APPENDIX

GET DATA

```
DATA = '/content/gdrive/My Drive/PROJECT 19.K1.0005 - DAVIN CHANG/Dataset/processed.cleveland.data'
DATA_NAMES = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']

df = pd.read_csv(DATA, sep=',', header=None, names=DATA_NAMES)
```

REPLACING 1-4 TARGET HEART DISEASE PRESENCE TO 1 AS TRUE

```
df['target'] = df['target'].replace({
                    2 : 1,
                    3 : 1,
                    4 : 1,
                })
```

DROPPING MISSING VALUES

```
check = df.isin(['?'])
selectedRow = df[check.any(axis=1)]
df_dropping = df.drop(selectedRow.index)
```

REPLACING MISSING VALUES WITH MEDIAN

```
df_median = df.replace({
          '?' : np.nan
     })

df_median['ca'] = df_median['ca'].fillna(df_median['ca'].median())
df_median['thal'] = df_median['thal'].fillna(df_median['thal'].median())
```

REPLACING MISSING VALUES WITH MODE

```
df_modus = df.replace({
          '?' : np.nan
     })

df_modus['ca'] = df_modus['ca'].fillna(df_modus['ca'].mode()[0])
df_modus['thal'] = df_modus['thal'].fillna(df_modus['thal'].mode()[0])
```

EVALUATE MODEL FUNCTION

```
1. def evaluate_model(model, cv, X, y):
2.
3.     scores_precision     =     cross_val_score(model,     X,     y,
   scoring='precision', cv=cv)
4.     scores_recall      =      cross_val_score(model,      X,      y,
   scoring='recall', cv=cv)
5.     scores_f1 = cross_val_score(model, X, y, scoring='f1', cv=cv)
6.     scores_roc_auc     =     cross_val_score(model,     X,     y,
   scoring='roc_auc', cv=cv)
7.     scores_accuracy     =     cross_val_score(model,     X,     y,
   scoring='accuracy', cv=cv)
8.
9.     return    np.mean(scores_precision),    np.mean(scores_recall),
   np.mean(scores_f1),                          np.mean(scores_roc_auc),
   np.mean(scores_accuracy)
```

10 FOLD CROSS VALIDATION FUNCTION

```
1. def do_10Fold(X, y, specific_metrics=None):
2.     model = XGBClassifier(
3.         seed = seed
4.     )
5.     cv = KFold(n_splits=10, shuffle=True, random_state=seed)
6.     precision_mean,     recall_mean,     f1_mean,     roc_auc_mean,
   accuracy_mean = evaluate_model(model, cv, X, y)
7.
8.     if(specific_metrics == "accuracy"):
9.         return accuracy_mean
10.         else :
11.             print('>   folds=10,   precision=%.4f,   recall=%.4f,
   f1=%.4f,   roc   area=%.4f,   accuracy=%.4f'   %   (precision_mean,
   recall_mean, f1_mean, roc_auc_mean, accuracy_mean))
```

MAKE BOXPLOT FUNCTION

```
1. def make_boxplot(data, text):
2.     plt.boxplot(data,
3.         vert=True,
4.         labels=[text])
5.
6.     plt.title(text)
7.     plt.ylabel("Value")
8.     plt.grid(axis = 'y')
9.     plt.show()
```

REMOVE OUTLIERS FUNCTION

```
1. def remove_outliers(feature):
2.     Q1 = np.percentile(X_outliers[feature], 25,
3.                        interpolation = 'midpoint')
4.
5.     Q3 = np.percentile(X_outliers[feature], 75,
6.                        interpolation = 'midpoint')
7.     IQR = Q3 - Q1
```

```
8.
9.      upper_value = (Q3+1.5*IQR)
10.         lower_value = (Q1-1.5*IQR)
11.         upper_len     =     len(np.where(X_outliers[feature]     >=
    upper_value)[0])
12.         lower_len     =     len(np.where(X_outliers[feature]     <=
    lower_value)[0])
13.
14.         for _ in range(upper_len):
15.             upper = np.where(X_outliers[feature] >= upper_value)
16.             X_outliers.drop(upper[0], inplace=True)
17.             Y_outliers.drop(upper[0], inplace=True)
18.             X_outliers.reset_index(drop=True, inplace=True)
19.             Y_outliers.reset_index(drop=True, inplace=True)
20.
21.         for _ in range(lower_len):
22.             lower = np.where(X_outliers[feature] <= lower_value)
23.             X_outliers.drop(lower[0], inplace=True)
24.             Y_outliers.drop(lower[0], inplace=True)
25.             X_outliers.reset_index(drop=True, inplace=True)
26.             Y_outliers.reset_index(drop=True, inplace=True)
27.
28.         print("Upper : ", upper_value)
29.         print("Lower : ", lower_value)
```

XGBCLASSIFIER HYPERPARAMETER EVALUATION FUNCTION

```
1. def xgbc(learning_rate, n_estimators, max_depth, min_child_weight,
   gamma, subsample, colsample_bytree):
2.
3.      model = XGBClassifier(
4.          learning_rate = learning_rate,
5.          n_estimators = int(n_estimators),
6.          max_depth = int(max_depth),
7.          min_child_weight = min_child_weight,
8.          gamma = gamma,
9.          subsample = subsample,
10.             colsample_bytree = colsample_bytree,
11.             seed = seed,
12.         )
13.
14.         cv = KFold(n_splits=10, shuffle=True, random_state=seed)
15.         accuracy     =     cross_val_score(model,     X,     Y,
    scoring='accuracy', cv=cv)
16.
17.         return np.mean(accuracy)
```

HYPERPARAMETER TUNING USING BAYESIAN OPTIMIZATION

```
1. hyperparameter = {
2.      'learning_rate': (0.1, 1), # default 0.1
3.      'n_estimators' : (100, 250), # default 100
4.      'max_depth': (1, 15), # default 3
5.      'min_child_weight' : (0, 1), # default 1
6.      'gamma' : (0, 1), # default 0
7.      'subsample' : (0.4, 1), # default 1
8.      'colsample_bytree' : (0.4, 1), # default 1
```

```
9. }
10.
11.     xgbcBO = BayesianOptimization(
12.                 f = xgbc,
13.                 pbounds =  hyperparameter,
14.                 random_state = seed
15.             )
16.
17.     xgbcBO.maximize()    return np.mean(accuracy)
```

CHI SQUARE FEATURE SELECTION FUNCTION

```
1. def chi_square(model, X, Y):
2.     print("\nChi Square Feature Selection")
3.     for i in range(X.shape[1], 0, -1) :
4.         selection = SelectKBest(score_func=chi2, k=i)
5.         select_X = selection.fit_transform(X, Y)
6.         feature_idx = selection.get_support()
7.         X_selected = pd.DataFrame(select_X)
8.         X_selected.columns = X.columns[feature_idx]
9.
10.             if(i == 13):
11.                 feature_selection_index                    =
   np.argsort(selection.scores_)[::-1][0:i]
12.                 best_feature_sorted                        =
   df.columns[feature_selection_index].values
13.                 print(best_feature_sorted)
14.
15.             print("n =", i, end=', ')
16.             do_10Fold(model, X_selected, Y)
17.             print(X_selected.columns)
18.             print()
```

MUTUAL INFORMATION FEATURE SELECTION FUNCTION

```
1. def mutual_information(model, X, Y):
2.     print("\nMutual Information Feature Selection")
3.     for i in range(X.shape[1], 0, -1) :
4.         selection   =   SelectKBest(score_func=mutual_info_classif,
   k=i)
5.         select_X = selection.fit_transform(X, Y)
6.         feature_idx = selection.get_support()
7.         X_selected = pd.DataFrame(select_X)
8.         X_selected.columns = X.columns[feature_idx]
9.
10.             if(i == 13):
11.                 feature_selection_index                    =
   np.argsort(selection.scores_)[::-1][0:i]
12.                 best_feature_sorted                        =
   df.columns[feature_selection_index].values
13.                 print(best_feature_sorted)
14.
15.             print("n =", i, end=', ')
16.             do_10Fold(model, X_selected, Y)
17.             print(X_selected.columns)
18.             print()
```

ANOVA FEATURE SELECTION FUNCTION

```
def anova(model, X, Y):
    print("\nAnova Feature Selection")
    for i in range(X.shape[1], 0, -1) :
        selection = SelectKBest(score_func=f_classif, k=i)
        select_X = selection.fit_transform(X, Y)
        feature_idx = selection.get_support()
        X_selected = pd.DataFrame(select_X)
        X_selected.columns = X.columns[feature_idx]

            if(i == 13):
                feature_selection_index                                    =
  np.argsort(selection.scores_)[::-1][0:i]
                best_feature_sorted                                        =
  df.columns[feature_selection_index].values
                print(best_feature_sorted)

            print("n =", i, end=', ')
            do_10Fold(model, X_selected, Y)
            print(X_selected.columns)
            print()
```

FORWARD FEATURE SELECTION FUNCTION

```
def forward_feature_selection(model, X, Y):
    cv = KFold(n_splits=10, shuffle=True, random_state=seed)
    print("\nForward Feature Selection")
    for i in range(X.shape[1]-1, 0, -1) :
        selection       =       SequentialFeatureSelector(model,
  n_features_to_select=i, cv=cv)
        select_X = selection.fit_transform(X, Y)
        feature_idx = selection.get_support()
        X_selected = pd.DataFrame(select_X)
        X_selected.columns = X.columns[feature_idx]

            print("n =", i, end=', ')
            do_10Fold(model, X_selected, Y)
            print(X_selected.columns)
            print()
```

BACKWARD FEATURE SELECTION FUNCTION

```
def backward_feature_selection(model, X, Y):
    cv = KFold(n_splits=10, shuffle=True, random_state=seed)
    print("\nBackward Feature Selection")
    for i in range(X.shape[1]-1, 0, -1) :
        selection       =       SequentialFeatureSelector(model,
  n_features_to_select=i, direction="backward", cv=cv)
        select_X = selection.fit_transform(X, Y)
        feature_idx = selection.get_support()
        X_selected = pd.DataFrame(select_X)
        X_selected.columns = X.columns[feature_idx]

            print("n =", i, end=', ')
            do_10Fold(model, X_selected, Y)
```

```
            print(X_selected.columns)
            print()
```

RECURSIVE FEATURE ELIMINATION FUNCTION

```
def recursive_feature_elimination(model, X, Y):
    print("\nRecursive Feature Elimination")
    for i in range(X.shape[1], 0, -1) :
        selection = RFE(model, n_features_to_select=i)
        select_X = selection.fit_transform(X, Y)
        feature_idx = selection.get_support()
        X_selected = pd.DataFrame(select_X)
        X_selected.columns = X.columns[feature_idx]

            print("n =", i, end=', ')
            do_10Fold(model, X_selected, Y)
            print(X_selected.columns)
            print()
```

FEATURE IMPORTANCE FEATURE SELECTION FUNCTION

```
def feature_importance(model, X, Y):
    model.fit(X, Y)
    thresholds = np.sort(model.feature_importances_)

    print("\nFeature Importance Feature Selection")
    for thresh in thresholds:
        selection = SelectFromModel(model, threshold=thresh,
prefit=True)
        feature_idx = selection.get_support()
        select_X = selection.transform(X.values)

            X_selected = pd.DataFrame(select_X)
            X_selected.columns = X.columns[feature_idx]

            print("n =", select_X.shape[1], end=', ')
            do_10Fold(model, X_selected, Y)
            print(X_selected.columns)
            print()
```

PLOT BAR HORIZONTAL MODEL FEATURE IMPORTANCE

```
thresholds = np.sort(model.feature_importances_)
plt.barh(X.columns, model.feature_importances_)
plt.show()
```

Similarity Report

PAPER NAME

TA-19.K1.0005.docx

WORD COUNT	CHARACTER COUNT
9322 Words	**50073 Characters**
PAGE COUNT	FILE SIZE
37 Pages	**88.5KB**
SUBMISSION DATE	REPORT DATE
Dec 16, 2022 3:48 PM GMT+7	**Dec 16, 2022 3:48 PM GMT+7**

11% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 5% Internet database
- Crossref database
- 9% Submitted Works database
- 6% Publications database
- Crossref Posted Content database

Summary