```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
import seaborn as sns

def load_credit_data():
    # Sample hardcoded data (can be replaced by real dataset)
    data = {
        'Income': [40000, 60000, 25000, 50000, 70000, 100000, 120000, 15000,
35000, 80000],
        'Age': [25, 45, 35, 40, 29, 55, 33, 60, 48, 30],
        'LoanAmount': [1000, 2000, 500, 1500, 1800, 2500, 3000, 700, 1100,
2300],
        'CreditScore': [0, 1, 0, 1, 1, 1, 1, 0, 0, 1]  # 1 = Good, 0 = Bad
    }
    return pd.DataFrame(data)

def preprocess_data(df):
    X = df[['Income', 'Age', 'LoanAmount']]
    y = df['CreditScore']
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return train_test_split(X_scaled, y, test_size=0.3, random_state=42)

def train_classifier(X_train, y_train):
    model = LogisticRegression()
    model.fit(X_train, y_train)
    return model

def evaluate_classifier(model, X_test, y_test):
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print("\n--- Evaluation Results ---")
    print(f"Accuracy: {acc:.2f}")
    print("Classification Report:\n", classification_report(y_test, y_pred))
    return y_test.values, y_pred, confusion_matrix(y_test, y_pred)

def plot_results(y_test, y_pred, cm):
    # Confusion Matrix
    sns.heatmap(cm, annot=True, fmt='d', cmap="YlGnBu", xticklabels=["Bad",
"Good"], yticklabels=["Bad", "Good"])
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

    # Actual vs Predicted
    x = np.arange(len(y_test))
    plt.figure()
    plt.bar(x - 0.2, y_test, width=0.4, label='Actual', color='gray')
    plt.bar(x + 0.2, y_pred, width=0.4, label='Predicted', color='orange')
    plt.xticks(x)
    plt.xlabel("Test Sample Index")
    plt.ylabel("Credit Score Class")
    plt.title("Actual vs Predicted Credit Scores")
    plt.legend()
    plt.show()

def main():
```

```
    print("=== Credit Score Classification Application ===")
    df = load_credit_data()
    X_train, X_test, y_train, y_test = preprocess_data(df)
    model = train_classifier(X_train, y_train)
    y_test, y_pred, cm = evaluate_classifier(model, X_test, y_test)
    plot_results(y_test, y_pred, cm)

if __name__ == "__main__":
    main()
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

def load_data():
    iris = load_iris()
    X = iris.data
    y = iris.target
    labels = iris.target_names
    return X, y, labels

def preprocess(X, y):
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return train_test_split(X_scaled, y, test_size=0.3, random_state=42)

def train_knn(X_train, y_train, k=3):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    return knn

def evaluate_model(model, X_test, y_test, labels):
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"\nAccuracy: {acc:.2f}")
    print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=labels))
    return y_test, y_pred, confusion_matrix(y_test, y_pred)

def visualize(cm, labels):
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=labels,
yticklabels=labels)
    plt.title("Confusion Matrix - KNN Iris Classifier")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

def main():
    print("=== Iris Flower Classification using KNN ===")
    X, y, labels = load_data()
    X_train, X_test, y_train, y_test = preprocess(X, y)
    model = train_knn(X_train, y_train, k=3)
    y_test_vals, y_pred_vals, cm = evaluate_model(model, X_test, y_test, labels)
    visualize(cm, labels)

if __name__ == "__main__":
    main()
5
```

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

def load_data():
    # Synthetic sample data for demo
    data = {
        'Mileage': [15000, 30000, 45000, 60000, 75000, 90000, 105000, 120000,
135000, 150000],
        'Age': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        'EngineSize': [1.2, 1.4, 1.6, 1.8, 2.0, 1.2, 1.4, 1.6, 1.8, 2.0],
        'Price': [25000, 22000, 20000, 18000, 16000, 14000, 12000, 10000, 8000,
6000]
    }
    return pd.DataFrame(data)

def train_model(df):
    X = df[['Mileage', 'Age', 'EngineSize']]
    y = df['Price']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    print(f"Model Coefficients: {model.coef_}")
    print(f"Intercept: {model.intercept_}")
    print(f"R2 Score: {r2_score(y_test, y_pred):.2f}")
    print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred):.2f}")

    return y_test, y_pred

def visualize_results(y_test, y_pred):
    plt.figure()
    plt.plot(y_test.values, label='Actual Price', marker='o')
    plt.plot(y_pred, label='Predicted Price', marker='x')
    plt.title("Car Price Prediction: Actual vs Predicted")
    plt.xlabel("Test Sample")
    plt.ylabel("Car Price")
    plt.legend()
    plt.grid(True)
    plt.show()

def main():
    print("=== Car Price Prediction using Linear Regression ===")
    df = load_data()
    y_test, y_pred = train_model(df)
    visualize_results(y_test, y_pred)

if __name__ == "__main__":
    main()
```

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

def load_data():
    # Synthetic sample house data
    data = {
        'Area': [1000, 1500, 1800, 2400, 3000, 3500, 4000, 4200, 5000, 5500],
        'Bedrooms': [2, 3, 3, 4, 4, 4, 5, 5, 6, 6],
        'Bathrooms': [1, 2, 2, 3, 3, 3, 4, 4, 5, 5],
        'Price': [150000, 200000, 220000, 300000, 350000, 380000, 450000,
470000, 550000, 580000]
    }
    return pd.DataFrame(data)

def train_model(df):
    X = df[['Area', 'Bedrooms', 'Bathrooms']]
    y = df['Price']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    print(f"Model Coefficients: {model.coef_}")
    print(f"Intercept: {model.intercept_}")
    print(f"R² Score: {r2_score(y_test, y_pred):.2f}")
    print(f"MSE: {mean_squared_error(y_test, y_pred):.2f}")

    return y_test, y_pred

def visualize(y_test, y_pred):
    plt.figure()
    plt.plot(y_test.values, label="Actual Price", marker='o')
    plt.plot(y_pred, label="Predicted Price", marker='x')
    plt.title("House Price Prediction - Actual vs Predicted")
    plt.xlabel("Test Sample Index")
    plt.ylabel("Price")
    plt.legend()
    plt.grid(True)
    plt.show()

def main():
    print("=== House Price Prediction Application ===")
    df = load_data()
    y_test, y_pred = train_model(df)
    visualize(y_test, y_pred)

if __name__ == "__main__":
    main()
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

def load_data():
    iris = load_iris()
    X = iris.data
    y = iris.target
    labels = iris.target_names
    return X, y, labels

def split_data(X, y):
    return train_test_split(X, y, test_size=0.3, random_state=42)

def train_naive_bayes(X_train, y_train):
    model = GaussianNB()
    model.fit(X_train, y_train)
    return model

def evaluate_model(model, X_test, y_test, labels):
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    print(f"\nAccuracy: {acc:.2f}")
    print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=labels))
    return cm

def visualize_confusion_matrix(cm, labels):
    sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=labels,
yticklabels=labels)
    plt.title("Naive Bayes - Confusion Matrix")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

def main():
    print("=== Iris Flower Classification using Naive Bayes ===")
    X, y, labels = load_data()
    X_train, X_test, y_train, y_test = split_data(X, y)
    model = train_naive_bayes(X_train, y_train)
    cm = evaluate_model(model, X_test, y_test, labels)
    visualize_confusion_matrix(cm, labels)

if __name__ == "__main__":
    main()
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

def load_data():
    iris = load_iris()
    return iris.data, iris.target, iris.target_names

def evaluate_model(name, model, X_train, X_test, y_train, y_test, target_names):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"\n{name} Accuracy: {acc:.2f}")
    print(f"{name} Classification Report:\n", classification_report(y_test,
y_pred, target_names=target_names))
    return name, acc

def visualize_accuracies(results):
    models = [x[0] for x in results]
    scores = [x[1] for x in results]

    plt.figure()
    plt.bar(models, scores, color='skyblue')
    plt.ylim(0, 1)
    plt.title("Classifier Accuracy Comparison")
    plt.ylabel("Accuracy Score")
    plt.xlabel("Classifier")
    plt.grid(axis='y')
    plt.show()

def main():
    print("=== Classification Algorithms Comparison ===")
    X, y, target_names = load_data()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

    classifiers = [
        ("K-Nearest Neighbors", KNeighborsClassifier(n_neighbors=3)),
        ("Naive Bayes", GaussianNB()),
        ("Decision Tree", DecisionTreeClassifier()),
        ("Logistic Regression", LogisticRegression(max_iter=200))
    ]

    results = []
    for name, model in classifiers:
        results.append(evaluate_model(name, model, X_train, X_test, y_train,
y_test, target_names))

    visualize_accuracies(results)

if __name__ == "__main__":
    main()
'
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

def load_data():
    # Sample synthetic dataset
    data = {
        'RAM': [4, 6, 8, 12, 4, 8, 6, 12, 8, 6],
        'ROM': [64, 128, 256, 256, 32, 64, 128, 512, 128, 64],
        'Battery': [3000, 3500, 4000, 4500, 3000, 5000, 3500, 6000, 4000, 3700],
        'Camera': [12, 16, 48, 64, 8, 12, 16, 108, 48, 20],
        'Screen': [5.5, 6.0, 6.3, 6.7, 5.0, 6.2, 6.1, 6.9, 6.5, 5.8],
        'Price': [15000, 20000, 30000, 40000, 12000, 17000, 19000, 60000, 35000,
18000]
    }
    return pd.DataFrame(data)

def train_model(df):
    X = df[['RAM', 'ROM', 'Battery', 'Camera', 'Screen']]
    y = df['Price']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    print(f"\nModel R2 Score: {r2_score(y_test, y_pred):.2f}")
    print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred):.2f}")

    return y_test, y_pred

def visualize(y_test, y_pred):
    plt.plot(y_test.values, label='Actual Price', marker='o')
    plt.plot(y_pred, label='Predicted Price', marker='x')
    plt.title('Mobile Price Prediction - Actual vs Predicted')
    plt.xlabel('Sample Index')
    plt.ylabel('Price (INR)')
    plt.legend()
    plt.grid(True)
    plt.show()

def main():
    print("=== Mobile Price Prediction using Random Forest ===")
    df = load_data()
    y_test, y_pred = train_model(df)
    visualize(y_test, y_pred)

if __name__ == "__main__":
    main()
```

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

def load_data():
    iris = load_iris()
    return iris.data, iris.target, iris.target_names

def train_perceptron(X_train, y_train):
    clf = Perceptron(max_iter=1000, eta0=0.1, random_state=0)
    clf.fit(X_train, y_train)
    return clf

def evaluate_model(clf, X_test, y_test, target_names):
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"\nPerceptron Accuracy: {acc:.2f}")
    print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=target_names))
    return y_test, y_pred

def visualize(y_test, y_pred):
    plt.scatter(range(len(y_test)), y_test, marker='o', label='Actual')
    plt.scatter(range(len(y_pred)), y_pred, marker='x', label='Predicted')
    plt.title('Perceptron - Iris Classification')
    plt.xlabel('Sample Index')
    plt.ylabel('Class')
    plt.legend()
    plt.grid(True)
    plt.show()

def main():
    print("=== Perceptron Based Iris Classification ===")
    X, y, target_names = load_data()

    # Standardize features for faster convergence
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

    clf = train_perceptron(X_train, y_train)
    y_test, y_pred = evaluate_model(clf, X_test, y_test, target_names)
    visualize(y_test, y_pred)

if __name__ == "__main__":
    main()
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

def load_data():
    # Synthetic data for Bank Loan Prediction
    data = {
        'Age': [25, 35, 45, 20, 30, 50, 40, 60, 48, 33],
        'Job': ['Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes',
'No'],
        'Income': [50000, 60000, 30000, 25000, 45000, 80000, 40000, 30000,
70000, 42000],
        'Credit_Score': ['Good', 'Good', 'Bad', 'Bad', 'Good', 'Good', 'Bad',
'Bad', 'Good', 'Bad'],
        'Loan': ['Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes',
'No']
    }
    df = pd.DataFrame(data)
    return df

def preprocess_data(df):
    le = LabelEncoder()
    for column in ['Job', 'Credit_Score', 'Loan']:
        df[column] = le.fit_transform(df[column])
    X = df[['Age', 'Job', 'Income', 'Credit_Score']]
    y = df['Loan']
    return train_test_split(X, y, test_size=0.3, random_state=42), le

def train_and_evaluate(X_train, X_test, y_train, y_test, label_encoder):
    model = GaussianNB()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    print(f"\nAccuracy: {acc:.2f}")
    print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=label_encoder.classes_))
    return cm, label_encoder.classes_

def visualize_confusion_matrix(cm, class_names):
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title("Naive Bayes - Confusion Matrix for Loan Prediction")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

def main():
    print("=== Bank Loan Prediction using Naive Bayes ===")
    df = load_data()
    (X_train, X_test, y_train, y_test), label_encoder = preprocess_data(df)
    cm, class_names = train_and_evaluate(X_train, X_test, y_train, y_test,
label_encoder)
    visualize_confusion_matrix(cm, class_names)

if __name__ == "__main__":
    main()
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

def generate_data():
    # Simulated monthly sales data
    months = np.array(range(1, 13)).reshape(-1, 1)  # Month 1 to 12
    sales = np.array([2500, 2600, 2700, 3000, 3200, 3300,
                      3500, 3700, 3900, 4200, 4500, 4700])
    return months, sales

def train_model(months, sales):
    model = LinearRegression()
    model.fit(months, sales)
    return model

def predict_future_sales(model, months):
    future_months = np.array(range(13, 19)).reshape(-1, 1)  # Predict next 6
months
    future_sales = model.predict(future_months)
    return future_months, future_sales

def visualize_sales(months, sales, future_months, future_sales):
    plt.plot(months, sales, marker='o', label='Actual Sales')
    plt.plot(future_months, future_sales, marker='x', linestyle='--',
label='Predicted Sales')
    plt.xlabel('Month')
    plt.ylabel('Sales')
    plt.title('Future Sales Prediction')
    plt.legend()
    plt.grid(True)
    plt.show()

def main():
    print("=== Future Sales Prediction using Linear Regression ===")
    months, sales = generate_data()
    model = train_model(months, sales)

    # Predict and show metrics
    predicted_sales = model.predict(months)
    print(f"R2 Score: {r2_score(sales, predicted_sales):.2f}")
    print(f"MSE: {mean_squared_error(sales, predicted_sales):.2f}")

    future_months, future_sales = predict_future_sales(model, months)
    visualize_sales(months, sales, future_months, future_sales)

if __name__ == "__main__":
    main()
5
```