Import

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

Read data

```
df = pd.read_csv('newdata.csv', index_col=[0])
df.index.name = 'id'
df.head()
```

```
    LDL  weight(kg)  systolic  Cholesterol  ALT  Gtp  triglyceride  \
id
0    75          60       135          172   25   27           300
1   126          65       146          194   23   37            55
2    93          75       118          178   31   53           197
3   102          95       131          180   27   30           203
4    93          60       121          155   13   17            87

    Urine protein  dental caries  height(cm)  smoking
id
0               1              0         165        1
1               1              1         165        0
2               1              0         170        1
3               1              1         180        0
4               1              0         165        1
```

Feature Engenering using(polynomil features)

```
trans = PolynomialFeatures(degree=2, include_bias = False)
print(df.columns)
Y=df['smoking']

df.drop('smoking',axis=1 , inplace =True)

data = trans.fit_transform(np.array(df)) # bta5od np array
feature_names = df.columns  # Original feature, names mn gher el
smoking

# Use get_feature_names_out method to get feature names for the
polynomial features
```

```
poly_feature_names =
trans.get_feature_names_out(input_features=feature_names)# model
feature_index_mapping = {i: name for i, name in
enumerate(poly_feature_names)}
# Access the feature names for each column of the transformed array
column_headers = [feature_index_mapping[i] for i in
range(data.shape[1])]# shape 1 3dd el columns

df = pd.DataFrame(data)
df.columns = column_headers
print(df.head())
```

```
Index(['LDL', 'weight(kg)', 'systolic', 'Cholesterol', 'ALT', 'Gtp',
       'triglyceride', 'Urine protein', 'dental caries', 'height(cm)',
       'smoking'],
      dtype='object')
     LDL  weight(kg)  systolic  Cholesterol   ALT   Gtp  triglyceride
\
0   75.0        60.0     135.0        172.0  25.0  27.0         300.0

1  126.0        65.0     146.0        194.0  23.0  37.0          55.0

2   93.0        75.0     118.0        178.0  31.0  53.0         197.0

3  102.0        95.0     131.0        180.0  27.0  30.0         203.0

4   93.0        60.0     121.0        155.0  13.0  17.0          87.0


   Urine protein  dental caries  height(cm)  ...  triglyceride^2  \
0            1.0            0.0       165.0  ...         90000.0
1            1.0            1.0       165.0  ...          3025.0
2            1.0            0.0       170.0  ...         38809.0
3            1.0            1.0       180.0  ...         41209.0
4            1.0            0.0       165.0  ...          7569.0

   triglyceride Urine protein  triglyceride dental caries  \
0                       300.0                         0.0
1                        55.0                        55.0
2                       197.0                         0.0
3                       203.0                       203.0
4                        87.0                         0.0

   triglyceride height(cm)  Urine protein^2  Urine protein dental
caries  \
0                  49500.0              1.0
0.0
1                   9075.0              1.0
1.0
2                  33490.0              1.0
```

```
0.0
3                   36540.0               1.0
1.0
4                   14355.0               1.0
0.0

   Urine protein height(cm)  dental caries^2  dental caries height(cm)
\
0                     165.0              0.0                       0.0

1                     165.0              1.0                     165.0

2                     170.0              0.0                       0.0

3                     180.0              1.0                     180.0

4                     165.0              0.0                       0.0


   height(cm)^2
0       27225.0
1       27225.0
2       28900.0
3       32400.0
4       27225.0

[5 rows x 65 columns]
```

choose best 10 features

```python
y = Y # smoking
X = df

from sklearn.feature_selection import SelectKBest, chi2
column_names = X.columns # asamy columns

df = pd.DataFrame(X, columns=column_names)

# Use SelectKBest with chi2 to select the top features
k_best = 10  # Number of top features to select
chi2_features = SelectKBest(chi2, k=k_best) # return object

X_kbest_features = chi2_features.fit_transform(X, y) # ab3tlo data

# Get the indices of the selected features
selected_feature_indices = chi2_features.get_support(indices=True)

# Get the names of the selected features
selected_feature_names = df.columns[selected_feature_indices]
```

```
print("Original Feature Names:", df.columns)
print("Selected Feature Names:", selected_feature_names)
```

```
Original Feature Names: Index(['LDL', 'weight(kg)', 'systolic',
'Cholesterol', 'ALT', 'Gtp',
       'triglyceride', 'Urine protein', 'dental caries', 'height(cm)',
'LDL^2',
       'LDL weight(kg)', 'LDL systolic', 'LDL Cholesterol', 'LDL ALT',
       'LDL Gtp', 'LDL triglyceride', 'LDL Urine protein', 'LDL dental
caries',
       'LDL height(cm)', 'weight(kg)^2', 'weight(kg) systolic',
       'weight(kg) Cholesterol', 'weight(kg) ALT', 'weight(kg) Gtp',
       'weight(kg) triglyceride', 'weight(kg) Urine protein',
       'weight(kg) dental caries', 'weight(kg) height(cm)',
'systolic^2',
       'systolic Cholesterol', 'systolic ALT', 'systolic Gtp',
       'systolic triglyceride', 'systolic Urine protein',
       'systolic dental caries', 'systolic height(cm)',
'Cholesterol^2',
       'Cholesterol ALT', 'Cholesterol Gtp', 'Cholesterol
triglyceride',
       'Cholesterol Urine protein', 'Cholesterol dental caries',
       'Cholesterol height(cm)', 'ALT^2', 'ALT Gtp', 'ALT
triglyceride',
       'ALT Urine protein', 'ALT dental caries', 'ALT height(cm)',
'Gtp^2',
       'Gtp triglyceride', 'Gtp Urine protein', 'Gtp dental caries',
       'Gtp height(cm)', 'triglyceride^2', 'triglyceride Urine
protein',
       'triglyceride dental caries', 'triglyceride height(cm)',
       'Urine protein^2', 'Urine protein dental caries',
       'Urine protein height(cm)', 'dental caries^2',
       'dental caries height(cm)', 'height(cm)^2'],
      dtype='object')
Selected Feature Names: Index(['LDL triglyceride', 'weight(kg)
triglyceride', 'systolic triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)'],
      dtype='object')
```

splitting the data

```
y = Y
x = df[selected_feature_names]
X_train, X_temp, y_train, y_temp = train_test_split(x, y,
test_size=0.3, random_state=42)
```

```
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)
```

Normalization using standard scaller

```
from sklearn.preprocessing import StandardScaler
data = X_train
t = y_train
col = data
# Assuming 'df' is your DataFrame
# print(col)
scaler = StandardScaler()

df_z_scaled = pd.DataFrame(scaler.fit_transform(col),
columns=col.columns) # minus mean / standard div
# 3yzen nlz2 feature m3 target
df_z_scaled.reset_index(drop=True, inplace=True)
t.reset_index(drop=True, inplace=True)

#  now we remove the outliers
df_z_scaled['smoking'] = t
print(df_z_scaled['smoking'].head(5))
cat, numerical = [], []
for col in df_z_scaled.columns:
    if df_z_scaled[col].nunique() > 10:
        numerical.append(col)
    else:
        cat.append(col)
for col in numerical:
    Q1 = df_z_scaled[col].quantile(0.25)
    Q3 = df_z_scaled[col].quantile(0.75)

    IQR = Q3 - Q1
    lower = Q1 - 1.5*IQR
    upper = Q3 + 1.5*IQR

# Create arrays of Boolean values indicating the outlier rows
    upper_array = np.where(df_z_scaled[col] >= upper)[0]
    lower_array = np.where(df_z_scaled[col] <= lower)[0]

    # Removing the outliers
    df_z_scaled.drop(index=upper_array, inplace=True, errors='ignore')
    df_z_scaled.drop(index=lower_array, inplace=True,
errors='ignore')


print(df_z_scaled.columns)
```

```
0    1
1    1
```

```
2    1
3    0
4    0
Name: smoking, dtype: int64
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
```

normalize validation

```
tv = y_val
col = X_val

scaler = StandardScaler()
df_val_scaled = pd.DataFrame(scaler.fit_transform(col),
columns=col.columns)
df_val_scaled.reset_index(drop=True, inplace=True)
tv.reset_index(drop=True, inplace=True)
#  now we remove the outliers

data_val = trans.fit_transform(np.array(df_val_scaled))
feature_names = df_val_scaled.columns  # Original feature names

# Use get_feature_names_out method to get feature names for the
polynomial features
poly_feature_names =
trans.get_feature_names_out(input_features=feature_names)

# Create a dictionary to map column indices to feature names
feature_index_mapping = {i: name for i, name in
enumerate(poly_feature_names)}

# Access the feature names for each column of the transformed array
column_headers = [feature_index_mapping[i] for i in
range(df_val_scaled.shape[1])]


df_val_scaled = pd.DataFrame(df_val_scaled)
df_val_scaled.columns = column_headers

df_val_scaled['smoking'] = tv

y_train = df_z_scaled['smoking']
df_z_scaled.drop('smoking', axis = 1, inplace=True)
Y_validation = df_val_scaled['smoking']
df_val_scaled.drop("smoking",axis =1 ,inplace =True )
```

Bagging

```
# Set the number of trees in the ensemble
# e7na dlw2t sh8alen b df_val_scaled w df_z_scaled
best_params = {}
best_accuracy = -1

param_ranges = {
        'n_trees': [2,4,8,16,32,64,128]
        }


# num_trees = 64
# for _ in range(n_iter):
#         params = {param: np.random.choice(values) for param, values
in param_ranges.items()}
for num in param_ranges['n_trees']:
    num_trees = num
# Create an array to store individual decision trees
    trees = []
    # y_train = df_z_scaled['smoking']
    # df_z_scaled.drop('smoking', axis = 1, inplace=True)
    #print(df_z_scaled.columns)
    #print(df_val_scaled.columns)
    for _ in range(num_trees):
        # Bootstrap sampling: randomly sample with replacement
        indices = np.random.choice(len(df_z_scaled),
size=len(df_z_scaled), replace=True)
        X_bootstrapped, y_bootstrapped = df_z_scaled.iloc[indices],
y_train.iloc[indices]

        # Train a decision tree on the bootstrapped dataset
        tree = DecisionTreeClassifier()
        tree.fit(X_bootstrapped, y_bootstrapped)

        # Add the trained tree to the ensemble
        trees.append(tree)

    # Make predictions on the test set and aggregate the results
    # Y_validation = df_val_scaled['smoking']
    # df_val_scaled.drop("smoking",axis =1 ,inplace =True )
    predictions = np.array([tree.predict(df_val_scaled) for tree in
trees])
    ensemble_predictions = np.median(predictions, axis=0)  # You can
use np.median() for classification

    # Convert predictions to integer values for classification
    ensemble_predictions = np.round(ensemble_predictions).astype(int)
#3shan lw 3dd even median hytl3 0.5
                                                                    #
```

```
round will predict 1
    # Calculate accuracy
    accuracy = accuracy_score(Y_validation, ensemble_predictions)
#vaid zy test 15%
    print("Ensemble Accuracy:", accuracy)
    if(accuracy > best_accuracy):
        best_accuracy =accuracy
        best_params['n_trees'] = num

print(best_params)
print(best_accuracy)
Bagging_best_parm=best_params
```

```
Ensemble Accuracy: 0.6321319435723555
Ensemble Accuracy: 0.6616015739461677
Ensemble Accuracy: 0.6892293524216166
Ensemble Accuracy: 0.7107873916865503
Ensemble Accuracy: 0.7250198836284483
Ensemble Accuracy: 0.7300012558081125
Ensemble Accuracy: 0.7305035790531207
{'n_trees': 128}
0.7305035790531207
```

Boosting

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

class AdaBoost:
    def __init__(self, n_estimators=50):
        self.n_estimators = n_estimators
        self.alphas = []
        self.models = []

    def fit(self, X, y):
        m, n = X.shape # m " number of samples"
        # Initialize weights for data points
        w = np.ones(m) / m
        for _ in range(self.n_estimators):
            # Create a weak learner (decision tree)
            model = DecisionTreeClassifier()
            # Fit the weak learner to the data with weighted samples
            model.fit(X, y, sample_weight=w) #deaful all weight are
equal
            # Make predictions
            predictions = model.predict(X)
            # Calculate weighted error
            weighted_error = np.sum(w * (predictions != y))
            # Calculate alpha (weight of the weak learner)
```

```
            alpha = 0.5 * np.log((1 - weighted_error) /
max(weighted_error, 1e-10))
            # Update weights
            #print(predictions)
            w *= np.exp(-alpha * y * predictions)# if true  --> mul *-
alpha else mul alpha
            w /= np.sum(w)
            # Save alpha and the weak learner
            self.alphas.append(alpha)
            self.models.append(model)

    def predict(self, X):
        # Make predictions using the weighted sum of weak learners
        weighted_sum = sum(alpha * model.predict(X) for alpha, model
in zip(self.alphas, self.models))#
        # Apply sign function to get final predictions
        predictions = np.sign(weighted_sum)# if >0 -- > 1 ,,,,,  asghr
htrg3 -1 ,,,, 0 htrg3 0
        #print(predictions," pr")
        return predictions.astype(int)
```

grid search to ada boost

```
best_params = {}
best_accuracy = -1

param_ranges = {
        'n_trees': [2,4,8,16,32,64,128],
    }

for num in param_ranges['n_trees']:
    num_trees = num

    boosting_model = AdaBoost(n_estimators=num_trees)
    #print(X_train.shape)
    boosting_model.fit(df_z_scaled, y_train.replace(0,-1))

    # Make predictions on the training data
    predictions_train = boosting_model.predict(df_val_scaled)

    # Calculate accuracy on the training data
    accuracy_train = accuracy_score(Y_validation.replace(0,-1),
predictions_train)
    print("validation Accuracy:", accuracy_train)
    if(accuracy_train > best_accuracy) :
        best_bossting_parm = num
        best_accuracy = accuracy_train
        best_params['n_trees'] = num
        print(best_accuracy)
```

```
print("Best acc in boosting is : \n ")
print(best_accuracy)
```

```
validation Accuracy: 0.6172715475741973
0.6172715475741973
validation Accuracy: 0.6351040227719871
0.6351040227719871
validation Accuracy: 0.6389970279208004
0.6389970279208004
validation Accuracy: 0.6410900414416677
0.6410900414416677
validation Accuracy: 0.6435597973962912
0.6435597973962912
validation Accuracy: 0.6430156138808657
validation Accuracy: 0.6367365733182636
Best acc in boosting is :

0.6435597973962912
```

random forest

```
import numpy as np
from collections import Counter
import random
import math


# Calculate Entropy
def entropy(y):
    hist = np.bincount(y)
    ps = hist/len(y)
    return - np.sum([p * np.log2(p) for p in ps if p > 0])


# Create Node
class Node:

    def __init__(self, feature=None, threshold=None, left=None,
right=None, *, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf_node(self):
        return self.value is not None


#Decision Tree
```

```
class DecissionTree:
    import numpy as np
    def __init__(self, min_samples_split=2, max_depth=100,
n_feats=None, max_features='auto'):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.n_feats = n_feats
        self.root = None
        self.max_features = max_features

    def fit(self, X, y):
        self.n_feats = X.shape[1] if not self.n_feats else
min(self.n_feats, X.shape[1])
        self.cols = list(X.columns)
        self.root = self.grow_tree(X, y)

    def grow_tree(self, X, y, depth=0):

        df = X.copy()
        df['smoking'] = y

        n_samples, n_features = X.shape
        n_labels = len(np.unique(y))

        # stopping criteria
        if (depth >= self.max_depth or n_labels == 1 or n_samples <
self.min_samples_split):
            leaf_value = self.most_common_label(y)
            return Node(value=leaf_value)

        # array of random columns in Dataset

        data = self.feature_sampling(X, self.max_features)

        feats_idxs = list(data.columns)

        best_feat, best_thresh = self.best_criteria(X, y.tolist(),
feats_idxs)

        left_df, right_df = df[df[best_feat]<=best_thresh].copy(),
df[df[best_feat]>best_thresh].copy()

        left = self.grow_tree(left_df.drop('smoking', axis=1),
left_df['smoking'].values, depth+1)
        right = self.grow_tree(right_df.drop('smoking', axis=1),
right_df['smoking'].values, depth+1)

        return Node(best_feat, best_thresh, left, right)
```

```
    def best_criteria(self, X, y, feats_idxs):
        import numpy as np

        best_gain = -1
        split_idx, split_tresh = None, None

        X = X.to_numpy()

        for feats_idx in feats_idxs:

            index = int(self.cols.index(feats_idx))

            df = pd.DataFrame(X[:, index], columns=['X_col'])
            df['y'] = y
            df = df.sort_values(by=['X_col'], ascending=True)

            X_col_2 = df.X_col
            y_2 = df.y

            X_col_2 = X_col_2.to_numpy()
            y_2 = y_2.to_numpy()

            for val in X_col_2:
                gain = self.information_gain(y_2, X_col_2, val)
                if gain > best_gain:
                    best_gain = gain
                    split_idx = feats_idx
                    split_tresh = val

        return split_idx, split_tresh

    def information_gain(self, y, X_col, thresh):
        import numpy as np

        parent_entropy = entropy(y)

        left, right = self.split(X_col, thresh)

        if len(left) == 0 or len(right) == 0:
            return 0

        n = len(y)
        n_l, n_r = len(left), len(right)
        e_l, e_r = entropy(y[left]), entropy(y[right])

        child_entropy = (n_l / n) * e_l + (n_r / n) * e_r

        ig = parent_entropy - child_entropy
        return ig
```

```
    def split(self, X_col, split_tresh):

        left_idxs = np.argwhere(X_col <= split_tresh).flatten()
        right_idxs = np.argwhere(X_col > split_tresh).flatten()

        return left_idxs, right_idxs

    def most_common_label(self, y):
        counter = Counter(y)
        most_common = counter.most_common(1)[0][0]
        return most_common

    def predict(self, X):
        import numpy as np

        X = X.to_numpy().tolist()
        return np.array([self.traverse_tree(x, self.root) for x in X])

    def traverse_tree(self, x, node):
        if node.is_leaf_node():
            return node.value

        index = int(self.cols.index(node.feature))

        if x[index] <= node.threshold:
            return self.traverse_tree(x, node.left)

        return self.traverse_tree(x, node.right)

    def feature_sampling(self, data, val):
        if type(val) == int:
            col = random.sample(data.columns.tolist()[:], val)
            new_df = data[col]
            return new_df
        elif type(val) == float:
            col = random.sample(data.columns.tolist()[:], int(val *
data.shape[1]))
            new_df = data[col]
            return new_df
        elif val == 'auto' or val == 'sqrt':
            col = random.sample(data.columns.tolist()[:],
int(math.sqrt(data.shape[1])))
            new_df = data[col]
            return new_df
        elif val == 'log2':
            col = random.sample(data.columns.tolist()[:],
int(math.log2(data.shape[1])))
            new_df = data[col]
            return new_df
        else:
```

```
            return data



class randomforestclassifier:
    def __init__(self, n_estimators=100, criterion='entropy',
max_depth=None, min_samples_split=2, bootstrap=True, max_samples=None,
                 max_features='auto'):
        self.n_estimators = n_estimators
        self.criterion = criterion
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.bootstrap = bootstrap
        self.max_samples = max_samples
        self.max_features = max_features


    def fit(self, X_train, y_train):
        dummy_data = X_train.copy()
        dummy_data['smoking'] = y_train

        self.tree_list = []
        print(dummy_data.columns)
        for i in range(self.n_estimators):

            if self.bootstrap == True:
                df = self.row_sampling(dummy_data, self.max_samples)
            else:
                df = dummy_data.copy()
            # print(df.columns)
            print(type(df))
            tree = DecissionTree(max_depth=self.max_depth,
min_samples_split=self.min_samples_split,
max_features=self.max_features)

            tree.fit(df.drop('smoking', axis=1), df['smoking'])



            self.tree_list.append(tree)


    def predict(self, X_test):
        y_preds = np.empty((X_test.shape[0], len(self.tree_list)))
        # Let each tree make a prediction on the data
        for i, tree in enumerate(self.tree_list):
            # Indices of the features that the tree has trained on
            # idx = tree.feature_indices
            # Make a prediction based on those features
            prediction = tree.predict(X_test)
```

```
            y_preds[:, i] = prediction

        y_pred = []
        # For each sample
        for sample_predictions in y_preds:
            # Select the most common class prediction

y_pred.append(np.bincount(sample_predictions.astype('int')).argmax())
        return y_pred

    def score(self, y_true=None, y_pred=None):
        acc = np.sum(y_true == y_pred)/len(y_true)
        return acc

    def row_sampling(self, data, val):
        new_df = data.sample(n=val, random_state=42)
        return new_df
```

random forest hypertuning

```
def random_search(x_t, y_t ,x_val ,y_val ,  n_iter):
        best_params = {}
        best_accuracy = -1

        param_ranges = {
            'n_trees': [4, 10, 16, 32, 64],
            'n_bootstrap': [1024,2048,4096,8000],
            'n_features':[2, 3, 4, 5],
            'dt_max_depth': [2, 3, 7, 9],
            'min_sample_split' : [2, 3, 5]
        }

        for _ in range(n_iter):
            params = {param: np.random.choice(values) for param,
values in param_ranges.items()}
            n_estimators = params['n_trees']
            max_samples = params['n_bootstrap']
            max_features = params['n_features']
            max_depth = params['dt_max_depth']
            min_sample_split = params['min_sample_split']
            # print(n_trees)
            # Call the random forest algorithm
            forest =
randomforestclassifier( n_estimators=n_estimators, max_depth=
max_depth,
                min_samples_split= min_sample_split, max_samples=
max_samples,
                max_features= max_features
            )
```

```
            forest.fit(x_t,y_t)
            # Make predictions on validation set
            predictions = forest.predict(x_val)

            # Calculate accuracy
            accuracy = forest.score(y_val, predictions)

            if accuracy > best_accuracy:
                best_accuracy = accuracy
                print(best_accuracy)
                best_params = params.copy()

        return best_params, best_accuracy


best_params, best_accuracy = random_search(df_z_scaled,y_train,
df_val_scaled,y_val, 10)
print(f' the best parameters so far are : {best_params}')
print(f'accuracy is: {best_accuracy}')
best_pram_of_randForest=best_params
```

```
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
0.6714387374942442
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
0.7001548830005442
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
 the best parameters so far are : {'n_trees': 10, 'n_bootstrap': 8000,
'n_features': 5, 'dt_max_depth': 3, 'min_sample_split': 2}
accuracy is: 0.7001548830005442
```

Normalize test data and test on all models

```
tv = y_test
col = X_test

scaler = StandardScaler() # - mean / sd
df_test_scaled = pd.DataFrame(scaler.fit_transform(col),
columns=col.columns)
df_test_scaled.reset_index(drop=True, inplace=True)
tv.reset_index(drop=True, inplace=True)
#  now we remove the outliers

data_val = trans.fit_transform(np.array(df_test_scaled))
feature_names = df_test_scaled.columns  # Original feature names

# Use get_feature_names_out method to get feature names for the
polynomial features
poly_feature_names =
trans.get_feature_names_out(input_features=feature_names)

# Create a dictionary to map column indices to feature names
feature_index_mapping = {i: name for i, name in
enumerate(poly_feature_names)}

# Access the feature names for each column of the transformed array
column_headers = [feature_index_mapping[i] for i in
range(df_test_scaled.shape[1])]


df_test_scaled = pd.DataFrame(df_test_scaled)
df_test_scaled.columns = column_headers

df_test_scaled['smoking'] = tv
```

```
print(best_params)
```

```
{'n_trees': 10, 'n_bootstrap': 8000, 'n_features': 5, 'dt_max_depth':
3, 'min_sample_split': 2}
```

test_set on bagging

```
df_test_scaled.drop('smoking',axis=1,inplace=True)
print(df_test_scaled.columns)
```

```
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)'],
      dtype='object')
```

```
trees = []
for _ in range(Bagging_best_parm['n_trees']):
    # Bootstrap sampling: randomly sample with replacement
    indices = np.random.choice(len(df_z_scaled),
size=len(df_z_scaled), replace=True)
    X_bootstrapped, y_bootstrapped = df_z_scaled.iloc[indices],
y_train.iloc[indices]

    # Train a decision tree on the bootstrapped dataset
    tree = DecisionTreeClassifier()
    tree.fit(X_bootstrapped, y_bootstrapped)

    # Add the trained tree to the ensemble
    trees.append(tree)

# Make predictions on the test set and aggregate the results
# Y_validation = df_val_scaled['smoking']
# df_val_scaled.drop("smoking",axis =1 ,inplace =True )
predictions = np.array([tree.predict(df_test_scaled) for tree in
trees])
ensemble_predictions = np.median(predictions, axis=0)  # You can use
np.median() for classification

# Convert predictions to integer values for classification
ensemble_predictions = np.round(ensemble_predictions).astype(int)
#3shan lw 3dd even median hytl3 0.5
                                                                    #
round will predict 1
# Calculate accuracy
accuracy = accuracy_score(y_test, ensemble_predictions) #vaid zy test
15%
print("Ensemble test bagging accuracy:", accuracy)
```

```
Ensemble test bagging accuracy: 0.7293620227729404
```

Test set on best parm of Bossting

```
boosting_model = AdaBoost(n_estimators=best_bossting_parm)
#print(X_train.shape)
boosting_model.fit(df_z_scaled, y_train.replace(0,-1))

# Make predictions on the training data
predictions_train = boosting_model.predict(df_test_scaled)

# Calculate accuracy on the training data
accuracy_train = accuracy_score(y_test.replace(0,-1),
predictions_train)
print("validation boost Accuracy on test :", accuracy_train)
```

```
validation boost Accuracy on test : 0.6386470194239786
```

Test set on best parm of rand forest

```
forest =
randomforestclassifier( n_estimators=best_pram_of_randForest['n_trees'
],
                                    max_depth=
best_pram_of_randForest['dt_max_depth'],
    min_samples_split= best_pram_of_randForest['min_sample_split'],
    max_samples= best_pram_of_randForest['n_bootstrap'],
    max_features= best_pram_of_randForest['n_features']
)

forest.fit(df_z_scaled,y_train)
# Make predictions on validation set
predictions = forest.predict(df_test_scaled)

# Calculate accuracy
accuracy = forest.score(y_test, predictions)

print("accuracy of random forest on test set",accuracy)
```

```
Index(['LDL triglyceride', 'weight(kg) triglyceride', 'systolic
triglyceride',
       'Cholesterol Gtp', 'Cholesterol triglyceride', 'Gtp^2',
       'Gtp triglyceride', 'Gtp height(cm)', 'triglyceride^2',
       'triglyceride height(cm)', 'smoking'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
accuracy of random forest on test set 0.7017330877427997
```