Ensemble learing

Voting Classifier

In [1]:

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import ExtraTreesClassifier, VotingClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# Load the dataset
df = pd.read_csv('loan_data.csv')  # Update with your actual path

# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode categorical features
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])

# Separate features and target variable
X = df.drop('not.fully.paid', axis=1)  # Features
y = df['not.fully.paid']  # Target variable

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

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_siz

# Define classifiers with increased max_iter for Logistic Regression
clf1 = LogisticRegression(max_iter=200)  # Increase max_iter
clf2 = DecisionTreeClassifier()
clf3 = ExtraTreesClassifier()
clf4 = SVC(probability=True)

# Evaluating each classifier individually
classifiers = [clf1, clf2, clf3, clf4]
for clf in classifiers:
    scores = cross_val_score(clf, X_train, y_train, cv=10, scoring='accur
    print(f"{clf.__class__.__name__} Cross-validated accuracy: {np.round(

# Hard Voting
hard_voting_clf = VotingClassifier(estimators=[('lr', clf1), ('dt', clf2)
hard_voting_scores = cross_val_score(hard_voting_clf, X_train, y_train, c
print("Hard Voting Classifier Cross-validated accuracy:", np.round(np.mea

# Soft Voting
soft_voting_clf = VotingClassifier(estimators=[('lr', clf1), ('dt', clf2)
soft_voting_scores = cross_val_score(soft_voting_clf, X_train, y_train, c
```

```python
print("Soft Voting Classifier Cross-validated accuracy:", np.round(np.mea

# Weighted Voting
for i in range(1, 4):
    for j in range(1, 4):
        for k in range(1, 4):
            weighted_voting_clf = VotingClassifier(estimators=[('lr', clf
                                                    voting='soft',
                                                    weights=[i, j, k])
            weighted_scores = cross_val_score(weighted_voting_clf, X_trai
            print(f"Weighted Voting (weights={i},{j},{k}) accuracy:", np.

# Classifiers of the same algorithm
# Using SVC with different kernels
svm1 = SVC(probability=True, kernel='linear')
svm2 = SVC(probability=True, kernel='poly', degree=2)
svm3 = SVC(probability=True, kernel='rbf')

svm_classifiers = [svm1, svm2, svm3]
for svm in svm_classifiers:
    svm_scores = cross_val_score(svm, X_train, y_train, cv=10, scoring='a
    print(f"{svm.__class__.__name__} Cross-validated accuracy: {np.round(

# Soft Voting with SVM classifiers
svm_voting_clf = VotingClassifier(estimators=[('svm1', svm1), ('svm2', sv
svm_voting_scores = cross_val_score(svm_voting_clf, X_train, y_train, cv=
print("SVM Soft Voting Classifier Cross-validated accuracy:", np.round(np

# Fit the best model and predict
best_model = soft_voting_clf  # Choose based on your evaluation
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

# Calculate accuracy on test set
accuracy = accuracy_score(y_test, y_pred)
print("Test set accuracy:", np.round(accuracy, 2))
```

```
LogisticRegression Cross-validated accuracy: 0.84
DecisionTreeClassifier Cross-validated accuracy: 0.74
ExtraTreesClassifier Cross-validated accuracy: 0.84
SVC Cross-validated accuracy: 0.84
Hard Voting Classifier Cross-validated accuracy: 0.84
Soft Voting Classifier Cross-validated accuracy: 0.84
Weighted Voting (weights=1,1,1) accuracy: 0.82
Weighted Voting (weights=1,1,2) accuracy: 0.83
Weighted Voting (weights=1,1,3) accuracy: 0.84
Weighted Voting (weights=1,2,1) accuracy: 0.73
Weighted Voting (weights=1,2,2) accuracy: 0.78
Weighted Voting (weights=1,2,3) accuracy: 0.82
Weighted Voting (weights=1,3,1) accuracy: 0.73
Weighted Voting (weights=1,3,2) accuracy: 0.73
Weighted Voting (weights=1,3,3) accuracy: 0.76
Weighted Voting (weights=2,1,1) accuracy: 0.84
Weighted Voting (weights=2,1,2) accuracy: 0.84
Weighted Voting (weights=2,1,3) accuracy: 0.84
Weighted Voting (weights=2,2,1) accuracy: 0.79
Weighted Voting (weights=2,2,2) accuracy: 0.82
Weighted Voting (weights=2,2,3) accuracy: 0.83
Weighted Voting (weights=2,3,1) accuracy: 0.74
Weighted Voting (weights=2,3,2) accuracy: 0.76
Weighted Voting (weights=2,3,3) accuracy: 0.8
Weighted Voting (weights=3,1,1) accuracy: 0.84
Weighted Voting (weights=3,1,2) accuracy: 0.84
Weighted Voting (weights=3,1,3) accuracy: 0.84
Weighted Voting (weights=3,2,1) accuracy: 0.82
Weighted Voting (weights=3,2,2) accuracy: 0.83
Weighted Voting (weights=3,2,3) accuracy: 0.83
Weighted Voting (weights=3,3,1) accuracy: 0.76
Weighted Voting (weights=3,3,2) accuracy: 0.8
Weighted Voting (weights=3,3,3) accuracy: 0.82
SVC Cross-validated accuracy: 0.84
SVC Cross-validated accuracy: 0.84
SVC Cross-validated accuracy: 0.84
SVM Soft Voting Classifier Cross-validated accuracy: 0.84
Test set accuracy: 0.84
```

Voting (regressor)

In [4]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import ExtraTreesRegressor, VotingRegressor
from sklearn.svm import SVR

# Load the dataset
dataframe = pd.read_csv('loan_data.csv')  # Update with your actual path

# Identify categorical columns
categorical_columns = dataframe.select_dtypes(include=['object']).columns

# Create a LabelEncoder object
label_encoder = LabelEncoder()
```

```python
# Encode categorical features
for column in categorical_columns:
    dataframe[column] = label_encoder.fit_transform(dataframe[column])

# Separate features and target variable
features = dataframe.drop('int.rate', axis=1)  # Features
target = dataframe['int.rate']  # Target variable

# Scale the data
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_spl

# Define regressors for voting
estimators = [
    ('linear_regression', LinearRegression()),
    ('decision_tree', DecisionTreeRegressor()),
    ('extra_trees', ExtraTreesRegressor()),
    ('support_vector_regression', SVR())
]

# Evaluate individual regressors
for name, regressor in estimators:
    scores = cross_val_score(regressor, features_train, target_train, sco
    print(f"{name} - Cross-validated R²: {np.round(np.mean(scores), 2)}")

# Voting Regressor
voting_regressor = VotingRegressor(estimators)
voting_regressor.fit(features_train, target_train)
voting_scores = cross_val_score(voting_regressor, features_train, target_
print("Voting Regressor - Cross-validated R²:", np.round(np.mean(voting_s

# Fit each regressor and get predictions
predictions_train = []
predictions_test = []

for name, regressor in estimators:
    regressor.fit(features_train, target_train)
    target_train_predicted = regressor.predict(features_train)
    target_test_predicted = regressor.predict(features_test)

    # Store predictions for averaging
    predictions_train.append(target_train_predicted)
    predictions_test.append(target_test_predicted)

    # Calculate and display metrics
    mse_train = mean_squared_error(target_train, target_train_predicted)
    r2_train = r2_score(target_train, target_train_predicted)
    mse_test = mean_squared_error(target_test, target_test_predicted)
    r2_test = r2_score(target_test, target_test_predicted)

    print(f"{name} - Train Set MSE: {np.round(mse_train, 2)}, R²: {np.rou
    print(f"{name} - Test Set MSE: {np.round(mse_test, 2)}, R²: {np.round

# Average predictions for train and test set
average_train_prediction = np.mean(predictions_train, axis=0)
average_test_prediction = np.mean(predictions_test, axis=0)
```

```
# Calculate MSE and R² for averaged predictions
mse_average_train = mean_squared_error(target_train, average_train_predic
r2_average_train = r2_score(target_train, average_train_prediction)

mse_average_test = mean_squared_error(target_test, average_test_predictio
r2_average_test = r2_score(target_test, average_test_prediction)

print("Average Predictions Train Set MSE:", np.round(mse_average_train, 2
print("Average Predictions Train Set R²:", np.round(r2_average_train, 2))
print("Average Predictions Test Set MSE:", np.round(mse_average_test, 2))
print("Average Predictions Test Set R²:", np.round(r2_average_test, 2))
```

```
linear_regression - Cross-validated R²: 0.65
decision_tree - Cross-validated R²: 0.5
extra_trees - Cross-validated R²: 0.74
support_vector_regression - Cross-validated R²: -0.34
Voting Regressor - Cross-validated R²: 0.65
linear_regression - Train Set MSE: 0.0, R²: 0.65
linear_regression - Test Set MSE: 0.0, R²: 0.64

decision_tree - Train Set MSE: 0.0, R²: 1.0
decision_tree - Test Set MSE: 0.0, R²: 0.52

extra_trees - Train Set MSE: 0.0, R²: 1.0
extra_trees - Test Set MSE: 0.0, R²: 0.74

support_vector_regression - Train Set MSE: 0.0, R²: -0.34
support_vector_regression - Test Set MSE: 0.0, R²: -0.32

Average Predictions Train Set MSE: 0.0
Average Predictions Train Set R²: 0.85
Average Predictions Test Set MSE: 0.0
Average Predictions Test Set R²: 0.66
```

Bagging (Regressor)

Simple linear Regression (MSE check)

In [30]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
dataframe = pd.read_csv('loan_data.csv')  # Update with your actual path

# Identify categorical columns
categorical_columns = dataframe.select_dtypes(include=['object']).columns

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode categorical features
for column in categorical_columns:
    dataframe[column] = label_encoder.fit_transform(dataframe[column])

# Separate features and target variable
features = dataframe.drop('int.rate', axis=1)  # Features
target = dataframe['int.rate']  # Target variable
```

```
# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_spl

# Initialize the Linear Regression model
model = LinearRegression()

# Fit the model on the training data
model.fit(features_train, target_train)

# Make predictions on the test data
target_pred = model.predict(features_test)

# Evaluate the model
mse = mean_squared_error(target_test, target_pred)
r2 = r2_score(target_test, target_pred)

print("Mean Squared Error:", np.round(mse, 2))
print("R^2 Score:", np.round(r2, 2))
```

```
Mean Squared Error: 0.0
R^2 Score: 0.64
```

In [6]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, cross
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.svm import SVR

# Load the dataset
dataframe = pd.read_csv('loan_data.csv')  # Update with your actual path

# Identify categorical columns
categorical_columns = dataframe.select_dtypes(include=['object']).columns

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode categorical features
for column in categorical_columns:
    dataframe[column] = label_encoder.fit_transform(dataframe[column])

# Separate features and target variable
features = dataframe.drop('int.rate', axis=1)  # Features
target = dataframe['int.rate']  # Target variable

# Scale the data
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_spl

# Define base regressors
regressors = {
    'Linear Regression': LinearRegression(),
```

```python
    'Decision Tree': DecisionTreeRegressor(),
    'SVR': SVR()
}

# Fit each regressor and get predictions
predictions_train = []
predictions_test = []

for name, reg in regressors.items():
    reg.fit(features_train, target_train)
    y_train_pred = reg.predict(features_train)
    y_test_pred = reg.predict(features_test)

    # Store predictions for averaging
    predictions_train.append(y_train_pred)
    predictions_test.append(y_test_pred)

    # Calculate and display metrics
    mse_train = mean_squared_error(target_train, y_train_pred)
    r2_train = r2_score(target_train, y_train_pred)
    mse_test = mean_squared_error(target_test, y_test_pred)
    r2_test = r2_score(target_test, y_test_pred)

    print(f"{name} - Train Set MSE: {np.round(mse_train, 2)}, R²: {np.rou
    print(f"{name} - Test Set MSE: {np.round(mse_test, 2)}, R²: {np.round

# Bagging Regressor with Decision Tree
bag_regressor = BaggingRegressor(estimator=DecisionTreeRegressor(), n_est
bag_regressor.fit(features_train, target_train)

# Predictions with Bagging Regressor
y_bag_pred_train = bag_regressor.predict(features_train)
y_bag_pred_test = bag_regressor.predict(features_test)

# Calculate metrics for Bagging Regressor
mse_bag_train = mean_squared_error(target_train, y_bag_pred_train)
r2_bag_train = r2_score(target_train, y_bag_pred_train)
mse_bag_test = mean_squared_error(target_test, y_bag_pred_test)
r2_bag_test = r2_score(target_test, y_bag_pred_test)

print("Bagging Regressor - Train Set MSE:", np.round(mse_bag_train, 2))
print("Bagging Regressor - Train Set R²:", np.round(r2_bag_train, 2))
print("Bagging Regressor - Test Set MSE:", np.round(mse_bag_test, 2))
print("Bagging Regressor - Test Set R²:", np.round(r2_bag_test, 2))

# Grid Search for Best Bagging Regressor
params = {
    'estimator': [LinearRegression(), DecisionTreeRegressor(), SVR()],
    'n_estimators': [50, 100],
    'max_samples': [0.5, 1.0],
    'max_features': [0.5, 1.0],
    'bootstrap': [True, False],
}

bagging_regressor_grid = GridSearchCV(BaggingRegressor(random_state=1), p
bagging_regressor_grid.fit(features_train, target_train)

print('Best R² Score Through Grid Search: %.3f' % bagging_regressor_grid.
print('Best Parameters: ', bagging_regressor_grid.best_params_)
```

```
Linear Regression - Train Set MSE: 0.0, R²: 0.65
Linear Regression - Test Set MSE: 0.0, R²: 0.64

Decision Tree - Train Set MSE: 0.0, R²: 1.0
Decision Tree - Test Set MSE: 0.0, R²: 0.48

SVR - Train Set MSE: 0.0, R²: -0.34
SVR - Test Set MSE: 0.0, R²: -0.32

Bagging Regressor - Train Set MSE: 0.0
Bagging Regressor - Train Set R²: 0.96
Bagging Regressor - Test Set MSE: 0.0
Bagging Regressor - Test Set R²: 0.74
Fitting 3 folds for each of 48 candidates, totalling 144 fits
Best R² Score Through Grid Search: 0.741
Best Parameters:  {'bootstrap': True, 'estimator': DecisionTreeRegressor(), 'max_features': 1.0, 'max_samples': 0.5, 'n_estimators': 100}
```

Bagging Classifier

In [15]:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

# Load the dataset
dataframe = pd.read_csv('loan_data.csv')  # Update with your actual path

# Identify categorical columns
categorical_columns = dataframe.select_dtypes(include=['object']).columns

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode categorical features
for column in categorical_columns:
    dataframe[column] = label_encoder.fit_transform(dataframe[column])

# Separate features and target variable
features = dataframe.drop('not.fully.paid', axis=1)  # Features
target = dataframe['not.fully.paid']  # Target variable

# Scale the data
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_spl

# Define base classifiers
classifiers = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
```

```python
    'Naive Bayes': GaussianNB(),
    'Random Forest': RandomForestClassifier()
}

# Fit each classifier and get predictions
predictions_train = []
predictions_test = []

for name, clf in classifiers.items():
    clf.fit(features_train, target_train)
    y_train_pred = clf.predict(features_train)
    y_test_pred = clf.predict(features_test)

    # Store predictions for potential averaging
    predictions_train.append(y_train_pred)
    predictions_test.append(y_test_pred)

    # Calculate and display metrics
    train_accuracy = accuracy_score(target_train, y_train_pred)
    test_accuracy = accuracy_score(target_test, y_test_pred)

    print(f"{name} - Train Accuracy: {np.round(train_accuracy, 2)}, Test 

# Bagging Classifier with Decision Tree
bag_classifier = BaggingClassifier(estimator=DecisionTreeClassifier(), n_
bag_classifier.fit(features_train, target_train)

# Predictions with Bagging Classifier
y_bag_pred_train = bag_classifier.predict(features_train)
y_bag_pred_test = bag_classifier.predict(features_test)

# Calculate accuracy for Bagging Classifier
train_accuracy_bag = accuracy_score(target_train, y_bag_pred_train)
test_accuracy_bag = accuracy_score(target_test, y_bag_pred_test)

print("Bagging Classifier - Train Accuracy:", np.round(train_accuracy_bag
print("Bagging Classifier - Test Accuracy:", np.round(test_accuracy_bag,

# # Grid Search for Best Bagging Classifier
# params = {
#     'estimator': [LogisticRegression(), DecisionTreeClassifier(), KNeig
#     'n_estimators': [50, 100],
#     'max_samples': [0.5, 1.0],
#     'max_features': [0.5, 1.0],
#     'bootstrap': [True, False],
# }

# bagging_classifier_grid = GridSearchCV(BaggingClassifier(random_state=1
# bagging_classifier_grid.fit(features_train, target_train)

# print('Best Accuracy Score Through Grid Search: %.3f' % bagging_classif
# print('Best Parameters: ', bagging_classifier_grid.best_params_)
```

```
Logistic Regression - Train Accuracy: 0.84, Test Accuracy: 0.84

Decision Tree - Train Accuracy: 1.0, Test Accuracy: 0.76

K-Nearest Neighbors - Train Accuracy: 0.85, Test Accuracy: 0.83

Naive Bayes - Train Accuracy: 0.79, Test Accuracy: 0.79

Random Forest - Train Accuracy: 1.0, Test Accuracy: 0.84

Bagging Classifier - Train Accuracy: 1.0
Bagging Classifier - Test Accuracy: 0.84
```

Comparison of Bagging Classifier and Random Forest Classifier

Bagging Classifier

- **Definition**: Stands for "Bootstrap Aggregating." Creates multiple copies of training data by s with replacement.
- **Purpose**: Reduces variance and improves accuracy by combining predictions from multiple
- **Model Type**: Can use any kind of model (e.g., decision trees, linear models).
- **Decision Making**: Final prediction is made by averaging (for regression) or voting (for class from all models.

Random Forest Classifier

- **Definition**: A specific type of Bagging that uses decision trees as base models and adds randomness in feature selection.
- **Purpose**: Makes trees less correlated, improving overall performance.
- **Model Type**: Primarily based on decision trees.
- **Decision Making**: Combines predictions from multiple trees through voting or averaging.

Key Differences

- **Model Variability**: Bagging can use various models, while Random Forest uses only decisi
- **Feature Selection**: Random Forest selects a random subset of features for each tree, addi randomness.
- **Complexity**: Random Forest generally performs better than basic Bagging, especially with datasets.

Summary: Bagging improves model performance by averaging predictions from multiple copies Random Forest is a specialized version that focuses on decision trees with added randomness f results.

In [5]:
```python
import numpy as np
import pandas as pd
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
```

```python
# Load the dataset
dataframe = pd.read_csv('loan_data.csv')  # Update with your actual path

# Identify categorical columns
categorical_columns = dataframe.select_dtypes(include=['object']).columns

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode categorical features
for column in categorical_columns:
    dataframe[column] = label_encoder.fit_transform(dataframe[column])

# Separate features and target variable
features = dataframe.drop('not.fully.paid', axis=1)  # Features
target = dataframe['not.fully.paid']  # Target variable

# Scale the data
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_spl

# Bagging Classifier with Decision Tree
bag_classifier = BaggingClassifier(estimator=DecisionTreeClassifier(), n_
bag_classifier.fit(features_train, target_train)

# Predictions with Bagging Classifier
y_bag_pred_train = bag_classifier.predict(features_train)
y_bag_pred_test = bag_classifier.predict(features_test)

# Calculate accuracy for Bagging Classifier
train_accuracy_bag = accuracy_score(target_train, y_bag_pred_train)
test_accuracy_bag = accuracy_score(target_test, y_bag_pred_test)

print("Bagging Classifier - Train Accuracy:", np.round(train_accuracy_bag
print("Bagging Classifier - Test Accuracy:", np.round(test_accuracy_bag,

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=1)
rf_classifier.fit(features_train, target_train)

# Predictions with Random Forest Classifier
y_rf_pred_train = rf_classifier.predict(features_train)
y_rf_pred_test = rf_classifier.predict(features_test)

# Calculate accuracy for Random Forest Classifier
train_accuracy_rf = accuracy_score(target_train, y_rf_pred_train)
test_accuracy_rf = accuracy_score(target_test, y_rf_pred_test)

print("Random Forest Classifier - Train Accuracy:", np.round(train_accura
print("Random Forest Classifier - Test Accuracy:", np.round(test_accuracy

# Comparison of accuracy
print(f"\nComparison of Classifiers:")
print(f"Bagging Classifier Test Accuracy: {np.round(test_accuracy_bag, 2)
print(f"Random Forest Classifier Test Accuracy: {np.round(test_accuracy_r
```

```
Bagging Classifier - Train Accuracy: 1.0
Bagging Classifier - Test Accuracy: 0.84
Random Forest Classifier - Train Accuracy: 1.0
Random Forest Classifier - Test Accuracy: 0.84

Comparison of Classifiers:
Bagging Classifier Test Accuracy: 0.84
Random Forest Classifier Test Accuracy: 0.84
```

heart data set randomforest vs bagging

In [9]:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier

# Load the dataset
dataframe = pd.read_csv('heart.csv')  # Update with your actual path

# Summary of the dataset
print(dataframe.info())

# Identify categorical columns
categorical_columns = dataframe.select_dtypes(include=['object']).columns

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode categorical features
for column in categorical_columns:
    dataframe[column] = label_encoder.fit_transform(dataframe[column])

# Separate features and target variable
features = dataframe.drop('target', axis=1)  # Features
target = dataframe['target']  # Target variable

# Scale the data
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_spl

# Bagging Classifier with Decision Tree
bag_classifier = BaggingClassifier(estimator=DecisionTreeClassifier(), n_
bag_classifier.fit(features_train, target_train)

# Predictions with Bagging Classifier
y_bag_pred_train = bag_classifier.predict(features_train)
y_bag_pred_test = bag_classifier.predict(features_test)

# Calculate accuracy for Bagging Classifier
train_accuracy_bag = accuracy_score(target_train, y_bag_pred_train)
test_accuracy_bag = accuracy_score(target_test, y_bag_pred_test)

print("Bagging Classifier - Train Accuracy:", np.round(train_accuracy_bag
print("Bagging Classifier - Test Accuracy:", np.round(test_accuracy_bag,
```

```python
# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=1)
rf_classifier.fit(features_train, target_train)

# Predictions with Random Forest Classifier
y_rf_pred_train = rf_classifier.predict(features_train)
y_rf_pred_test = rf_classifier.predict(features_test)

# Calculate accuracy for Random Forest Classifier
train_accuracy_rf = accuracy_score(target_train, y_rf_pred_train)
test_accuracy_rf = accuracy_score(target_test, y_rf_pred_test)

print("Random Forest Classifier - Train Accuracy:", np.round(train_accura
print("Random Forest Classifier - Test Accuracy:", np.round(test_accuracy

# Comparison of accuracy
print(f"\nComparison of Classifiers:")
print(f"Bagging Classifier Test Accuracy: {np.round(test_accuracy_bag, 2)
print(f"Random Forest Classifier Test Accuracy: {np.round(test_accuracy_r
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
None
Bagging Classifier - Train Accuracy: 1.0
Bagging Classifier - Test Accuracy: 0.84
Random Forest Classifier - Train Accuracy: 1.0
Random Forest Classifier - Test Accuracy: 0.84

Comparison of Classifiers:
Bagging Classifier Test Accuracy: 0.84
Random Forest Classifier Test Accuracy: 0.84
```

Boosting Classifier

In [20]:

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifi
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```python
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier  # Correct import for XGBoost

# Load the dataset
dataframe = pd.read_csv('loan_data.csv')  # Update with your actual path

# Identify categorical columns
categorical_columns = dataframe.select_dtypes(include=['object']).columns

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode categorical features
for column in categorical_columns:
   dataframe[column] = label_encoder.fit_transform(dataframe[column])

# Separate features and target variable
features = dataframe.drop('not.fully.paid', axis=1)  # Features
target = dataframe['not.fully.paid']  # Target variable

# Scale the data
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
5
# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_spl

# AdaBoost Classifier
ada_classifier = AdaBoostClassifier(estimator=DecisionTreeClassifier(), n
ada_classifier.fit(features_train, target_train)

# Predictions with AdaBoost Classifier
y_ada_pred_train = ada_classifier.predict(features_train)
y_ada_pred_test = ada_classifier.predict(features_test)

# Calculate accuracy for AdaBoost Classifier
train_accuracy_ada = accuracy_score(target_train, y_ada_pred_train)
test_accuracy_ada = accuracy_score(target_test, y_ada_pred_test)

print("AdaBoost Classifier - Train Accuracy:", np.round(train_accuracy_ad
print("AdaBoost Classifier - Test Accuracy:", np.round(test_accuracy_ada,

# Gradient Boosting Classifier
gb_classifier = GradientBoostingClassifier(n_estimators=100, random_state
gb_classifier.fit(features_train, target_train)

# Predictions with Gradient Boosting Classifier
y_gb_pred_train = gb_classifier.predict(features_train)
y_gb_pred_test = gb_classifier.predict(features_test)

# Calculate accuracy for Gradient Boosting Classifier
train_accuracy_gb = accuracy_score(target_train, y_gb_pred_train)
test_accuracy_gb = accuracy_score(target_test, y_gb_pred_test)

print("Gradient Boosting Classifier - Train Accuracy:", np.round(train_ac
print("Gradient Boosting Classifier - Test Accuracy:", np.round(test_accu

# Histogram-based Gradient Boosting Classifier
hist_gb_classifier = HistGradientBoostingClassifier(max_iter=100, random_
hist_gb_classifier.fit(features_train, target_train)
```

```
# Predictions with Histogram-based Gradient Boosting Classifier
y_hist_gb_pred_train = hist_gb_classifier.predict(features_train)
y_hist_gb_pred_test = hist_gb_classifier.predict(features_test)

# Calculate accuracy for Histogram-based Gradient Boosting Classifier
train_accuracy_hist_gb = accuracy_score(target_train, y_hist_gb_pred_trai
test_accuracy_hist_gb = accuracy_score(target_test, y_hist_gb_pred_test)

print("Histogram-based Gradient Boosting Classifier - Train Accuracy:", n
print("Histogram-based Gradient Boosting Classifier - Test Accuracy:", np

# XGBoost Classifier
xgb_classifier = XGBClassifier(n_estimators=100, random_state=1)
xgb_classifier.fit(features_train, target_train)

# Predictions with XGBoost Classifier
y_xgb_pred_train = xgb_classifier.predict(features_train)
y_xgb_pred_test = xgb_classifier.predict(features_test)

# Calculate accuracy for XGBoost Classifier
train_accuracy_xgb = accuracy_score(target_train, y_xgb_pred_train)
test_accuracy_xgb = accuracy_score(target_test, y_xgb_pred_test)

print("XGBoost Classifier - Train Accuracy:", np.round(train_accuracy_xgb
print("XGBoost Classifier - Test Accuracy:", np.round(test_accuracy_xgb,

# Comparison of accuracy
print(f"\nComparison of Classifiers:")
print(f"AdaBoost Classifier Test Accuracy: {np.round(test_accuracy_ada, 2
print(f"Gradient Boosting Classifier Test Accuracy: {np.round(test_accura
print(f"Histogram-based Gradient Boosting Classifier Test Accuracy: {np.r
print(f"XGBoost Classifier Test Accuracy: {np.round(test_accuracy_xgb, 2)
```

```
C:\Users\Hp\miniconda3\envs\machine_learning\lib\site-
packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMM
algorithm to circumvent this warning.
  warnings.warn(
AdaBoost Classifier - Train Accuracy: 1.0
AdaBoost Classifier - Test Accuracy: 0.75
Gradient Boosting Classifier - Train Accuracy: 0.85
Gradient Boosting Classifier - Test Accuracy: 0.84
Histogram-based Gradient Boosting Classifier - Train Accuracy: 0.9
Histogram-based Gradient Boosting Classifier - Test Accuracy: 0.84
XGBoost Classifier - Train Accuracy: 0.96
XGBoost Classifier - Test Accuracy: 0.83

Comparison of Classifiers:
AdaBoost Classifier Test Accuracy: 0.75
Gradient Boosting Classifier Test Accuracy: 0.84
Histogram-based Gradient Boosting Classifier Test Accuracy: 0.84
XGBoost Classifier Test Accuracy: 0.83
```

In [19]:
```
dataframe.head()
```

Out[19]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.w
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	56:
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	27(
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	47:
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	26!
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	40(

Boosting Regressor

In [27]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import AdaBoostRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
import lightgbm as lgb
from catboost import CatBoostRegressor

# Load the dataset
dataframe = pd.read_csv('loan_data.csv')  # Update with your actual path

# Identify categorical columns
categorical_columns = dataframe.select_dtypes(include=['object']).columns

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode categorical features
for column in categorical_columns:
    dataframe[column] = label_encoder.fit_transform(dataframe[column])

# Separate features and target variable
features = dataframe.drop(['log.annual.inc'], axis=1)  # Features
target = dataframe['log.annual.inc']  # Target variable (log-transformed

# Scale the data
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_spl

# Initialize the models
models = {
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=100, rand
    "XGBoost": XGBRegressor(n_estimators=100, random_state=1),
    "LightGBM": lgb.LGBMRegressor(n_estimators=100, random_state=1),
    "CatBoost": CatBoostRegressor(n_estimators=100, random_state=1, verbo
    "AdaBoost": AdaBoostRegressor(n_estimators=100, random_state=1),
    "Histogram-based Gradient Boosting": HistGradientBoostingRegressor(ma
}

# Train and evaluate each model
for model_name, model in models.items():
```

```
    model.fit(features_train, target_train)
    y_train_pred = model.predict(features_train)
    y_test_pred = model.predict(features_test)

    # Calculate Mean Squared Error
    train_mse = mean_squared_error(target_train, y_train_pred)
    test_mse = mean_squared_error(target_test, y_test_pred)

    print(f"{model_name} - Train MSE: {np.round(train_mse, 2)}")
    print(f"{model_name} - Test MSE: {np.round(test_mse, 2)}\n")

# Optional: Reverse the log transformation to get actual income predictio
actual_income_train = np.exp(y_train_pred)
actual_income_test = np.exp(y_test_pred)

print("Sample actual income predictions (Train):", actual_income_train[:5
print("Sample actual income predictions (Test):", actual_income_test[:5])
```

```
Gradient Boosting - Train MSE: 0.18
Gradient Boosting - Test MSE: 0.2

XGBoost - Train MSE: 0.06
XGBoost - Test MSE: 0.23

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
was 0.002362 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1595
[LightGBM] [Info] Number of data points in the train set: 7662, number of use
features: 13
[LightGBM] [Info] Start training from score 10.934129
LightGBM - Train MSE: 0.13
LightGBM - Test MSE: 0.2

CatBoost - Train MSE: 0.13
CatBoost - Test MSE: 0.21

AdaBoost - Train MSE: 0.25
AdaBoost - Test MSE: 0.27

Histogram-based Gradient Boosting - Train MSE: 0.12
Histogram-based Gradient Boosting - Test MSE: 0.2

Sample actual income predictions (Train): [ 46418.42390139  49340.22199464
53346.07728535 104318.40494611
  48289.29526851]
Sample actual income predictions (Test): [50801.79070877 61794.28943989
40188.18258602 41352.70111008
 47600.41188874]
```

To Apply the OneHot Encoding

In [25]:
```
import pandas as pd

# Load the dataset
dataframe = pd.read_csv('loan_data.csv')  # Update with your actual path

# Display the initial summary
print(dataframe.info())
```

```python
# One-hot encoding for the 'purpose' column
dataframe = pd.get_dummies(dataframe, columns=['purpose'], drop_first=Tru

# Display the updated dataframe summary
print(dataframe.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   credit.policy      9578 non-null   int64  
 1   purpose            9578 non-null   object 
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64  
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64  
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64  
 11  delinq.2yrs        9578 non-null   int64  
 12  pub.rec            9578 non-null   int64  
 13  not.fully.paid     9578 non-null   int64  
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 19 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   credit.policy               9578 non-null   int64  
 1   int.rate                    9578 non-null   float64
 2   installment                 9578 non-null   float64
 3   log.annual.inc              9578 non-null   float64
 4   dti                         9578 non-null   float64
 5   fico                        9578 non-null   int64  
 6   days.with.cr.line           9578 non-null   float64
 7   revol.bal                   9578 non-null   int64  
 8   revol.util                  9578 non-null   float64
 9   inq.last.6mths              9578 non-null   int64  
 10  delinq.2yrs                 9578 non-null   int64  
 11  pub.rec                     9578 non-null   int64  
 12  not.fully.paid              9578 non-null   int64  
 13  purpose_credit_card         9578 non-null   bool   
 14  purpose_debt_consolidation  9578 non-null   bool   
 15  purpose_educational         9578 non-null   bool   
 16  purpose_home_improvement    9578 non-null   bool   
 17  purpose_major_purchase      9578 non-null   bool   
 18  purpose_small_business      9578 non-null   bool   
dtypes: bool(6), float64(6), int64(7)
memory usage: 1.0 MB
None
```

In [26]:
```python
dataframe.head()
```

Out[26]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.ba
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740

In [ ]: