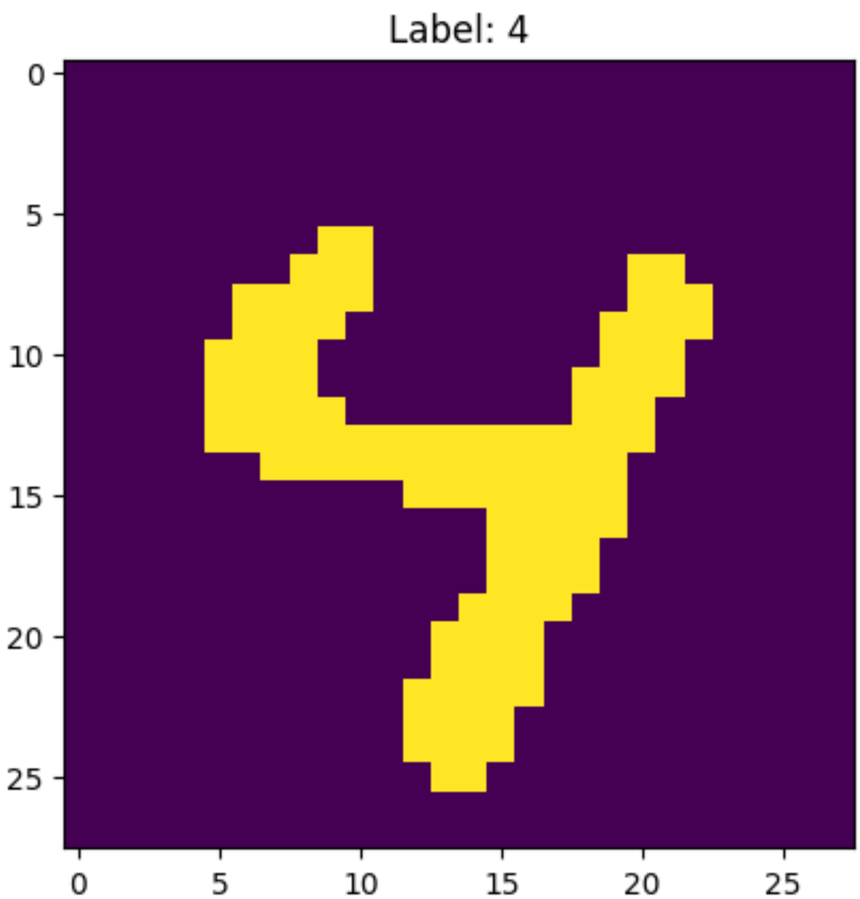
Problem 1: PCA

In [1]:
```
import torch
import torchvision
from matplotlib import pyplot as plt
import numpy as np
```

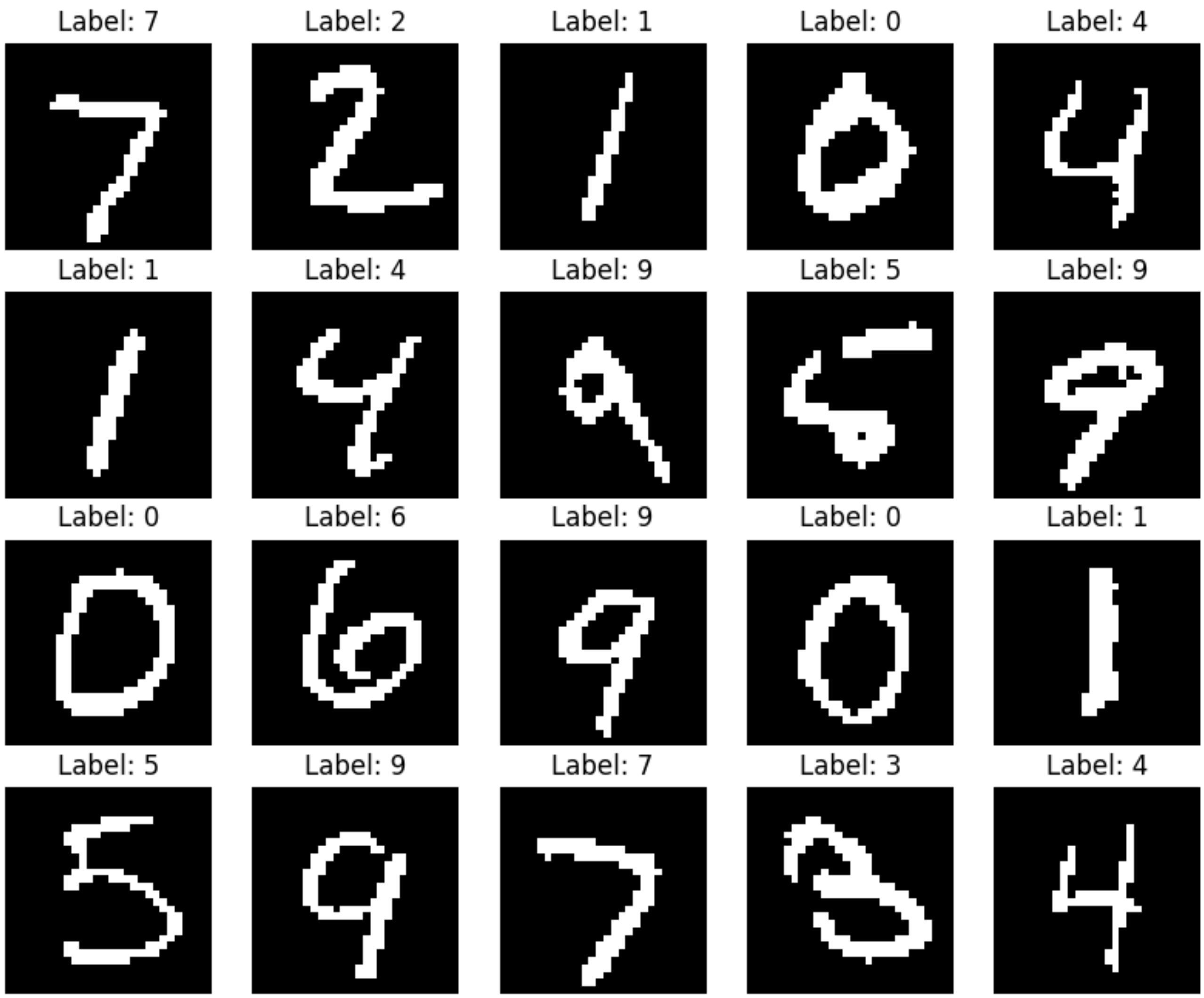
In [2]:
```
# Get MNIST
data_transform = torchvision.transforms.Compose ([
    torchvision.transforms.ToTensor(),
    lambda x: torch.floor(x*255/128).squeeze(dim=0),
])
mnist_test = torchvision.datasets.MNIST(
    root='./temp', train=False, transform=data_transform, download=False
)
x, y = mnist_test.__getitem__(1010)
plt.imshow(x.numpy())
plt.title(f'Label: {y}');

# All images and all labels
X = mnist_test.data     # 10000 x 28 x 28
Y = mnist_test.targets  # 10000
```



In [3]:
```
# Question 1a
print("Number of samples: ", X.shape[0])
print("Number of feature dimensions: ", X.shape[1]* X.shape[2])
```

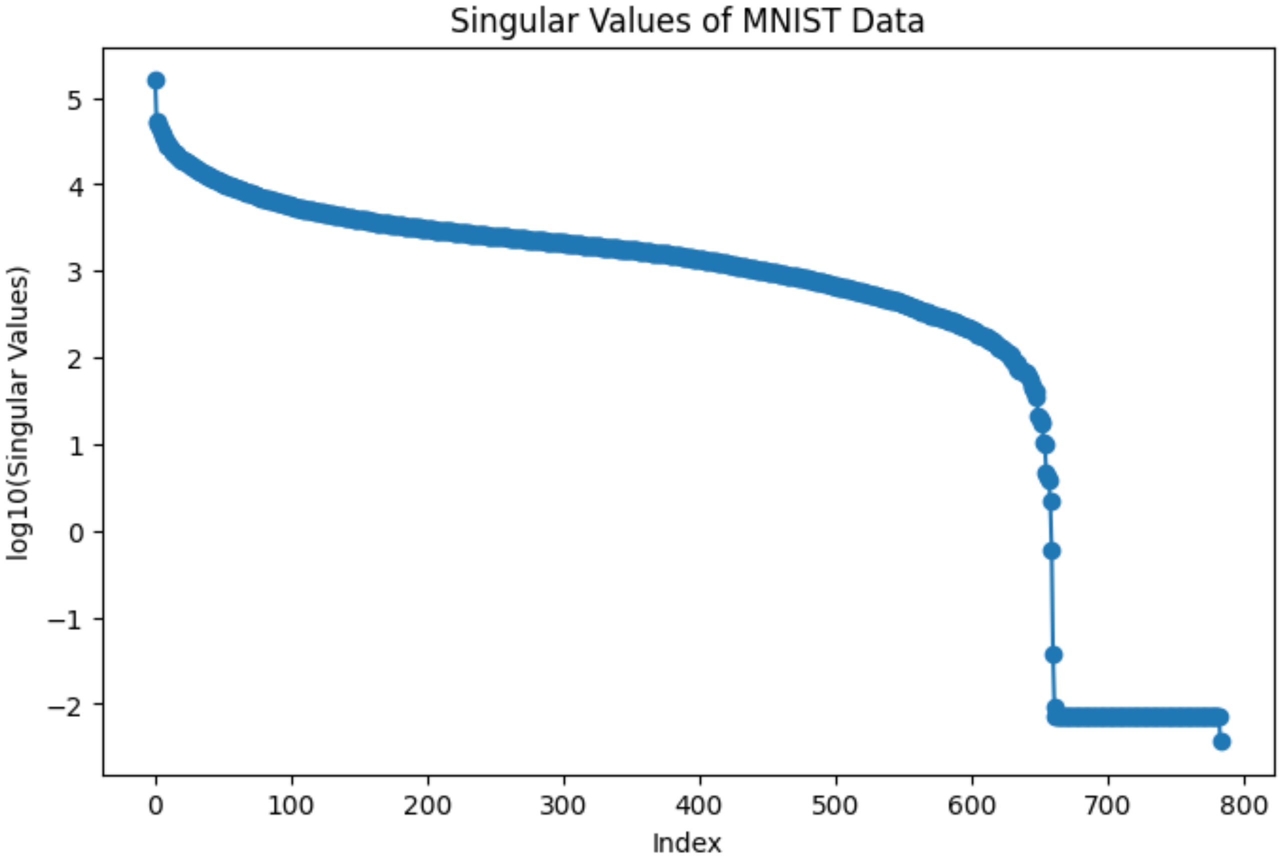
```
Number of samples:  10000
Number of feature dimensions:  784
```

In [4]:
```
# Question 1b
fig, axes = plt.subplots(4, 5, figsize=(10, 8))
for i, ax in enumerate(axes.flatten()):
    img, label = mnist_test[i]
    ax.imshow(img.numpy(), cmap='gray')
    ax.set_title(f"Label: {label}")
    ax.axis('off')

plt.show()
```



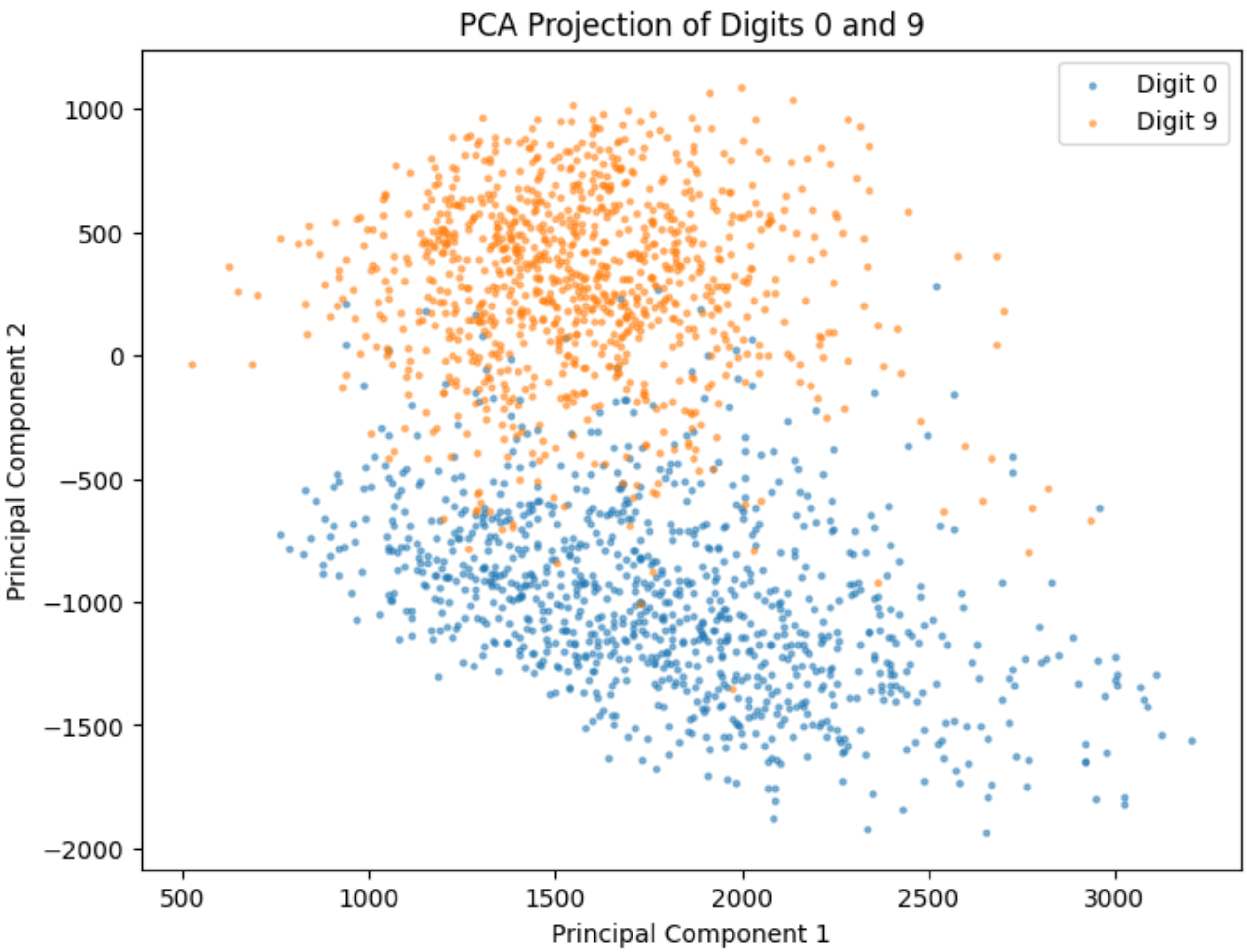
In [5]:
```
# Question 1c
X = mnist_test.data.reshape(10000, 784).float()

U, S, Vt = torch.linalg.svd(X, full_matrices=False)

plt.figure(figsize=(8, 5))
```

```python
plt.plot(np.log10(S.numpy()), marker='o')
plt.xlabel("Index")
plt.ylabel("log10(Singular Values)")
plt.title("Singular Values of MNIST Data")
plt.show()
```



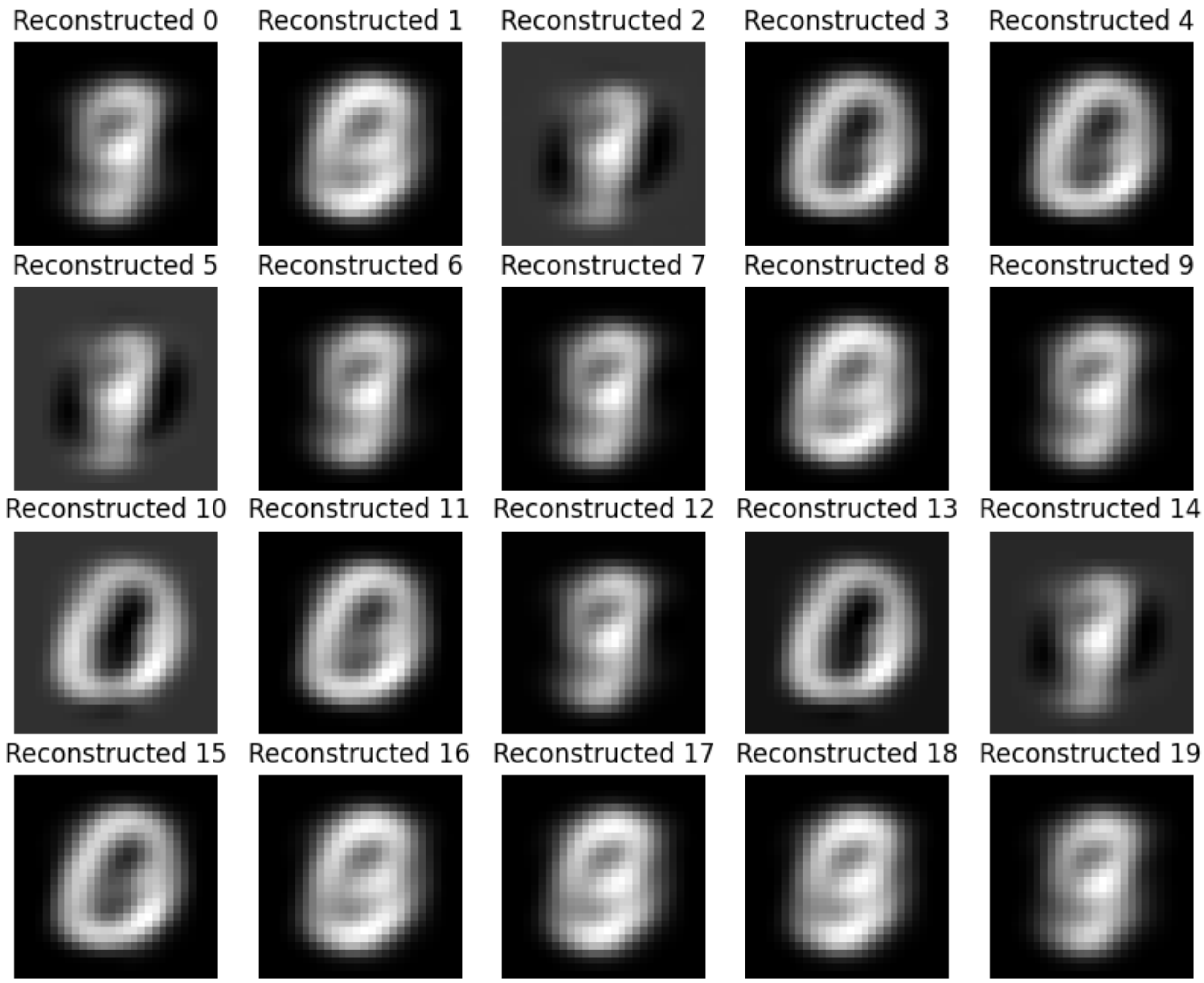
In [6]:

```python
# Question 1d
V = Vt[:2, :].T
Z = X @ V
Y = mnist_test.targets.numpy()
Z_0 = Z[Y == 0]
Z_9 = Z[Y == 9]
plt.figure(figsize=(8, 6))
plt.scatter(Z_0[:, 0], Z_0[:, 1], label='Digit 0', alpha=0.5, s=5)
plt.scatter(Z_9[:, 0], Z_9[:, 1], label='Digit 9', alpha=0.5, s=5)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Projection of Digits 0 and 9")
plt.legend()
plt.show()
```



In [7]:

```
# Question 1e
X_hat = Z @ V.T
fig, axes = plt.subplots(4, 5, figsize=(10, 8))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(X_hat[i].reshape(28, 28), cmap='gray')
    ax.set_title(f"Reconstructed {i}")
    ax.axis('off')

plt.show()
```



Problem 2: kNN

In [8]:
```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
import sklearn.metrics as metrics

# for easier reading np
np.set_printoptions(precision=3,suppress=True)
```

In [9]:
```
with open('./winequality-red.csv', 'r') as f:
  temp = np.genfromtxt(f,delimiter=',',skip_header=1)
  X = temp[:,:-1]
  y = temp[:,-1]
  Labels = np.unique(y)    # class labels
  print('Class labels are: ', Labels)
```

```
Class labels are:  [3. 4. 5. 6. 7. 8.]
```

In [10]:
```
# Question 2a
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
print("Training data shape: ", X_train.shape)
print("Testing data shape: ", X_test.shape)
```

```
Training data shape:  (1279, 11)
Testing data shape:  (320, 11)
```

In [11]:
```
# Question 2b
from sklearn.model_selection import KFold, cross_val_score
kf = KFold(n_splits=3, shuffle=True)

accuracy_scores = {}
for k in range(1, 50, 1):
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=kf, scoring='accuracy
    accuracy_scores[k] = np.mean(scores)

best_k = max(accuracy_scores, key=accuracy_scores.get)
print("Best k:", best_k)
```

```
Best k: 1
```

In [12]:
```
# Question 2c
from sklearn.metrics import confusion_matrix, classification_report
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Optional: Print detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, zero_division=0))
```

```
Confusion Matrix:
[[ 0  0  1  1  1  0]
 [ 2  0  4  4  0  0]
 [ 0  7 87 34  7  0]
 [ 0  6 28 73 13  0]
 [ 0  0  5 15 28  1]
 [ 0  0  0  2  1  0]]

Classification Report:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         3
         4.0       0.00      0.00      0.00        10
         5.0       0.70      0.64      0.67       135
         6.0       0.57      0.61      0.59       120
         7.0       0.56      0.57      0.57        49
         8.0       0.00      0.00      0.00         3

    accuracy                           0.59       320
   macro avg       0.30      0.30      0.30       320
weighted avg       0.59      0.59      0.59       320
```

Problem 3: Decision Trees & Ensembles

In [32]:

```python
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, GridS
from sklearn.preprocessing import StandardScaler
import sklearn.metrics as metrics
```

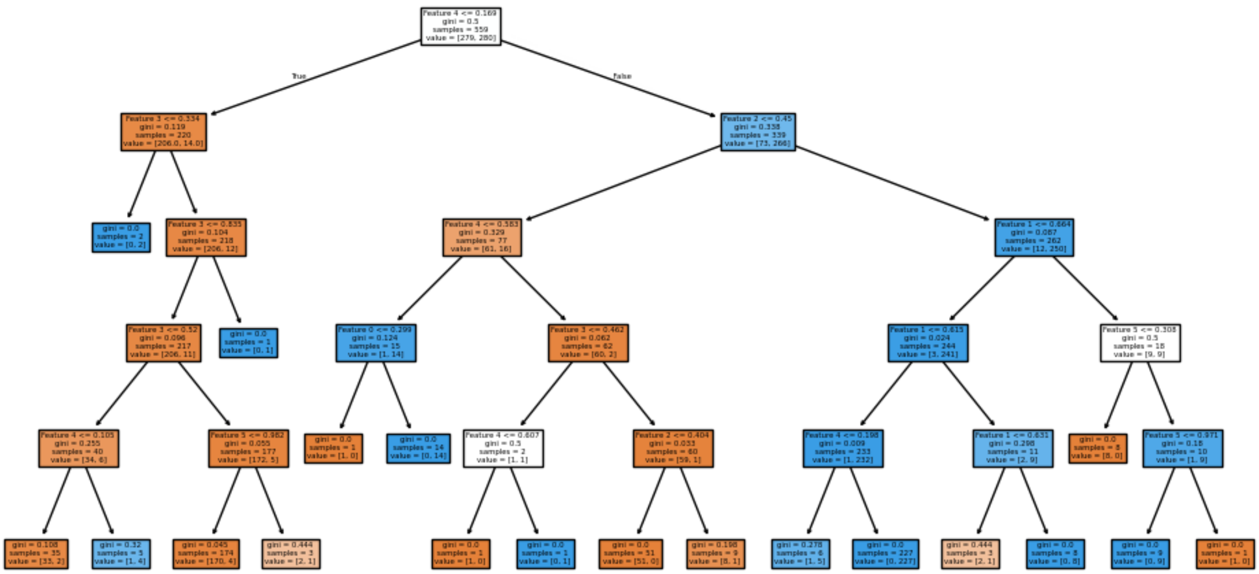
In [14]:

```python
with open('./Xray.csv', 'r') as f:
  X = np.genfromtxt(f,delimiter=',',skip_header=1)
  X, y = X[:, :-1], X[:, -1].astype(int)
```

In [77]:

```python
# Part a) Use sklearn.tree.DecisionTreeClassifier
# Documentation: https://scikit-learn.org/dev/modules/generated/sklearn.tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.30, st
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.

# Hyperparameter tuning using GridSearchCV
param_grid = {'max_depth': [3, 5, 10, None], 'min_samples_split': [2, 5, 10]

dt = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid, cv=5,
dt.fit(X_train, y_train)

# Best model
best_dt = dt.best_estimator_

# Evaluate on the test set
y_pred_dt = best_dt.predict(X_test)
```

```python
accuracy_dt = metrics.accuracy_score(y_test, y_pred_dt)
cm_dt = confusion_matrix(y_test, y_pred_dt)

# Plot decision tree
plt.figure(figsize=(12, 6))
plot_tree(best_dt, filled=True, feature_names=[f'Feature {i}' for i in range
plt.show()

print("Decision Tree Test Accuracy:", accuracy_dt)
print("Decision Tree Confusion Matrix:\n", cm_dt)
```



```
Decision Tree Test Accuracy: 0.9916666666666667
Decision Tree Confusion Matrix:
 [[59  1]
 [ 0 60]]
```

In [60]:

```python
# Part b) Use sklearn.ensemble.RandomForestClassifier
# Documentation: https://scikit-learn.org/1.5/modules/generated/sklearn.ense
from sklearn.ensemble import RandomForestClassifier
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.30, st
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.

param_grid = {
    'n_estimators': [50, 100, 200],  # Number of trees
    'max_depth': [10, 20, None],  # Maximum depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split a
    'min_samples_leaf': [1, 2, 4]  # Minimum samples at a leaf node
}

# rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=
# rf.fit(X_train, y_train)
best_rf = RandomForestClassifier(random_state=42, max_depth=10, min_samples_
best_rf.fit(X_train, y_train)
# best_rf = rf.best_estimator_
# print("Best Hyperparameters:", rf.best_params_)
```

```
y_pred_rf = best_rf.predict(X_test)

accuracy_rf = metrics.accuracy_score(y_test, y_pred_rf)
cm_rf = confusion_matrix(y_test, y_pred_rf)

print("\nRandom Forest Test Accuracy:", accuracy_rf)
print("\nRandom Forest Confusion Matrix:\n", cm_rf)
```

```
Random Forest Test Accuracy: 0.9833333333333333

Random Forest Confusion Matrix:
 [[60  0]
 [ 2 58]]
```

In [76]:

```
# Part c) Use xgboost.xgb
# Documentation: https://xgboost.readthedocs.io/en/stable/python/python_intr
from xgboost import XGBClassifier
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.30, st
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.

xgb = XGBClassifier(
    random_state=42,
    eval_metric="logloss",
    tree_method="hist",
    n_estimators=50,
    learning_rate=0.1,
    max_depth=3,
    subsample=0.8,
    colsample_bytree=0.8
)
xgb.fit(X_train, y_train)

# best_xgb = xgb.best_estimator_
# print("Best Hyperparameters:", xgb.best_params_)

y_pred_xgb = xgb.predict(X_test)

accuracy_xgb = metrics.accuracy_score(y_test, y_pred_xgb)
cm_xgb = confusion_matrix(y_test, y_pred_xgb)

print("\nXGBoost Test Accuracy:", accuracy_xgb)
print("\nXGBoost Confusion Matrix:\n", cm_xgb)
```

```
XGBoost Test Accuracy: 0.9333333333333333

XGBoost Confusion Matrix:
 [[57  3]
 [ 5 55]]
```

Part d)

i) The accuracy for the xgboost and Random Forest model is similar, both in the 95% < range

ii) The training time was the same for both or at least not noticable with this little data

iii) We can get good performance with a lower number of esitmators with the xgboost model