1. Implement a function to train a linear regression model using stochastic gradient descent (SGD) with mini-batch updates. The function should include options for different learning rates, batch sizes, and a regularization term.

In [1]:

```
# Input:
# X = [[1, 1], [1, 2], [1, 3], [1, 4]]
# y = [3, 4, 5, 6]
# learning_rate = 0.01
# batch_size = 2
# num_iterations = 1000
# regularization_term = 0.1
# random_state=42
# Expected Output:
# Optimized coefficients: array([0, 1])
```

In [2]:

```
X = [[1, 1], [1, 2], [1, 3], [1, 4]]
y = [3, 4, 5, 6]
learning_rate = 0.01
batch_size = 2
num_iterations = 1000
regularization_term = 0.1
random_state=42
```

In [3]:

```
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
```

In [4]:

```
model=SGDRegressor()
```

In [5]:

```
s=StandardScaler()
X=s.fit_transform(X)
n_samples=X.shape[0]
for i in range(num_iterations):
    a=0
    while a<n_samples:
        e=a+batch_size
        X_s=X[a:e]
        y_s=y[a:e]
        model.partial_fit(X_s,y_s)
        a=e
```

In [6]:

```
model.coef_
```

Out[6]:

```
array([0.        , 1.11652325])
```

2. Write a function to implement linear regression with Lasso regularization (L1 regularization) using coordinate descent. The function should allow for different regularization parameters and tolerance levels for convergence.

In [7]:

```
# Input:
# X = [[1, 1], [1, 2], [1, 3], [1, 4]]
# y = [3, 4, 5, 6]
# regularization_param = 0.1
# tolerance = 0.001
# Expected Output:
# Optimized coefficients: [1.6, 0.8]
```

In [8]:

```
X = [[1, 1], [1, 2], [1, 3], [1, 4]]
y = [3, 4, 5, 6]
regularization_param = 0.1
tolerance = 0.001
```

In [9]:

```
from sklearn.linear_model import Lasso
```

In [10]:

```
model=Lasso(alpha=regularization_param,tol=tolerance)
```

In [11]:

```
model.fit(X,y)
```

Out[11]:

```
▾ Lasso
Lasso(alpha=0.1, tol=0.001)
```

In [12]:

```
model.coef_
```

Out[12]:

```
array([0.  , 0.92])
```

3.Create a program that performs logistic regression with L1 regularization (Lasso) using coordinate descent

In [13]:

```
# Input:
# X = [[1, 2], [2, 3], [3, 4], [4, 5]]
# y = [0, 0, 1, 1]
# regularization_param = 0.1
# tolerance = 0.001
```

In [14]:

```
X = [[1, 2], [2, 3], [3, 4], [4, 5]]
y = [0, 0, 1, 1]
regularization_param = 0.1
tolerance = 0.001
```

In [15]:

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

In [16]:

```
model=LogisticRegression(penalty='l1',C=1/regularization_param,solver='liblinear')
```

In [17]:

```
model.fit(X,y)
```

Out[17]:

```
LogisticRegression
LogisticRegression(C=10.0, penalty='l1', solver='liblinear')
```

In [18]:

```
model.coef_
```

Out[18]:

```
array([[2.24590863, 0.        ]])
```

4.Write a program to calculate the area under the ROC curve (AUC) for a logistic regression model.

In [19]:

```
# Input:
# y_true = [0, 1, 1, 0, 1]
# y_pred = [0.2, 0.8, 0.6, 0.3, 0.9]
```

In [20]:

```
y_true = [0, 1, 1, 0, 1]
y_pred = [0.2, 0.8, 0.6, 0.3, 0.9]
```

In [21]:

```
from sklearn.metrics import roc_auc_score
```

In [22]:

```
roc_auc_score(y_true,y_pred)
```

Out[22]:

```
1.0
```

5.Write a program to calculate the log loss (binary cross-entropy) for a logistic regression model using vectorized operations.

In [23]:

```
# Input:
# y_true = [0, 1, 1, 0]
# y_pred = [0.2, 0.8, 0.9, 0.3]
```

In [24]:

```
y_true = [0, 1, 1, 0]
y_pred = [0.2, 0.8, 0.9, 0.3]
```

In [25]:

```
from sklearn.metrics import log_loss
```

In [26]:

```
log_loss(y_true,y_pred)
```

Out[26]:

```
0.22708064055624455
```

6. Write a program to predict the class labels for new input data using a trained decision tree classifier.

In [27]:

```
#input
# X=[[1,2],[3,4],[4,5],[5,6]]
# y=[0,0,1,1]
```

In [28]:

```
X=[[1,2],[3,4],[4,5],[5,6]]
y=[0,0,1,1]
```

In [29]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [30]:

```
model=DecisionTreeClassifier()
```

In [31]:

```
model.fit(X,y)
```

Out[31]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [32]:

```
X_new=[[3,4],[6,7]]
```

In [33]:

```
model.predict(X_new)
```

Out[33]:

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

7.Create a function to visualize a decision tree using a graph representation.

In [34]:

```
# Input:
# X = [[1, 2], [2, 3], [3, 4], [4, 5]]
# y = [0, 0, 1, 1]
```

In [35]:

```
X = [[1, 2], [2, 3], [3, 4], [4, 5]]
y = [0, 0, 1, 1]
```

In [36]:

```
from sklearn.tree import DecisionTreeClassifier,plot_tree
import matplotlib.pyplot as plt
```

In [37]:

```
model=DecisionTreeClassifier()
model.fit(X,y)
```

Out[37]:

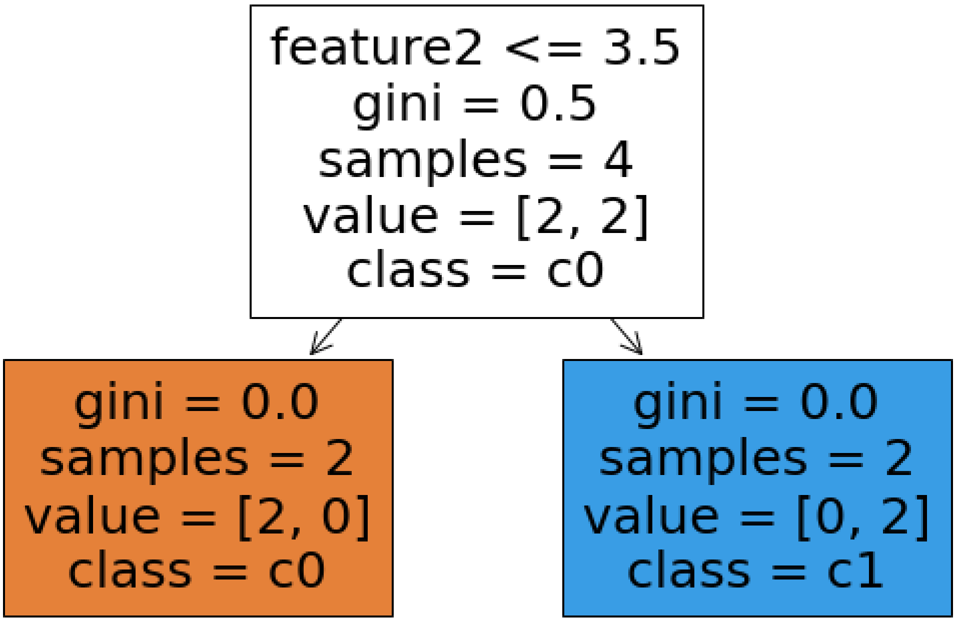
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [38]:

```
plt.figure(figsize=(10,6))
plot_tree(model,filled=True,class_names=['c0','c1'],feature_names=[f'feature{i+1}' for i
```

Out[38]:

```
[Text(0.5, 0.75, 'feature2 <= 3.5\ngini = 0.5\nsamples = 4\nvalue = [2, 2]
\nclass = c0'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]\nclass = c0'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = c1')]
```



8.Write a program to perform hierarchical clustering using the complete linkage method.

In [39]:

```
# Input:
# X = [[1, 2], [2, 3], [10, 12], [11, 13], [20, 25], [22, 24]]
```

In [40]:

```
from sklearn.cluster import AgglomerativeClustering
```

In [41]:

```
model=AgglomerativeClustering()
```

In [42]:

```
X = [[1, 2], [2, 3], [10, 12], [11, 13], [20, 25], [22, 24]]
```

In [43]:

```
model.fit(X)
```

Out[43]:

```
▾ AgglomerativeClustering
AgglomerativeClustering()
```

In [44]:

```
model.labels_
```

Out[44]:

```
array([0, 0, 0, 0, 1, 1], dtype=int64)
```

9.Implement a program to perform density-based clustering using the DBSCAN algorithm.

In [45]:

```
# Input:
# X = [[1, 2], [2, 3], [10, 12], [11, 13], [20, 25], [22, 24]]
# epsilon = 3
# min_samples = 2
```

In [46]:

```
X = [[1, 2], [2, 3], [10, 12], [11, 13], [20, 25], [22, 24]]
epsilon = 3
min_samples = 2
```

In [47]:

```
from sklearn.cluster import DBSCAN
```

In [48]:

```
model=DBSCAN(min_samples=min_samples,eps=epsilon)
```

In [49]:

```
model.fit_predict(X)
```

Out[49]:

```
array([0, 0, 1, 1, 2, 2], dtype=int64)
```

10. Write a program to perform clustering using the fuzzy C-means algorithm.

In [50]:

```
#input:
#X = [[1, 2], [2, 3], [10, 12], [11, 13], [20, 25], [22, 24]]
```

In [51]:

```
from skfuzzy.cluster import cmeans
import numpy as np
```

In [52]:

```
X = np.array([[1, 2], [2, 3], [10, 12], [11, 13], [20, 25], [22, 24]])
```

In [53]:

```
model,u,_,_,_,_,_=cmeans(X.T,c=2,m=2,error=0.01,maxiter=5000)
```

In [54]:

```
u.argmax(axis=0)
```

Out[54]:

```
array([1, 1, 1, 1, 0, 0], dtype=int64)
```