1. Can we use Bagging for regression problems ?

Ans: Yes — **bagging is fully applicable to regression workloads**.

From a delivery standpoint, bagging (Bootstrap Aggregating) **strategically reduces variance** by training multiple regression models on different bootstrapped samples of the same dataset and **aggregating their outputs via averaging** rather than voting.

In short, at an execution level:

- Multiple regression models are trained in parallel on resampled data
- Each model generates a numeric prediction
- Final output is the **mean of all predictions**
- Net impact: **improved stability and generalization**, especially for high-variance regressors (e.g., decision tree regression)

Business value proposition:

- Mitigates overfitting risk
- Enhances predictive consistency
- Scales well with ensemble-friendly models

Real-world example:
Random Forest Regression is essentially a **production-grade bagging implementation** with decision trees as base learners.

2. What is the difference between multiple model training and single model training?

Ans: At a high level, the distinction can be framed as an execution and scalability decision:

- **Single model training** is a *focused delivery approach* where one model is trained to address a specific problem. It minimizes operational complexity, reduces compute overhead, and is easier to monitor and maintain, but it may underperform when the problem space is diverse or highly variable.
- **Multiple model training** is a *portfolio-based strategy* where several models are trained in parallel or sequence, often for comparison, specialization, or ensembling. This approach increases performance robustness and coverage across scenarios but introduces higher computational cost, orchestration effort, and lifecycle management complexity.

3. Explain the concept of feature randomness in Random Forest.

Ans: At a high level, **feature randomness** is a built-in diversification lever that Random Forests use to optimize portfolio performance across decision trees.

In practical terms, when each tree in the forest is trained, it does **not evaluate all input features** at every split. Instead, it samples a **random subset of features** and selects the best split only from that subset.

Business impact / value add:

- Reduces correlation between trees, avoiding groupthink.
- Improves generalization by minimizing overfitting risk.
- Increases model robustness and stability on unseen data.
- Delivers more reliable aggregate predictions when trees are combined.

4. What is OOB (Out-of-Bag) Score?

Ans: **Out-of-Bag (OOB) Score** is a built-in **performance KPI** used in **Random Forests** to evaluate model accuracy **without allocating a separate validation dataset**.

In execution terms:

- Each tree is trained on a bootstrapped sample (~63% of data).
- The remaining ~37% (out-of-bag data) acts as a **live validation set**.
- Predictions on this unused data are aggregated to compute the **OOB score**.

Bottom line:
OOB score provides a **cost-efficient, unbiased estimate of generalization performance** while optimizing data utilization and reducing validation overhead.

5. How can you measure the importance of features in a Random Forest model?

Ans: At a high level, feature importance in a Random Forest can be **operationalized** using a few standard levers:

- **Mean Decrease in Impurity (Gini / MSE importance):**
  Quantifies how much each feature contributes to reducing node impurity across all trees. Higher cumulative reduction ⇒ higher strategic value.
- **Permutation Importance:**
  Measures the drop in model performance when a feature's values are randomly shuffled. A larger performance degradation signals higher business impact.
- **Feature Usage Frequency:**
  Tracks how often a feature is selected for splits across the forest, serving as a proxy for influence.

6. Explain the working principle of a Bagging Classifier.

Ans: At a high level, a **Bagging (Bootstrap Aggregating) Classifier** operates on the principle of **risk diversification through parallelized learning**.

In execution terms:

- Multiple **bootstrap samples** are generated from the original dataset via random sampling with replacement.
- A **base classifier** (typically a high-variance model like a decision tree) is trained independently on each sample.
- All trained models run in parallel and produce individual predictions.
- The final output is **aggregated using majority voting**, ensuring consensus-driven decision-making.

7. How do you evaluate a Bagging Classifier's performance ?

Ans: From a stakeholder-aligned evaluation standpoint, a Bagging Classifier's performance is assessed by **benchmarking aggregated predictive outcomes against predefined success KPIs**.

In short:

- **Accuracy / Error Rate**: Measure overall prediction effectiveness on unseen data.
- **Precision, Recall, F1-score**: Validate class-wise performance, especially under imbalance risk.
- **Confusion Matrix**: Diagnose misclassification patterns and operational leakage.
- **Cross-validation scores**: Ensure performance stability and variance reduction benefits.
- **Out-of-bag (OOB) score**: Leverage internal validation to estimate generalization without extra data.

Collectively, these metrics provide a **holistic, risk-adjusted view of model robustness, scalability, and business readiness**.

8. How does a Bagging Regressor work?

Ans: At a high-level alignment, a **Bagging Regressor** operationalizes risk diversification by parallelizing multiple weak learners to stabilize output variance and improve predictive reliability.

Working mechanism (short, executive view):

- **Bootstrap Sampling:** The original dataset is replicated into multiple subsets via random sampling *with replacement*.
- **Model Training:** A separate regression model (typically a decision tree) is trained independently on each subset.

- **Parallel Execution:** All models learn in isolation, reducing correlation and overfitting exposure.
- **Aggregation:** Final prediction is produced by **averaging** the outputs of all models.

Net business impact:
Lower variance, higher robustness, and more consistent regression performance without increasing model bias.

9. What is the main advantage of ensemble techniques ?

Ans: At a high level, the **core value proposition of ensemble techniques** is **performance uplift through collective intelligence**.

In practical terms, ensembles **strategically combine multiple models to reduce individual weaknesses**, thereby **improving overall accuracy, robustness, and generalization** compared to any single model operating in isolation.

10. What is the main challenge of ensemble methods?

Ans: At a high level, the **primary challenge of ensemble methods** is **increased computational complexity**.

From a delivery standpoint:

- They require **higher training time and memory footprint**
- They introduce **operational complexity** in tuning and deployment
- They reduce **model interpretability**, making stakeholder explainability harder

Net-net, while ensembles optimize performance and robustness, they come with a **cost in scalability, speed, and transparency**.

11. Explain the key idea behind ensemble techniques.

Ans: At a high level, ensemble techniques **leverage strategic aggregation of multiple models to drive superior overall performance**.

Instead of relying on a single predictor, they **combine diverse learners to mitigate individual model weaknesses, reduce variance and bias, and improve robustness**, ultimately delivering more stable and accurate outcomes than any standalone model.

12. What is a Random Forest Classifier?

Ans: From a high-level stakeholder perspective, a **Random Forest Classifier** is a **robust ensemble learning model** that delivers reliable classification outcomes by aggregating the decisions of multiple decision trees.

In short:

- It builds **many decision trees** on different random subsets of data and features.
- Each tree provides a **classification vote**.
- The final output is determined by **majority voting**, improving accuracy and stability.
- This approach **reduces overfitting**, enhances generalization, and scales well for real-world datasets.

In essence, it's a **low-risk, high-confidence classification strategy** designed to improve predictive performance through diversification.

13. What are the main types of ensemble techniques?

Ans: At a high level, ensemble methodologies can be strategically aligned into three core buckets:

- **Bagging (Bootstrap Aggregation):** Parallel model deployment to reduce variance and improve stability.
- **Boosting:** Sequential model orchestration to minimize bias and incrementally improve weak learners.
- **Stacking (Stacked Generalization):** Layered model integration where a meta-learner optimizes final predictions.

These techniques collectively enable performance uplift through diversified model collaboration and risk mitigation.

14. What is ensemble learning in machine learning?

Ans: In alignment with industry best practices, **ensemble learning** is a machine learning strategy that **combines multiple models to drive superior predictive performance** versus any single model in isolation.

In simple terms:
multiple weak or diverse models are strategically aggregated to produce a more accurate, stable, and robust outcome.

Common enterprise-grade ensemble approaches include:

- **Bagging** (e.g., Random Forest) – reduces variance
- **Boosting** (e.g., AdaBoost, Gradient Boosting) – reduces bias
- **Stacking** – optimizes results via a meta-model

15. When should we avoid using ensemble methods?

Ans: From a strategic standpoint, you should avoid ensemble methods when:

- **Data is very small** → Ensembles can overfit and add unnecessary complexity.
- **Interpretability is crucial** → Individual models are easier to explain than ensembles.
- **Computational resources are limited** → Ensembles require more memory and processing.
- **Base model is already extremely strong** → Gains from ensembling may be negligible.

It's about balancing **performance vs complexity and cost**.

16. How does Bagging help in reducing overfitting ?

Ans: Bagging (Bootstrap Aggregating) reduces overfitting by **combining predictions from multiple models trained on different random subsets of the data**. This **averages out errors and variance**, so the final model is less sensitive to noise or peculiarities in a single dataset, leading to **more stable and generalizable predictions**.

17. Why is Random Forest better than a single Decision Tree?

Ans: Random Forest outperforms a single Decision Tree because it **aggregates multiple trees** to reduce **overfitting** and **variance**, leading to **more stable and accurate predictions**. Essentially, it leverages the "wisdom of the crowd" instead of relying on a single tree's decision.

18. What is the role of bootstrap sampling in Bagging?

Ans: In a **nutshell**, **bootstrap sampling** in Bagging is used to **create multiple random subsets** of the original dataset **with replacement**, so that each base model trains on a slightly different dataset.

This **reduces variance** and **improves overall model stability** by averaging predictions from diverse models.

Think of it as giving each model its **own mini "view" of the data** to make the ensemble more robust.

19. What are some real-world applications of ensemble techniques ?

Ans: Leveraging a synergy of predictive models, ensemble techniques optimize decision-making across multiple domains. Key real-world applications include:

- **Finance:** Credit scoring, fraud detection, stock price prediction.

- **Healthcare:** Disease diagnosis, patient risk stratification, medical image analysis.
- **E-commerce:** Product recommendation systems, customer churn prediction.
- **Cybersecurity:** Intrusion detection, spam filtering, malware classification.
- **Marketing:** Customer segmentation, campaign response prediction.
- **Weather & Environment:** Rainfall prediction, climate modeling, air quality forecasting.

In essence, ensembles enhance accuracy, robustness, and generalization by integrating the strengths of multiple models.

20. What is the difference between Bagging and Boosting?

Ans: Here's a concise corporate-style breakdown:

Bagging (Bootstrap Aggregating):

- Objective: Reduce **variance**.
- Approach: Train **multiple models independently** on different random subsets of data.
- Voting/Averaging: Combine predictions **equally** (majority vote or average).
- Example: **Random Forest**.

Boosting:

- Objective: Reduce **bias** (and some variance).
- Approach: Train **models sequentially**, where each new model focuses on **errors of previous models**.
- Weighted Voting: Combine predictions with **weights based on model performance**.
- Example: **AdaBoost, XGBoost**.

TL;DR: Bagging = parallel & reduces variance; Boosting = sequential & reduces bias.

If you want, I can make a **1-line super-easy memory trick** for them.

Practical

21. Train a Bagging Classifier using Decision Trees on a sample dataset and print model accuracy.

Ans:

```
# Import required libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split dataset into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize base estimator
base_tree = DecisionTreeClassifier(random_state=42)

# Initialize Bagging Classifier (updated keyword)
bagging_model = BaggingClassifier(estimator=base_tree,  # changed from base_estimator
                                  n_estimators=50,      # Number of trees
                                  random_state=42,
                                  bootstrap=True)
```

```
# Train the model
bagging_model.fit(X_train, y_train)

# Make predictions
y_pred = bagging_model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Bagging Classifier Accuracy: {accuracy:.2f}")
```

O/P: Bagging Classifier Accuracy: 1.00

22. Train a Bagging Regressor using Decision Trees and evaluate using Mean Squared Error (MSE).

Ans:

```
# Import necessary libraries
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Load California Housing dataset
data = fetch_california_housing()
X = data.data
y = data.target

# Split dataset into training and testing sets
```

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

# Initialize Decision Tree Regressor
base_model = DecisionTreeRegressor(random_state=42)

# Initialize Bagging Regressor with Decision Trees (modern sklearn)
bagging_model = BaggingRegressor(
    estimator=base_model,   # <- updated parameter name
    n_estimators=100,
    random_state=42
)

# Fit the model
bagging_model.fit(X_train, y_train)

# Predict on test data
y_pred = bagging_model.predict(X_test)

# Evaluate performance using Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.4f}")
```

O/P : Mean Squared Error (MSE): 0.2559

23. Train a Random Forest Classifier on the Breast Cancer dataset and print feature importance scores.

Ans:
```
# Step 1: Import necessary libraries
from sklearn.datasets import load_breast_cancer
```

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd

# Step 2: Load the dataset
data = load_breast_cancer()
X = data.data
y = data.target
feature_names = data.feature_names

# Step 3: Initialize and train the Random Forest Classifier
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X, y)

# Step 4: Get feature importance scores
feature_importances = rf_clf.feature_importances_

# Step 5: Create a DataFrame to display feature importance
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Step 6: Print the feature importance
print(importance_df)
```

O/P :

	Feature	Importance
23	worst area	0.139357
27	worst concave points	0.132225
7	mean concave points	0.107046
20	worst radius	0.082848
22	worst perimeter	0.080850

2 mean perimeter 0.067990

6 mean concavity 0.066917

3 mean area 0.060462

26 worst concavity 0.037339

0 mean radius 0.034843

13 area error 0.029553

25 worst compactness 0.019864

21 worst texture 0.017485

1 mean texture 0.015225

10 radius error 0.014264

24 worst smoothness 0.012232

5 mean compactness 0.011597

12 perimeter error 0.010085

28 worst symmetry 0.008179

4 mean smoothness 0.007958

19 fractal dimension error 0.005942

16 concavity error 0.005820

15 compactness error 0.005612

14 smoothness error 0.004722

...

11 texture error 0.003744

18 symmetry error 0.003546

8 mean symmetry 0.003423

9 mean fractal dimension 0.002615

24. Train a Random Forest Regressor and compare its performance with a single Decision Tree.

Ans: # Import necessary libraries

import numpy as np

from sklearn.datasets import fetch_california_housing # Modern regression dataset

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
data = fetch_california_housing()
X = data.data
y = data.target

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize models
dt_model = DecisionTreeRegressor(random_state=42)
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train Decision Tree
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

# Train Random Forest
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Evaluate performance
def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f"{model_name} Performance:")
```

```
    print(f"  Mean Squared Error: {mse:.3f}")
    print(f"  R2 Score: {r2:.3f}\n")

evaluate_model(y_test, y_pred_dt, "Decision Tree")
evaluate_model(y_test, y_pred_rf, "Random Forest")
```

O/P : Decision Tree Performance:
  Mean Squared Error: 0.495
  R2 Score: 0.622

Random Forest Performance:
  Mean Squared Error: 0.255
  R2 Score: 0.805

25. Compute the Out-of-Bag (OOB) Score for a Random Forest Classifier.

Ans:
```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split into training and testing sets (optional, since OOB is internal)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Random Forest with OOB enabled
rf = RandomForestClassifier(n_estimators=100, oob_score=True, random_state=42)
```

```
# Fit the model
rf.fit(X_train, y_train)

# Compute Out-of-Bag score
oob_score = rf.oob_score_

print(f"Out-of-Bag Score: {oob_score:.4f}")
```

O/P : Out-of-Bag Score: 0.9167

26. Train a Bagging Classifier using SVM as a base estimator and print accuracy.

Ans:

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define SVM as base estimator
svm_base = SVC(kernel='linear', probability=True, random_state=42)

# Create Bagging Classifier using SVM as base estimator (use 'estimator' now)
bagging_model = BaggingClassifier(estimator=svm_base, n_estimators=10,
random_state=42)
```

```
# Train the model
bagging_model.fit(X_train, y_train)

# Predict on test set
y_pred = bagging_model.predict(X_test)

# Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Bagging Classifier Accuracy with SVM: {accuracy:.4f}")
```

O/P : Bagging Classifier Accuracy with SVM: 1.0000

27. Train a Random Forest Classifier with different numbers of trees and compare accuracy.

Ans:

```
# Importing necessary libraries
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X = data.data
y = data.target

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Define different numbers of trees to evaluate
n_trees_options = [10, 50, 100, 200, 500]

# Store accuracy results
accuracy_results = {}

# Iterative model training and evaluation
for n_trees in n_trees_options:
    clf = RandomForestClassifier(n_estimators=n_trees, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_results[n_trees] = accuracy
    print(f"Number of Trees: {n_trees}, Accuracy: {accuracy:.4f}")

# Optional: Visualize results
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
plt.plot(list(accuracy_results.keys()), list(accuracy_results.values()), marker='o')
plt.title("Random Forest Accuracy vs Number of Trees")
plt.xlabel("Number of Trees")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```

O/P : Bagging Classifier Accuracy with SVM: 1.0000

28. Train a Bagging Classifier using Logistic Regression as a base estimator and print AUC score.

Ans:

```
# Import required libraries
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import roc_auc_score

# Step 1: Generate a synthetic binary classification dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
                           n_redundant=5, random_state=42)

# Step 2: Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Initialize Logistic Regression as the base estimator
base_lr = LogisticRegression(solver='liblinear', random_state=42)

# Step 4: Initialize Bagging Classifier (use 'estimator' instead of 'base_estimator')
bagging_clf = BaggingClassifier(estimator=base_lr, n_estimators=50,
                                random_state=42, n_jobs=-1)

# Step 5: Train the Bagging Classifier
bagging_clf.fit(X_train, y_train)

# Step 6: Predict probabilities on the test set
y_prob = bagging_clf.predict_proba(X_test)[:, 1]  # Probability for positive class
```

```
# Step 7: Calculate AUC score
auc = roc_auc_score(y_test, y_prob)
print(f"AUC Score: {auc:.4f}")
```

O/P : AUC Score: 0.9137

29. Train a Random Forest Regressor and analyze feature importance scores.

Ans:

```
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load California Housing dataset
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target, name='MedHouseVal')

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train Random Forest Regressor
rf_model = RandomForestRegressor(
    n_estimators=100,
```

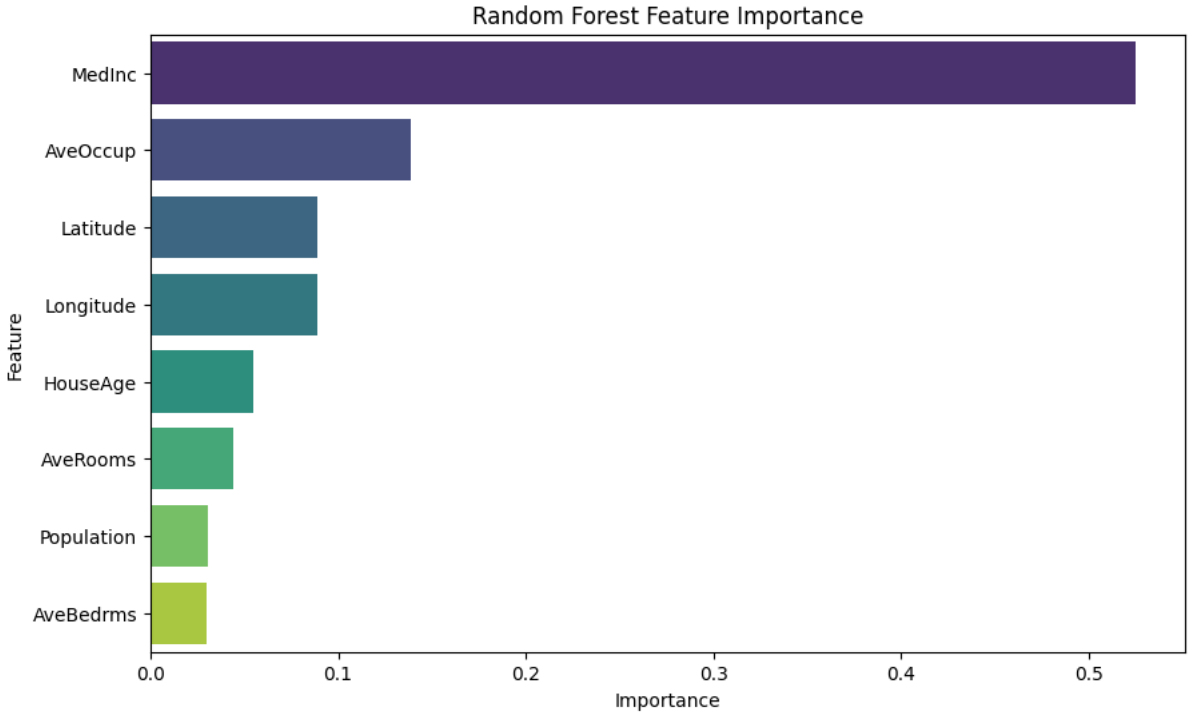
```
    max_depth=None,
    random_state=42
)
rf_model.fit(X_train, y_train)

# Predict and evaluate
y_pred = rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.3f}")
print(f"R² Score: {r2:.3f}")

# Feature importance analysis
feature_importances = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

print("\nFeature Importance Scores:")
print(feature_importances)

# Visualize feature importance
plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feature_importances, palette='viridis')
plt.title('Random Forest Feature Importance')
plt.show()
```

O/P: Mean Squared Error: 0.255

R² Score: 0.805

Feature Importance Scores:

	Feature	Importance
0	MedInc	0.524871
5	AveOccup	0.138443
6	Latitude	0.088936
7	Longitude	0.088629
1	HouseAge	0.054593
2	AveRooms	0.044272
4	Population	0.030650
3	AveBedrms	0.029606



30. Train an ensemble model using both Bagging and Random Forest and compare accuracy.

Ans:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Bagging with Decision Tree as base estimator
bagging_model = BaggingClassifier(
    estimator=DecisionTreeClassifier(),  # updated argument name
    n_estimators=100,
    random_state=42
)

# Random Forest
rf_model = RandomForestClassifier(
    n_estimators=100,
    random_state=42
)

# Train models
bagging_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)

# Predictions
```

```
bagging_preds = bagging_model.predict(X_test)
rf_preds = rf_model.predict(X_test)

# Accuracy evaluation
bagging_acc = accuracy_score(y_test, bagging_preds)
rf_acc = accuracy_score(y_test, rf_preds)

# Compare results
print(f"Bagging Classifier Accuracy: {bagging_acc:.4f}")
print(f"Random Forest Classifier Accuracy: {rf_acc:.4f}")

if bagging_acc > rf_acc:
    print("Bagging outperforms Random Forest on this dataset.")
elif bagging_acc < rf_acc:
    print("Random Forest outperforms Bagging on this dataset.")
else:
    print("Both models achieve the same accuracy.")
```

O/P: Bagging Classifier Accuracy: 1.0000
Random Forest Classifier Accuracy: 1.0000
Both models achieve the same accuracy.

31. Train a Random Forest Classifier and tune hyperparameters using GridSearchCV.

Ans:

```
# Step 1: Import required libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
# Step 2: Load dataset (example: Iris dataset)
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

# Step 3: Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

# Step 4: Initialize the Random Forest Classifier
rf = RandomForestClassifier(random_state=42)

# Step 5: Define hyperparameter grid for optimization
param_grid = {
   'n_estimators': [50, 100, 200],
   'max_depth': [None, 5, 10, 20],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4],
   'max_features': ['auto', 'sqrt', 'log2']
}

# Step 6: Configure GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

# Step 7: Fit the model
grid_search.fit(X_train, y_train)

# Step 8: Output best parameters and best score
print("Best Hyperparameters:", grid_search.best_params_)
```

```
print("Best Accuracy Score:", grid_search.best_score_)

# Step 9: Evaluate on test set
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nTest Accuracy:", accuracy_score(y_test, y_pred))
```

32. Train a Bagging Regressor with different numbers of base estimators and compare performance.

Ans:

```
# Import necessary libraries
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load California Housing dataset
data = fetch_california_housing()
X, y = data.data, data.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define different numbers of base estimators to test
n_estimators_list = [5, 10, 20, 50, 100]
```

```
# Store results
mse_list = []

# Loop over different numbers of base estimators
for n_estimators in n_estimators_list:
    # Initialize Bagging Regressor with Decision Tree as base estimator
    bagging_model = BaggingRegressor(
        estimator=DecisionTreeRegressor(),
        n_estimators=n_estimators,
        random_state=42
    )

    # Train the model
    bagging_model.fit(X_train, y_train)

    # Make predictions
    y_pred = bagging_model.predict(X_test)

    # Calculate Mean Squared Error
    mse = mean_squared_error(y_test, y_pred)
    mse_list.append(mse)

    print(f"Number of Estimators: {n_estimators}, MSE: {mse:.4f}")

# Plot performance vs. number of estimators
plt.figure(figsize=(8,5))
plt.plot(n_estimators_list, mse_list, marker='o')
plt.title("Bagging Regressor Performance vs Number of Estimators")
plt.xlabel("Number of Estimators")
```

plt.ylabel("Mean Squared Error")

plt.grid(True)

plt.show()

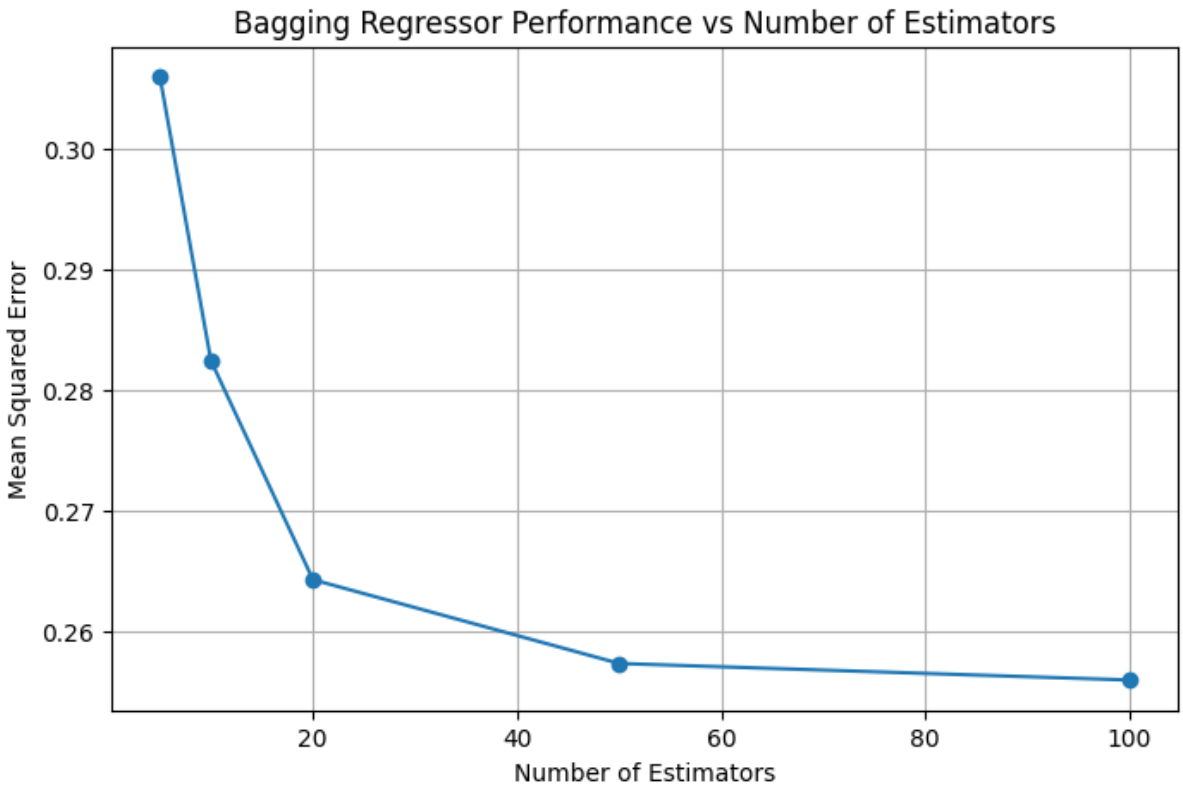
O/P : Number of Estimators: 5, MSE: 0.3060

Number of Estimators: 10, MSE: 0.2824

Number of Estimators: 20, MSE: 0.2643

Number of Estimators: 50, MSE: 0.2573

Number of Estimators: 100, MSE: 0.2559



33. Train a Random Forest Classifier and analyze misclassified samples.

Ans: # Step 1: Import Libraries

import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix


# Step 2: Load Dataset

data = load_iris()

X = data.data

y = data.target

feature_names = data.feature_names

target_names = data.target_names


# Step 3: Split into Train and Test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)


# Step 4: Train Random Forest Classifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

rf_model.fit(X_train, y_train)


# Step 5: Make Predictions

y_pred = rf_model.predict(X_test)


# Step 6: Evaluate Performance

print("Classification Report:\n", classification_report(y_test, y_pred,
target_names=target_names))

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))


# Step 7: Analyze Misclassified Samples

misclassified_idx = np.where(y_test != y_pred)[0]

print(f"Number of misclassified samples: {len(misclassified_idx)}\n")


if len(misclassified_idx) > 0:
```

```
    misclassified_samples = pd.DataFrame(X_test[misclassified_idx],
columns=feature_names)
    misclassified_samples['True Label'] = y_test[misclassified_idx]
    misclassified_samples['Predicted Label'] = y_pred[misclassified_idx]
    print("Misclassified Samples:\n", misclassified_samples)
else:
    print("No misclassified samples!")

# Optional: Feature Importance Analysis
import matplotlib.pyplot as plt
import seaborn as sns

feature_importances = rf_model.feature_importances_
sns.barplot(x=feature_importances, y=feature_names)
plt.title("Feature Importance in Random Forest")
plt.show()
```

O/P :

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	0.82	0.90	0.86	10
virginica	0.89	0.80	0.84	10
accuracy			0.90	30
macro avg	0.90	0.90	0.90	30
weighted avg	0.90	0.90	0.90	30

Confusion Matrix:

```
 [[10  0  0]
```

[0 9 1]

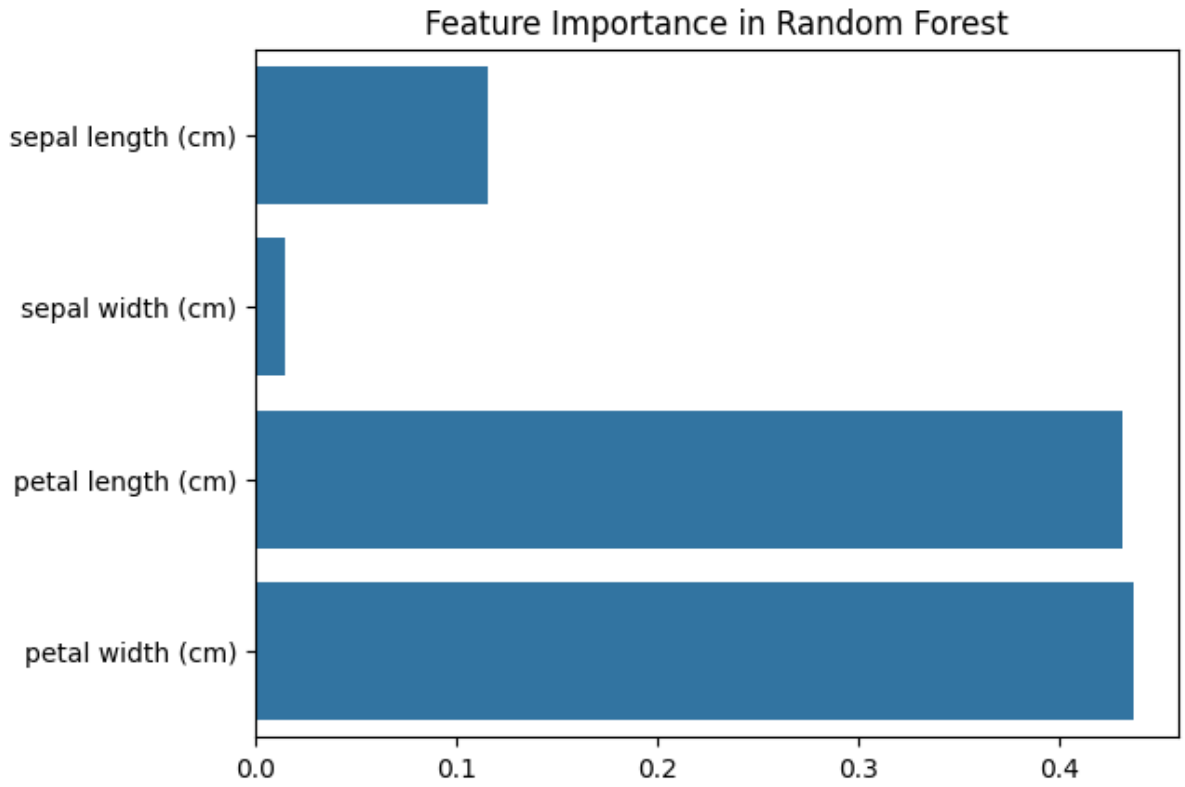
[0 2 8]]

Number of misclassified samples: 3

Misclassified Samples:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
0	6.0	3.0	4.8	1.8	
1	6.1	2.6	5.6	1.4	
2	6.7	3.0	5.0	1.7	

	True Label	Predicted Label
0	2	1
1	2	1
2	1	2



34. Train a Bagging Classifier and compare its performance with a single Decision Tree Classifier.

Ans: # Step 1: Import necessary libraries

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score, classification_report

# Step 2: Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 3: Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 4: Initialize a single Decision Tree Classifier
dt = DecisionTreeClassifier(random_state=42)

# Step 5: Train the Decision Tree
dt.fit(X_train, y_train)

# Step 6: Predict and evaluate Decision Tree
y_pred_dt = dt.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Decision Tree Accuracy:", accuracy_dt)
print("Decision Tree Classification Report:\n", classification_report(y_test, y_pred_dt))

# Step 7: Initialize a Bagging Classifier with Decision Tree as base estimator
```

```
bagging = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(),
    n_estimators=50,          # number of trees
    max_samples=0.8,          # fraction of samples for each tree
    max_features=1.0,         # fraction of features for each tree
    random_state=42
)

# Step 8: Train the Bagging Classifier
bagging.fit(X_train, y_train)

# Step 9: Predict and evaluate Bagging Classifier
y_pred_bag = bagging.predict(X_test)
accuracy_bag = accuracy_score(y_test, y_pred_bag)
print("Bagging Classifier Accuracy:", accuracy_bag)
print("Bagging Classifier Classification Report:\n", classification_report(y_test, y_pred_bag))

# Step 10: Comparison Summary
print(f"\nPerformance Comparison:\nDecision Tree: {accuracy_dt:.4f}\nBagging Classifier: {accuracy_bag:.4f}")
```

O/P : Decision Tree Accuracy: 1.0

Decision Tree Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45

macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

35. Train a Random Forest Classifier and visualize the confusion matrix.

Ans:

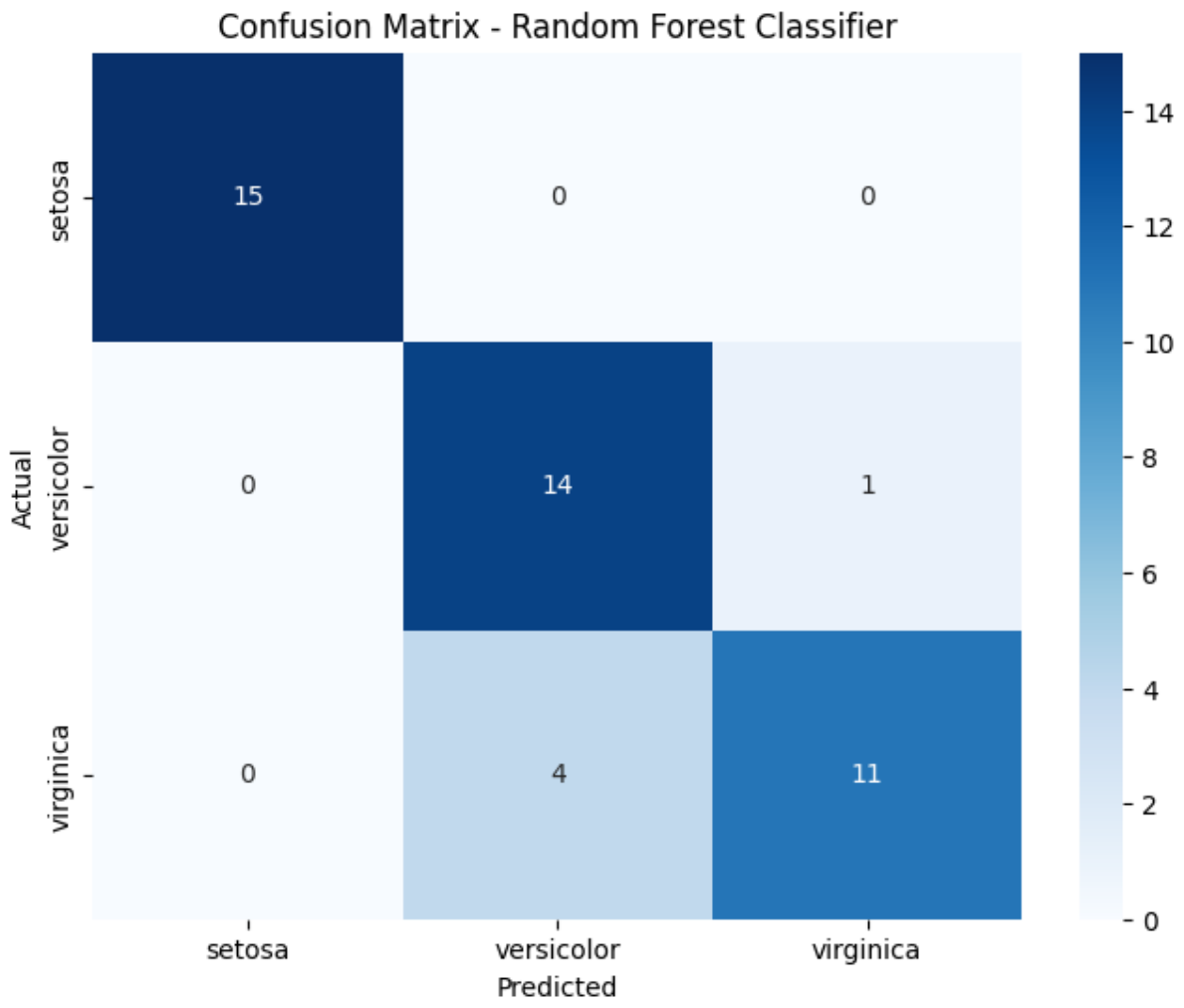
```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report

# Step 1: Load dataset (example: Iris dataset)
data = load_iris()
X = data.data
y = data.target
class_names = data.target_names

# Step 2: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Step 3: Initialize and train Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Step 4: Make predictions
y_pred = rf_model.predict(X_test)
```

```
# Step 5: Evaluate model
print("Classification Report:\n", classification_report(y_test, y_pred))

# Step 6: Confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Step 7: Visualize confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Random Forest Classifier')
plt.show()
```

O/P:

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	15
1	0.78	0.93	0.85	15
2	0.92	0.73	0.81	15
accuracy			0.89	45
macro avg	0.90	0.89	0.89	45
weighted avg	0.90	0.89	0.89	45



36. Train a Stacking Classifier using Decision Trees, SVM, and Logistic Regression, and compare accuracy.

Ans: # Import necessary libraries

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import StackingClassifier

from sklearn.metrics import accuracy_score

```
# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Scale features (important for SVM & Logistic Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define base classifiers
base_classifiers = [
    ('decision_tree', DecisionTreeClassifier(random_state=42)),
    ('svm', SVC(probability=True, random_state=42)),  # probability=True needed for stacking
    ('logreg', LogisticRegression(max_iter=1000, random_state=42))
]

# Define stacking classifier with Logistic Regression as final estimator
stacking_clf = StackingClassifier(
    estimators=base_classifiers,
    final_estimator=LogisticRegression(),
    passthrough=True  # optional: pass original features to final estimator
)

# Train stacking classifier
stacking_clf.fit(X_train_scaled, y_train)
```

```
# Predictions
y_pred_stack = stacking_clf.predict(X_test_scaled)

# Accuracy of stacking classifier
stacking_accuracy = accuracy_score(y_test, y_pred_stack)
print(f"Stacking Classifier Accuracy: {stacking_accuracy:.4f}")

# Compare with individual base classifiers
for name, clf in base_classifiers:
    clf.fit(X_train_scaled, y_train)
    y_pred = clf.predict(X_test_scaled)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {acc:.4f}")
```

O/P : Stacking Classifier Accuracy: 0.9333
decision_tree Accuracy: 0.9111
svm Accuracy: 0.9333
logreg Accuracy: 0.9111

37. Train a Random Forest Classifier and print the top 5 most important features.

Ans:

```
# Importing necessary libraries
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris  # Example dataset

# Load dataset (replace with your own dataset if needed)
data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
```

```
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Feature importance extraction
feature_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
top_5_features = feature_importances.sort_values(ascending=False).head(5)

print("Top 5 Most Important Features:")
print(top_5_features)
```

O/P : Top 5 Most Important Features:

```
petal width (cm)     0.433982
petal length (cm)    0.417308
sepal length (cm)    0.104105
sepal width (cm)     0.044605
dtype: float64
```

38. Train a Bagging Classifier and evaluate performance using Precision, Recall, and F1-score.

Ans:

```
# Step 1: Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
```

```
# Step 2: Load the dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 3: Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Step 4: Initialize the Bagging Classifier
# Using DecisionTree as base estimator
bagging_model = BaggingClassifier(
    estimator=DecisionTreeClassifier(),  # updated parameter
    n_estimators=50,                     # number of trees
    random_state=42
)

# Step 5: Train the model
bagging_model.fit(X_train, y_train)

# Step 6: Make predictions
y_pred = bagging_model.predict(X_test)

# Step 7: Evaluate performance
report = classification_report(y_test, y_pred, target_names=iris.target_names)
print("Performance Metrics:\n")
print(report)
```

O/P : # Step 1: Import necessary libraries

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

# Step 2: Load the dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 3: Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Step 4: Initialize the Bagging Classifier
# Using DecisionTree as base estimator
bagging_model = BaggingClassifier(
    estimator=DecisionTreeClassifier(),  # updated parameter
    n_estimators=50,                     # number of trees
    random_state=42
)

# Step 5: Train the model
bagging_model.fit(X_train, y_train)

# Step 6: Make predictions
y_pred = bagging_model.predict(X_test)
```

```
# Step 7: Evaluate performance
report = classification_report(y_test, y_pred, target_names=iris.target_names)
print("Performance Metrics:\n")
print(report)
```

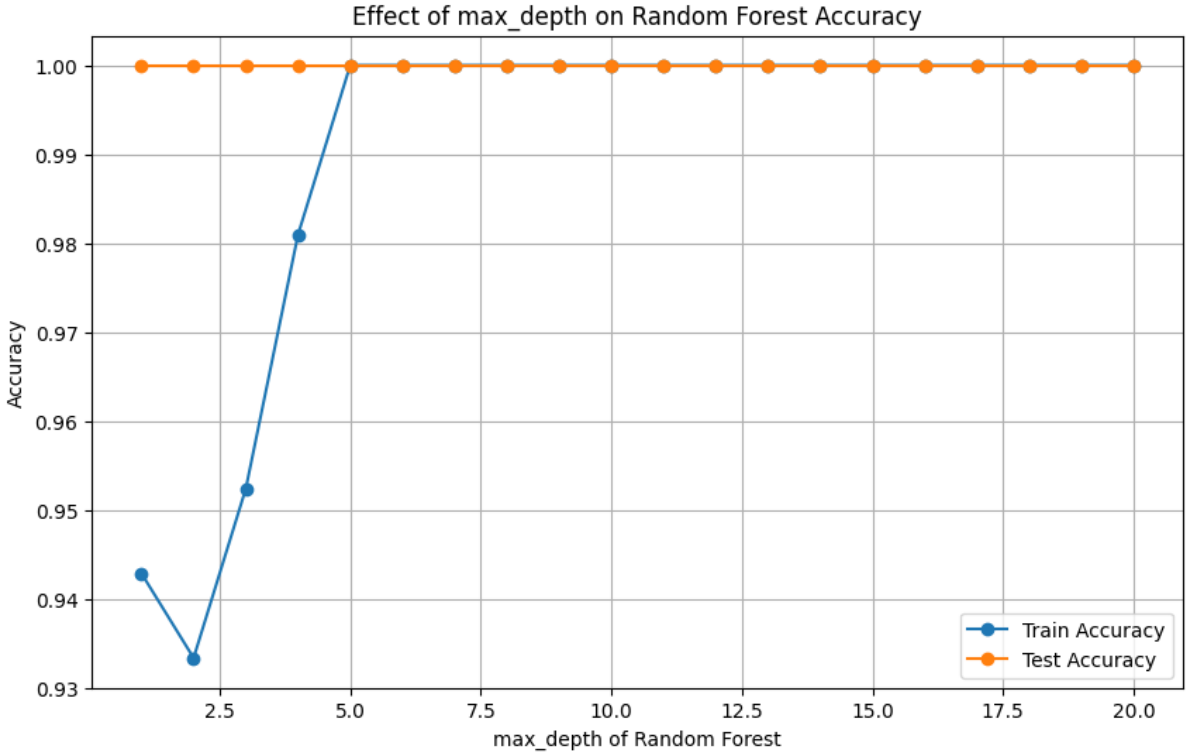
39. Train a Random Forest Classifier and analyze the effect of max_depth on accuracy.

Ans:

```
# Step 1: Import libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Step 2: Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 3: Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 4: Initialize variables to store results
max_depth_values = list(range(1, 21))  # Testing max_depth from 1 to 20
train_accuracies = []
test_accuracies = []

# Step 5: Train Random Forest with different max_depth values
for depth in max_depth_values:
    clf = RandomForestClassifier(max_depth=depth, n_estimators=100, random_state=42)
```

```
    clf.fit(X_train, y_train)

    # Predict on training and test data
    y_train_pred = clf.predict(X_train)
    y_test_pred = clf.predict(X_test)

    # Calculate accuracy
    train_accuracies.append(accuracy_score(y_train, y_train_pred))
    test_accuracies.append(accuracy_score(y_test, y_test_pred))

# Step 6: Plot the results
plt.figure(figsize=(10,6))
plt.plot(max_depth_values, train_accuracies, label='Train Accuracy', marker='o')
plt.plot(max_depth_values, test_accuracies, label='Test Accuracy', marker='o')
plt.xlabel('max_depth of Random Forest')
plt.ylabel('Accuracy')
plt.title('Effect of max_depth on Random Forest Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

O/P :



40. Train a Bagging Regressor using different base estimators (DecisionTree and KNeighbors) and compare performance.

Ans: # Import necessary libraries

```
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score


# Load dataset
data = fetch_california_housing()
X, y = data.data, data.target


# Split into train and test sets
```

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

# Initialize base estimators
dtree = DecisionTreeRegressor(random_state=42)
knn = KNeighborsRegressor(n_neighbors=5)

# Create Bagging Regressors (use 'estimator' instead of 'base_estimator')
bag_dtree = BaggingRegressor(estimator=dtree, n_estimators=50, random_state=42)
bag_knn = BaggingRegressor(estimator=knn, n_estimators=50, random_state=42)

# Train the models
bag_dtree.fit(X_train, y_train)
bag_knn.fit(X_train, y_train)

# Make predictions
y_pred_dtree = bag_dtree.predict(X_test)
y_pred_knn = bag_knn.predict(X_test)

# Evaluate performance
print("Bagging with Decision Tree Regressor")
print("R2 Score:", r2_score(y_test, y_pred_dtree))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_dtree))

print("\nBagging with KNeighbors Regressor")
print("R2 Score:", r2_score(y_test, y_pred_knn))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_knn))
```

O/P : Bagging with Decision Tree Regressor

R2 Score: 0.8036499747356253

Mean Squared Error: 0.2572988359842641

Bagging with KNeighbors Regressor

R2 Score: 0.1786722263202739

Mean Squared Error: 1.0762752887085227

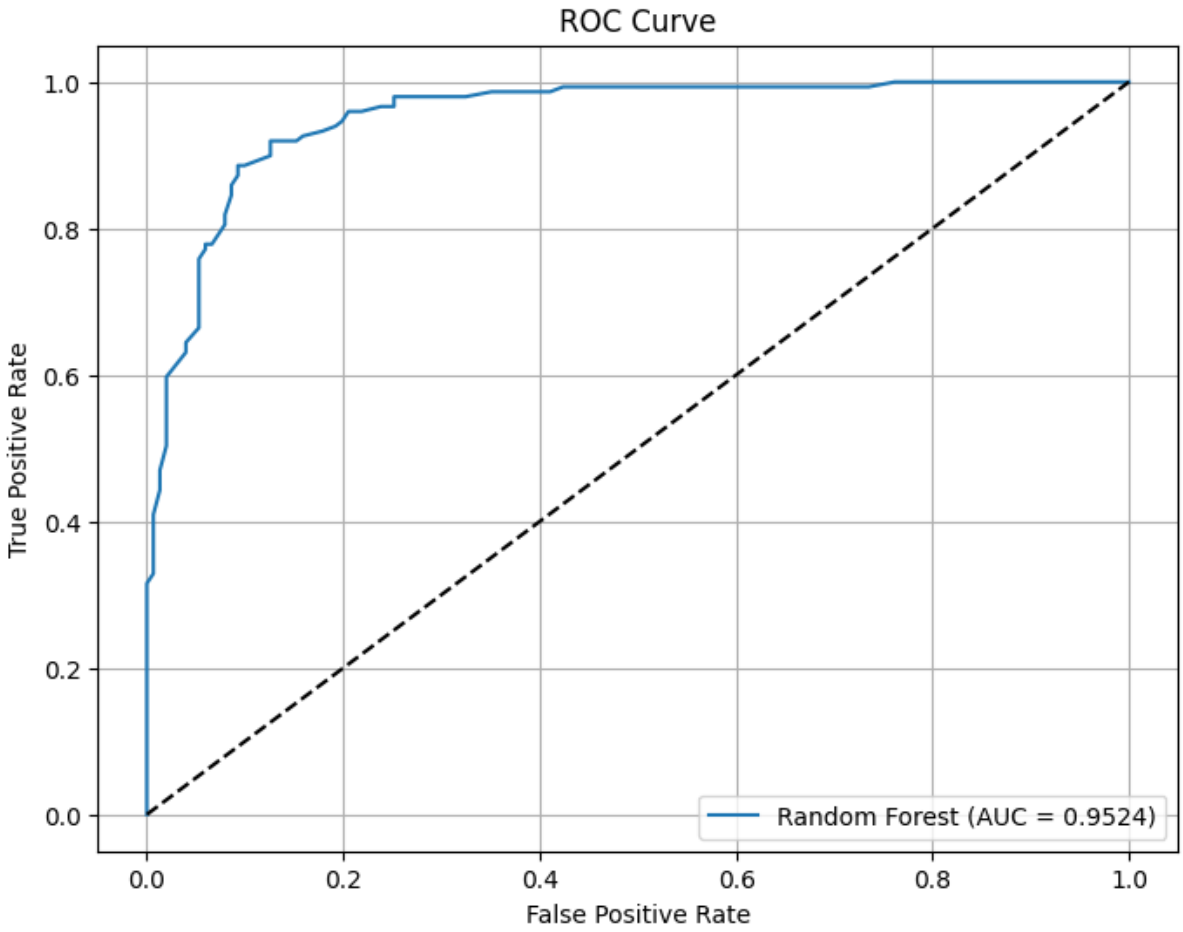
41. Train a Random Forest Classifier and evaluate its performance using ROC-AUC Score.

Ans: # Step 1: Import the necessary libraries

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, roc_curve

# Step 2: Generate a synthetic dataset (you can replace this with your own dataset)
X, y = make_classification(n_samples=1000, n_features=20,
                           n_informative=15, n_redundant=5,
                           n_classes=2, random_state=42)

# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=42,
                                                    stratify=y)

# Step 4: Initialize and train the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Step 5: Get predicted probabilities for the positive class
y_pred_proba = rf_model.predict_proba(X_test)[:, 1]
```

```
# Step 6: Calculate the ROC-AUC score
roc_auc = roc_auc_score(y_test, y_pred_proba)
print(f"ROC-AUC Score: {roc_auc:.4f}")

# Optional: Plot ROC Curve
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'Random Forest (AUC = {roc_auc:.4f})')
plt.plot([0,1], [0,1], 'k--')  # Diagonal line for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```

O/P : ROC-AUC Score: 0.9524



42. Train a Bagging Classifier and evaluate its performance using cross-validatio.

Ans: # Import necessary libraries

from sklearn.datasets import load_iris

from sklearn.model_selection import cross_val_score

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

import numpy as np

Load dataset

data = load_iris()

X = data.data

y = data.target

```
# Initialize base estimator
base_estimator = DecisionTreeClassifier()

# Initialize Bagging Classifier with updated parameter
bagging_clf = BaggingClassifier(
    estimator=base_estimator,  # <-- updated from base_estimator
    n_estimators=50,
    max_samples=0.8,
    max_features=1.0,
    random_state=42
)

# Evaluate using cross-validation
cv_scores = cross_val_score(bagging_clf, X, y, cv=5, scoring='accuracy')

# Print results
print("Cross-Validation Accuracy Scores:", cv_scores)
print("Mean CV Accuracy:", np.mean(cv_scores))
print("Standard Deviation:", np.std(cv_scores))
```

O/P : Cross-Validation Accuracy Scores: [0.96666667 0.96666667 0.93333333 0.93333333 1.]

Mean CV Accuracy: 0.96

Standard Deviation: 0.024944382578492935

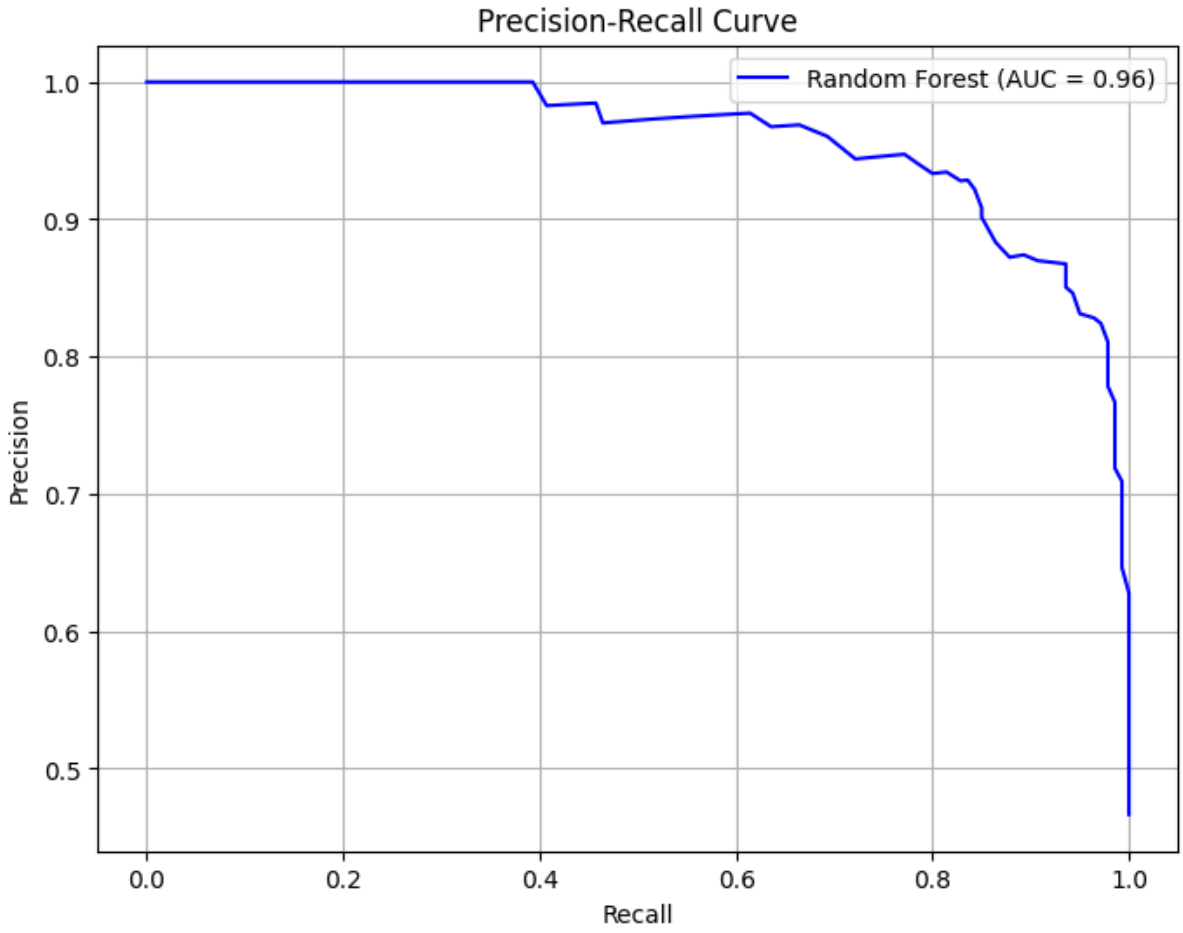
43. Train a Random Forest Classifier and plot the Precision-Recall curv.

Ans:

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_recall_curve, auc

# Step 1: Generate a sample dataset (binary classification)
X, y = make_classification(n_samples=1000, n_features=20,
                           n_informative=15, n_redundant=5,
                           random_state=42)

# Step 2: Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=42)

# Step 3: Train a Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Step 4: Predict probabilities for the positive class
y_probs = rf.predict_proba(X_test)[:, 1]

# Step 5: Compute Precision-Recall values
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)
pr_auc = auc(recall, precision)

# Step 6: Plot the Precision-Recall curve
plt.figure(figsize=(8,6))
plt.plot(recall, precision, label=f'Random Forest (AUC = {pr_auc:.2f})', color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
```

```
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True)
plt.show()
```



44. Train a Stacking Classifier with Random Forest and Logistic Regression and compare accuracy.

Ans:
```
# Import libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define base models
rf = RandomForestClassifier(n_estimators=100, random_state=42)
lr = LogisticRegression(max_iter=1000, random_state=42)

# Define Stacking Classifier with Logistic Regression as final estimator
stacking_clf = StackingClassifier(
    estimators=[('rf', rf), ('lr', lr)],
    final_estimator=LogisticRegression(),
    cv=5
)

# Train individual models
rf.fit(X_train, y_train)
lr.fit(X_train, y_train)
stacking_clf.fit(X_train, y_train)

# Make predictions
rf_pred = rf.predict(X_test)
lr_pred = lr.predict(X_test)
stack_pred = stacking_clf.predict(X_test)

# Compute and display accuracy
print(f"Random Forest Accuracy: {accuracy_score(y_test, rf_pred):.4f}")
```

```
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, lr_pred):.4f}")
print(f"Stacking Classifier Accuracy: {accuracy_score(y_test, stack_pred):.4f}")
```

O/P : Random Forest Accuracy: 1.0000

Logistic Regression Accuracy: 1.0000

Stacking Classifier Accuracy: 1.0000

45. Train a Bagging Regressor with different levels of bootstrap samples and compare performance.

Ans:
```
# Step 1: Import libraries
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Step 2: Load dataset
data = fetch_california_housing()
X = data.data
y = data.target

# Step 3: Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Define bootstrap sample sizes to test
bootstrap_levels = [0.5, 0.7, 1.0]  # Fraction of training samples for each estimator

# Step 5: Train Bagging Regressor with different bootstrap samples
results = {}
```

```
for sample_frac in bootstrap_levels:
    bag_reg = BaggingRegressor(
        estimator=DecisionTreeRegressor(),
        n_estimators=100,
        max_samples=sample_frac,
        bootstrap=True,
        random_state=42
    )
    bag_reg.fit(X_train, y_train)
    y_pred = bag_reg.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    results[sample_frac] = mse
    print(f"Bootstrap fraction: {sample_frac}, Test MSE: {mse:.4f}")

# Step 6: Compare results
best_fraction = min(results, key=results.get)
print(f"\nBest bootstrap fraction: {best_fraction} with MSE: {results[best_fraction]:.4f}")
```

O/P : Bootstrap fraction: 0.5, Test MSE: 0.2641

Bootstrap fraction: 0.7, Test MSE: 0.2590

Bootstrap fraction: 1.0, Test MSE: 0.2559

Best bootstrap fraction: 1.0 with MSE: 0.2559