Program For Linear regression.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Generate some example data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the Linear Regression model
model = LinearRegression()

# Train the model using the training sets
model.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = model.predict(X_test)

# Print coefficients
print("Coefficients: \n", model.coef_)
print("Intercept: \n", model.intercept_)

# Plot outputs
plt.scatter(X_test, y_test, color='black', label='Actual data')
plt.plot(X_test, y_pred, color='blue', linewidth=3, label='Regression line')

plt.xlabel("X")
plt.ylabel("y")
plt.title("Linear Regression")
plt.legend()
plt.show()

# Evaluate the model
print(f"Mean squared error: {np.mean((y_pred - y_test) ** 2)}")
print(f"Coefficient of determination (R^2): {model.score(X_test, y_test)}")
```

Program For Multivariate Linear Regression.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Generate some example data
np.random.seed(0)
X1 = 2 * np.random.rand(100, 1)
X2 = 3 * np.random.rand(100, 1)
y = 4 + 3 * X1 + 2 * X2 + np.random.randn(100, 1)

# Combine X1 and X2 into a single matrix
X = np.hstack((X1, X2))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the Linear Regression model
model = LinearRegression()

# Train the model using the training sets
model.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = model.predict(X_test)

# Print coefficients
print("Coefficients: \n", model.coef_)
print("Intercept: \n", model.intercept_)

# 3D Plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_test[:, 0], X_test[:, 1], y_test, color='black', label='Actual data')
ax.scatter(X_test[:, 0], X_test[:, 1], y_pred, color='blue', label='Predicted data')

ax.set_xlabel("X1")
ax.set_ylabel("X2")
ax.set_zlabel("y")
plt.title("Multivariate Linear Regression")
plt.legend()
plt.show()

# Evaluate the model
print(f"Mean squared error: {np.mean((y_pred - y_test) ** 2)}")
print(f"Coefficient of determination (R^2): {model.score(X_test, y_test)}")
```

Program For Logistics Regression.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Generate some example data
np.random.seed(0)
X = np.random.rand(100, 2)  # 100 samples with 2 features
y = (X[:, 0] + X[:, 1] > 1).astype(int)  # Binary target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the Logistic Regression model
model = LogisticRegression()

# Train the model using the training sets
model.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Optional: Visualize the decision boundary
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Spectral)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.Spectral)

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Logistic Regression Decision Boundary')
plt.show()
```

Program for SVM:

```
# Load the important packages
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

# Load the datasets
cancer = load_breast_cancer()
X = cancer.data[:, :2]  # Use only the first two features for visualization
y = cancer.target

# Build the model
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)

# Train the model
svm.fit(X, y)

# Plot Decision Boundary
DecisionBoundaryDisplay.from_estimator(
   svm,
   X,
   response_method="predict",
   cmap=plt.cm.Spectral,
   alpha=0.8,
   xlabel=cancer.feature_names[0],
   ylabel=cancer.feature_names[1],
)

# Scatter plot
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolors='k')
plt.title("SVM Decision Boundary on Breast Cancer Dataset")
plt.show()
```

Program for DBSCAN:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Generate synthetic data
n_samples = 750
X, _ = make_blobs(n_samples=n_samples, centers=4, cluster_std=0.60, random_state=0)

# Standardize features
X = StandardScaler().fit_transform(X)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=10)
dbscan.fit(X)

# Get cluster labels
labels = dbscan.labels_

# Number of clusters (excluding noise)
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
print(f"Number of clusters found: {n_clusters}")
print(f"Number of noise points: {list(labels).count(-1)}")

# Plotting the results
plt.figure(figsize=(10, 6))

# Plotting different clusters
unique_labels = set(labels)
for label in unique_labels:
    if label == -1:
        # Noise points
        plt.scatter(X[labels == label, 0], X[labels == label, 1], s=50, c='black', marker='x', label='Noise')
    else:
        plt.scatter(X[labels == label, 0], X[labels == label, 1], s=50, label=f'Cluster {label}')

plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Program for PCA: Principal Component Analysis

```
# Import all libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer
# Load the breast cancer dataset
data = load_breast_cancer()
# Check the output classes
print(data['target_names'])
# Check the input attributes
print(data['feature_names'])
# Construct a DataFrame using pandas
df1 = pd.DataFrame(data['data'], columns=data['feature_names'])
# Scale data before applying PCA
scaling = StandardScaler()
# Use fit and transform method
Scaled_data = scaling.fit_transform(df1)
# Set n_components=3
principal = PCA(n_components=3)
x = principal.fit_transform(Scaled_data)
# Check the dimensions of data after PCA
print(x.shape)
# Check the values of eigen vectors produced by principal components
print(principal.components_)
# 2D Scatter Plot
plt.figure(figsize=(10, 10))
plt.scatter(x[:, 0], x[:, 1], c=data['target'], cmap='plasma')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA of Breast Cancer Dataset')
plt.colorbar(label='Classes')
plt.show()
# 3D Scatter Plot
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10, 10))
axis = fig.add_subplot(111, projection='3d')
axis.scatter(x[:, 0], x[:, 1], x[:, 2], c=data['target'], cmap='plasma')
axis.set_xlabel("PC1", fontsize=10)
axis.set_ylabel("PC2", fontsize=10)
axis.set_zlabel("PC3", fontsize=10)
axis.set_title('3D PCA of Breast Cancer Dataset')
plt.show()
# Check how much variance is explained by each principal component
print(principal.explained_variance_ratio_)
```