5.Attempting to increase accuracy

we seem to have reached a bottleneck accuracy of around 70%. we will attempt to explore means of improving the model's performance.

5.1 using engineered feature

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.base import clone
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Load the dataset
data = pd.read_csv("MyData.csv")

# Drop the first column by index
data = data.drop(data.columns[0], axis=1)

# save the updated dataset back to a CSV file
data.to_csv("MyData_updated.csv", index=False)

# Display the first few rows
print("First few rows of the dataset:")
display(data.head())

# Overview of the dataset
print("\nDataset Information:")
data.info()

print("\nStatistical Summary:")
display(data.describe())
```

```
First few rows of the dataset:

   hearing(left)  Cholesterol  ALT  eyesight(left)  waist(cm)
hearing(right)  \
0              1          172   25             0.5       81.0
1
1              2          194   23             0.6       89.0
2
```

```
2              1         178   31             0.4       81.0
1
3              1         180   27             1.5      105.0
1
4              1         155   13             1.5       80.5
1

   dental caries  hemoglobin  weight(kg)  serum creatinine  smoking
0              0        16.5          60               1.0        1
1              1        16.2          65               1.1        0
2              0        17.4          75               0.8        1
3              1        15.9          95               1.0        0
4              0        15.4          60               0.8        1


Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159256 entries, 0 to 159255
Data columns (total 11 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   hearing(left)     159256 non-null  int64
 1   Cholesterol       159256 non-null  int64
 2   ALT               159256 non-null  int64
 3   eyesight(left)    159256 non-null  float64
 4   waist(cm)         159256 non-null  float64
 5   hearing(right)    159256 non-null  int64
 6   dental caries     159256 non-null  int64
 7   hemoglobin        159256 non-null  float64
 8   weight(kg)        159256 non-null  int64
 9   serum creatinine  159256 non-null  float64
 10  smoking           159256 non-null  int64
dtypes: float64(4), int64(7)
memory usage: 13.4 MB

Statistical Summary:

       hearing(left)    Cholesterol            ALT  eyesight(left)  \
count  159256.000000  159256.000000  159256.000000   159256.000000
mean        1.023974     195.796165      26.550296        1.005798
std         0.152969      28.396959      17.753070        0.402113
min         1.000000      77.000000       1.000000        0.100000
25%         1.000000     175.000000      16.000000        0.800000
50%         1.000000     196.000000      22.000000        1.000000
75%         1.000000     217.000000      32.000000        1.200000
max         2.000000     393.000000    2914.000000        9.900000

           waist(cm)  hearing(right)  dental caries     hemoglobin  \
count  159256.000000   159256.000000  159256.000000  159256.000000
mean       83.001990        1.023421       0.197996      14.796965
```

```
std          8.957937        0.151238       0.398490       1.431213
min         51.000000        1.000000       0.000000       4.900000
25%         77.000000        1.000000       0.000000      13.800000
50%         83.000000        1.000000       0.000000      15.000000
75%         89.000000        1.000000       0.000000      15.800000
max        127.000000        2.000000       1.000000      21.000000

          weight(kg)  serum creatinine        smoking
count  159256.000000     159256.000000  159256.000000
mean       67.143662          0.892764       0.437365
std        12.586198          0.179346       0.496063
min        30.000000          0.100000       0.000000
25%        60.000000          0.800000       0.000000
50%        65.000000          0.900000       0.000000
75%        75.000000          1.000000       1.000000
max       130.000000          9.900000       1.000000
```

```
# Check for missing values
missing_values = data.isnull().sum()
print("\nMissing Values in Each Column:")
print(missing_values[missing_values > 0])

#Handle missing values:
data.fillna(data.median(), inplace=True)

#remove outliers using
IQR!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
df = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 *
IQR))).any(axis=1)] ## remvoe outliers

#scaling (Normalization)
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

```
Missing Values in Each Column:
Series([], dtype: int64)
```

```
features =[ 'waist(cm)', 'hemoglobin', 'weight(kg)', 'serum
creatinine','eyesight(left)','Cholesterol']

# Split the data into training and testing sets
X = df_scaled[features]
y = df['smoking']

# Convert X and y to numpy arrays for clarity
X_scaled = np.array(X)
```

```
y = np.array(y)

# Engineered feature: Multiply features across columns for each sample
E_feature = X_scaled[:, 1] * X_scaled[:, 2]

# Concatenate the engineered feature to the original features
E_X = np.column_stack((X_scaled, E_feature))  # Shape will now be
(109386, 5)

# Split into train, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(E_X, y,
test_size=0.4, random_state=42, stratify=y)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42, stratify=y_temp)



class RandomForest:
    def __init__(self, base_estimators=None, n_estimators=100,
max_features='sqrt', random_state=None):
        """
        Random Forest classifier that can use multiple base
estimators.

        Parameters:
        - base_estimators: List of base models to use for ensemble
(e.g., [DecisionTree, LogisticRegression]).
        - n_estimators: Total number of models to train.
        - max_features: The number of features to use for each model.
Options: 'sqrt', 'log2', or an integer.
        - random_state: Random seed for reproducibility.
        """
        self.base_estimators = base_estimators or
[DecisionTreeClassifier(random_state=random_state)]
        self.n_estimators = n_estimators
        self.max_features = max_features
        self.random_state = random_state
        self.models = []

    def fit(self, X, y):
        """
        Train the RandomForest classifier using bootstrap sampling and
feature selection.
        """
        np.random.seed(self.random_state)
        self.models = []

        n_samples, n_features = X.shape
        n_estimators_per_model = self.n_estimators //
len(self.base_estimators)
```

```
        for base_estimator in self.base_estimators:
            for _ in range(n_estimators_per_model):
                # Bootstrap sampling
                indices = np.random.choice(n_samples, size=n_samples,
replace=True)
                X_bootstrap = X[indices]
                y_bootstrap = y[indices]


                max_features = n_features

                features = np.random.choice(n_features,
size=max_features, replace=False)
                X_bootstrap = X_bootstrap[:, features]

                # Train a model on the bootstrap sample with a random
subset of features
                model = clone(base_estimator)
                model.fit(X_bootstrap, y_bootstrap)
                self.models.append((model, features))

    def predict(self, X):
        """
        Predict class labels using majority voting.
        """
        predictions = np.zeros((len(self.models), len(X)))
        for i, (model, features) in enumerate(self.models):
            X_subset = X[:, features]
            predictions[i, :] = model.predict(X_subset)

        # Majority vote (for classification)
        return np.round(np.mean(predictions, axis=0)).astype(int)

best_estimators = [
    DecisionTreeClassifier(max_depth=3, random_state=42),
    KNeighborsClassifier(n_neighbors=3)
]

X_combined = np.concatenate((X_train, X_valid), axis=0)
y_combined= np.concatenate((y_train,y_valid), axis=0)

    # Train the Random Forest ensemble
rf_model = RandomForest(base_estimators=best_estimators,
n_estimators=150, random_state=42,  max_features=3)
rf_model.fit(X_combined, y_combined)

# Make predictions
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.7093427187128623
```