IMPORTS

```
# Imports
import numpy as np
import random
import matplotlib.pyplot as plt
import pandas as pd
from pandas.plotting import scatter_matrix
import seaborn as sns
from sklearn.model_selection import
train_test_split,RandomizedSearchCV
from sklearn.ensemble import BaggingClassifier,AdaBoostClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.utils import resample
from scipy import stats
from sklearn.tree import DecisionTreeClassifier
import collections
from sklearn import metrics
!pip install scikit-optimize
from skopt import BayesSearchCV
from skopt.space import Real, Integer
from scipy.stats import randint


Requirement already satisfied: scikit-optimize in
/usr/local/lib/python3.10/dist-packages (0.9.0)
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.3.2)
Requirement already satisfied: pyaml>=16.9 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize)
(23.9.7)
Requirement already satisfied: numpy>=1.13.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize)
(1.23.5)
Requirement already satisfied: scipy>=0.19.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize)
(1.11.4)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.2.2)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.10/dist-packages (from pyaml>=16.9->scikit-
optimize) (6.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0-
>scikit-optimize) (3.2.0)
```

Loading the Data

```
# Load the CSV file

from google.colab import drive
drive.mount('/content/drive')

train_set=pd.read_csv('/content/drive/MyDrive/train_set.csv')
val_set=pd.read_csv('/content/drive/MyDrive/validation_set.csv')
test_set=pd.read_csv('/content/drive/MyDrive/test_set.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
train_set.head()
```

```
        ALT  eyesight(left)  hemoglobin       age  waist(cm)
hearing(right)  \
0  0.400546        0.642857    0.608696  0.384615   0.423684
0.0
1  0.352112        0.642857    0.608696  0.461538   0.328947
0.0
2  0.276602        0.571429    0.521739  0.307692   0.171053
0.0
3  0.362285        0.571429    0.496894  0.615385   0.328947
0.0
4  0.285462        0.428571    0.565217  0.615385   0.315789
1.0

   height(cm)  systolic       HDL  smoking
0   -0.030268  1.060261 -1.150036        1
1   -0.597229 -0.432361  1.448097        1
2   -0.597229 -2.239220  0.551718        1
3   -1.164191 -1.767866  0.484563        0
4   -1.164191  1.060261 -0.677961        0
```

```
X_train_scaled = train_set.drop("smoking", axis=1)
y_train = train_set['smoking'].copy()

X_valid_scaled = val_set.drop("smoking", axis=1)
y_valid = val_set['smoking'].copy()

X_test_scaled = test_set.drop("smoking", axis=1)
y_test = test_set['smoking'].copy()
```

Training Models

1) Bagging

Our problem is considered a "Binary Classification" problem, so using Logistic Regression and Decision Tree Classifier in Bagging were the most suitable models.

1.1) Using Built-In bagging classifier

```
dt = DecisionTreeClassifier()
bootstrap_samples = 20
bagging = BaggingClassifier(estimator = dt, n_estimators =
bootstrap_samples, random_state = 42)

# Train the Bagging classifier
bagging.fit(X_train_scaled, y_train)

# Make predictions on the test set
predictions = bagging.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)

print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.7112478546611412
```

Tuning Built-In

1.2) Linear Regression Implementation

This version of bagging uses Linear Regression Model as base model.

```
def Bagging_LR(X, y, test, samples):
    predictions = []
    for i in range (samples):
        # The resample function created bootstrap samples (sampling by
replacement)
        X_sample, y_sample = resample(X,y)
        log_reg = LogisticRegression()
        log_reg.fit(X_sample, y_sample)
        pred = log_reg.predict(test)
        predictions.append(pred)
    predictions = np.array(predictions)
    # Majority Voting: The mode is the value that appears most
frequently
    final_predictions = stats.mode(predictions, axis = 0)[0]
    return final_predictions
```

Tuning LR

Testing for different values of n_estimators to check if the accuracy is improved.

```
n_estimators_values = [5, 10, 15, 20, 50, 100]
results = []
for n in n_estimators_values:
    predictions = Bagging_LR(X_train_scaled, y_train, X_test_scaled,
n)
    accuracy = accuracy_score(y_test, predictions)
    results.append({"n_estimators": n, "Accuracy %": accuracy*100})

results_df = pd.DataFrame(results)
results_df
```

	n_estimators	Accuracy %
0	5	71.283854
1	10	71.254552
2	15	71.292227
3	20	71.279668
4	50	71.258738
5	100	71.304785

1.3) Decision Tree Implementation

This version of bagging uses Decision Tree Model as base model.

```
def Bagging_DT(X_train, y_train, X_test, n_estimators):
    predictions = []
    for _ in range(n_estimators):
        X_sample, y_sample = resample(X_train, y_train)
        model = DecisionTreeClassifier()
        model.fit(X_sample, y_sample)
        predictions.append(model.predict(X_test))

    aggregated_predictions = stats.mode(predictions, axis=0).mode
    return aggregated_predictions
```

Tuning DT

Testing for different values of n_estimators to check if the accuracy is improved.

```
n_estimators_values = [5, 10, 15, 20, 50, 100]
results = []
for n in n_estimators_values:
    predictions = Bagging_DT(X_train_scaled, y_train, X_test_scaled,
n)
    accuracy = accuracy_score(y_test, predictions)
    results.append({"n_estimators": n, "Accuracy %": accuracy*100})
```

```
results_df = pd.DataFrame(results)
results_df
```

```
   n_estimators  Accuracy %
0             5   69.873163
1            10   69.366654
2            15   71.179204
3            20   71.530830
4            50   72.401524
5           100   72.606639
```

When we used Linear Regression Model with Bagging, the accuracy stopped improving at almost 71.39%, but with Decision Trees it reached 72.46% which is better than DT model.

So, in the final model we will use the Bagging_DT model with n_estimators = 100 to ensure the best accuracy.

2) Boosting

```
# Create AdaBoost classifier with a decision stump as a base estimator
base_estimator = DecisionTreeClassifier(max_depth=1)
adaboost_classifier =
AdaBoostClassifier(base_estimator=base_estimator)

# Define the search space for Bayesian optimization
param_space = {'n_estimators': Integer(5, 100)}  # Integer values
between 5 and 100

# Perform Bayesian optimization
bayes_search = BayesSearchCV(adaboost_classifier, param_space,
n_iter=10, cv=5, scoring='accuracy', random_state=42)
bayes_search.fit(X_train_scaled, y_train)

# Get the best parameters and the corresponding model
best_n_estimators = bayes_search.best_params_['n_estimators']
best_model = bayes_search.best_estimator_

# Train the best model on the entire training set
best_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = best_model.predict(X_valid_scaled)

# Evaluate accuracy
accuracy = accuracy_score(y_valid, y_pred)
print(f"Best n_estimators: {best_n_estimators}")
print(f"Accuracy: {accuracy}")
```

```
class AdaBoostClassifier:
    def __init__(self, n_estimators=50):
        self.n_estimators = n_estimators
        self.alphas = []  # Store the weights of weak learners
        self.models = []  # Store the weak learners

    def fit(self, X, y):
        n_samples, n_features = X.shape
        # Initialize weights uniformly
        weights = np.ones(n_samples) / n_samples
        for _ in range(self.n_estimators):
            # Train a weak learner
            model = self._train_weak_learner(X, y, weights)
            # Make predictions
            predictions = model.predict(X)
            # Calculate error
            error = np.sum(weights * (predictions != y))
            # Calculate alpha (weight of the weak learner)
            alpha = 0.5 * np.log((1 - error) / max(error, 1e-10))
            # Update weights
            weights *= np.exp(-alpha * y * predictions)
            weights /= np.sum(weights)
            # Store the weak learner and its weight
            self.models.append(model)
            self.alphas.append(alpha)

    def _train_weak_learner(self, X, y, weights):
        # For simplicity, using a decision stump (depth=1) as a weak
learner
        weak_learner = DecisionTreeClassifier(max_depth=1)
        weak_learner.fit(X, y, sample_weight=weights)
        return weak_learner

    def predict(self, X):
        # Combine weak learners' predictions with their weights
        predictions = np.array([model.predict(X) for model in
self.models])
        weighted_predictions = np.dot(self.alphas, predictions)
        # Convert to binary predictions
        return np.sign(weighted_predictions)


# Create and train AdaBoost classifier
n_estimators=50
adaboost_classifier = AdaBoostClassifier(n_estimators=50)
adaboost_classifier.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = adaboost_classifier.predict(X_valid_scaled)
```

```
# Evaluate accuracy
accuracy = accuracy_score(y_valid, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.6827277294038848
```

```
# Create and train AdaBoost classifier
adaboost_classifier = AdaBoostClassifier(n_estimators=96)
adaboost_classifier.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = adaboost_classifier.predict(X_test_scaled)

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

```
Accuracy: 0.6844154213236218
```

3) Random Forests

```
class RandomForest:
    def __init__(self, n_estimators=100, max_feat=3, max_depth=None):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.max_feat = max_feat
        self.trees = []

    def fit(self, X, y):
        for _ in range(self.n_estimators):
            X_sample, y_sample = resample(X,y)

            tree = DecisionTreeClassifier(max_depth=self.max_depth,
max_features=self.max_feat, random_state=42)
            tree.fit(X_sample, y_sample)
            self.trees.append(tree)

    def predict(self, X):
        predictions = [tree.predict(X) for tree in self.trees]
        predictions = np.array(predictions).T  # Transposing to get
correct format

        actual_predictions = []
        for prediction in predictions:
            counter = collections.Counter(prediction)
            most_common = counter.most_common(1)[0][0]
            actual_predictions.append(most_common)
```

```
        return actual_predictions

n=20
performance =
pd.DataFrame(columns=['n_estimators','n_feat','train_accuracy','val_ac
curacy'])

for n_feat in range (2,7):
  row=[]

  random_forest = RandomForest( n, n_feat)
  # Train the classifier
  random_forest.fit(X_train_scaled, y_train)

  # Make predictions
  y_pred = random_forest.predict(X_train_scaled)
  # Evaluate the model
  train_accuracy = metrics.accuracy_score(y_train, y_pred)

  y_pred = random_forest.predict(X_valid_scaled)
  # Evaluate the model
  val_accuracy = metrics.accuracy_score(y_valid, y_pred)
  row = {'n_estimators': [n],'n_feat': [n_feat],'train_accuracy':
[train_accuracy],'val_accuracy': [val_accuracy]}
  new_row = pd.DataFrame(row)

  performance = pd.concat([performance, new_row], ignore_index=True)
#performance.head()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Print the DataFrame
print(performance)
```

```
  n_estimators n_feat  train_accuracy  val_accuracy
0           20      2        0.996717      0.718059
1           20      3        0.997040      0.723669
2           20      4        0.996699      0.717348
3           20      5        0.996466      0.719064
4           20      6        0.996968      0.721283
```

```
n=50
performance =
pd.DataFrame(columns=['n_estimators','n_feat','train_accuracy','val_ac
curacy'])

for n_feat in range (2,7):
  row=[]

  random_forest = RandomForest( n, n_feat)
```

```
  # Train the classifier
  random_forest.fit(X_train_scaled, y_train)

  # Make predictions
  y_pred = random_forest.predict(X_train_scaled)
  # Evaluate the model
  train_accuracy = metrics.accuracy_score(y_train, y_pred)

  y_pred = random_forest.predict(X_valid_scaled)
  # Evaluate the model
  val_accuracy = metrics.accuracy_score(y_valid, y_pred)
  row = {'n_estimators': [n],'n_feat': [n_feat],'train_accuracy':
[train_accuracy],'val_accuracy': [val_accuracy]}
  new_row = pd.DataFrame(row)

  performance = pd.concat([performance, new_row], ignore_index=True)
#performance.head()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Print the DataFrame
print(performance)
```

```
  n_estimators n_feat  train_accuracy  val_accuracy
0           50      2        0.999794      0.730785
1           50      3        0.999812      0.728232
2           50      4        0.999758      0.728860
3           50      5        0.999776      0.726222
4           50      6        0.999803      0.726013
```

```
n=70
performance =
pd.DataFrame(columns=['n_estimators','n_feat','train_accuracy','val_ac
curacy'])

for n_feat in range (2,7):
  row=[]

  random_forest = RandomForest( n, n_feat)
  # Train the classifier
  random_forest.fit(X_train_scaled, y_train)

  # Make predictions
  y_pred = random_forest.predict(X_train_scaled)
  # Evaluate the model
  train_accuracy = metrics.accuracy_score(y_train, y_pred)

  y_pred = random_forest.predict(X_valid_scaled)
  # Evaluate the model
  val_accuracy = metrics.accuracy_score(y_valid, y_pred)
```

```
  row = {'n_estimators': [n],'n_feat': [n_feat],'train_accuracy':
[train_accuracy],'val_accuracy': [val_accuracy]}
  new_row = pd.DataFrame(row)

  performance = pd.concat([performance, new_row], ignore_index=True)
#performance.head()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Print the DataFrame
print(performance)
```

```
  n_estimators n_feat  train_accuracy  val_accuracy
0           70      2        0.999973      0.734427
1           70      3        0.999982      0.729781
2           70      4        0.999937      0.730157
3           70      5        0.999910      0.730283
4           70      6        0.999919      0.726725
```

```
n=100
performance =
pd.DataFrame(columns=['n_estimators','n_feat','train_accuracy','val_ac
curacy'])

for n_feat in range (2,7):
  row=[]

  random_forest = RandomForest( n, n_feat)
  # Train the classifier
  random_forest.fit(X_train_scaled, y_train)

  # Make predictions
  y_pred = random_forest.predict(X_train_scaled)
  # Evaluate the model
  train_accuracy = metrics.accuracy_score(y_train, y_pred)

  y_pred = random_forest.predict(X_valid_scaled)
  # Evaluate the model
  val_accuracy = metrics.accuracy_score(y_valid, y_pred)
  row = {'n_estimators': [n],'n_feat': [n_feat],'train_accuracy':
[train_accuracy],'val_accuracy': [val_accuracy]}
  new_row = pd.DataFrame(row)
  performance = pd.concat([performance, new_row], ignore_index=True)
#performance.head()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Print the DataFrame
print(performance)
```

```
  n_estimators n_feat  train_accuracy  val_accuracy
0          100      2        0.999991      0.731957
1          100      3        0.999991      0.730702
2          100      4        0.999973      0.729320
3          100      5        0.999973      0.730074
4          100      6        0.999991      0.727436
```

```
n=150
performance =
pd.DataFrame(columns=['n_estimators','n_feat','train_accuracy','val_ac
curacy'])

for n_feat in range (2,7):
  row=[]

  random_forest = RandomForest( n, n_feat)
  # Train the classifier
  random_forest.fit(X_train_scaled, y_train)

  # Make predictions
  y_pred = random_forest.predict(X_train_scaled)
  # Evaluate the model
  train_accuracy = metrics.accuracy_score(y_train, y_pred)

  y_pred = random_forest.predict(X_valid_scaled)
  # Evaluate the model
  val_accuracy = metrics.accuracy_score(y_valid, y_pred)
  row = {'n_estimators': [n],'n_feat': [n_feat],'train_accuracy':
[train_accuracy],'val_accuracy': [val_accuracy]}
  new_row = pd.DataFrame(row)
  performance = pd.concat([performance, new_row], ignore_index=True)
#performance.head()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Print the DataFrame
print(performance)
```

```
  n_estimators n_feat  train_accuracy  val_accuracy
0          150      2             1.0      0.732627
1          150      3             1.0      0.732627
2          150      4             1.0      0.731330
3          150      5             1.0      0.730618
4          150      6             1.0      0.729069
```

```
n=200
performance =
pd.DataFrame(columns=['n_estimators','n_feat','train_accuracy','val_ac
curacy'])

for n_feat in range (2,7):
```

```
  row=[]

  random_forest = RandomForest( n, n_feat)
  # Train the classifier
  random_forest.fit(X_train_scaled, y_train)

  # Make predictions
  y_pred = random_forest.predict(X_train_scaled)
  # Evaluate the model
  train_accuracy = metrics.accuracy_score(y_train, y_pred)

  y_pred = random_forest.predict(X_valid_scaled)
  # Evaluate the model
  val_accuracy = metrics.accuracy_score(y_valid, y_pred)
  row = {'n_estimators': [n],'n_feat': [n_feat],'train_accuracy':
[train_accuracy],'val_accuracy': [val_accuracy]}
  new_row = pd.DataFrame(row)

  performance = pd.concat([performance, new_row], ignore_index=True)
#performance.head()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Print the DataFrame
print(performance)
```

```
  n_estimators n_feat  train_accuracy  val_accuracy
0          200      2             1.0      0.733799
1          200      3             1.0      0.732083
2          200      4             1.0      0.732334
3          200      5             1.0      0.731037
4          200      6             1.0      0.728943
```

```
n=250
performance =
pd.DataFrame(columns=['n_estimators','n_feat','train_accuracy','val_ac
curacy'])

for n_feat in range (2,7):
  row=[]

  random_forest = RandomForest( n, n_feat)
  # Train the classifier
  random_forest.fit(X_train_scaled, y_train)

  # Make predictions
  y_pred = random_forest.predict(X_train_scaled)
  # Evaluate the model
  train_accuracy = metrics.accuracy_score(y_train, y_pred)

  y_pred = random_forest.predict(X_valid_scaled)
```

```
  # Evaluate the model
  val_accuracy = metrics.accuracy_score(y_valid, y_pred)
  row = {'n_estimators': [n],'n_feat': [n_feat],'train_accuracy':
[train_accuracy],'val_accuracy': [val_accuracy]}
  new_row = pd.DataFrame(row)

  performance = pd.concat([performance, new_row], ignore_index=True)
#performance.head()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Print the DataFrame
print(performance)
```

```
  n_estimators n_feat  train_accuracy  val_accuracy
0          250      2             1.0      0.734972
1          250      3             1.0      0.732418
2          250      4             1.0      0.733423
3          250      5             1.0      0.731790
4          250      6             1.0      0.728190
```

```
n=300
performance =
pd.DataFrame(columns=['n_estimators','n_feat','train_accuracy','val_ac
curacy'])

for n_feat in range (2,7):
  row=[]

  random_forest = RandomForest( n, n_feat)
  # Train the classifier
  random_forest.fit(X_train_scaled, y_train)

  # Make predictions
  y_pred = random_forest.predict(X_train_scaled)
  # Evaluate the model
  train_accuracy = metrics.accuracy_score(y_train, y_pred)

  y_pred = random_forest.predict(X_valid_scaled)
  # Evaluate the model
  val_accuracy = metrics.accuracy_score(y_valid, y_pred)
  row = {'n_estimators': [n],'n_feat': [n_feat],'train_accuracy':
[train_accuracy],'val_accuracy': [val_accuracy]}
  new_row = pd.DataFrame(row)

  performance = pd.concat([performance, new_row], ignore_index=True)
#performance.head()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

```
# Print the DataFrame
print(performance)
```

```
  n_estimators n_feat  train_accuracy  val_accuracy
0          300      2             1.0      0.734469
1          300      3             1.0      0.734009
2          300      4             1.0      0.730116
3          300      5             1.0      0.731999
4          300      6             1.0      0.729571
```

```
n=350
performance =
pd.DataFrame(columns=['n_estimators','n_feat','train_accuracy','val_ac
curacy'])

for n_feat in range (2,7):
  row=[]

  random_forest = RandomForest( n, n_feat)
  # Train the classifier
  random_forest.fit(X_train_scaled, y_train)

  # Make predictions
  y_pred = random_forest.predict(X_train_scaled)
  # Evaluate the model
  train_accuracy = metrics.accuracy_score(y_train, y_pred)

  y_pred = random_forest.predict(X_valid_scaled)
  # Evaluate the model
  val_accuracy = metrics.accuracy_score(y_valid, y_pred)
  row = {'n_estimators': [n],'n_feat': [n_feat],'train_accuracy':
[train_accuracy],'val_accuracy': [val_accuracy]}
  new_row = pd.DataFrame(row)

  performance = pd.concat([performance, new_row], ignore_index=True)
#performance.head()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Print the DataFrame
print(performance)
```

```
  n_estimators n_feat  train_accuracy  val_accuracy
0          350      2             1.0      0.734260
1          350      3             1.0      0.732753
2          350      4             1.0      0.731497
3          350      5             1.0      0.733088
4          350      6             1.0      0.729990
```

from this we observe that the best model the model with n_estmimators=250 and maximum features considered in each split = 2 .So we'll use this model for testing.

```
n_estimators = 250
max_feat=2
random_forest = RandomForest( n_estimators, max_feat)
# Train the classifier
random_forest.fit(X_train_scaled, y_train)
y_pred = random_forest.predict(X_test_scaled)
test_accuracy = metrics.accuracy_score(y_test, y_pred)
print("Testing accuracy=",test_accuracy)
```

```
Testing accuracy= 0.7378291263761564
```