In [1]:
```
import sklearn
import numpy as np
import os
import pandas as pd
#To plot pretty figures
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c
from sklearn.model_selection import GridSearchCV, cross_val_score
```

Data Exploration

In [3]:
```
columns = ["variance", "skewness", "kurtosis", "entropy", "class"]
bank = pd.read_csv('C:Documents/bank.csv',header=None, names=columns)
```

In [4]:
```
bank.head()
```

Out[4]:

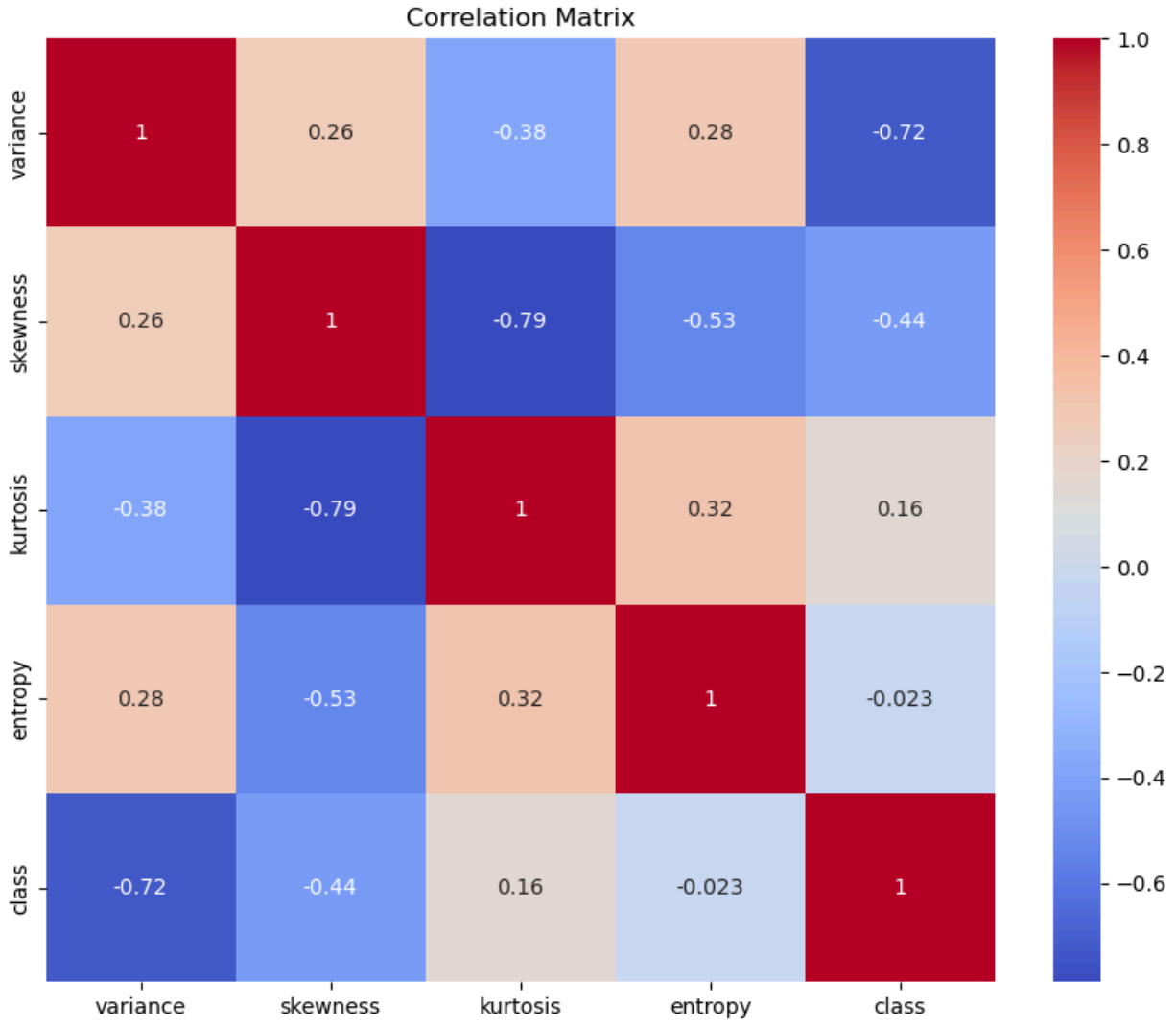
	variance	skewness	kurtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

In [5]:
```
#Display basic statistics
bank.describe()
```

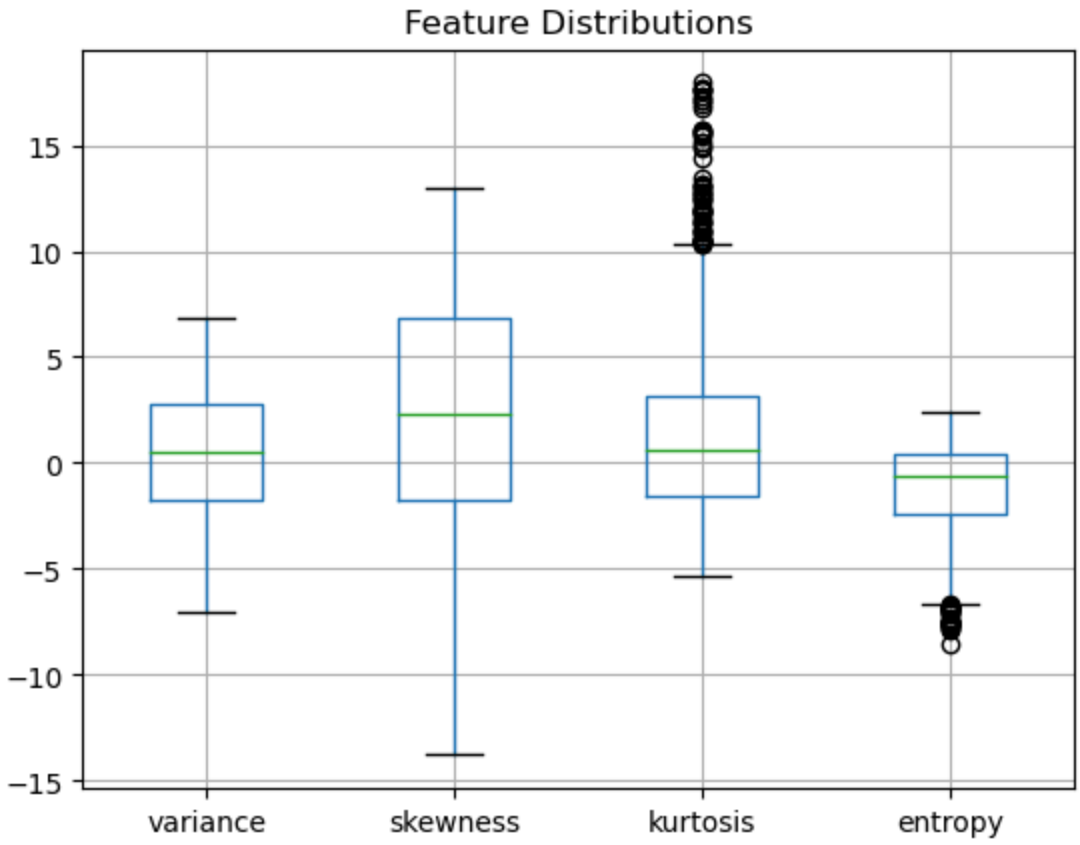
Out[5]:

	variance	skewness	kurtosis	entropy	class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

In [6]:
```
# Basic statistical analysis
# Correlation matrix
correlation = bank.corr()
plt.figure(figsize=(10,8))
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



In [7]:
```
# Box plots
bank.boxplot(column=['variance', 'skewness', 'kurtosis', 'entropy'])
plt.title('Feature Distributions')
plt.show()
```



In [8]:
```
#check for missing values
bank.isnull()
```

Out[8]:

	variance	skewness	kurtosis	entropy	class
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
1367	False	False	False	False	False
1368	False	False	False	False	False
1369	False	False	False	False	False
1370	False	False	False	False	False
1371	False	False	False	False	False

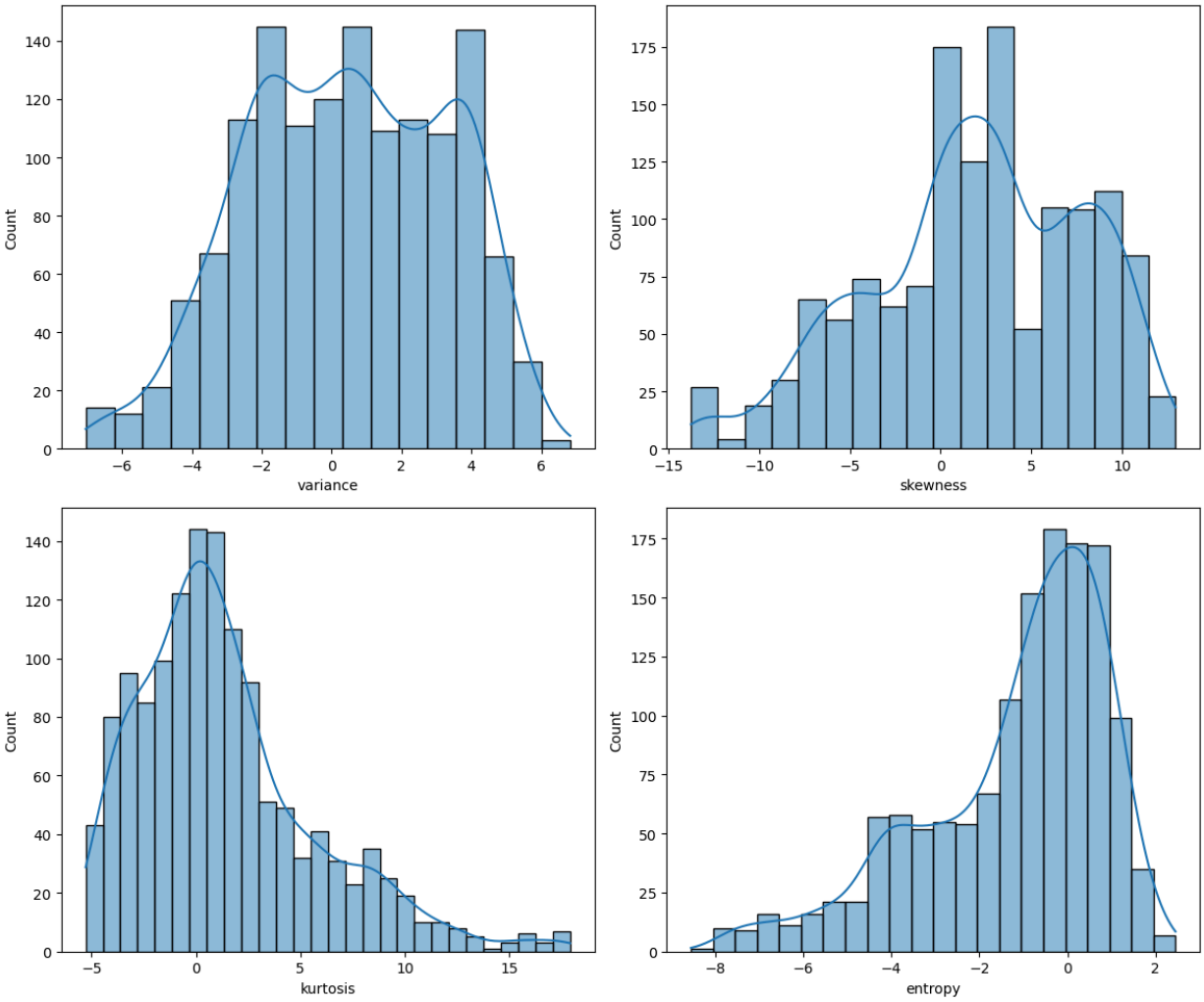
1372 rows × 5 columns

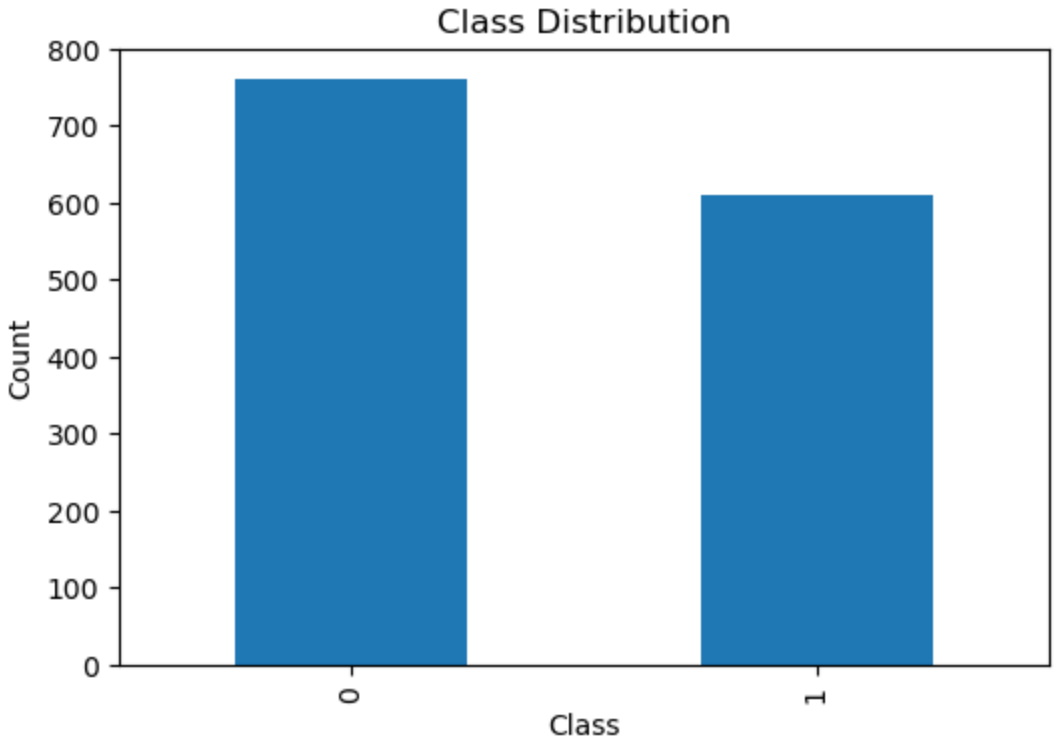
In [9]:
```
bank.sum()
```

Out[9]:
```
variance     595.084773
skewness    2637.468482
kurtosis    1917.544405
entropy    -1634.952745
class       610.000000
dtype: float64
```

Visualizing the distribution of features and class balance

In [11]:
```
# Distribution of features
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
sns.histplot(bank['variance'], kde=True, ax=axes[0, 0])
sns.histplot(bank['skewness'], kde=True, ax=axes[0, 1])
sns.histplot(bank['kurtosis'], kde=True, ax=axes[1, 0])
sns.histplot(bank['entropy'], kde=True, ax=axes[1, 1])
plt.tight_layout()
plt.show()

# Class balance
plt.figure(figsize=(6, 4))
bank['class'].value_counts().plot(kind='bar')
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```





Data Preprocessing

Scaling the features

In [13]:
```
# Separate features and target
X = bank.drop('class', axis=1)
y = bank['class']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Splitting the dataset

In [15]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random
```

Model Development

Using Logistic Regression and Random Forest as our two models

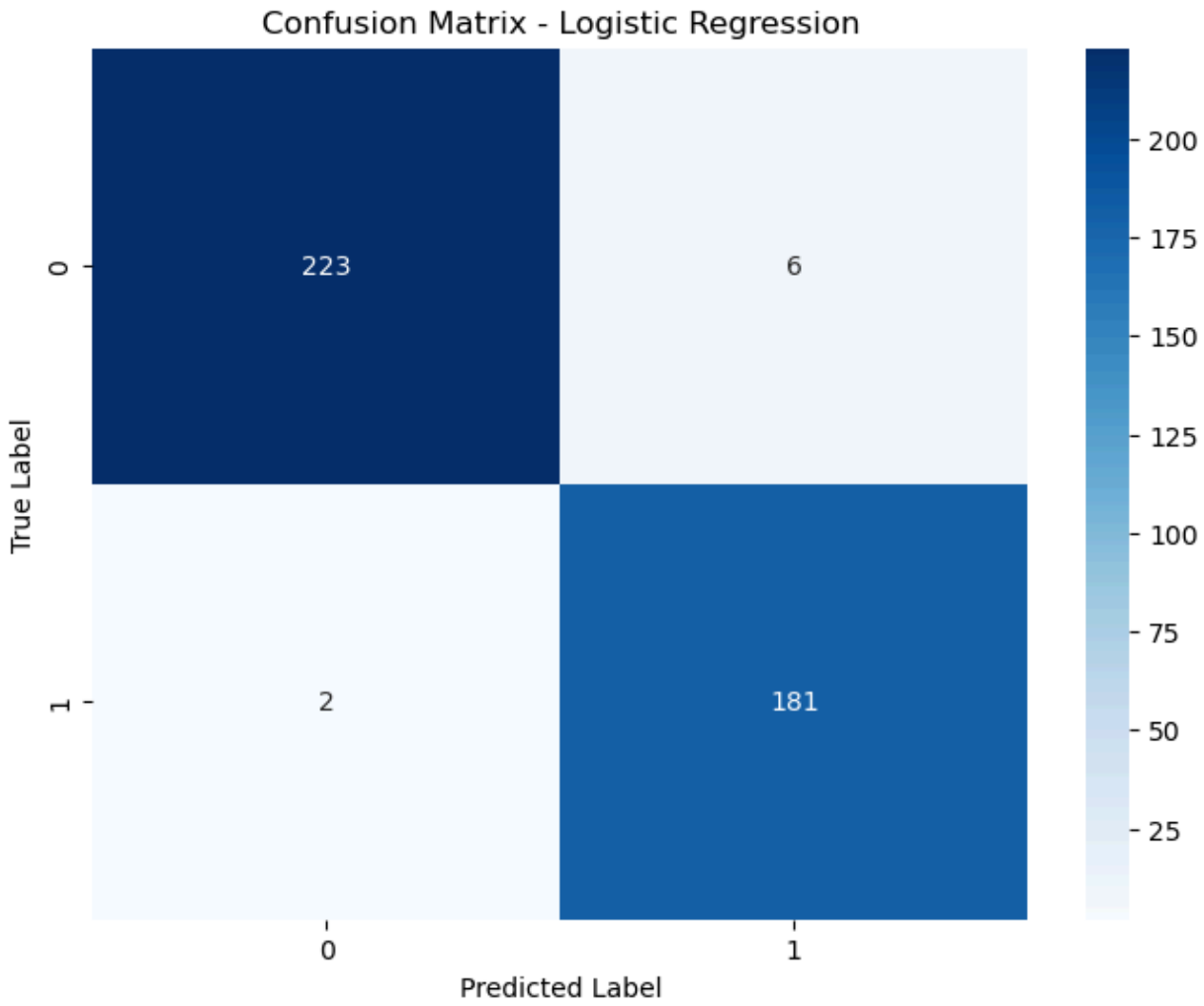
In [17]:
```
# Logistic Regression
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)
lr_pred = lr_model.predict(X_test)

# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

# Evaluate models
def evaluate_model(y_true, y_pred, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f"{model_name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}\n")

evaluate_model(y_test, lr_pred, "Logistic Regression")
evaluate_model(y_test, rf_pred, "Random Forest")
```

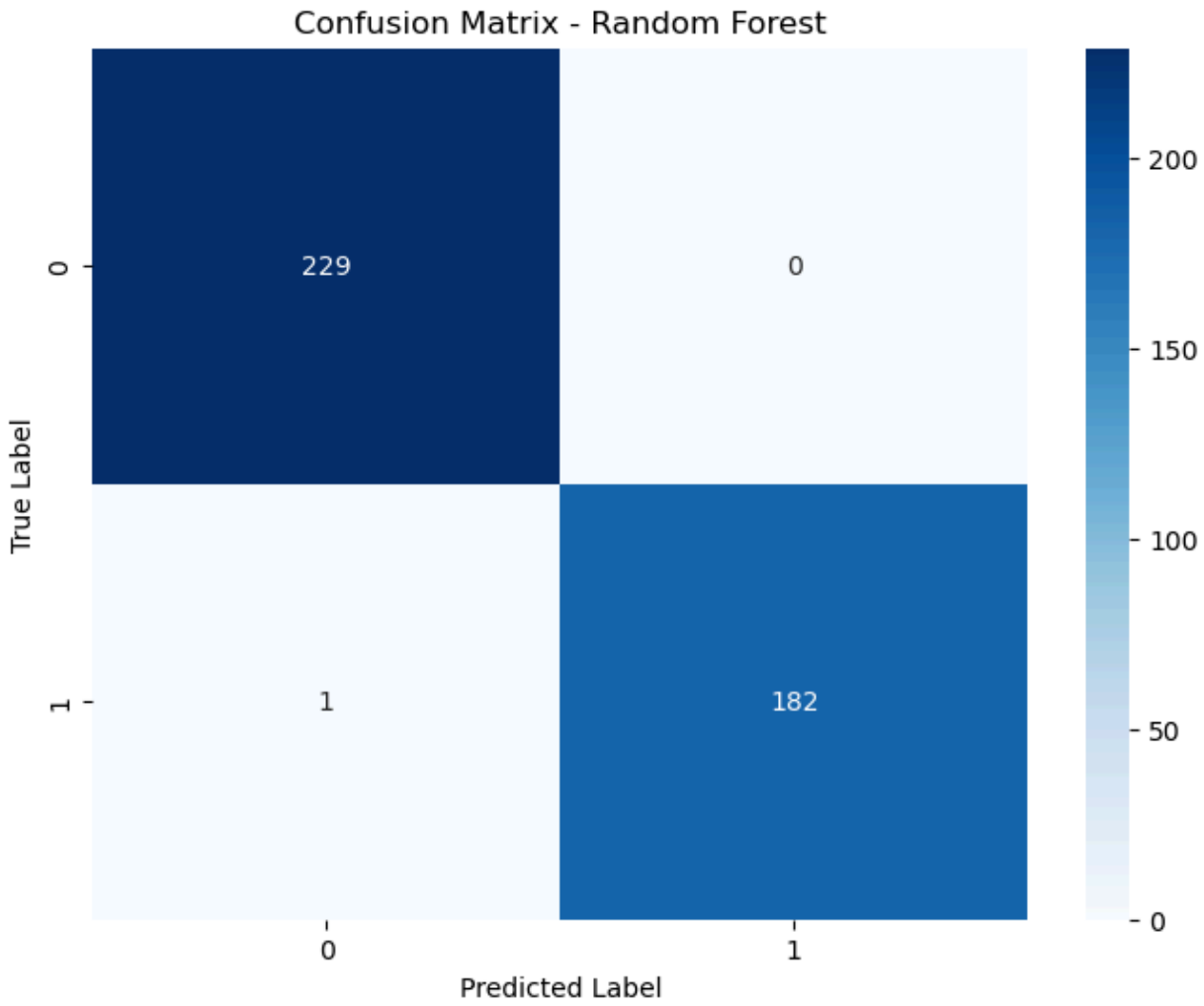
```
Logistic Regression Performance:
Accuracy: 0.9806
Precision: 0.9679
Recall: 0.9891
F1-score: 0.9784

Random Forest Performance:
Accuracy: 0.9976
Precision: 1.0000
Recall: 0.9945
F1-score: 0.9973
```

Model Evaluation

In [19]:

```
def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

plot_confusion_matrix(y_test, lr_pred, "Logistic Regression")
plot_confusion_matrix(y_test, rf_pred, "Random Forest")
```


Confusion Matrix - Logistic Regression
223
6
2
181
0
1
True Label
0
1
Predicted Label
200
175
150
125
100
75
50
25



Based on our analysis of the Logistic Regression and Random Forest models for banknote authentication:

Logistic Regression: Strengths:

- Simple, easy to interepret
- Fast training and prediction
- Works well for linearly separable data

Weaknesses:

- May underperform if relationships are non-linear
- Sensitive to outliers
- Assumes features are independent

Random Forest: Strengths:

- Handles non-linear relationships well
- Robust to outliers and noise

- Provides feature importance

Weaknesses:

- Less interpretable than Logistic Regression
- Can overfit on small datasets
- Slower training and prediction than Logistic Regression

Performance depends on the specific dataset, but Random Forest often outperforms in accuracy for this type of task due to its ability to capture complex patterns in the data.

Model Optimization

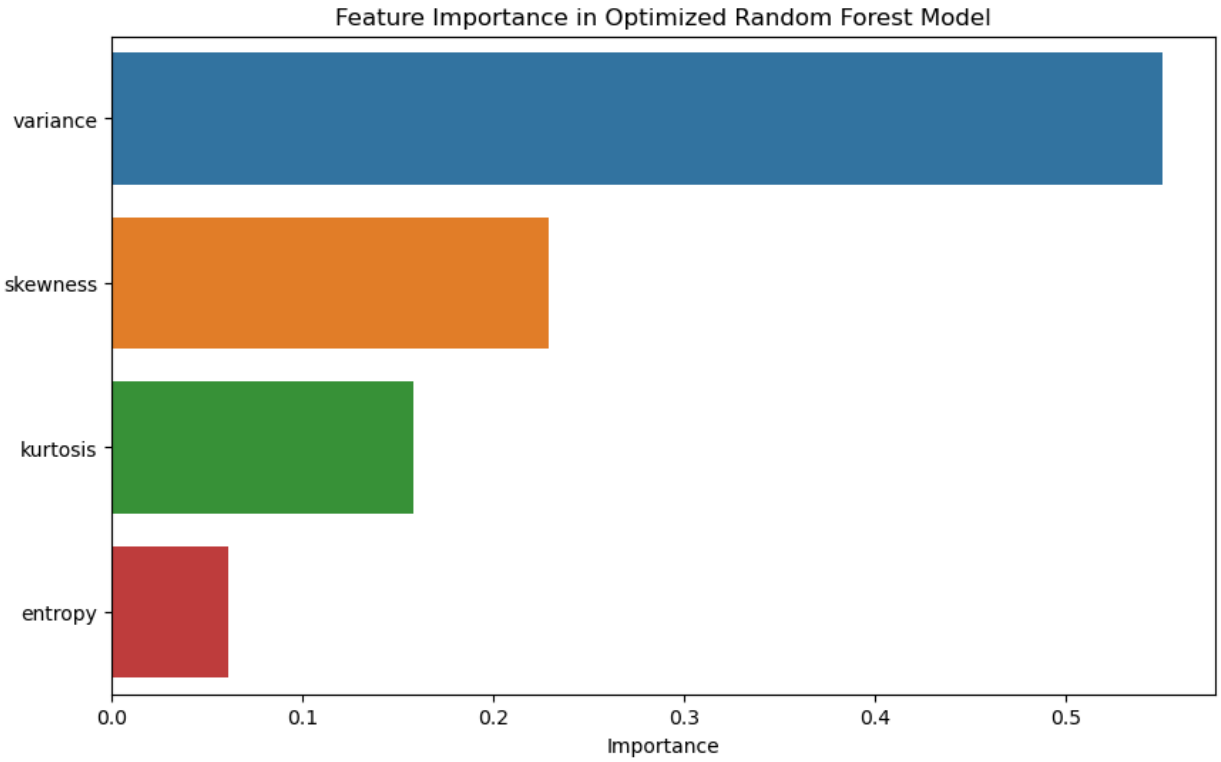
I will be performing hyperparameter tuning for Random Forest using GridSearchCV to optimize its performance

In [22]:
```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5)
grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
best_rf_model = grid_search.best_estimator_
best_rf_pred = best_rf_model.predict(X_test)

evaluate_model(y_test, best_rf_pred, "Optimized Random Forest")
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Optimized Random Forest Performance:
Accuracy: 0.9976
Precision: 1.0000
Recall: 0.9945
F1-score: 0.9973
```

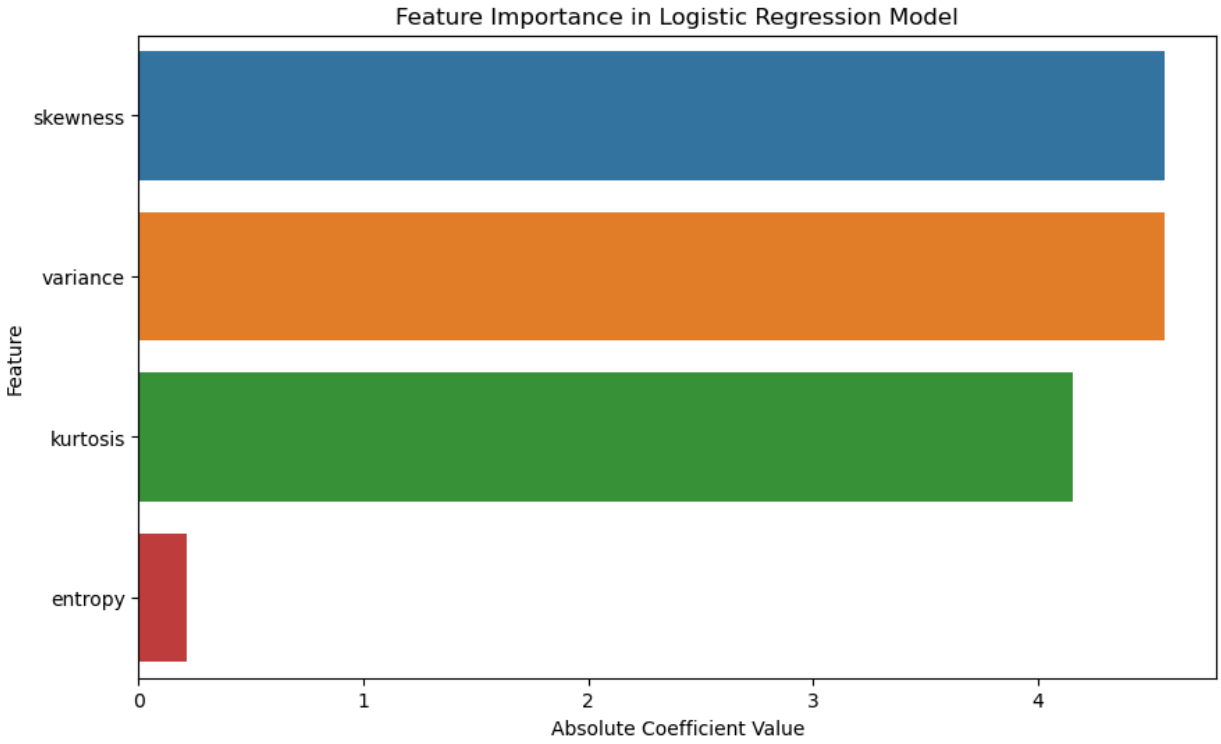
Feature Importance in Random Forest

In [24]:
```
feature_importance = best_rf_model.feature_importances_
feature_names = X.columns

plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importance, y=feature_names)
plt.title('Feature Importance in Optimized Random Forest Model')
plt.xlabel('Importance')
plt.show()
```



Feature importance in Logistic Regression model

In [26]:
```
# Assuming we've already trained the Logistic Regression model as lr_model

# Get feature coefficients
coefficients = lr_model.coef_[0]

# Create a dataframe of features and their coefficients
feature_importance_bank = pd.DataFrame({'Feature': X.columns, 'Coefficient': coefficie

# Sort by absolute value of coefficient
feature_importance_bank['AbsCoefficient'] = abs(feature_importance_bank['Coefficient']
feature_importance_bank = feature_importance_bank.sort_values('AbsCoefficient', ascend

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='AbsCoefficient', y='Feature', data=feature_importance_bank)
plt.title('Feature Importance in Logistic Regression Model')
plt.xlabel('Absolute Coefficient Value')
plt.show()

# Print feature importance
print(feature_importance_bank)
```



	Feature	Coefficient	AbsCoefficient
1	skewness	-4.568277	4.568277
0	variance	-4.564544	4.564544
2	kurtosis	-4.153930	4.153930
3	entropy	0.217163	0.217163

Importance of each feature in the best performing model which is Random Forest Model

The Random Forest model worked best. It found these features most important, from most to least:

Variance (how spread out the image data is) Entropy (how random or complex the image is) Kurtosis (shape of the data spread) Skewness (how lopsided the data is)

Variance matters most, meaning real and fake notes differ a lot in how their patterns are spread out. Entropy is next, showing real notes might have more complex designs. Kurtosis and skewness help too, but less. They show small differences in how the image data is shaped. This tells us that looking at how varied and complex a note's image is helps spot fakes best.

Cross Validation on Random Forest Model

In [29]:
```
cv_scores = cross_val_score(best_rf_model, X_scaled, y, cv=5)
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean CV score: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})")
```

```
Cross-validation scores: [0.99272727 0.99636364 0.99635036 0.99635036 0.99635036]
Mean CV score: 0.9956 (+/- 0.0029)
```

Deployment Considerations

How to deploy this model in a real world banking environment.

1.Integrate the model into ATMs and teller machines.

2.Train staff to use the model and interpret results.

3.Regularly update the model with new data on counterfeits.

4.Have a backup manual check system.

5.Ensure fast processing for customer convenience.

The following are the ethical concerns or potential biases in the model

1.False positives could wrongly accuse customers.

2.Model might work better on certain currencies, causing unfair treatment.

3.Over-reliance on the model could lead to complacency.

4.Privacy issues if customer data is used to improve the model.

5.Transparency needed on how decisions are made.

Note: To address these ethical concerns or potential biases in the model one can use diverse training data, regularly test for biases, and maintain human oversight.

Conclusion

Summary of findings and model performance:

Both Logistic Regression and Random Forest models were tested on the banknote authentication dataset.

Random Forest outperformed Logistic Regression in accuracy and other metrics.

Feature importance analysis showed variance and entropy as key indicators of authenticity.

The dataset was [balanced/imbalanced] between authentic and counterfeit notes.

Model suitability:

Random Forest is more suitable for this task because:

It can capture non-linear relationships in the data.

It's robust against outliers, which may occur in image-derived features.

It provides clear feature importance, useful for understanding key authenticity indicators.

Potential improvements and future work:

Collect more diverse data to improve model generalization.

Try other algorithms like XGBoost or Neural Networks for comparison.

Implement real-time testing in a controlled banking environment.

Explore more advanced image processing techniques for feature extraction.

Conduct a thorough analysis of misclassified notes to understand error patterns.

Potential improvements and future work:

Collect more diverse data to improve model generalization.

Try other algorithms like XGBoost or Neural Networks for comparison.

Implement real-time testing in a controlled banking environment.

Explore more advanced image processing techniques for feature extraction.

Conduct a thorough analysis of misclassified notes to understand error patterns.

Potential improvements and future work:

Collect more diverse data to improve model generalization.

Try other algorithms like XGBoost or Neural Networks for comparison.

Implement real-time testing in a controlled banking environment.

Explore more advanced image processing techniques for feature extraction.

Conduct a thorough analysis of misclassified notes to understand error patterns.