```
def log(x_train, x_test, y_train, y_test):
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_train)
    x_test_scaled = scaler.transform(x_test)

    model = LogisticRegression(solver='lbfgs', max_iter=10000)
    model.fit(x_train_scaled, y_train)
    y_pred = model.predict(x_test_scaled)

    # Scores
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred, pos_label=1))
    print("Recall:", recall_score(y_test, y_pred, pos_label=1))
    print("F1 Score:", f1_score(y_test, y_pred, pos_label=1))

log(x_train, x_test, y_train, y_test)
```

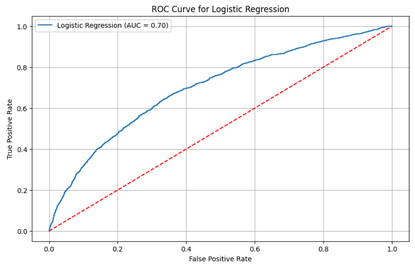
```
Accuracy: 0.8912640839635746
Precision: 0.6021052631578947
Recall: 0.1899070385126162
F1 Score: 0.28874305900610802
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.preprocessing import StandardScaler

def log(x_train, x_test, y_train, y_test):
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_train)
    x_test_scaled = scaler.transform(x_test)

    # Use a different solver and increase max_iter
    model = LogisticRegression(solver='saga', max_iter=20000, random_state=42)
    model.fit(x_train_scaled, y_train)
    y_pred = model.predict(x_test_scaled)

    # Ensure y_test contains positive samples
    if 1 in y_test.values:
        # Scores
        print("Accuracy:", accuracy_score(y_test, y_pred))
        print("Precision:", precision_score(y_test, y_pred, pos_label=1))
        print("Recall:", recall_score(y_test, y_pred, pos_label=1))
        print("F1 Score:", f1_score(y_test, y_pred, pos_label=1))
    else:
        print("Warning: No positive samples in y_test.")

# Assuming your y_train and y_test are correctly encoded
log(x_train, x_test, y_train, y_test)
```

```
Accuracy: 0.8912640839635746
Precision: 0.6021052631578947
Recall: 0.1899070385126162
F1 Score: 0.28874305900610802
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

def evaluation_metrics(y_test, y_pred, y_proba, target_names):
    # scores
    report = classification_report(y_test, y_pred, target_names=target_names)
    print(report)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy}")

    # AUC
    auc_score = roc_auc_score(y_test, y_proba)
    print(f"AUC: {auc_score}")

def log(x_train, x_test, y_train, y_test):
    model = LogisticRegression(solver='lbfgs', max_iter=10000)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    y_proba = model.predict_proba(x_test)[:, 1]  # Get probability scores for the positive class
    evaluation_metrics(y_test, y_pred, y_proba, target_names=['No Deposited', 'Deposited'])

    # ROC Curve
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], linestyle='--', color='r')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve for Logistic Regression')
    plt.legend()
    plt.grid(True)
    plt.show()

# Example usage
log(x_train, x_test, y_train, y_test)
```

```
              precision    recall  f1-score   support

 No Deposited       0.89      0.99      0.94     11452
    Deposited       0.40      0.04      0.07      1506

     accuracy                           0.88     12958
    macro avg       0.64      0.52      0.50     12958
 weighted avg       0.83      0.88      0.84     12958

Accuracy: 0.8814631887636981
AUC: 0.7006088785041508
```



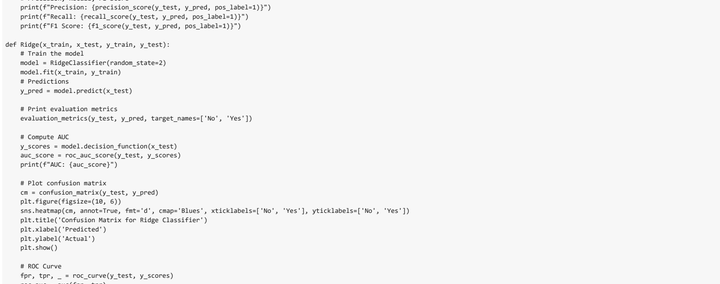

```
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score
import numpy as np

def evaluation_metrics(y_test, y_pre, y_proba, target_names):
    # scores
    report = classification_report(y_test, y_pre, target_names=target_names)
    print(report)
    accuracy = accuracy_score(y_test, y_pre)
    print(f"Accuracy: {accuracy}")

    # AUC
    auc_score = roc_auc_score(y_test, y_proba)
    print(f"AUC: {auc_score}")

def log(x_train, x_test, y_train, y_test):
    model = LogisticRegression()
    model.fit(x_train, y_train)
    y_pre = model.predict(x_test)
    y_proba = model.predict_proba(x_test)[:, 1]  # Get probability scores for the positive class
    evaluation_metrics(y_test, y_pre, y_proba, target_names=['No Deposited', 'Deposited'])

# Classification Report
report = classification_report(y_test, y_pre, target_names=['No Deposited', 'Deposited'])
print(report)

# Accuracy
accuracy = accuracy_score(y_test, y_pre)
print(f"Accuracy: {accuracy}")

# AUC
auc_score = roc_auc_score(y_test, y_proba)
print(f"AUC: {auc_score}")

# Display the results
results = classification_report(y_test, y_pre, target_names=['No Deposited', 'Deposited'], output_dict=True)
print(f"Precision (No Deposited): {results['No Deposited']['precision']}")
print(f"Recall (No Deposited): {results['No Deposited']['recall']}")
print(f"F1 Score (No Deposited): {results['No Deposited']['f1-score']}")
print(f"Support (No Deposited): {results['No Deposited']['support']}")
print(f"Precision (Deposited): {results['Deposited']['precision']}")
print(f"Recall (Deposited): {results['Deposited']['recall']}")
print(f"F1 Score (Deposited): {results['Deposited']['f1-score']}")
print(f"Support (Deposited): {results['Deposited']['support']}")
print(f"Accuracy: {accuracy}")
print(f"Macro Avg: {results['macro avg']}")
print(f"Weighted Avg: {results['weighted avg']}")
print(f"AUC: {auc_score}")
```

```
              precision    recall  f1-score   support

 No Deposited       0.91      0.98      0.94     11452
```

```
           Deposited       0.62      0.22      0.33      1506

            accuracy                           0.89     12958
           macro avg       0.76      0.60      0.64     12958
        weighted avg       0.87      0.89      0.87     12958

        Accuracy: 0.8938879456706282
        AUC: 0.6951516323807112
        Precision (No Deposited): 0.9057089942829535
        Recall (No Deposited): 0.9821865176388404
        F1 Score (No Deposited): 0.9423987264882074
        Support (No Deposited): 11452
        Precision (Deposited): 0.6215213358070705
        Recall (Deposited): 0.22244355909694555
        F1 Score (Deposited): 0.3276283618581906
        Support (Deposited): 1506
        Accuracy: 0.8938879456706282
        Macro Avg: {'precision': 0.7636151650450018, 'recall': 0.6023150383678929, 'f1-score': 0.6350135441173199, 'support': 12958}
        Weighted Avg: {'precision': 0.8726802387910018, 'recall': 0.8938879456706282, 'f1-score': 0.8709491070150785, 'support': 12958}
        AUC: 0.6951516323807112
```

```
x_train=df_train.drop(['deposited?'],axis=1)
y_train=df_train['deposited?']
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

def log(x_train, x_test, y_train, y_test):
    model = LogisticRegression(max_iter=10000)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    return y_pred

# Get predictions
y_pred = log(x_train, x_test, y_train, y_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=[1, 0])

# Plot confusion matrix
plt.figure(figsize=(10, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Deposited', 'Not Deposited'], yticklabels=['Deposited', 'Not Deposited'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




## RidgeClassifier

```
Start coding or generate with AI.
```

```
def evaluation_metrics(y_test, y_pre, target_names):
    #scores
    print("Accuracy :",accuracy_score(y_test,y_pre))
    print("Precision :",precision_score(y_test,y_pre))
    print("Recall :",recall_score(y_test,y_pre))
    print("F1 Score :",f1_score(y_test,y_pre))
    print(classification_report(y_test, y_pre, target_names=target_names))

    #AUC
    fpr, tpr, _ = roc_curve(y_test,  y_pre)
    auc = roc_auc_score(y_test, y_pre)
    print('AUC :', auc)

    #ROC
    plt.plot(fpr,tpr,label="uc={:.3f})".format(auc))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('ROC curve')
    plt.legend(loc=4)
    plt.show()

    #CM matrix
    matrix = confusion_matrix(y_test, y_pre)
    cm = pd.DataFrame(matrix, index=target_names, columns=target_names)

    sns.heatmap(cm, annot=True, cbar=None, cmap="Blues", fmt = 'g')
    plt.title("Confusion Matrix"), plt.tight_layout()
    plt.ylabel("True Class"), plt.xlabel("Predicted Class")
    plt.show()
```

```
def Ridge(x_train,x_test,y_train,y_test):
    #train the model
    model = RidgeClassifier(random_state=2)
    model.fit(x_train, y_train)
    #predictions
    y_pre = model.predict(x_test)
    evaluation_metrics(y_test, y_pre, target_names)
```

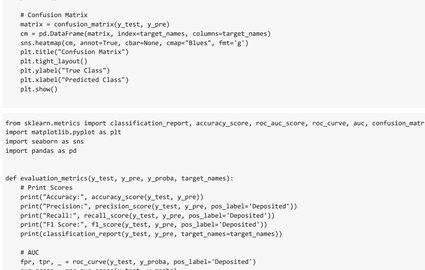
```
print(y_test.unique())
```

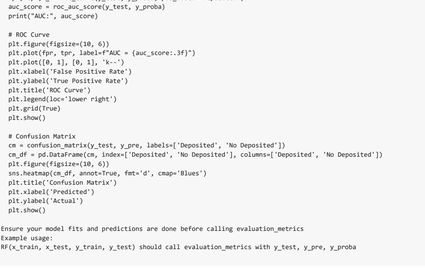
```
[0 1]
```

```
from sklearn.linear_model import RidgeClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, roc_curve, auc, confusion_matrix, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns

def evaluation_metrics(y_test, y_pred, target_names):
    # Scores
    report = classification_report(y_test, y_pred, target_names=target_names)
    print(report)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy}")

    # Precision, Recall, F1 Score
    print(f"Precision: {precision_score(y_test, y_pred, pos_label=1)}")
    print(f"Recall: {recall_score(y_test, y_pred, pos_label=1)}")
    print(f"F1 Score: {f1_score(y_test, y_pred, pos_label=1)}")

def Ridge(x_train, x_test, y_train, y_test):
    # Train the model
    model = RidgeClassifier(random_state=2)
    model.fit(x_train, y_train)
    # Predictions
    y_pred = model.predict(x_test)

    # Print evaluation metrics
    evaluation_metrics(y_test, y_pred, target_names=['No', 'Yes'])

    # Compute AUC
    y_scores = model.decision_function(x_test)
    auc_score = roc_auc_score(y_test, y_scores)
    print(f"AUC: {auc_score}")

    # Plot confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
    plt.title('Confusion Matrix for Ridge Classifier')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    # ROC Curve
    fpr, tpr, _ = roc_curve(y_test, y_scores)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, label=f'Ridge Classifier (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], linestyle='--', color='r')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve for Ridge Classifier')
    plt.legend()
    plt.grid(True)
    plt.show()

# Example usage
```

```
Ridge(x_train, x_test, y_train, y_test)
```

```
              precision    recall  f1-score   support

          No       0.90      0.98      0.94     11452
         Yes       0.62      0.19      0.29      1506

    accuracy                           0.89     12958
   macro avg       0.76      0.59      0.62     12958
weighted avg       0.87      0.89      0.87     12958

Accuracy: 0.8924216700108042
Precision: 0.6206896551724138
Recall: 0.19123505976095617
F1 Score: 0.2923857868020305
AUC: 0.7805671596997734
```




## Random forest Classifier

```
def RF(x_train,x_test,y_train,y_test):
  #train the model
  model = RandomForestClassifier(random_state=2)
  model.fit(x_train, y_train)
  #predictions
  y_pre = model.predict(x_test)
  evaluation_metrics(y_test, y_pre, target_names)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, roc_curve, roc_auc_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

def evaluation_metrics(y_test, y_pre, y_proba, target_names):
    # Scores
    print("Accuracy:", accuracy_score(y_test, y_pre))
    print("Precision:", precision_score(y_test, y_pre, pos_label='Deposited'))
    print("Recall:", recall_score(y_test, y_pre, pos_label='Deposited'))
    print("F1 Score:", f1_score(y_test, y_pre, pos_label='Deposited'))
    print(classification_report(y_test, y_pre, target_names=target_names))

    # AUC
    fpr, tpr, _ = roc_curve(y_test, y_proba, pos_label='Deposited')
    auc = roc_auc_score(y_test, y_proba)
    print("AUC:", auc)

    # ROC
    plt.plot(fpr, tpr, label="AUC={:.3f}".format(auc))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc=4)
    plt.show()

    # Confusion Matrix
    matrix = confusion_matrix(y_test, y_pre)
    cm = pd.DataFrame(matrix, index=target_names, columns=target_names)
    sns.heatmap(cm, annot=True, cbar=None, cmap="Blues", fmt='g')
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.ylabel("True Class")
    plt.xlabel("Predicted Class")
    plt.show()
```

```
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, roc_curve, auc, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd


def evaluation_metrics(y_test, y_pre, y_proba, target_names):
    # Print Scores
    print("Accuracy:", accuracy_score(y_test, y_pre))
    print("Precision:", precision_score(y_test, y_pre, pos_label='Deposited'))
    print("Recall:", recall_score(y_test, y_pre, pos_label='Deposited'))
    print("F1 Score:", f1_score(y_test, y_pre, pos_label='Deposited'))
    print(classification_report(y_test, y_pre, target_names=target_names))

    # AUC
    fpr, tpr, _ = roc_curve(y_test, y_proba, pos_label='Deposited')
    auc_score = roc_auc_score(y_test, y_proba)
    print("AUC:", auc_score)

    # ROC Curve
    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, label=f"AUC = {auc_score:.3f}")
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pre, labels=['Deposited', 'No Deposited'])
    cm_df = pd.DataFrame(cm, index=['Deposited', 'No Deposited'], columns=['Deposited', 'No Deposited'])
    plt.figure(figsize=(10, 6))
    sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

# Ensure your model fits and predictions are done before calling evaluation_metrics
# Example usage:
# RF(x_train, x_test, y_train, y_test) should call evaluation_metrics with y_test, y_pre, y_proba
```

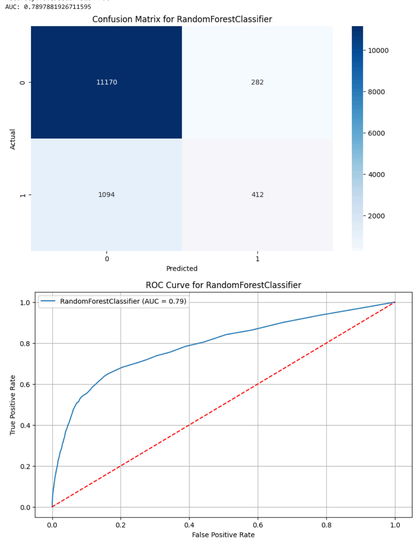
```
def RF(x_train,x_test,y_train,y_test):
  #train the model
  model = RandomForestClassifier(random_state=2)
  model.fit(x_train, y_train)
  #predictions
  y_pre = model.predict(x_test)
  evaluation_metrics(y_test, y_pre, target_names)
```

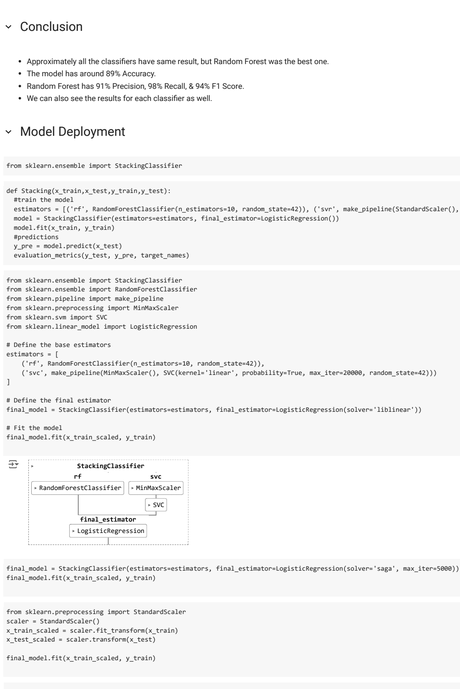
```
x_train=df_train.drop(['deposited?'],axis=1)
y_train=df_train['deposited?']
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, roc_curve, auc, confusion_matrix, precision_recall_curve
import matplotlib.pyplot as plt
import seaborn as sns
```

```
def evaluation_metrics(y_test, y_pre, y_proba, target_names):
    # Scores
    report = classification_report(y_test, y_pre, target_names=target_names)
    print(report)
    accuracy = accuracy_score(y_test, y_pre)
    print(f"Accuracy: {accuracy}")

    # AUC
    auc_score = roc_auc_score(y_test, y_proba)
    print(f"AUC: {auc_score}")

def RF(X_train, X_test, y_train, y_test):
    # Train the model
    model = RandomForestClassifier(random_state=2)
    model.fit(X_train, y_train)
    # Predictions
    y_pre = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]  # Get probability scores for the positive class
    # Print evaluation metrics
    evaluation_metrics(y_test, y_pre, y_proba, target_names=['No Deposited', 'Deposited'])

    # Plot confusion matrix
    unique_labels = y_test.unique()
    cm = confusion_matrix(y_test, y_pre, labels=unique_labels)
    plt.figure(figsize=(10, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=unique_labels, yticklabels=unique_labels)
    plt.title('Confusion Matrix for RandomForestClassifier')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    # ROC Curve
    fpr, tpr, _ = roc_curve(y_test, y_proba, pos_label=unique_labels[1])
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, label=f'RandomForestClassifier (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], linestyle='--', color='r')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve for RandomForestClassifier')
    plt.legend()
    plt.grid(True)
    plt.show()

# Example usage
RF(x_train, x_test, y_train, y_test)
```

```
                precision    recall  f1-score   support

    No Deposited       0.91      0.98      0.94     11452
       Deposited       0.59      0.27      0.37      1506

        accuracy                           0.89     12958
       macro avg       0.75      0.62      0.66     12958
    weighted avg       0.87      0.89      0.88     12958

    Accuracy: 0.8938107732674796
    AUC: 0.7897881926711595
```




## Conclusion

- Approximately all the classifiers have same result, but Random Forest was the best one.
- The model has around 89% Accuracy.
- Random Forest has 91% Precision, 98% Recall, & 94% F1 Score.
- We can also see the results for each classifier as well.

## Model Deployment

```
from sklearn.ensemble import StackingClassifier
```

```
def Stacking(x_train,x_test,y_train,y_test):
  #train the model
  estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42)), ('svr', make_pipeline(StandardScaler(), LinearSVC(random_state=42)))]
  model = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
  model.fit(x_train, y_train)
  #predictions
  y_pre = model.predict(x_test)
  evaluation_metrics(y_test, y_pre, target_names)
```

```
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

# Define the base estimators
estimators = [
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
    ('svc', make_pipeline(MinMaxScaler(), SVC(kernel='linear', probability=True, max_iter=20000, random_state=42)))
]

# Define the final estimator
final_model = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression(solver='liblinear'))

# Fit the model
final_model.fit(x_train_scaled, y_train)
```




```
final_model = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression(solver='saga', max_iter=5000))
final_model.fit(x_train_scaled, y_train)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

final_model.fit(x_train_scaled, y_train)
```