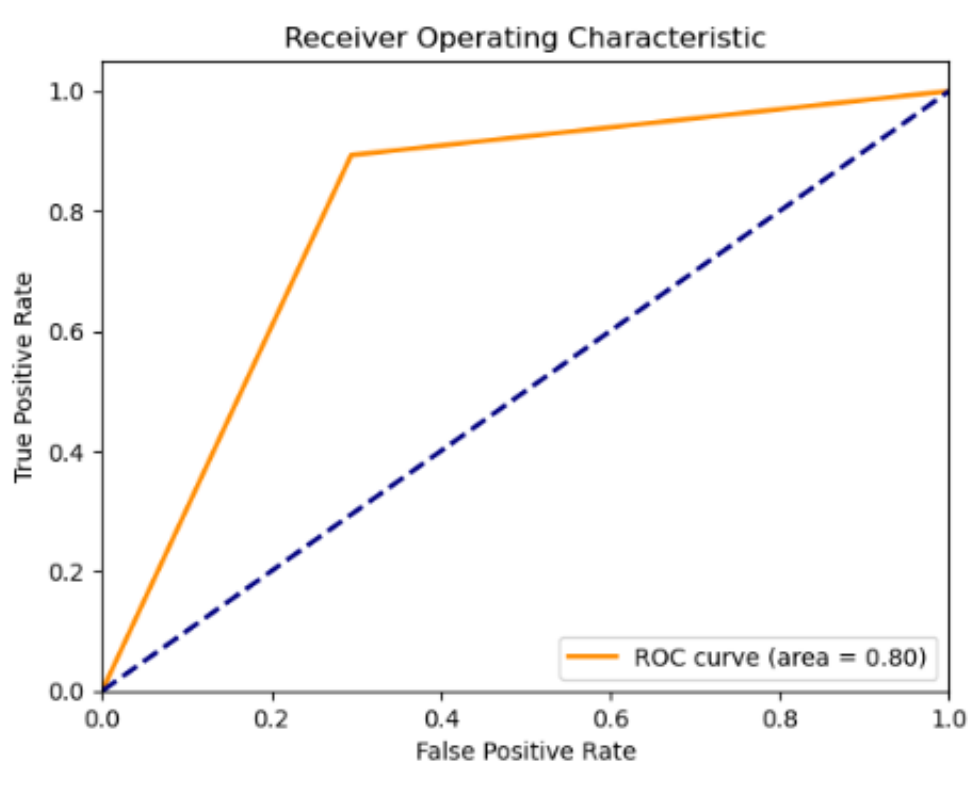
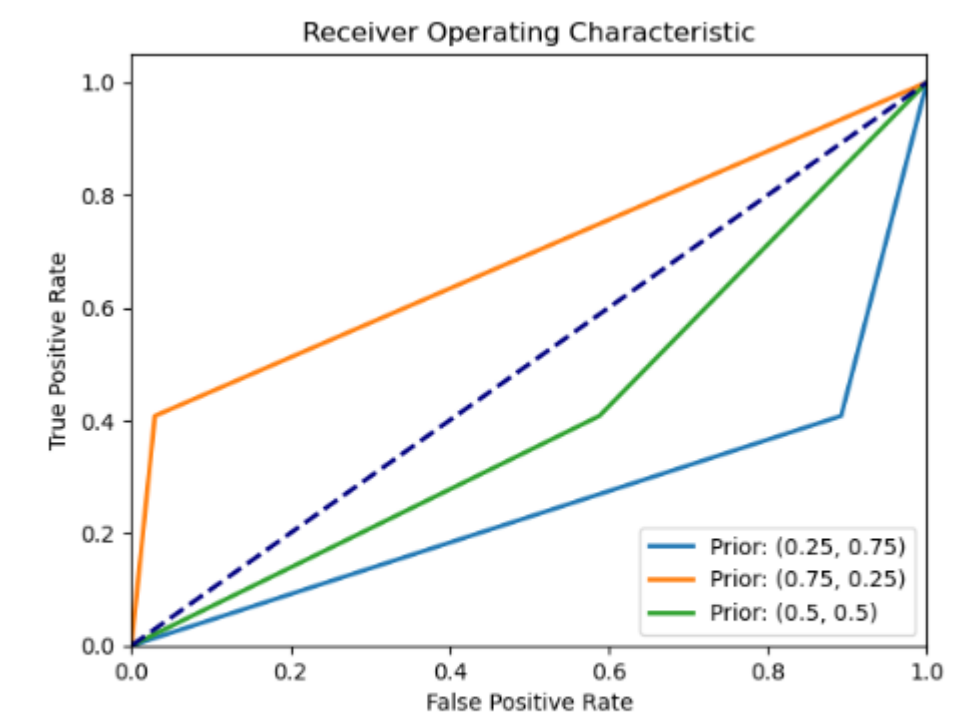
```python
data = pd.read_csv('heart.csv')
X = data.drop('target', axis=1)
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

y_pred = model.predict(X_test)
s
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)

fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Recall:", recall)
print("Precision:", precision)
print("False Positive Rate:", fpr)
```



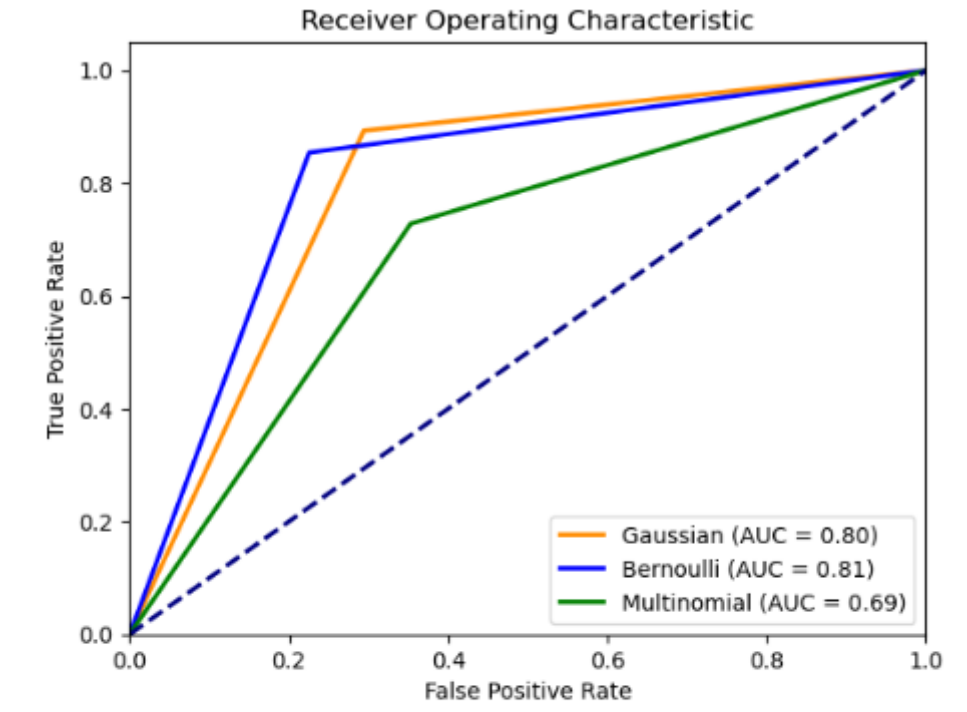

```
Accuracy: 0.8
Recall: 0.8932038834951457
Precision: 0.7540983606557377
False Positive Rate: [0.         0.29411765 1.        ]
```

```python
prior_probs = [(0.25, 0.75), (0.75, 0.25), (0.5, 0.5)]

accuracies = []
recalls = []
precisions = []
fprs = []
roc_aucs = []

for prior in prior_probs:
    class_dist = np.array(prior)
    modified_counts = class_dist * len(y_train)
    modified_counts = modified_counts.astype(int)
    modified_y_train = np.concatenate([np.full(count, i) for i, count in enumerate(modified_counts)])

    model = GaussianNB()
    model.fit(X_train, modified_y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)

    fpr, tpr, _ = roc_curve(y_test, y_pred)
    roc_auc = auc(fpr, tpr)

    accuracies.append(accuracy)
    recalls.append(recall)
    precisions.append(precision)
    fprs.append(fpr)
    roc_aucs.append(roc_auc)
```



Prior Probabilities	Accuracy	Recall		Precision
(0.25, 0.75)		0.5415	0.9709	0.5236
(0.75, 0.25)		0.4878	0.0097	0.2500
(0.5, 0.5)		0.4098	0.4078	0.4118

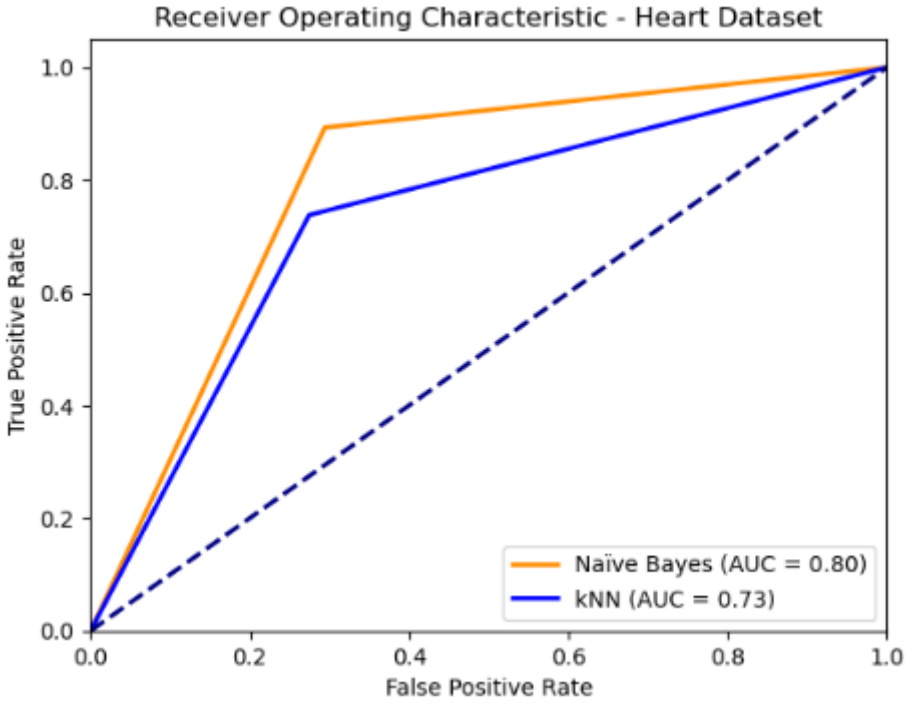
```python
gaussian_model = GaussianNB()
gaussian_model.fit(X_train, y_train)
gaussian_y_pred = gaussian_model.predict(X_test)

bernoulli_model = BernoulliNB()
bernoulli_model.fit(X_train, y_train)
bernoulli_y_pred = bernoulli_model.predict(X_test)

multinomial_model = MultinomialNB()
multinomial_model.fit(X_train, y_train)
multinomial_y_pred = multinomial_model.predict(X_test)

gaussian_accuracy = accuracy_score(y_test, gaussian_y_pred)
gaussian_recall = recall_score(y_test, gaussian_y_pred)
gaussian_precision = precision_score(y_test, gaussian_y_pred)

bernoulli_accuracy = accuracy_score(y_test, bernoulli_y_pred)
bernoulli_recall = recall_score(y_test, bernoulli_y_pred)
bernoulli_precision = precision_score(y_test, bernoulli_y_pred)

multinomial_accuracy = accuracy_score(y_test, multinomial_y_pred)
multinomial_recall = recall_score(y_test, multinomial_y_pred)
multinomial_precision = precision_score(y_test, multinomial_y_pred)

gaussian_fpr, gaussian_tpr, _ = roc_curve(y_test, gaussian_y_pred)
gaussian_roc_auc = auc(gaussian_fpr, gaussian_tpr)

bernoulli_fpr, bernoulli_tpr, _ = roc_curve(y_test, bernoulli_y_pred)
bernoulli_roc_auc = auc(bernoulli_fpr, bernoulli_tpr)

multinomial_fpr, multinomial_tpr, _ = roc_curve(y_test, multinomial_y_pred)
multinomial_roc_auc = auc(multinomial_fpr, multinomial_tpr)
```

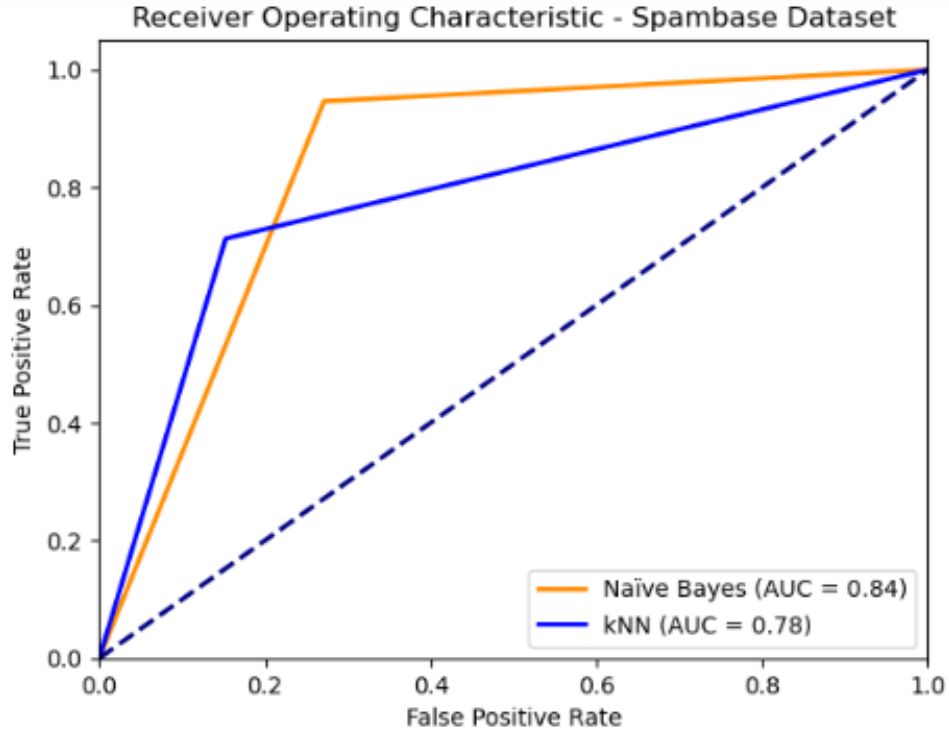


Naïve Bayes Classifier	Accuracy	Recall	Precision
Gaussian	0.8000	0.8932	0.7541
Bernoulli	0.8146	0.8544	0.7928
Multinomial	0.6878	0.7282	0.6757

```python
spambase_data = pd.read_csv('spambase.csv')
X_spam = spambase_data.drop('class', axis=1)
y_spam = spambase_data['class']

heart_data = pd.read_csv('heart.csv')
X_heart = heart_data.drop('target', axis=1)
y_heart = heart_data['target']

X_spam_train, X_spam_test, y_spam_train, y_spam_test = train_test_split(X_spam, y_spam, test_size=0.2, random_state=42)

X_heart_train, X_heart_test, y_heart_train, y_heart_test = train_test_split(X_heart, y_heart, test_size=0.2, random_state=42)

spam_model = GaussianNB()
spam_model.fit(X_spam_train, y_spam_train)
spam_y_pred = spam_model.predict(X_spam_test)
heart_model = GaussianNB()
heart_model.fit(X_heart_train, y_heart_train)
heart_y_pred = heart_model.predict(X_heart_test)

spam_accuracy = accuracy_score(y_spam_test, spam_y_pred)
spam_recall = recall_score(y_spam_test, spam_y_pred)
spam_precision = precision_score(y_spam_test, spam_y_pred)
spam_fpr, spam_tpr, _ = roc_curve(y_spam_test, spam_y_pred)
spam_roc_auc = auc(spam_fpr, spam_tpr)

heart_accuracy = accuracy_score(y_heart_test, heart_y_pred)
heart_recall = recall_score(y_heart_test, heart_y_pred)
heart_precision = precision_score(y_heart_test, heart_y_pred)
heart_fpr, heart_tpr, _ = roc_curve(y_heart_test, heart_y_pred)
heart_roc_auc = auc(heart_fpr, heart_tpr)
```

Dataset	Accuracy	Recall	Precision
Spambase	0.8208	0.9462	0.7193
Heart	0.8000	0.8932	0.7541






```
Heart Dataset Metrics:
Classifier      Accuracy        Recall          Precision
Naïve Bayes     0.8000  0.8932          0.7541
kNN             0.7317  0.7379          0.7308

Spambase Dataset Metrics:
Classifier      Accuracy        Recall          Precision
Naïve Bayes     0.8208  0.9462          0.7193
kNN             0.7904  0.7128          0.7744
```

```python
# Train Gaussian Naïve Bayes classifier on heart dataset
heart_nb_model = GaussianNB()
heart_nb_model.fit(X_heart_train, y_heart_train)
heart_nb_y_pred = heart_nb_model.predict(X_heart_test)

# Train kNN classifier on heart dataset
heart_knn_model = KNeighborsClassifier(n_neighbors=5)
heart_knn_model.fit(X_heart_train, y_heart_train)
heart_knn_y_pred = heart_knn_model.predict(X_heart_test)

# Train Gaussian Naïve Bayes classifier on spambase dataset
spam_nb_model = GaussianNB()
spam_nb_model.fit(X_spam_train, y_spam_train)
spam_nb_y_pred = spam_nb_model.predict(X_spam_test)

# Train kNN classifier on spambase dataset
spam_knn_model = KNeighborsClassifier(n_neighbors=5)
spam_knn_model.fit(X_spam_train, y_spam_train)
spam_knn_y_pred = spam_knn_model.predict(X_spam_test)

# Calculate evaluation metrics for heart dataset
heart_nb_accuracy = accuracy_score(y_heart_test, heart_nb_y_pred)
heart_nb_recall = recall_score(y_heart_test, heart_nb_y_pred)
heart_nb_precision = precision_score(y_heart_test, heart_nb_y_pred)
heart_nb_fpr, heart_nb_tpr, _ = roc_curve(y_heart_test, heart_nb_y_pred)
heart_nb_roc_auc = auc(heart_nb_fpr, heart_nb_tpr)

heart_knn_accuracy = accuracy_score(y_heart_test, heart_knn_y_pred)
heart_knn_recall = recall_score(y_heart_test, heart_knn_y_pred)
heart_knn_precision = precision_score(y_heart_test, heart_knn_y_pred)
heart_knn_fpr, heart_knn_tpr, _ = roc_curve(y_heart_test, heart_knn_y_pred)
heart_knn_roc_auc = auc(heart_knn_fpr, heart_knn_tpr)
```

```python
# Calculate evaluation metrics for spambase dataset
spam_nb_accuracy = accuracy_score(y_spam_test, spam_nb_y_pred)
spam_nb_recall = recall_score(y_spam_test, spam_nb_y_pred)
spam_nb_precision = precision_score(y_spam_test, spam_nb_y_pred)
spam_nb_fpr, spam_nb_tpr, _ = roc_curve(y_spam_test, spam_nb_y_pred)
spam_nb_roc_auc = auc(spam_nb_fpr, spam_nb_tpr)

spam_knn_accuracy = accuracy_score(y_spam_test, spam_knn_y_pred)
spam_knn_recall = recall_score(y_spam_test, spam_knn_y_pred)
spam_knn_precision = precision_score(y_spam_test, spam_knn_y_pred)
spam_knn_fpr, spam_knn_tpr, _ = roc_curve(y_spam_test, spam_knn_y_pred)
spam_knn_roc_auc = auc(spam_knn_fpr, spam_knn_tpr)
```

```python
from sklearn.neighbors import KNeighborsRegressor
def evaluate_knn_regression(X_train, y_train, X_test, y_test, k):
    knn_regression = KNeighborsRegressor(n_neighbors=k)
    knn_regression.fit(X_train, y_train)
    y_pred = knn_regression.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    mean_relative_error = np.mean(np.abs((y_test - y_pred) / y_test))

    return r2, mean_relative_error
k_values = [1, 3, 9]
print("Wave Dataset:")
for k in k_values:
    r2, mean_relative_error = evaluate_knn_regression(X_wave_train, y_wave_train, X_wave_test, y_wave_test, k)
    print(f"k = {k}:")
    print("R-squared:", r2)
    print("Mean Relative Error:", mean_relative_error)
    print()
print("Boston Housing Dataset:")
for k in k_values:
    r2, mean_relative_error = evaluate_knn_regression(X_boston_train, y_boston_train, X_boston_test, y_boston_test, k)
    print(f"k = {k}:")
    print("R-squared:", r2)
    print("Mean Relative Error:", mean_relative_error)
    print()
```

```
Wave Dataset:
k = 1:
R-squared: 0.19047619047619024
Mean Relative Error: nan

k = 3:
R-squared: 0.49206349206349187
Mean Relative Error: nan

k = 9:
R-squared: 0.5737801293356848
Mean Relative Error: nan

Boston Housing Dataset:
k = 1:
R-squared: 0.4179206827765607
Mean Relative Error: 0.1955707831064553

k = 3:
R-squared: 0.7046442656646525
Mean Relative Error: 0.1745151867940191

k = 9:
R-squared: 0.5226990227025877
Mean Relative Error: 0.21097048084751058
```

```python
wave = load_wine()
boston = load_boston()

X_wave = wave.data
y_wave = wave.target

X_boston = boston.data
y_boston = boston.target

X_wave_train, X_wave_test, y_wave_train, y_wave_test = train_test_split(X_wave, y_wave, test_size=0.2, random_state=42
X_boston_train, X_boston_test, y_boston_train, y_boston_test = train_test_split(X_boston, y_boston, test_size=0.2, ran

linear_regression = LinearRegression()
linear_regression.fit(X_wave_train, y_wave_train)
y_wave_pred_linear = linear_regression.predict(X_wave_test)

linear_regression.fit(X_boston_train, y_boston_train)
y_boston_pred_linear = linear_regression.predict(X_boston_test)

knn_regression = KNeighborsRegressor()
knn_regression.fit(X_wave_train, y_wave_train)
y_wave_pred_knn = knn_regression.predict(X_wave_test)

knn_regression.fit(X_boston_train, y_boston_train)
y_boston_pred_knn = knn_regression.predict(X_boston_test)

r2_linear_wave = r2_score(y_wave_test, y_wave_pred_linear)
r2_knn_wave = r2_score(y_wave_test, y_wave_pred_knn)

r2_linear_boston = r2_score(y_boston_test, y_boston_pred_linear)
r2_knn_boston = r2_score(y_boston_test, y_boston_pred_knn)

mean_relative_error_linear_wave = np.mean(np.abs((y_wave_test - y_wave_pred_linear) / y_wave_test))
mean_relative_error_knn_wave = np.mean(np.abs((y_wave_test - y_wave_pred_knn) / y_wave_test))

mean_relative_error_linear_boston = np.mean(np.abs((y_boston_test - y_boston_pred_linear) / y_boston_test))
mean_relative_error_knn_boston = np.mean(np.abs((y_boston_test - y_boston_pred_knn) / y_boston_test))
```

```
Wave Dataset:
Linear Regression - R-squared: 0.8825140263270393
kNN Regression - R-squared: 0.5028571428571427
Linear Regression - Mean Relative Error: inf
kNN Regression - Mean Relative Error: nan

Boston Housing Dataset:
Linear Regression - R-squared: 0.6687594935356289
kNN Regression - R-squared: 0.6473640882039258
Linear Regression - Mean Relative Error: 0.16866394539378796
kNN Regression - Mean Relative Error: 0.18885949164539265
```

```python
def calculate_r2_score(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    linear_regression = LinearRegression()
    linear_regression.fit(X_train, y_train)
    y_pred = linear_regression.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    return r2

dataset_variations = 10
r2_scores = []

for i in range(dataset_variations):
    X, y = mglearn.datasets.load_extended_boston()

    r2 = calculate_r2_score(X, y)
    r2_scores.append(r2)

mean_r2 = np.mean(r2_scores)
std_r2 = np.std(r2_scores)
```

```
R² Score Analysis:
Number of Dataset Variations: 10
Mean R² Score: 0.805582944797216
Standard Deviation of R² Scores: 0.0
R² Scores: [0.805582944797216, 0.805582944797216, 0.805582944797216, 0.805582944797216, 0.805582944797216, 0.80558294479721
6, 0.805582944797216, 0.805582944797216, 0.805582944797216, 0.805582944797216]
```

```python
X_wave = wave.data
y_wave = wave.target

X_boston = boston.data
y_boston = boston.target

X_wave_train, X_wave_test, y_wave_train, y_wave_test = train_test_spl
X_boston_train, X_boston_test, y_boston_train, y_boston_test = train_

linear_regression = LinearRegression()
linear_regression.fit(X_wave_train, y_wave_train)
y_wave_pred_linear = linear_regression.predict(X_wave_test)

linear_regression.fit(X_boston_train, y_boston_train)
y_boston_pred_linear = linear_regression.predict(X_boston_test)

knn_regression = KNeighborsRegressor()
knn_regression.fit(X_wave_train, y_wave_train)
y_wave_pred_knn = knn_regression.predict(X_wave_test)

knn_regression.fit(X_boston_train, y_boston_train)
y_boston_pred_knn = knn_regression.predict(X_boston_test)

r2_linear_wave = r2_score(y_wave_test, y_wave_pred_linear)
r2_knn_wave = r2_score(y_wave_test, y_wave_pred_knn)

r2_linear_boston = r2_score(y_boston_test, y_boston_pred_linear)
r2_knn_boston = r2_score(y_boston_test, y_boston_pred_knn)
```

```
Wave Dataset:
Linear Regression - R-squared: 0.8825140263270393
kNN Regression - R-squared: 0.5028571428571427

Boston Housing Dataset:
Linear Regression - R-squared: 0.6687594935356289
kNN Regression - R-squared: 0.6473640882039258
```

```
Wave Dataset:
Linear Regression - R-squared: 0.8825140263270393
kNN Regression - R-squared: 0.5028571428571427
Linear Regression - Mean Relative Error: inf
kNN Regression - Mean Relative Error: nan

Boston Housing Dataset:
Linear Regression - R-squared: 0.6687594935356289
kNN Regression - R-squared: 0.6473640882039258
Linear Regression - Mean Relative Error: 0.16866394539378796
kNN Regression - Mean Relative Error: 0.18885949164539265
```