SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE NAME: Machine Learning Laboratory **COURSE CODE:** DJS22L602
CLASS: Third Year B. Tech **SEM:** VI
Name :Anish Sharma

EXPERIMENT NO. 6

CO Measured:

CO3 Apply various machine learning techniques

TITLE: To implement ensemble methods by performing comparative analysis on bagging and boosting techniques used for prediction.

AIM / OBJECTIVE:
Perform ensemble methods using python libraries for the following methods over the suitable selected dataset and compare results.

- Simple Ensemble Techniques
  ○ Max Voting ○ Averaging
- Advanced Ensemble techniques ○ Stacking ○ Blending ○ Bagging
  ○ Boosting

DESCRIPTION OF EXPERIMENT:

Explain the Basic ensemble methods?
1. Averaging method: It is mainly used for regression problems. The method consists of building multiple models independently and returning the average of the prediction of all the models. In general, the combined output is better than an individual output because variance is reduced.

2. Max voting: It is mainly used for classification problems. The method consists of building multiple models independently and getting their individual output called 'vote'. The class with maximum votes is returned as output.

Explain the Advanced ensemble methods?

1. Stacking: It is an ensemble method that combines multiple models (classification or regression) via meta-model (meta-classifier or meta-regression). The base models are trained on the complete dataset, then the meta-model is trained on features returned (as output) from base models. The base models in stacking are typically different. The meta-model helps to find the features from base models to achieve the best accuracy.

2. Blending: It is similar to the stacking method explained above, but rather than using the whole dataset for training the base-models, a validation dataset is kept separate to make predictions.

3. Bagging: It is also known as a bootstrapping method. Base models are run on bags to get a fair distribution of the whole dataset. A bag is a subset of the dataset along with a replacement to make the size of the bag the same as the whole dataset. The final output is formed after combining the output of all base models.

4. Boosting: Boosting is a sequential method–it aims to prevent a wrong base model from affecting the final output. Instead of combining the base models, the method focuses on building a new model that is dependent on the previous one. A new model tries to remove the errors made by its previous one. Each of these models is called weak learners. The final model (aka strong learner) is formed by getting the weighted mean of all the weak learners.

PROCEDURE:

Code :

```
import pandas as pd


heart_data = pd.read_csv('heart.csv')  heart_data.head()
```

1 to 5 of 5 entries Filter

index	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

Steps to perform Stacking:

1. Split the train dataset into n parts
2. A base model (say linear regression) is fitted on n-1 parts and predictions are made for the nth part. This is done for each one of the n part of the train set.

3. The base model is then fitted on the whole train dataset.
4. This model is used to predict the test dataset.
5. The Steps 2 to 4 are repeated for another base model which results in another set of predictions for the train and test dataset.
6. The predictions on train data set are used as a feature to build the new model.
7. This final model is used to make the predictions on test dataset

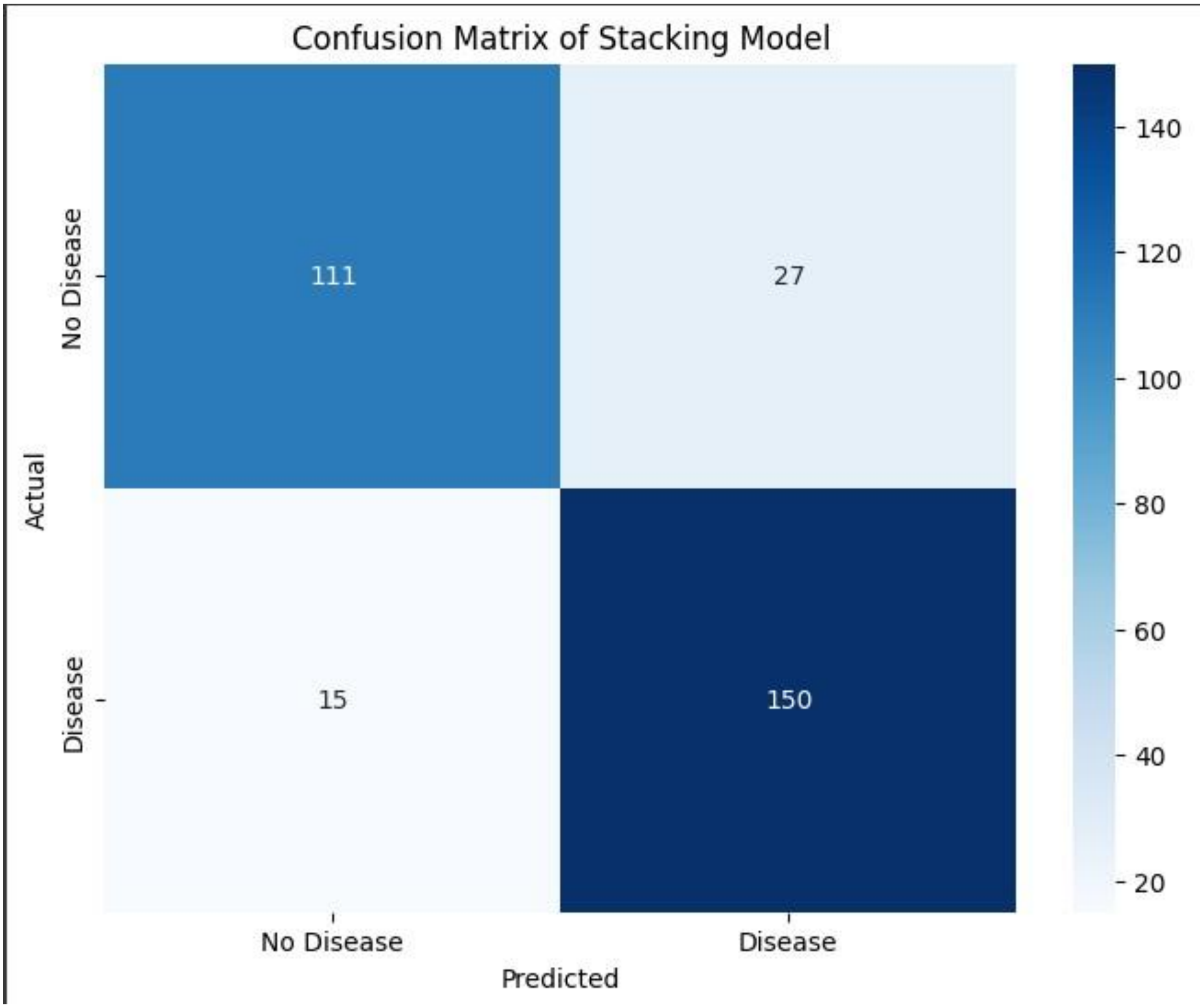
```
!pip install scikit-learn matplotlib seaborn
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
heart_data = pd.read_csv('heart.csv')

# Prepare data
X = heart_data.drop('target', axis=1)
y = heart_data['target']

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

# Split data into training and testing
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Base models
base_model_1 = LogisticRegression(max_iter=1000)
base_model_2 = RandomForestClassifier()

# Meta-model (stacking model)
meta_model = LogisticRegression(max_iter=1000)

# Store base model predictions and targets
train_predictions = []
```

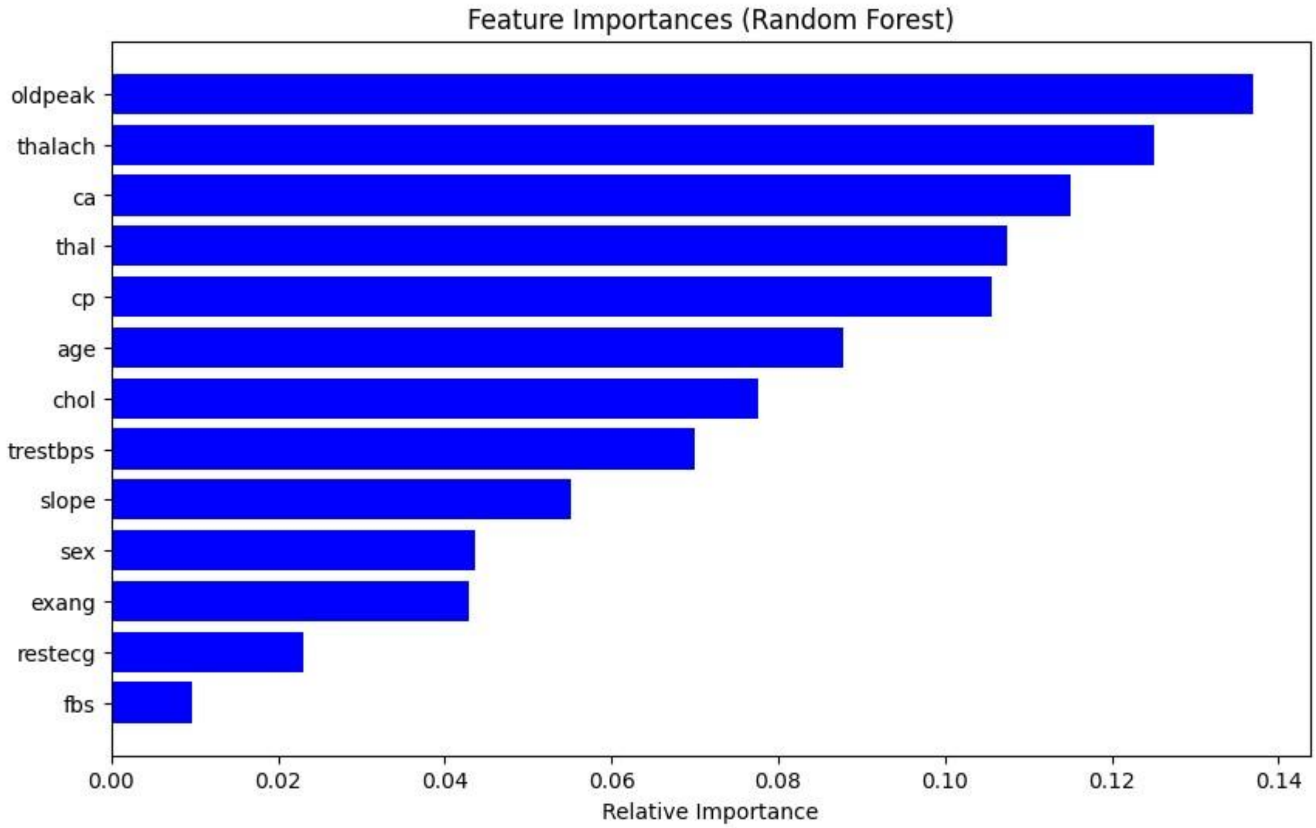
```
    train_targets = []

# Perform cross-validation for stacking
for train_index, test_index in kf.split(X_scaled):
X_train, X_test = X_scaled[train_index], X_scaled[test_index]
y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        base_model_1.fit(X_train, y_train)
        base_model_2.fit(X_train, y_train)
```

```

        base_pred_1 = base_model_1.predict(X_test)
        base_pred_2 = base_model_2.predict(X_test)

        train_predictions.extend(np.column_stack([base_pred_1,
        base_pred_2]))
        train_targets.extend(y_test)

# Combine predictions for training the meta-model
train_meta_features = np.array(train_predictions).reshape(-1, 2)
meta_model.fit(train_meta_features, train_targets)

   # Get predictions for the entire dataset
   base_pred_1_final = base_model_1.predict(X_scaled)
   base_pred_2_final = base_model_2.predict(X_scaled)
   test_meta_features = np.column_stack([base_pred_1_final,
   base_pred_2_final])
   final_predictions = meta_model.predict(test_meta_features)

   # Evaluate
   stacking_accuracy = accuracy_score(y, final_predictions)
   print(f"Stacking Model Accuracy: {stacking_accuracy}")

               # Visualization 1: Confusion Matrix
               cm = confusion_matrix(y, final_predictions)
               plt.figure(figsize=(8, 6))
               sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
               xticklabels=['No Disease', 'Disease'],
               yticklabels=['No Disease', 'Disease'])
               plt.title('Confusion Matrix of Stacking Model')
               plt.xlabel('Predicted')
               plt.ylabel('Actual')
               plt.show()

```

```
        # Visualization 2: Feature Importance of Base Model 2 (Random Forest)
        if hasattr(base_model_2, 'feature_importances_'):
            importances = base_model_2.feature_importances_
            features = X.columns
            indices = np.argsort(importances)

            plt.figure(figsize=(10, 6))
            plt.title('Feature Importances (Random Forest)')
```

```
        plt.barh(range(len(indices)), importances[indices], color='b', align='center')
        plt.yticks(range(len(indices)), [features[i] for i in indices])
        plt.xlabel('Relative Importance')
        plt.show()
    else:
        print("Base model 2 does not have feature_importances_ attribute.")
```


Confusion Matrix of Stacking Model
No Disease
Disease
Actual
111
27
15
150
No Disease
Disease
Predicted
140
120
100
80
60
40
20



Steps to perform Blending:

1. Split the training dataset into train, test and validation dataset.
2. Fit all the base models using train dataset.
3. Make predictions on validation and test dataset.
4. These predictions are used as features to build a second level model
5. This model is used to make predictions on test and meta-features

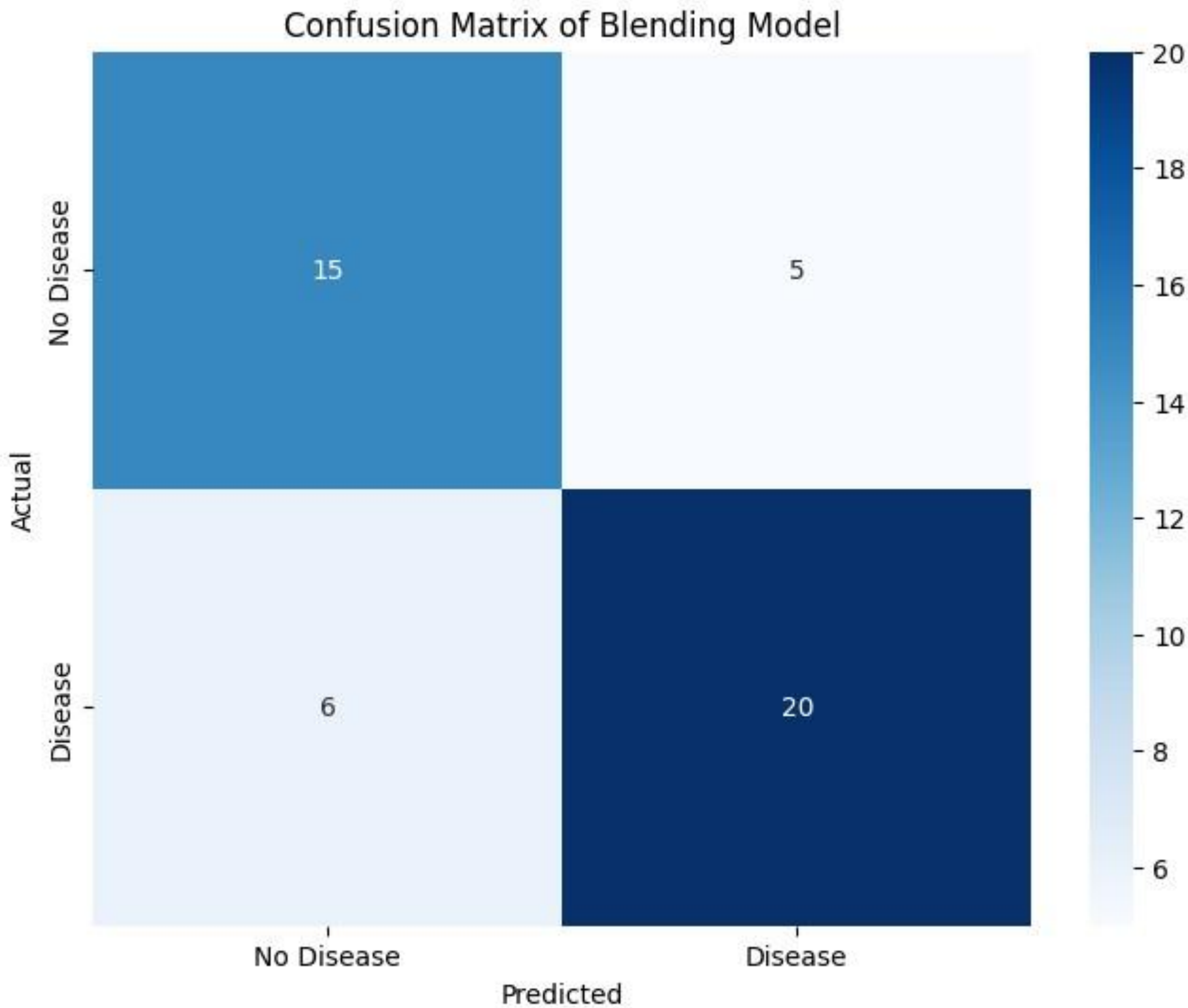
```
!pip install scikit-learn matplotlib seaborn --upgrade
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Load and split the dataset
```

```
heart_data = pd.read_csv('heart.csv')  # Assuming 'heart.csv' is your dataset
X = heart_data.drop('target', axis=1)
y = heart_data['target']
```

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

    # 2. Fit base models
    base_model_1 = LogisticRegression(max_iter=1000)
    base_model_2 = RandomForestClassifier()
    base_model_1.fit(X_train, y_train)
    base_model_2.fit(X_train, y_train)

    # 3. Make predictions
    val_pred_1 = base_model_1.predict(X_val)
    val_pred_2 = base_model_2.predict(X_val)
    test_pred_1 = base_model_1.predict(X_test)
    test_pred_2 = base_model_2.predict(X_test)

    # 4. Create meta-features
    val_meta_features = pd.DataFrame({'pred_1': val_pred_1, 'pred_2': val_pred_2})
    test_meta_features = pd.DataFrame({'pred_1': test_pred_1, 'pred_2': test_pred_2})

    # 5. Build and train meta-model (blender)
    meta_model = LogisticRegression(max_iter=1000)
    meta_model.fit(val_meta_features, y_val)

# 6. Make final predictions and evaluate
final_predictions = meta_model.predict(test_meta_features)
blending_accuracy = accuracy_score(y_test, final_predictions)
print(f"Blending Model Accuracy: {blending_accuracy}")

    # Visualizations
    from sklearn.metrics import confusion_matrix

```

```
        # Confusion Matrix
        cm = confusion_matrix(y_test, final_predictions)
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
        xticklabels=['No Disease', 'Disease'],
        yticklabels=['No Disease', 'Disease'])
        plt.title('Confusion Matrix of Blending Model')
        plt.xlabel('Predicted')
```

```
    plt.ylabel('Actual')
    plt.show()
```



Steps to perform Bagging:

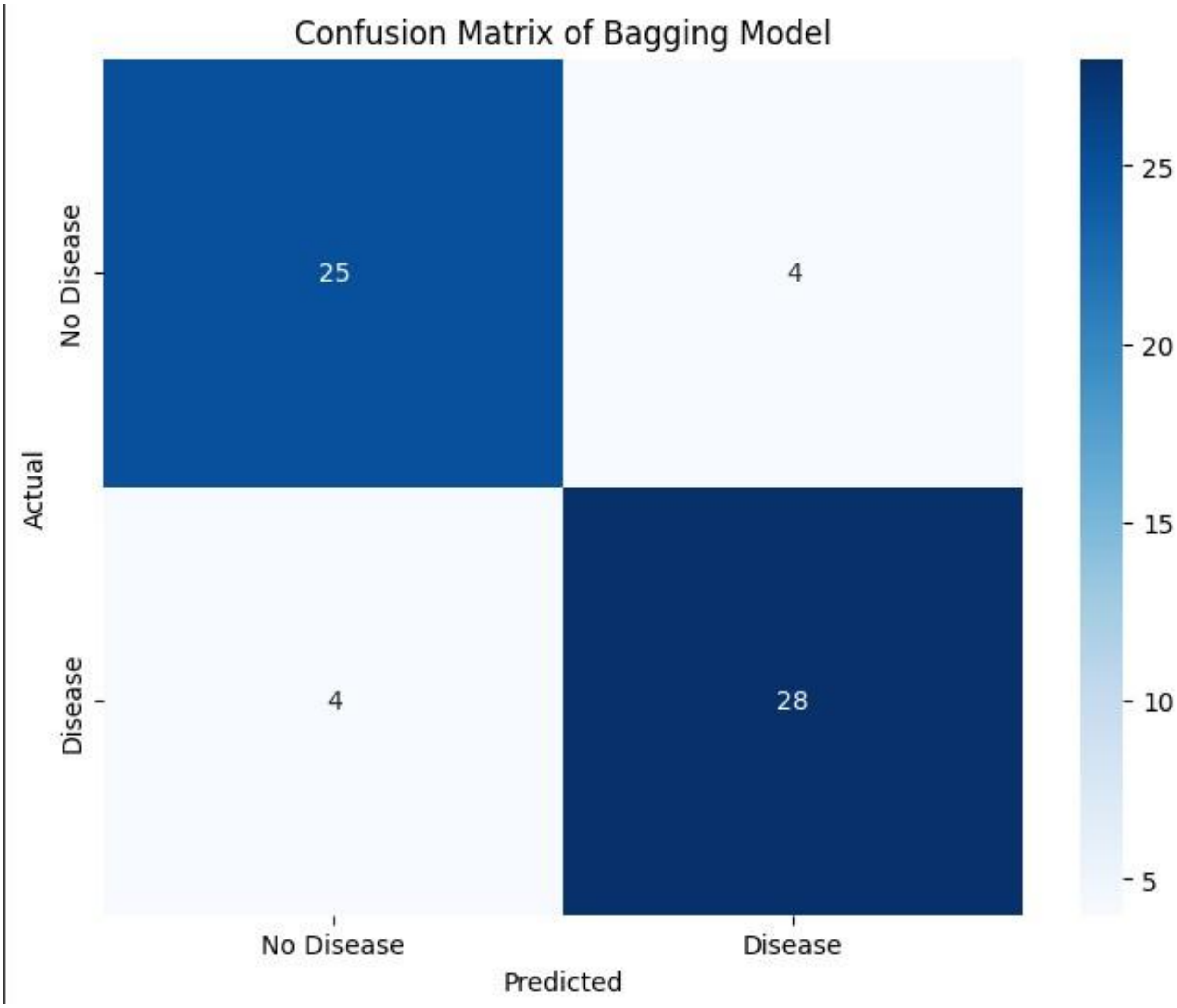
1. Create multiple datasets from the train dataset by selecting observations with replacements
2. Run a base model on each of the created datasets independently
3. Combine the predictions of all the base models to each the final output

```
!pip install scikit-learn matplotlib seaborn --upgrade
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import BaggingClassifier,
RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```

    # Load the dataset
    heart_data = pd.read_csv('heart.csv')

    # Prepare data
    X = heart_data.drop('target', axis=1)
    y = heart_data['target']

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

    # Base model (Random Forest)
    base_model = RandomForestClassifier()

    # Bagging Classifier (using 'estimator')
    bagging_model = BaggingClassifier(estimator=base_model,
    n_estimators=10, random_state=42)

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

    # Make predictions on test set
    predictions = bagging_model.predict(X_test)

    # Evaluate
    bagging_accuracy = accuracy_score(y_test, predictions)
    print(f"Bagging Model Accuracy: {bagging_accuracy}")

            # Visualization: Confusion Matrix
            cm = confusion_matrix(y_test, predictions)
            plt.figure(figsize=(8, 6))
            sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['No Disease', 'Disease'],
```

```
                    yticklabels=['No Disease', 'Disease'])
                    plt.title('Confusion Matrix of Bagging Model')
                    plt.xlabel('Predicted')
                    plt.ylabel('Actual')
                    plt.show()
```



Steps to perform Boosting:

1. Take a subset of the train dataset.
2. Train a base model on that dataset.
3. Use third model to make predictions on the whole dataset.
4. Calculate errors using the predicted values and actual values.
5. Initialize all data points with same weight.
6. Assign higher weight to incorrectly predicted data points.
7. Make another model, make predictions using the new model in such a way that errors made by the previous model are mitigated/corrected.
8. Similarly, create multiple models–each successive model correcting the errors of the previous model.

9. The final model (strong learner) is the weighted mean of all the previous models (weak learners)

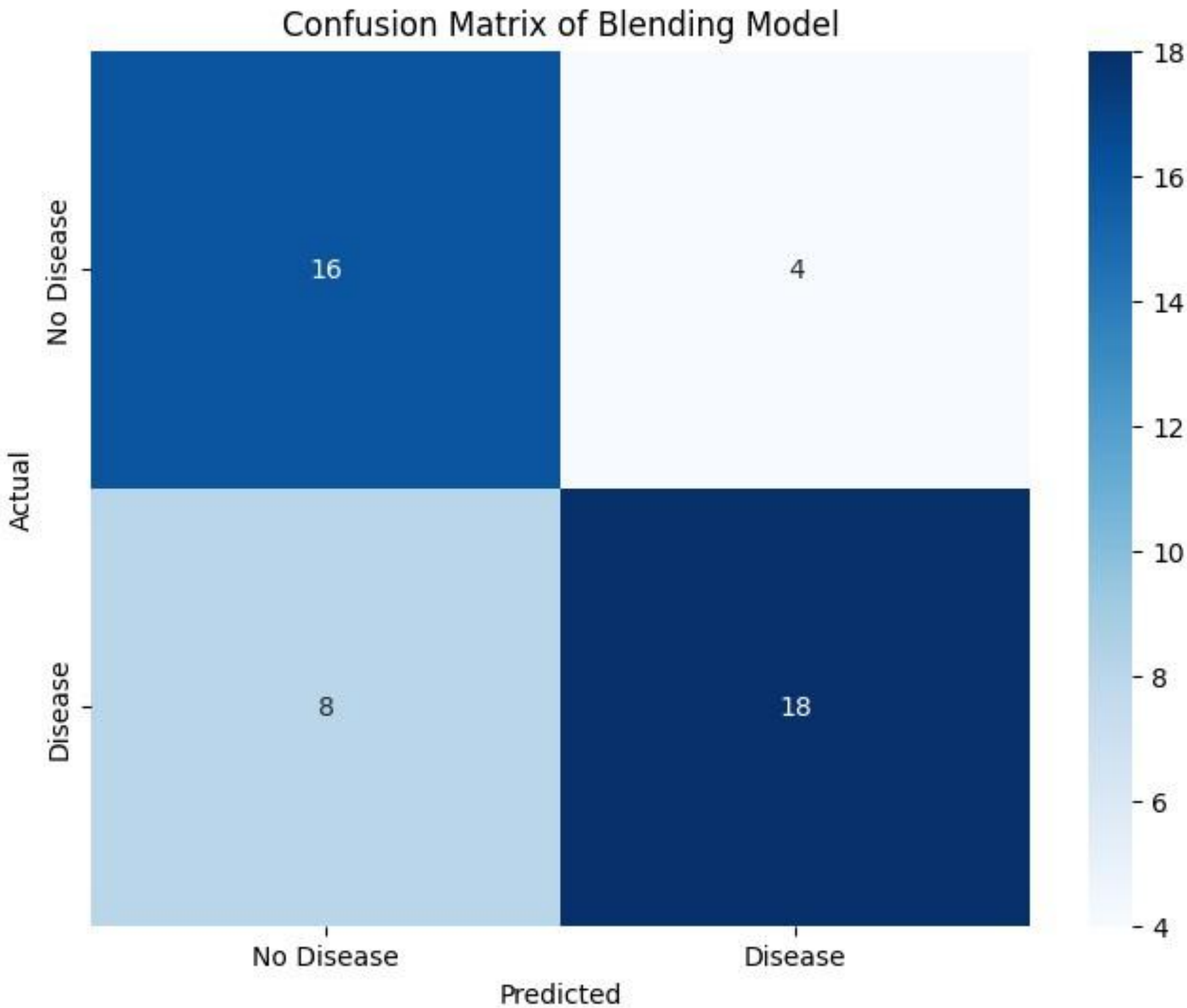
```
10.     !pip install scikit-learn matplotlib seaborn
11.     import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
heart_data = pd.read_csv('heart.csv')

# Prepare data
X = heart_data.drop('target', axis=1)
y = heart_data['target']

# Split data into train, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Base models
base_model_1 = LogisticRegression(max_iter=1000)
base_model_2 = RandomForestClassifier()

# Train base models on the training set
base_model_1.fit(X_train, y_train)
base_model_2.fit(X_train, y_train)

# Make predictions on validation and test sets
val_pred_1 = base_model_1.predict(X_val)
val_pred_2 = base_model_2.predict(X_val)
test_pred_1 = base_model_1.predict(X_test)
test_pred_2 = base_model_2.predict(X_test)
```

```
    # Create meta-features for validation and test sets
    val_meta_features = pd.DataFrame({'pred_1': val_pred_1, 'pred_2': val_pred_2})
    test_meta_features = pd.DataFrame({'pred_1': test_pred_1, 'pred_2': test_pred_2})

    # Meta-model (blender)
    meta_model = LogisticRegression(max_iter=1000)

```

```
    # Train meta-model on validation set
    meta_model.fit(val_meta_features, y_val)

    # Make final predictions on test set
    final_predictions = meta_model.predict(test_meta_features)

    # Evaluate
    blending_accuracy = accuracy_score(y_test, final_predictions)
    print(f"Blending Model Accuracy: {blending_accuracy}")

    # Visualization: Confusion Matrix
    cm = confusion_matrix(y_test, final_predictions)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=['No Disease', 'Disease'],
                yticklabels=['No Disease', 'Disease'])
    plt.title('Confusion Matrix of Blending Model')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

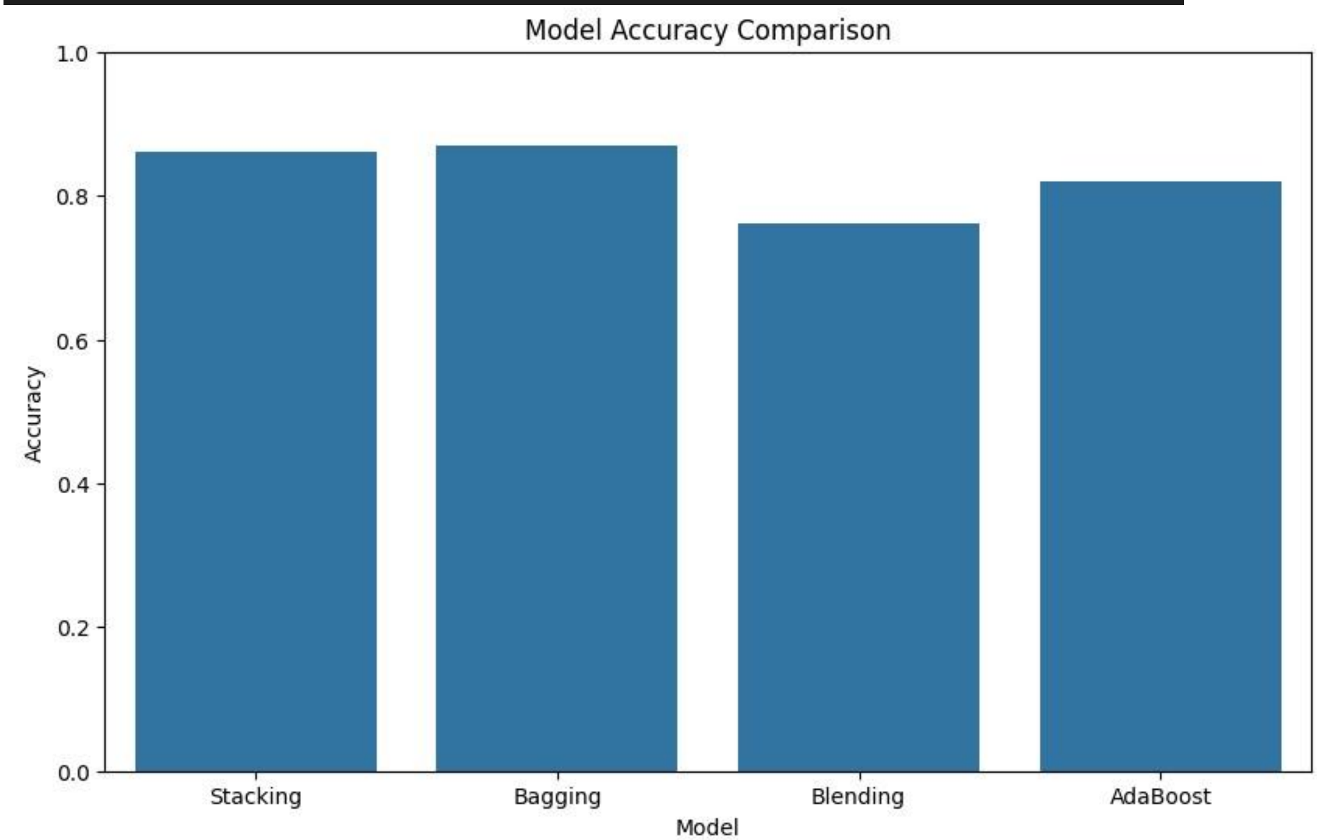



```
import pandas as pd
 data = {'Model': ['Stacking', 'Bagging', 'Blending',
'AdaBoost'],
        'Accuracy': [stacking_accuracy, bagging_accuracy, blending_accuracy,
accuracy]}

comparison_table = pd.DataFrame(data) print(comparison_table)
```

```
      Model  Accuracy
0  Stacking  0.861386
1   Bagging  0.868852
2  Blending  0.760870
3  AdaBoost  0.819672
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
 plt.figure(figsize=(10, 6)) sns.barplot(x='Model', y='Accuracy',
data=comparison_table) plt.title('Model Accuracy Comparison')
plt.xlabel('Model') plt.ylabel('Accuracy') plt.ylim(0, 1)  # Set y-
axis limits for better visualization plt.show()
```



Dataset:

1. Dataset from the UCI repository: Alcohol QCM Sensor Donated on 7/21/2019
   https://archive.ics.uci.edu/dataset/496/alcohol+qcm+sensor+dataset
2. Dataset from Kaggle: Cardiac features of patients from the "heart.csv" dataset:
   https://www.kaggle.com/datasets/arezaei81/heartcsv

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

OBSERVATIONS / DISCUSSION OF RESULT:

1. Compare the results of Basic ensemble methods and the Advanced ensemble methods?

CONCLUSION:

Based on the results, discuss the conclusions; describe the meaning of the experiment and the implications of your results.

REFERENCES:

(List the references as per format given below and citations to be included the document)

1. Ethem Alpaydın, “Introduction to Machine Learning”, 4th Edition, The MIT Press, 2020.
2. Peter Harrington, “Machine Learning in Action”, 1st Edition, Dreamtech Press, 2012.
3. Tom Mitchell, “Machine Learning”, 1st Edition, McGraw Hill, 2017.
4. Andreas C, Müller and Sarah Guido, “Introduction to Machine Learning with Python: A Guide for Data Scientists”, 1st Edition, O'reilly, 2016. 5. Kevin P. Murphy, “Machine Learning: A Probabilistic Perspective”, 1st Edition, MIT Press, 2012.

Website References:

Ensemble Learning Techniques Tutorial. Available Online: Ensemble Learning Techniques Tutorial, https://www.kaggle.com/code/pavansanagapati/ensemble-learning-techniques-tutorial
Ensembles: Gradient boosting, random forests, bagging, voting, stacking. Available Online: https://scikit-learn.org/stable/modules/ensemble.html