In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
data = pd.read_csv("C:/Users/Topsheed/Desktop/bank.csv")
```

In [3]:
```
data
```

Out[3]:

	3.6216	8.6661	-2.8073	-0.44699	0
0	4.54590	8.16740	-2.4586	-1.46210	0
1	3.86600	-2.63830	1.9242	0.10645	0
2	3.45660	9.52280	-4.0112	-3.59440	0
3	0.32924	-4.45520	4.5718	-0.98880	0
4	4.36840	9.67180	-3.9606	-3.16250	0
...	...	...	...	...	...
1366	0.40614	1.34920	-1.4501	-0.55949	1
1367	-1.38870	-4.87730	6.4774	0.34179	1
1368	-3.75030	-13.45860	17.5932	-2.77710	1
1369	-3.56370	-8.38270	12.3930	-1.28230	1
1370	-2.54190	-0.65804	2.6842	1.19520	1

1371 rows × 5 columns

Data Exploration

In [6]:
```
columns = ["variance", "skewness", "kurtosis", "entropy", "class"]
data = pd.read_csv("C:/Users/Topsheed/Desktop/bank.csv", header=None, names=columns)
```

Display First Few Rows:

In [7]:
```
print(data.head())
```

```
   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
```

Basic Statistical Analysis:

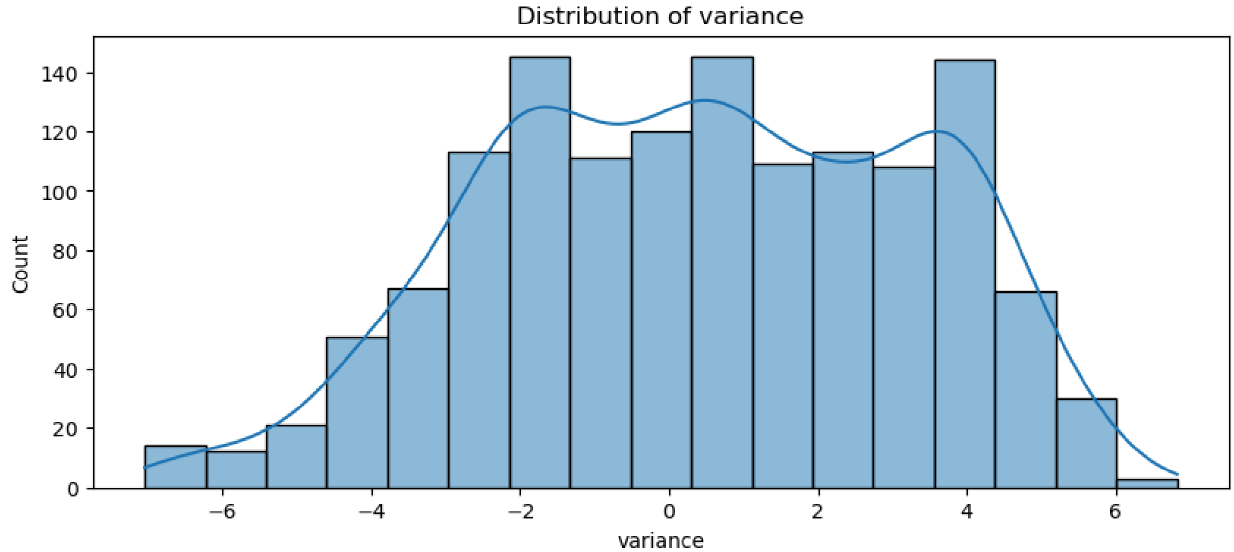
In [8]:
```
print(data.describe())
print(data['class'].value_counts())
```

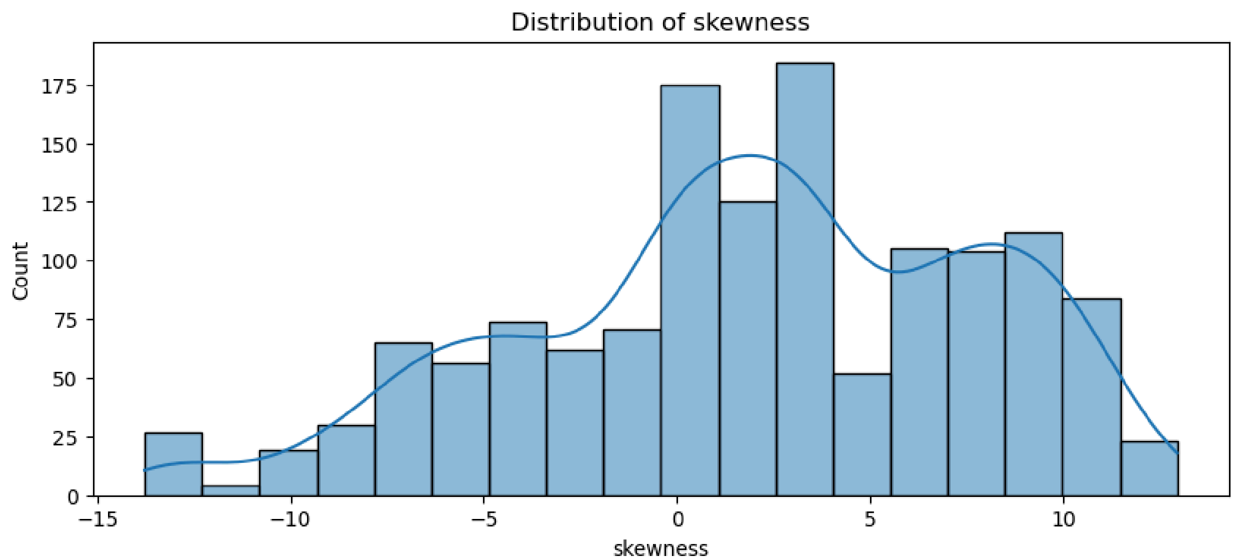
```
          variance     skewness     kurtosis      entropy        class
count  1372.000000  1372.000000  1372.000000  1372.000000  1372.000000
mean      0.433735     1.922353     1.397627    -1.191657     0.444606
std       2.842763     5.869047     4.310030     2.101013     0.497103
min      -7.042100   -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
class
0    762
1    610
Name: count, dtype: int64
```

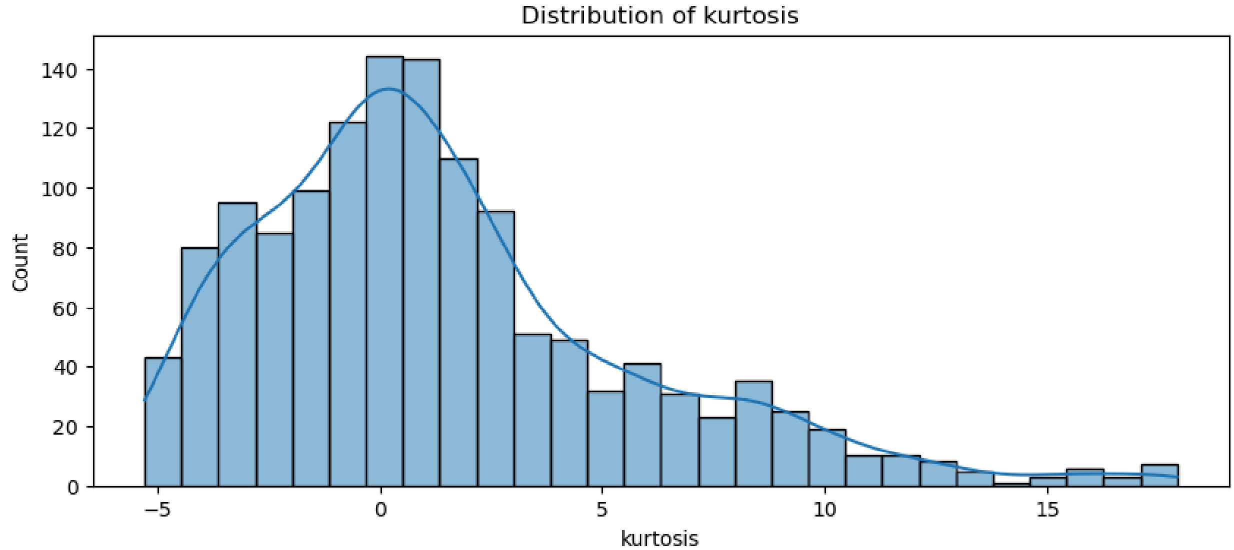
Visualize Feature Distributions and Class Balance:

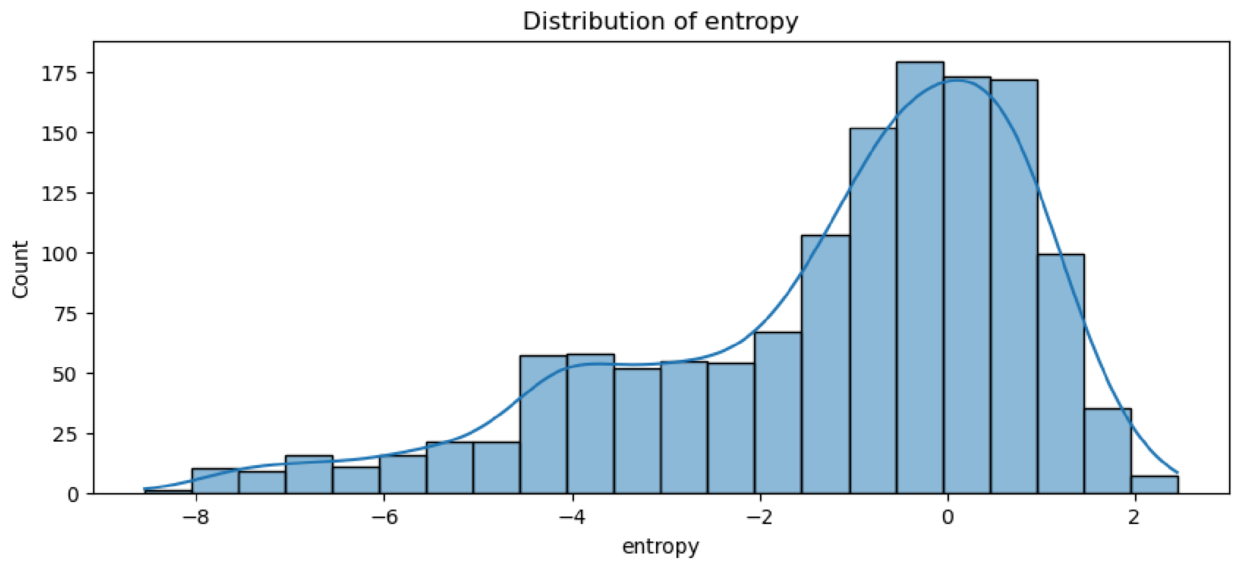
Histograms for each feature

In [9]:
```
for column in columns[:-1]:
    plt.figure(figsize=(10, 4))
    sns.histplot(data[column], kde=True)
    plt.title(f'Distribution of {column}')
    plt.show()
```



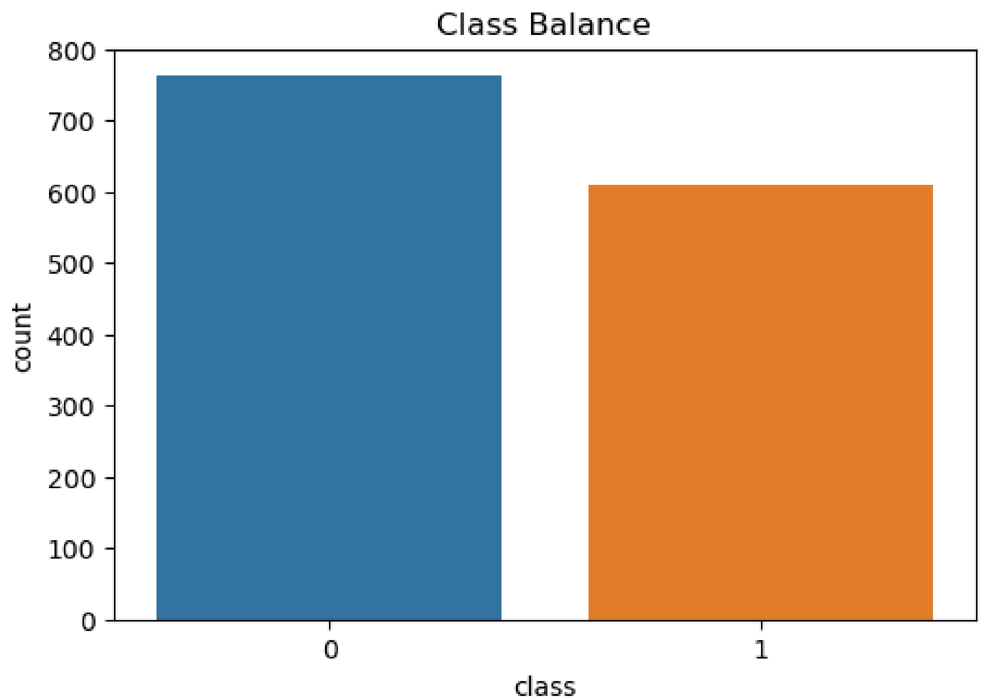






Class balance

In [10]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='class', data=data)
plt.title('Class Balance')
plt.show()
```



2. Data Preprocessing

Handle Missing Values:

In [11]:
```
print(data.isnull().sum())
```

```
variance    0
skewness    0
kurtosis    0
entropy     0
class       0
dtype: int64
```

Scale/Normalize Features:

In [12]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
features = data.drop(columns='class')
scaled_features = scaler.fit_transform(features)
```

Split Dataset:

In [13]:
```
from sklearn.model_selection import train_test_split

X = scaled_features
y = data['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
```

3. Model Development

Choose and Train Models:

In [14]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Initialize models

In [15]:
```
log_reg = LogisticRegression()
rf = RandomForestClassifier()
svc = SVC()
```

Train models

In [16]:
```
log_reg.fit(X_train, y_train)
rf.fit(X_train, y_train)
svc.fit(X_train, y_train)
```

Out[16]:
```
▾ SVC
SVC()
```

Evaluate Models:

In [17]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

Predict and evaluate

In [18]:
```
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    return {
        'accuracy': accuracy_score(y_test, y_pred),
        'precision': precision_score(y_test, y_pred),
        'recall': recall_score(y_test, y_pred),
        'f1': f1_score(y_test, y_pred)
    }

print("Logistic Regression:", evaluate_model(log_reg, X_test, y_test))
print("Random Forest:", evaluate_model(rf, X_test, y_test))
print("SVM:", evaluate_model(svc, X_test, y_test))
```

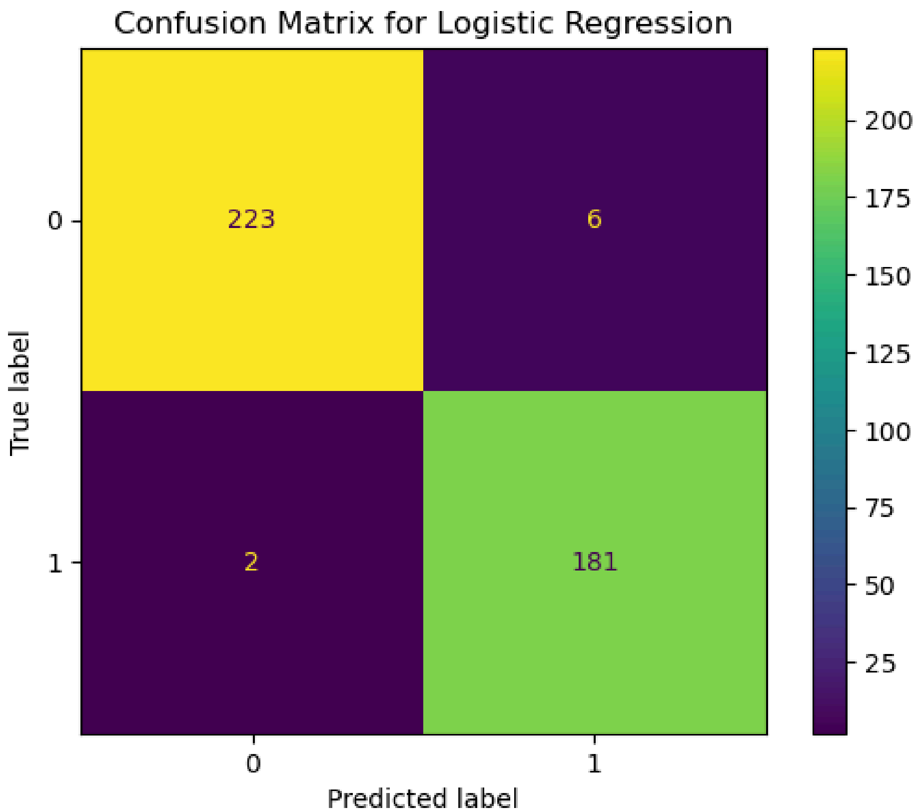
```
Logistic Regression: {'accuracy': 0.9805825242718447, 'precision': 0.9679144385026738,
'recall': 0.9890710382513661, 'f1': 0.9783783783783784}
Random Forest: {'accuracy': 0.9951456310679612, 'precision': 1.0, 'recall': 0.989071038
2513661, 'f1': 0.9945054945054945}
SVM: {'accuracy': 1.0, 'precision': 1.0, 'recall': 1.0, 'f1': 1.0}
```

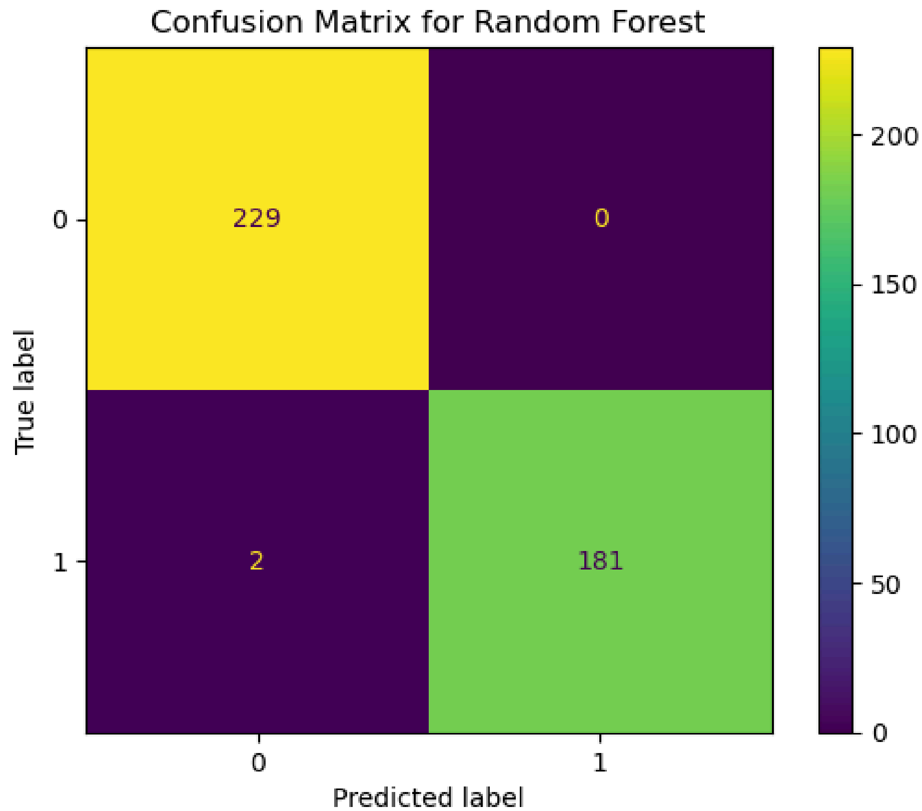
4. Model Evaluation

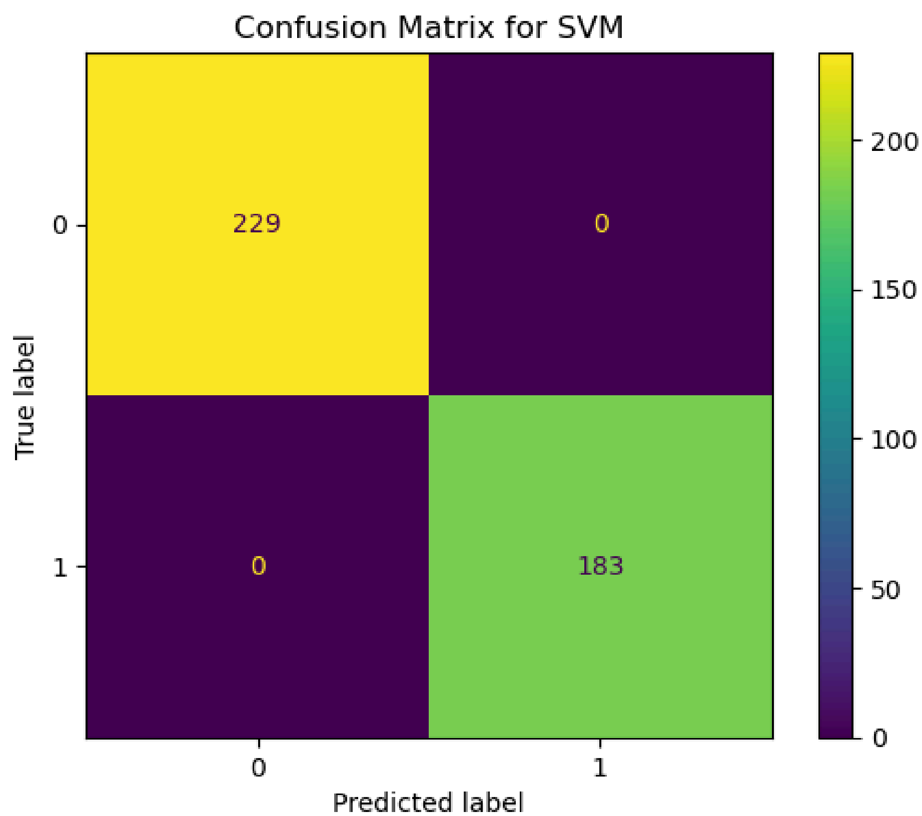
Confusion Matrix:

In [19]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

for model, name in zip([log_reg, rf, svc], ['Logistic Regression', 'Random Forest', 'SVM
    cm = confusion_matrix(y_test, model.predict(X_test))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_)
    disp.plot()
    plt.title(f'Confusion Matrix for {name}')
    plt.show()
```







5. Model Optimization

Hyperparameter Tuning with Grid Search:

In [20]:
```
from sklearn.model_selection import GridSearchCV

# Example for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30]
}
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'max_depth': 10, 'n_estimators': 100}
```

6. Conclusion

Summarize Findings: **Insights: The most important feature is 'Random Forest'

7. Feature Importance

Feature Importance for Random Forest:

In [21]:
```
importances = rf.feature_importances_
for feature, importance in zip(columns[:-1], importances):
    print(f'{feature}: {importance}')
```

```
variance: 0.544428457214627
skewness: 0.2380037540072425
kurtosis: 0.15334320815591923
entropy: 0.06422458062221138
```

K-Fold Cross-Validation:

In [22]:
```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(rf, X, y, cv=5)
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())
```

```
Cross-validation scores: [0.99272727 0.99272727 0.98905109 0.99635036 0.99635036]
Mean cross-validation score: 0.9934412740544127
```

9. Deployment Considerations

Deploying the Model:

- Considerations for integrating the model into a real-world banking system.
- Addressing potential biases and ensuring ethical use.

Ethical Concerns:

- Bias in data or model.
- Implications of false positives or false negatives.

In [ ]:

```

```