Epoch 17/50, accuracy: 0.00
Epoch 18/50, accuracy: 0.00
Epoch 19/50, accuracy: 0.00
Epoch 20/50, accuracy: 0.00
Epoch 21/50, accuracy: 0.00
Epoch 22/50, accuracy: 0.00
Epoch 23/50, accuracy: 0.00
Epoch 24/50, accuracy: 0.00
Epoch 25/50, accuracy: 0.00
Epoch 26/50, accuracy: 0.00
Epoch 27/50, accuracy: 0.00
Epoch 28/50, accuracy: 0.00
Epoch 29/50, accuracy: 0.00
Epoch 30/50, accuracy: 0.00
Epoch 31/50, accuracy: 0.00
Epoch 32/50, accuracy: 0.00
Epoch 33/50, accuracy: 0.00
Epoch 34/50, accuracy: 0.00
Epoch 35/50, accuracy: 0.00
Epoch 36/50, accuracy: 0.00
Epoch 37/50, accuracy: 0.00
Epoch 38/50, accuracy: 0.00
Epoch 39/50, accuracy: 0.00
Epoch 40/50, accuracy: 0.00
Epoch 41/50, accuracy: 0.00
Epoch 42/50, accuracy: 0.00
Epoch 43/50, accuracy: 0.00
Epoch 44/50, accuracy: 0.00
Epoch 45/50, accuracy: 0.00
Epoch 46/50, accuracy: 0.00
Epoch 47/50, accuracy: 0.00
Epoch 48/50, accuracy: 0.00
Epoch 50/50, accuracy: 0.00
```

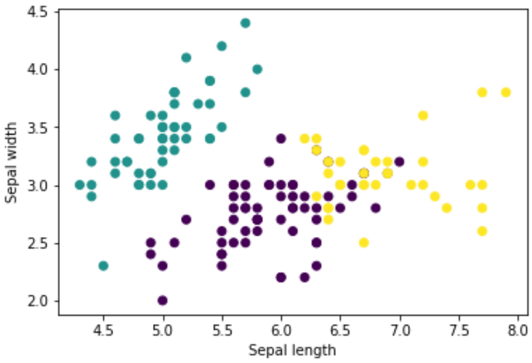
In [ ]:

```

```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Question 14 Consider the dataset iris. Apply hierarchical clustering algorithm to cluster the dataset into 3 clusters.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering

# Load the iris dataset
data = load_iris()
X = data.data

# Apply hierarchical clustering algorithm
model = AgglomerativeClustering(n_clusters=3)
model.fit(X)
labels = model.labels_

# Plot the scatter graph to visualize the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```



In [ ]:

Question 15 Write a program to implement 2-layered ANN for classifying digits datasets from sklearn. Use 70% data for training the model and check the accuracy of the model on remaining 30% data. Use softmax activation function in the last layer and relu function in the hidden layer

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.utils import to_categorical
```

In [2]:
```
data=load_digits()
```

In [3]:
```
X_train,X_test,y_train,y_test=train_test_split(data.data, data.target, test_size=0.3,random_state=0)
```

In [4]:
```
ann=Sequential()
ann.add(Dense(units=128, input_dim=X_train.shape[1]))
ann.add(Activation('relu'))
ann.add(Dense(units=10))
ann.add(Activation('softmax'))
```

In [5]:
```
ann.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [6]:
```
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)
ann.fit(X_train,y_train_cat,epochs=50,batch_size=32,validation_data=(X_test,y_test_cat))
```

```
Epoch 1/50
40/40 [==============================] - 1s 10ms/step - loss: 2.2648 - accuracy: 0.5091 - val_loss: 0.6395 - val_accuracy: 0.
7889
Epoch 2/50
40/40 [==============================] - 0s 3ms/step - loss: 0.3718 - accuracy: 0.8846 - val_loss: 0.3106 - val_accuracy: 0.9
056
Epoch 3/50
40/40 [==============================] - 0s 3ms/step - loss: 0.2083 - accuracy: 0.9348 - val_loss: 0.2748 - val_accuracy: 0.9
000
Epoch 4/50
40/40 [==============================] - 0s 4ms/step - loss: 0.1471 - accuracy: 0.9602 - val_loss: 0.1980 - val_accuracy: 0.9
407
Epoch 5/50
40/40 [==============================] - 0s 3ms/step - loss: 0.1121 - accuracy: 0.9722 - val_loss: 0.1735 - val_accuracy: 0.9
463
Epoch 6/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0801 - accuracy: 0.9801 - val_loss: 0.1681 - val_accuracy: 0.9
537
Epoch 7/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0656 - accuracy: 0.9881 - val_loss: 0.1713 - val_accuracy: 0.9
426
Epoch 8/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0573 - accuracy: 0.9873 - val_loss: 0.1464 - val_accuracy: 0.9
648
Epoch 9/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0426 - accuracy: 0.9944 - val_loss: 0.1242 - val_accuracy: 0.9
667
Epoch 10/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0352 - accuracy: 0.9960 - val_loss: 0.1366 - val_accuracy: 0.9
685
Epoch 11/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0316 - accuracy: 0.9952 - val_loss: 0.1226 - val_accuracy: 0.9
704
Epoch 12/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0313 - accuracy: 0.9968 - val_loss: 0.1199 - val_accuracy: 0.9
704
Epoch 13/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0226 - accuracy: 0.9968 - val_loss: 0.1175 - val_accuracy: 0.9
648
Epoch 14/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0177 - accuracy: 0.9984 - val_loss: 0.1170 - val_accuracy: 0.9
704
Epoch 15/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0172 - accuracy: 0.9984 - val_loss: 0.1152 - val_accuracy: 0.9
704
Epoch 16/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0171 - accuracy: 0.9984 - val_loss: 0.1069 - val_accuracy: 0.9
704
Epoch 17/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0186 - accuracy: 0.9960 - val_loss: 0.1107 - val_accuracy: 0.9
704
Epoch 18/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0137 - accuracy: 0.9976 - val_loss: 0.1120 - val_accuracy: 0.9
685
Epoch 19/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0132 - accuracy: 0.9992 - val_loss: 0.1020 - val_accuracy: 0.9
722
Epoch 20/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0103 - accuracy: 0.9992 - val_loss: 0.1041 - val_accuracy: 0.9
722
Epoch 21/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0087 - accuracy: 1.0000 - val_loss: 0.0996 - val_accuracy: 0.9
741
Epoch 22/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0091 - accuracy: 0.9992 - val_loss: 0.0989 - val_accuracy: 0.9
741
Epoch 23/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0070 - accuracy: 1.0000 - val_loss: 0.0971 - val_accuracy: 0.9
759
Epoch 24/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0061 - accuracy: 1.0000 - val_loss: 0.0980 - val_accuracy: 0.9
741
Epoch 25/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0058 - accuracy: 1.0000 - val_loss: 0.1032 - val_accuracy: 0.9
704
Epoch 26/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0054 - accuracy: 1.0000 - val_loss: 0.0993 - val_accuracy: 0.9
741
Epoch 27/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0986 - val_accuracy: 0.9
722
Epoch 28/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0043 - accuracy: 1.0000 - val_loss: 0.0990 - val_accuracy: 0.9
741
Epoch 29/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0041 - accuracy: 1.0000 - val_loss: 0.0957 - val_accuracy: 0.9
759
Epoch 30/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 0.0975 - val_accuracy: 0.9
759
Epoch 31/50
```

```
40/40 [==============================] - 0s 4ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.0982 - val_accuracy: 0.9
722
Epoch 32/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.0987 - val_accuracy: 0.9
759
Epoch 33/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.1001 - val_accuracy: 0.9
759
Epoch 34/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0030 - accuracy: 1.0000 - val_loss: 0.0992 - val_accuracy: 0.9
741
Epoch 35/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0029 - accuracy: 1.0000 - val_loss: 0.0965 - val_accuracy: 0.9
741
Epoch 36/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.0976 - val_accuracy: 0.9
759
Epoch 37/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.0983 - val_accuracy: 0.9
741
Epoch 38/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 0.0987 - val_accuracy: 0.9
759
Epoch 39/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0023 - accuracy: 1.0000 - val_loss: 0.0977 - val_accuracy: 0.9
759
Epoch 40/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.0987 - val_accuracy: 0.9
778
Epoch 41/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.0978 - val_accuracy: 0.9
759
Epoch 42/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.0981 - val_accuracy: 0.9
759
Epoch 43/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.0985 - val_accuracy: 0.9
759
Epoch 44/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.0969 - val_accuracy: 0.9
778
Epoch 45/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.0983 - val_accuracy: 0.9
759
Epoch 46/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.0966 - val_accuracy: 0.9
759
Epoch 47/50
40/40 [==============================] - 0s 4ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.0978 - val_accuracy: 0.9
778
Epoch 48/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.0974 - val_accuracy: 0.9
759
Epoch 49/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.0970 - val_accuracy: 0.9
778
Epoch 50/50
40/40 [==============================] - 0s 3ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.0956 - val_accuracy: 0.9
778
```

Out[6]:

```
<keras.callbacks.History at 0x160721031f0>
```

In [7]:

```python
loss,accuracy=ann.evaluate(X_test, y_test_cat, batch_size=32)
print("Test Loss:", loss)
print("Test Accuracy: " + str(accuracy))
```

```
17/17 [==============================] - 0s 2ms/step - loss: 0.0956 - accuracy: 0.9778
Test Loss: 0.09560243785381317
Test Accuracy: 0.9777777791023254
```

In [ ]: