In [5]:
```
import pandas as pd

df = pd.read_csv('takheer_data_cleaned.csv')
```

In [6]:
```
columns = df.columns.tolist()
columns.remove("smoking")
features_x = df[columns]
class_y = df["smoking"]
```

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

xtrain, xtest, ytrain, ytest = train_test_split(features_x, class_y, test_size=0.30, shuffle=False, tra
in_size=0.70)
```

In [8]:
```
import numpy as np
from sklearn.tree import DecisionTreeClassifier

class BaggingClassifier:
    def __init__(self, n_estimators=100, max_depth=None, threshold=0.5):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.threshold = threshold
        self.models = [DecisionTreeClassifier(max_depth=self.max_depth) for _ in range(n_estimators)]


    def fit(self, X, y):
        for model in self.models:
            indices = np.random.choice(len(x), len(x), replace=True)
            x_subset, y_subset = x.iloc[indices], y.iloc[indices]
            model.fit(x_subset, y_subset)

        return self


    def predict(self, x):
        pred = np.zeros((len(x), self.n_estimators))
        for i, model in enumerate(self.models):
            pred[:, i] = model.predict(x)
        avg_predictions = np.mean(pred, axis=1)
        binary_predictions = (avg_predictions >= self.threshold).astype(int)
        return binary_predictions

    def get_params(self, deep=True):
        return {
            'n_estimators': self.n_estimators,
            'max_depth': self.max_depth,
            'threshold': self.threshold,
        }


    def set_params(self, **params):
        if not params:
            return self

        for param, value in params.items():
            setattr(self, param, value)

        return self
```

In [9]:
```
class AdaBoost:
    def __init__(self, n_estimators=50, max_depth=4):
        self.n_estimators = n_estimators
        self.alphas = []
        self.models = []
        self.max_depth = max_depth

    def fit(self, x, y):
        x = np.array(x)
        y = np.array(y)
        # Get the number of rows and columns
        n_samples, n_features = x.shape
        # Initialize the weights to 1/N
        weights = np.ones(n_samples) / n_samples

        for _ in range(self.n_estimators):
            model = DecisionTreeClassifier(max_depth=self.max_depth)
            model.fit(x, y, sample_weight=weights)
            y_pred = model.predict(x)
            weighted_error = np.sum(weights[y_pred != y])
            if weighted_error >= 0.5:
                break

            alpha = 0.5 * np.log((1.0 - weighted_error) / max(weighted_error, 1e-10))
            self.alphas.append(alpha)
            self.models.append(model)

            # Update weights
            weights *= np.exp(-alpha * y * y_pred)
            weights /= np.sum(weights)

        return self

    def predict(self, x):
        x = np.array(x)
        pred = np.zeros(len(x))
        for alpha, model in zip(self.alphas, self.models):
            pred += alpha * model.predict(x)
        return np.sign(pred)

    def get_params(self, deep=True):
        return {
            'n_estimators': self.n_estimators,
            'max_depth': self.max_depth,
        }

    def set_params(self, **params):
        if not params:
            return self

        for param, value in params.items():
            setattr(self, param, value)

        return self
```

In [ ]:
```
class RandomForestClassifier:
    def __init__(self, n_estimators=100, max_depth=None, min_samples_split=2, min_samples_leaf=1):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf
        self.models = []

    def fit(self, x, y):
        x = np.array(x)
        y = np.array(y)
        n_samples, n_features = x.shape
        for _ in range(self.n_estimators):
            # Randomly select a subset of features
            selected_features = np.random.choice(n_features, size=int(np.sqrt(n_features)), replace=Fal
se)
            x_subset = x[:, selected_features]

            # Create a decision tree with random features
            tree = DecisionTreeClassifier(
                max_depth=self.max_depth,
                min_samples_split=self.min_samples_split,
                min_samples_leaf=self.min_samples_leaf
            )
            tree.fit(x_subset, y)
            self.models.append((tree, selected_features))
        return self

    def predict(self, x):
        x = np.array(x)
        pred = np.zeros((x.shape[0], self.n_estimators))
        for i, (tree, selected_features) in enumerate(self.models):
            x_subset = x[:, selected_features]
            pred[:, i] = tree.predict(x_subset)

        # Use majority voting for the final prediction
        final_predictions = np.apply_along_axis(lambda x: np.bincount(x.astype(int)).argmax(), axis=1,
arr=pred)
        return final_predictions

    def get_params(self, deep=True):
        return {
            'n_estimators': self.n_estimators,
            'max_depth': self.max_depth,
            'min_samples_split': self.min_samples_split,
            'min_samples_leaf': self.min_samples_leaf
        }

    def set_params(self, **params):
        if not params:
            return self

        for param, value in params.items():
            setattr(self, param, value)

        return self
```

In [30]:
```
bagging_model = BaggingClassifier(n_estimators=100, max_depth=60)
bagging_model.fit(xtrain, ytrain)

bagging_predictions = bagging_model.predict(xtest)
bagging_accuracy = accuracy_score(ytest, bagging_predictions)
print(f"Bagging Accuracy: {bagging_accuracy}")
```

```
Bagging Accuracy: 0.7316913595880292
```

In [31]:
```
adaboost = AdaBoost(n_estimators=300)
adaboost.fit(xtrain, ytrain)

predictions = adaboost.predict(xtest)

accuracy = accuracy_score(ytest, predictions)
print(f'Boosting Accuracy: {accuracy}')
```

```
Boosting Accuracy: 0.7387117470769491
```

In [32]:
```
random_forest = RandomForestClassifier(n_estimators=150, max_depth=3, min_samples_split=2, min_samples_
leaf=1)
random_forest.fit(xtrain, ytrain)

predictions = random_forest.predict(xtest)
accuracy = accuracy_score(ytest, predictions)
print(f'Random Forest Accuracy: {accuracy}')
```

```
Random Forest Accuracy: 0.7062597610907095
```

Hyperparameter Tuning

We will use grid search and randomized search methods to choose better hyperparameters

In [22]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from skopt import BayesSearchCV
import multiprocessing

n_jobs = multiprocessing.cpu_count()-1

np.int = int

def print_results(search_results):
    best_params = search_results.best_params_
    best_score = search_results.best_score_

    print("Best Parameters:", best_params)
    print("Best Score:", best_score)
```

Tuning Bagging model

In [23]:
```
params = {
    'n_estimators': [100, 200, 250, 300],
    'max_depth': [50, 60, 70, 90],
    'threshold': [0.3, 0.5, 0.7],
}

bagging = BaggingClassifier()
grid_search = GridSearchCV(estimator=bagging, param_grid=params, scoring='accuracy', n_jobs=n_jobs)
grid_search.fit(xtrain, ytrain)
print("Grid search - Bagging: ")
print_results(grid_search)
```

```
Grid search - Bagging:
Best Parameters: {'max_depth': 50, 'n_estimators': 100, 'threshold': 0.5}
Best Score: 0.7288794679743616
```

In [24]:
```
random_search = RandomizedSearchCV(estimator=bagging, param_distributions=params, n_iter=4, scoring='ac
curacy', random_state=42, n_jobs=n_jobs)

random_search.fit(xtrain, ytrain)
print("Random search - Bagging: ")
print_results(random_search)
```

```
Random search - Bagging:
Best Parameters: {'threshold': 0.5, 'n_estimators': 100, 'max_depth': 90}
Best Score: 0.7281558532037508
```

In [ ]:
```
bayes_optimization = BayesSearchCV(bagging, params, n_iter=6, scoring='accuracy', random_state=42, n_jo
bs=n_jobs)

bayes_optimization.fit(xtrain, ytrain)
print("Bayesian Optimization - Bagging: ")
print_results(bayes_optimization)
```

Tuning AdaBoost model

In [15]:
```
params = {
    'n_estimators': [100, 200, 250, 300, 500, 1000],
}

adaboost = AdaBoost()
grid_search = GridSearchCV(estimator=adaboost, param_grid=params, scoring='accuracy', n_jobs=n_jobs)
grid_search.fit(xtrain, ytrain)
print("Grid search - AdaBoost: ")
print_results(grid_search)
```

```
Grid search - AdaBoost:
Best Parameters: {'n_estimators': 100}
Best Score: 0.7142623573542956
```

In [37]:
```
random_search = RandomizedSearchCV(estimator=adaboost, param_distributions=params, n_iter=4, scoring='a
ccuracy', random_state=42, n_jobs=n_jobs)

random_search.fit(xtrain, ytrain)
print("Random search - AdaBoost: ")
print_results(random_search)
```

```
Random search - AdaBoost:
Best Parameters: {'n_estimators': 100}
Best Score: 0.7142623573542956
```

In [18]:
```
bayes_optimization = BayesSearchCV(adaboost, params, n_iter=6, scoring='accuracy', random_state=42, n_j
obs=n_jobs)

bayes_optimization.fit(xtrain, ytrain)
print("Bayesian Optimization - AdaBoost: ")
print_results(bayes_optimization)
```

```
Bayesian Optimization - AdaBoost:
Best Parameters: OrderedDict([('n_estimators', 250)])
Best Score: 0.7142623573542956
```

Tuning Random Forest model

In [38]:
```
params = {
    'n_estimators': [100, 200, 250],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 3],
    'min_samples_leaf': [1, 2, 4],
}

rf_model = RandomForestClassifier()
grid_search = GridSearchCV(estimator=rf_model, param_grid=params, scoring='accuracy', n_jobs=n_jobs)
grid_search.fit(xtrain, ytrain)

print("Grid search - Random Forest: ")
print_results(grid_search)
```

```
Grid search - Random Forest:
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 25
0}
Best Score: 0.7214714514997054
```

In [40]:
```
random_search = RandomizedSearchCV(estimator=rf_model, param_distributions=params, n_iter=6, scoring='a
ccuracy', random_state=42, n_jobs=n_jobs)
random_search.fit(xtrain, ytrain)

print("Random Search - Random Forest: ")
print_results(random_search)
```

```
Random Search - Random Forest:
Best Parameters: {'n_estimators': 250, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_depth': 1
0}
Best Score: 0.7211367772692194
```

In [20]:
```
bayes_optimization = BayesSearchCV(rf_model, params, n_iter=6, scoring='accuracy', random_state=42, n_j
obs=n_jobs)

bayes_optimization.fit(xtrain, ytrain)
print("Bayesian Optimization - Random Forest: ")
print_results(bayes_optimization)
```

```
Bayesian Optimization - Random Forest:
Best Parameters: OrderedDict([('max_depth', 10), ('min_samples_leaf', 2), ('min_samples_split', 3),
('n_estimators', 200)])
Best Score: 0.7210064431050265
```