1. Feature extraction using mnist dataset

```
from sklearn.datasets import fetch_openml
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd

X, y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False)

print("Original shape of X:", X.shape)
print("Original shape of y:", y.shape)
X_pca = PCA(n_components=20).fit_transform(StandardScaler().fit_transform(X))

df_pca = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(20)])
print("\nPCA Features (first 5 rows):")
print(df_pca.head())
```

2.Feature extraction using iris dataset

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

iris = load_iris()

X, y =iris.data, iris.target

pairs=[(0, 1, 'Sepal Length', 'Sepal Width'), (2, 3, 'Petal Length', 'Petal width')]
for i, j, xlabel, ylabel in pairs:
  plt.figure(figsize=(6, 4))
  plt.scatter(X[:, i], X[:, j], c=y, cmap='viridis', edgecolors='k', s=100)
  plt.title(f'(xlabel) vs (ylabel)')
  plt.xlabel(f'(xlabel) (cm)')
  plt.ylabel(f'(ylabel) (cm)')
  plt.colorbar(label='Species')
  plt.grid(True)
  plt.show()
```

3.Feature selection using Kbest

```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest, f_classif
import pandas as pd
iris =load_iris()
X_selected =SelectKBest(f_classif, k=2).fit_transform(iris.data, iris.target)
features=[iris.feature_names[i] for i in SelectKBest (f_classif, k=2)

    .fit(iris.data, iris.target).get_support(indices=True)]
print("Selected feature names:", features)
print("\nSelected Features (first 5 rows):")
print(pd.DataFrame(X_selected, columns=features).head())
```

4. Feature selection using iris dataset

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
import matplotlib.pyplot as plt
iris=load_iris()
rfe=RFE(LogisticRegression(max_iter=1000), n_features_to_select=2).fit(iris.data, iris.target)
print("Selected Features:", [f for f, s in zip(iris.feature_names, rfe.support_) if s])
plt.bar(iris.feature_names, rfe.ranking_, color='skyblue', edgecolor='k')
plt.xlabel("Feature")
plt.ylabel("Ranking (1 = selected)")
plt.title("Feature Ranking by RFE")
plt.show()
```

5.Decision tree regression

```
import numpy as np, matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
X = np.sort(5* np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel(); y[::5] += 0.5 * (0.5 - np.random.rand(16))
model=DecisionTreeRegressor (max_depth=3).fit(X, y)
X_test = np.arange(0, 5, 0.01) [:, None]
y_pred = model.predict(X_test)
plt.scatter(X, y, c="darkorange", edgecolor="k", s=20, label="Data")
plt.plot(X_test, y_pred, "navy", lw=2, label="Prediction")
plt.xlabel("X"); plt.ylabel("y"); plt.title("Decision Tree Regression")
plt.legend(); plt.grid(); plt.show()
```

6.Random Forest Regression

```
import numpy as np, matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
X= np.sort(5*np.random.rand(80,1),0)
y = np.sin(X).ravel(); y[::5] += 0.5*(0.5-np.random.rand(16))
model= RandomForestRegressor(n_estimators=100, random_state=42).fit(X,y)
X_test = np.arange(0,5,0.01)[:, None]; y_pred=model.predict(X_test)
plt.scatter (X,y,c="darkorange", edgecolor="k", s=20, label="Data")
plt.plot(X_test,y_pred, "navy", lw=2, label="Prediction")
plt.xlabel(""); plt.ylabel("y"); plt.title("Random Forest Regression")
plt.legend(); plt.grid(); plt.show()
```

7.svm classifier

```
import numpy as np, matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.preprocessing import StandardScaler
iris = datasets.load_iris()
X, Y=iris.data[:, :2], iris.target
X, Y = X[Y != 2], Y[Y != 2]
X = StandardScaler().fit_transform(X)
model=svm.SVC(kernel='linear', C=1).fit(X, Y)
XX, YY =np.meshgrid (np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 300),
            np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 300))
Z=model.predict(np.c_[XX.ravel(), YY.ravel()]).reshape(XX.shape)
plt.contourf(XX, YY, Z, cmap=plt.cm.coolwarm, alpha=0.3)
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.coolwarm, edgecolors='k')
plt.title("SVM Decision Boundary"); plt.grid(True); plt.show()
```

8.KNN

```
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

import numpy as np, matplotlib.pyplot as plt
X, y = datasets.load_iris(return_X_y=True)

X = StandardScaler().fit_transform(X[:, :2])
knn=KNeighborsClassifier(5).fit(X, Y)
XX, YY=np.meshgrid(np.arange(X[:,0].min()-1, X[:, 0].max()+1, 0.02),

          np.arange(X[:,1].min()-1, X[:,1].max()+1, 0.02))
Z =knn.predict(np.c_[XX.ravel(),YY.ravel()]).reshape(XX.shape)
plt.contourf(XX, YY, Z, cmap="Pastel1", alpha=0.8)
plt.scatter(X[:,0], X[:,1], c=y, cmap="Set1", edgecolors="k")
plt.title("KNN (k=5)"); plt.grid(True)
plt.show()
```

9. kmeans clustering

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
X,_=make_blobs(n_samples=300, centers=3, cluster_std=0.6, random_state=8)
y_kmeans=KMeans (n_clusters=3, random_state=0).fit_predict(X)
plt.scatter (X[:,0], X[:,1], c=y_kmeans, cmap="viridis", s=50)
plt.scatter(KMeans (n_clusters=3, random_state=0).fit(X).cluster_centers_[:,0], KMeans
(n_clusters=3, random_state=0).fit(X).cluster_centers_[:,1], c="red", s=200, marker="X")
plt.title("K-Means Clustering")
plt.show()
```

10.d b scan clustering

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
X,_= make_moons(n_samples=300, noise=0.05, random_state=0)
X = StandardScaler().fit_transform(X)
labels= DBSCAN(eps=0.3, min_samples=5).fit_predict(X)
for l in set(labels):
   c = "k" if l== -1 else plt.cm.Set1(l/max(labels))
   plt.scatter(X[labels == l, 0], X[labels == l, 1], c=[c],
           label=("Noise" if l==-1 else f"Cluster {l}"))
plt.title("DBSCAN Clustering")
plt.legend()
plt.show()
```

11. Ann for binary classification

```
 from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = keras.Sequential([
   keras.Input(shape=(X_train.shape[1],)),
   layers.Dense(16, activation='relu'),
   layers.Dense(8, activation='relu'),
   layers.Dense(1, activation='sigmoid')
])
model.compile(
   optimizer='adam',
   loss='binary_crossentropy',
   metrics=['accuracy']
)
history = model.fit(
   X_train, y_train,
   validation_data=(X_test, y_test),
   epochs=20,
   batch_size=16
)
plt.figure(figsize=(8,4))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.legend()
plt.show()
plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.legend()
plt.show()
```

12.boston housing

```
 from tensorflow.keras.datasets import boston_housing
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
(X_train, y_train), (X_test, y_test) = boston_housing.load_data()
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = keras.Sequential([
   layers.Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
   layers.Dense(1)
])
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
history = model.fit(X_train, y_train, epochs=30, batch_size=16, validation_data=(X_test, y_test))
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.legend()
plt.show()
plt.plot(history.history['mae'], label='train_mae')
plt.plot(history.history['val_mae'], label='val_mae')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.legend()
plt.show()
```