Ensemble Learning with Data Balancing Techniques

Ensemble learning: its a technique that combines multiple base models (weak learners) to produce a more robust and accurate predictive model. **Key Objectives**: Improve predictive performance, Reduce variance and overfitting, Enhance model stability.

This project demonstrates the use of **ensemble machine learning techniques** combined with **data balancing methods** to address imbalanced classification problems. By applying techniques such as **Random Forest (Bagging)**, **AdaBoost (Boosting)**, and **Stacking**, along with **SMOTE**, **Random Under-Sampling**, and **class weighting**, the project aims to improve prediction accuracy and robustness for imbalanced datasets.

🎯 Objective

The primary goals of this project are:

- ✅ To improve the predictive performance of machine learning models using ensemble techniques.
- ✅ To address the **class imbalance problem** using oversampling and undersampling methods.
- ✅ To evaluate and compare multiple ensemble models on a synthetic imbalanced dataset.
- ✅ To illustrate how **class weights** in logistic regression can help handle imbalance.

  ## 🧰 Technologies & Libraries Used

- Python
- Pandas, NumPy
- scikit-learn
- imbalanced-learn (`imblearn`)
- Matplotlib, Seaborn

📊 Step-by-Step Workflow

1️⃣ Data Generation

- A synthetic dataset is generated using `make_classification()` with a **class imbalance** (90% class 0, 10% class 1).

2️⃣ Data Splitting

- The dataset is split into **training and testing** sets (80/20 ratio).

3️⃣ Data Balancing

- **SMOTE**: Creates synthetic samples for the minority class to balance the dataset.
- **Random Under-Sampling**: Removes samples from the majority class to achieve balance.
- Visualizations show class distribution before and after balancing.

4️⃣ Model Training (Ensemble Techniques)

- **Random Forest** (Bagging)

- **AdaBoost** (Boosting)
- **Stacking Classifier** (combines Decision Tree + SVM with Logistic Regression as meta-learner) All models are trained using the **SMOTE-balanced data** for better generalization.

5️⃣ Model Evaluation

- Each ensemble model is evaluated using:
  - Classification Report (Precision, Recall, F1-Score)
  - Confusion Matrix (via heatmap)

6️⃣ Bonus: Logistic Regression with Class Weights

- Demonstrates how setting `class_weight='balanced'` in Logistic Regression helps handle imbalanced datasets **without resampling**.

🔍 Results & Observations

- **Random Forest and AdaBoost** generally provide better recall on the minority class compared to a simple logistic regression.
- **Stacking Classifier** performs competitively by combining the strengths of different models.
- **SMOTE** significantly improves classification performance by generating realistic synthetic examples.
- Using **class weights** is a useful alternative to resampling when working with small datasets.

🧪 Key Learnings

- Ensemble methods like Bagging, Boosting, and Stacking **reduce overfitting** and **increase stability**.
- Handling data imbalance is **crucial** for model performance in real-world scenarios like fraud detection, medical diagnoses, etc.
- Combining balancing techniques with ensemble models leads to more **robust and fair predictions**.

📁 File Structure

dataset1.ipynb # Jupyter Notebook with complete code and outputs

📌 Future Enhancements

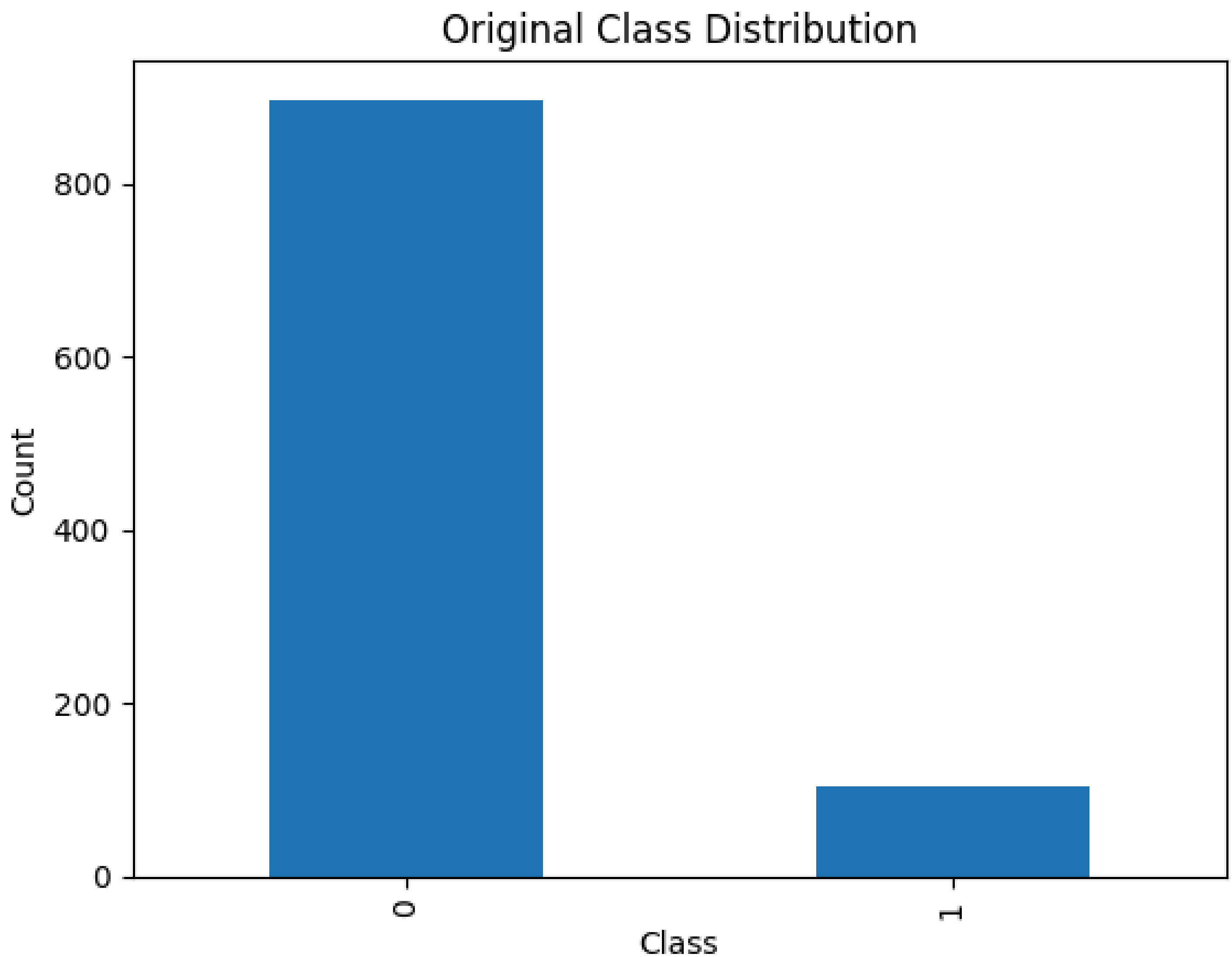
- Apply to real-world imbalanced datasets (e.g., credit card fraud).
- Compare with other boosting methods like **XGBoost**, **LightGBM**, or **CatBoost**.
- Integrate **cross-validation** to evaluate model stability.
- Deploy the best model via a Flask or Streamlit app.

Ensemble_balancing

```
# 📦 Step 1: Import Libraries
import pandas as pd
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
```

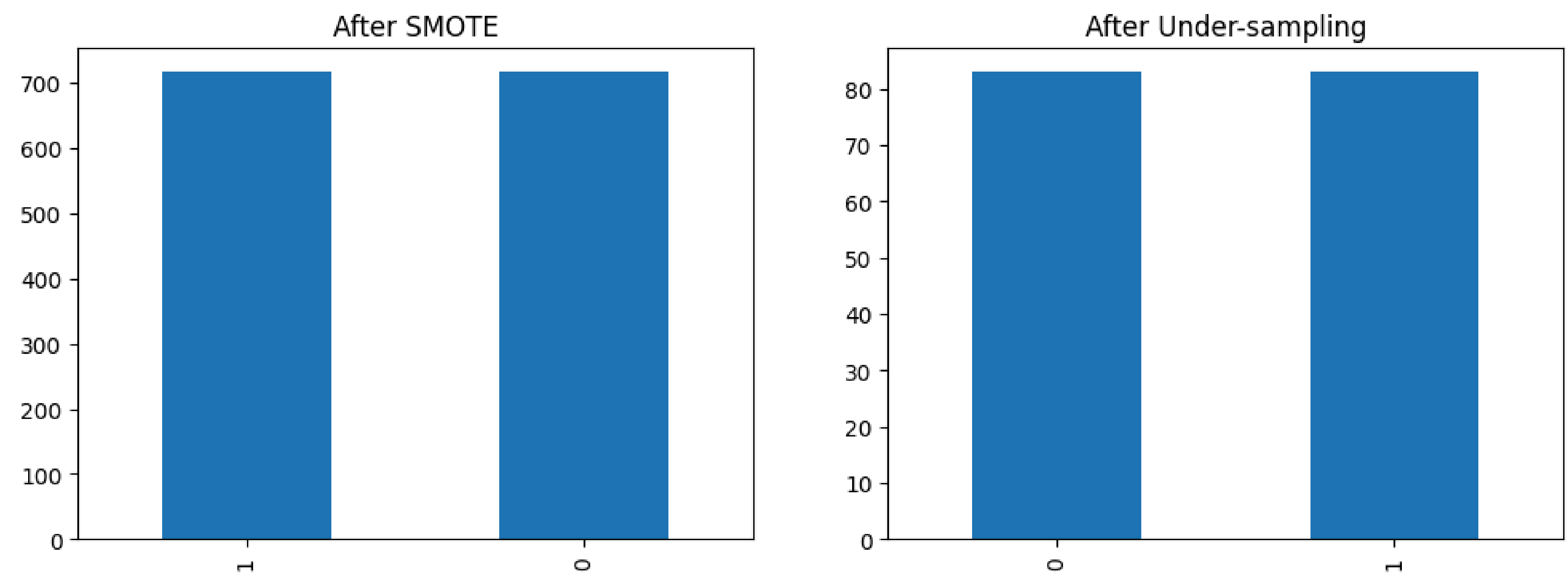
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
import matplotlib.pyplot as plt
import seaborn as sns

# Ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

```
# 🧰 Step 2: Create an Imbalanced Dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=2,
                           n_classes=2, weights=[0.9, 0.1], random_state=42)

# Visualize class distribution
pd.Series(y).value_counts().plot(kind='bar', title='Original Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



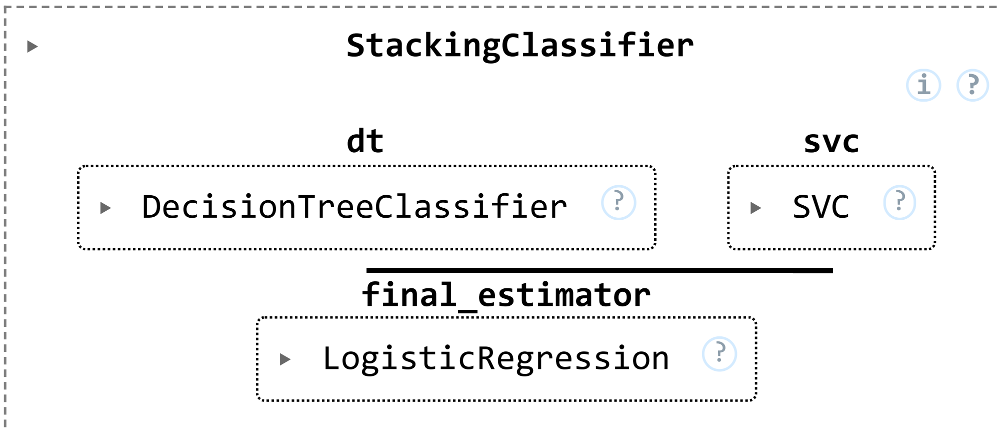

```
# ✂️ Step 3: Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# ⚖️ Step 4: Data Balancing Techniques

# ➕ SMOTE (Over-sampling)
smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X_train, y_train)

# ➖ Under-sampling
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X_train, y_train)

# Plot both
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
pd.Series(y_smote).value_counts().plot(kind='bar', ax=ax[0], title='After SMOTE')
pd.Series(y_rus).value_counts().plot(kind='bar', ax=ax[1], title='After Under-sampling')
plt.show()
```




```
# 🌲 Step 5: Train Common Ensemble Models

# Bagging → Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_smote, y_smote)

# Boosting → AdaBoost
ada_model = AdaBoostClassifier(n_estimators=100, random_state=42)
ada_model.fit(X_smote, y_smote)
```

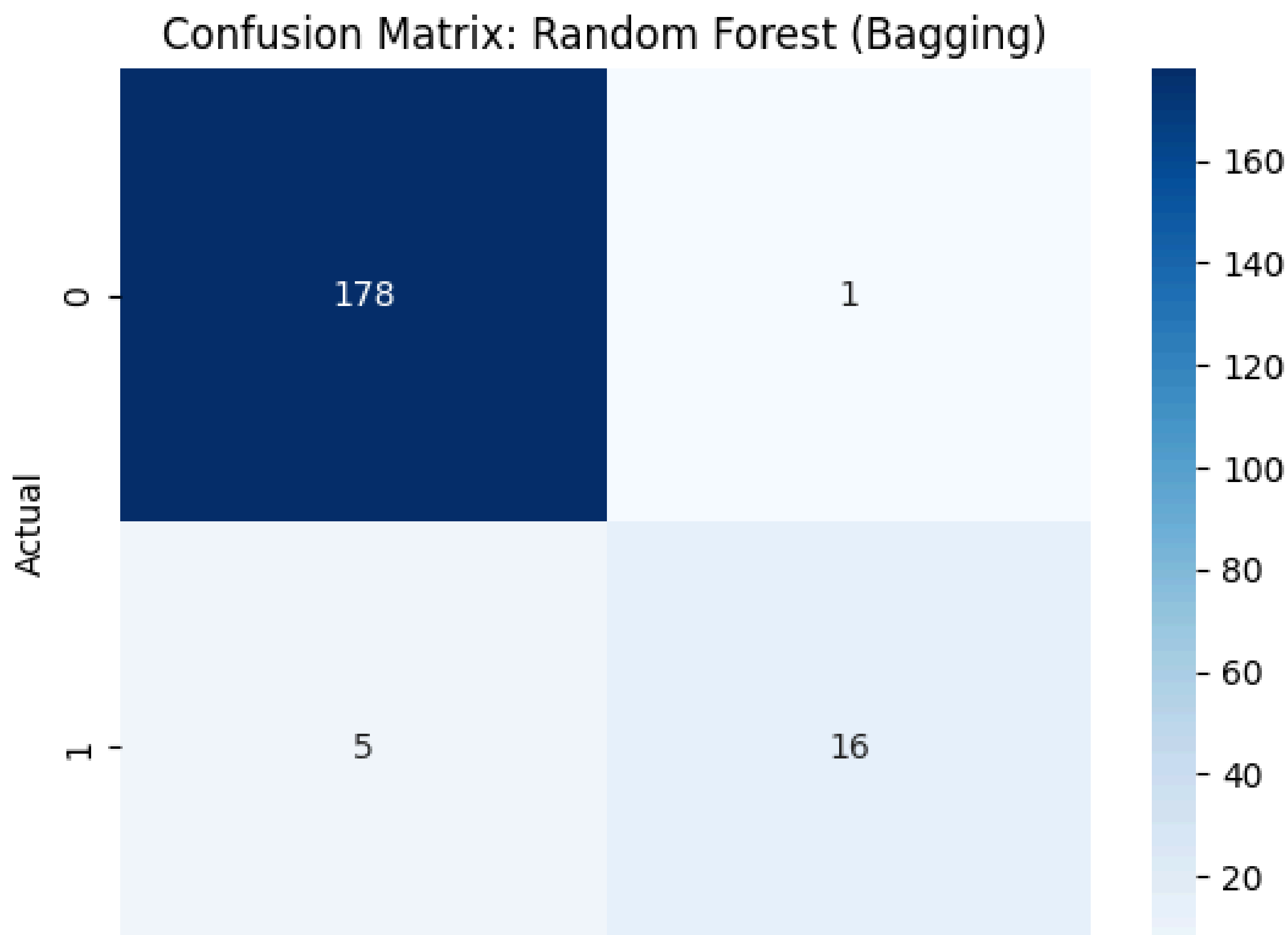
```python
# Stacking → Combine multiple base learners
base_learners = [
    ('dt', DecisionTreeClassifier()),
    ('svc', SVC(probability=True))
]
stack_model = StackingClassifier(estimators=base_learners, final_estimator=LogisticRegression())
stack_model.fit(X_smote, y_smote)
```




```python
# 🧪 Step 6: Evaluate All Models

models = {
    "Random Forest (Bagging)": rf_model,
    "AdaBoost (Boosting)": ada_model,
    "Stacking Classifier": stack_model
}

for name, model in models.items():
```

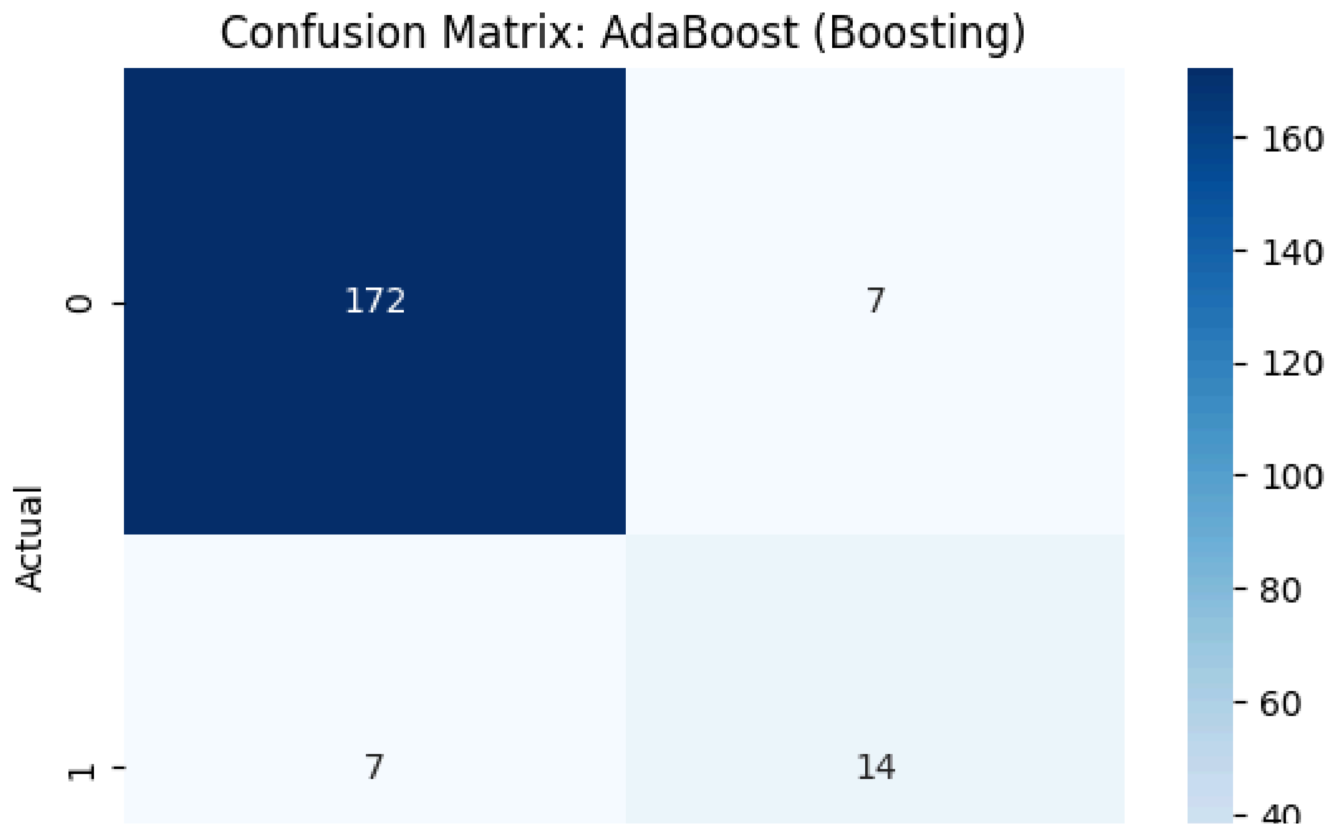
```
    print(f"\n🔎 Evaluation: {name}")
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
    plt.title(f"Confusion Matrix: {name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

```
🔎 Evaluation: Random Forest (Bagging)
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       179
           1       0.94      0.76      0.84        21

    accuracy                           0.97       200
   macro avg       0.96      0.88      0.91       200
weighted avg       0.97      0.97      0.97       200
```



🔎 Evaluation: AdaBoost (Boosting)

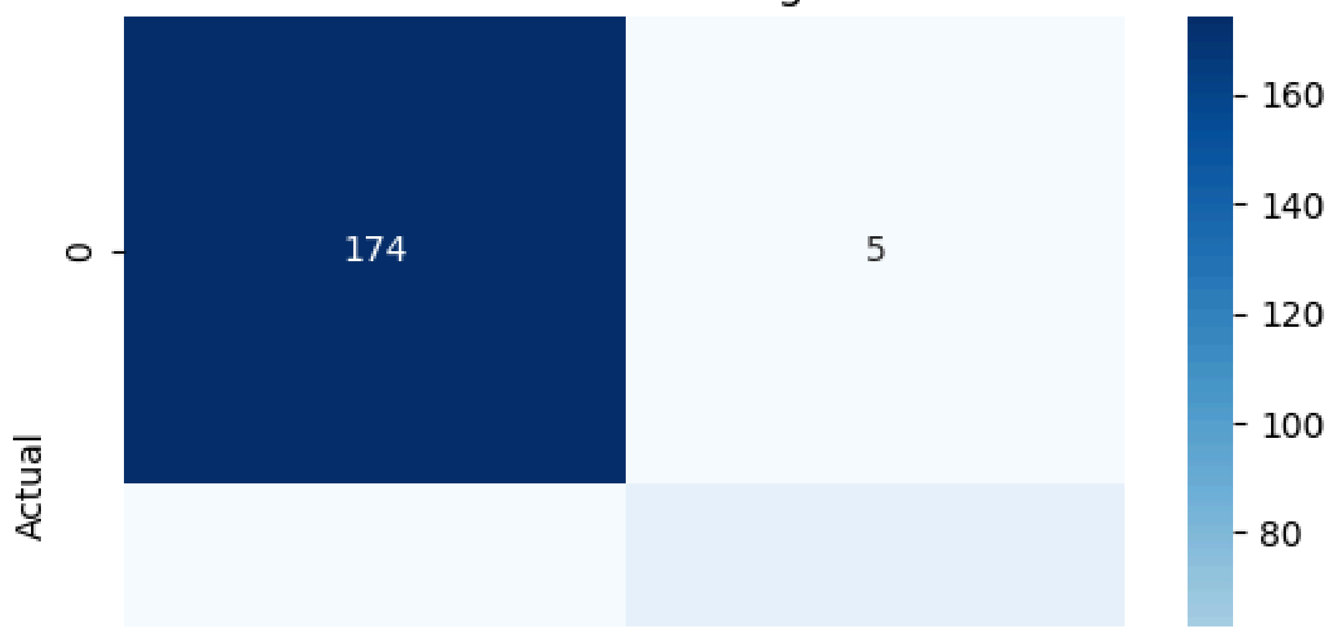
```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       179
           1       0.67      0.67      0.67        21

    accuracy                           0.93       200
   macro avg       0.81      0.81      0.81       200
weighted avg       0.93      0.93      0.93       200
```

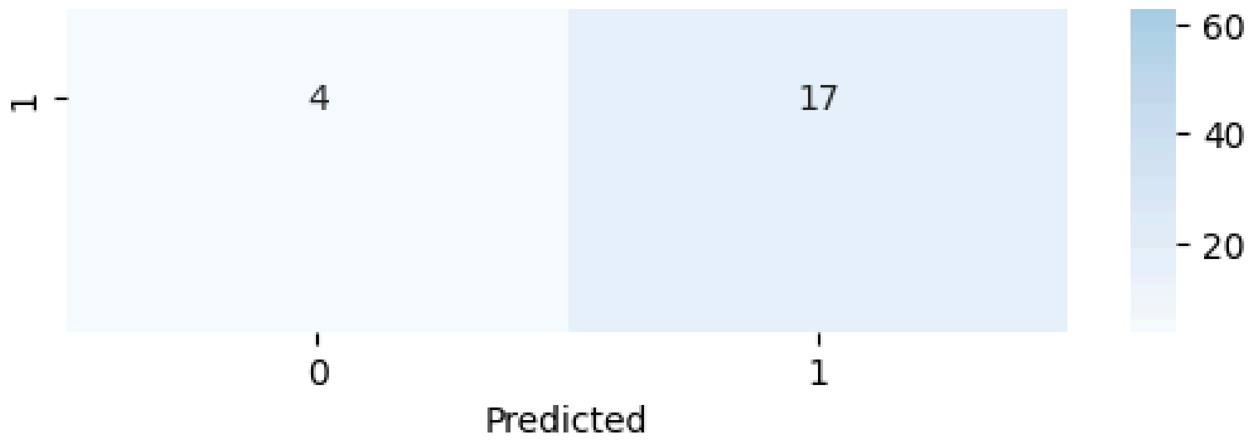


🔎 Evaluation: Stacking Classifier

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       179
           1       0.77      0.81      0.79        21

    accuracy                           0.95       200
   macro avg       0.88      0.89      0.88       200
weighted avg       0.96      0.95      0.96       200
```

Confusion Matrix: Stacking Classifier




1
4
17
60
40
20
0
1
Predicted

```
# 🎯 Step 7: Class Weights Example (Logistic Regression)
lr_weighted = LogisticRegression(class_weight='balanced')
lr_weighted.fit(X_train, y_train)
y_pred = lr_weighted.predict(X_test)

print("⚖️ Logistic Regression with Class Weights")
print(classification_report(y_test, y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Oranges')
```