Predicting 10 year risk of Coronary Heart Disease

Introduction/Objective

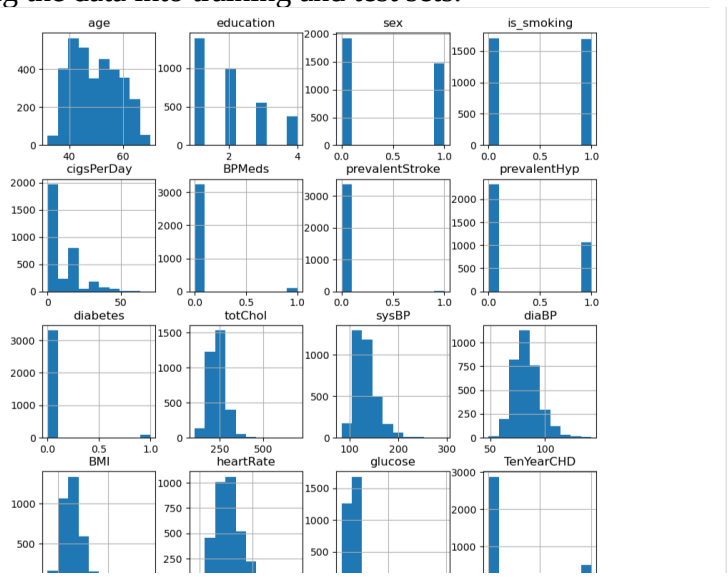
The main objective of this analysis is to develop classifiers model focused on prediction and compare them to see which model is the best. The analysis aims to provide accurate predictions of the 10-year risk of future coronary heart disease (CHD) for patients based on their demographic, behavioral, and medical attributes. The business or stakeholders of this data, such as healthcare providers or researchers, can benefit from the analysis by identifying individuals at high risk of CHD and implementing preventive measures or personalized interventions.

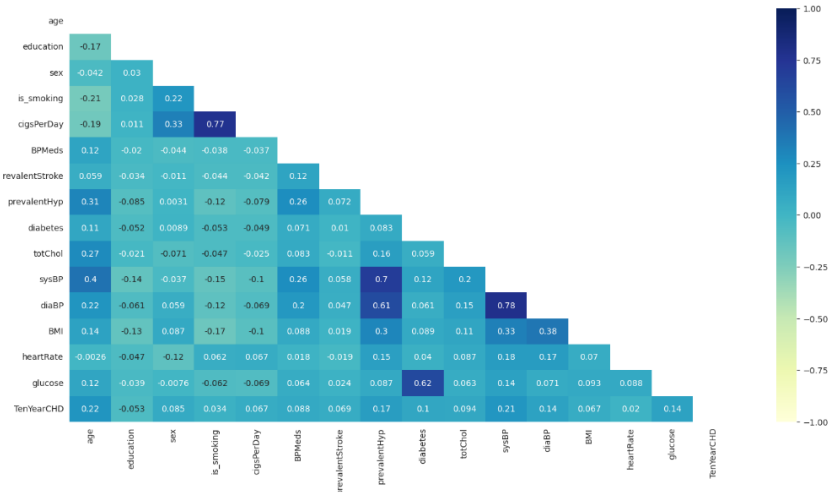
Background

The chosen dataset is the cardiovascular risk factor data from Kaggle(cardiovascular-risk-factor-data | Kaggle), which includes information on over 4,000 patients and 15 attributes representing potential risk factors for CHD. The attributes include demographic factors (e.g., sex, age, education), behavioral factors (e.g., smoking status, number of cigarettes per day), and medical history and current measurements (e.g., blood pressure, cholesterol levels, BMI).

EDA

The data exploration phase involved analyzing the distribution of variables, checking for missing values, and handling categorical variables through one-hot encoding or label encoding as appropriate. For data visualization I included a pie chart, histogram for each of the features, as well as a heat map to see the correlation between variables. Data cleaning and preprocessing steps included removing duplicate entries, handling missing data, scaling numerical features, and splitting the data into training and test sets.





Three classifier models were trained and evaluated Decision Tree, Logistic Regression, and Support Vector Machine (SVM). The models were trained using the same training and test splits, and performance metrics such as accuracy, precision, recall, F1 score, and confusion matrix were calculated to assess their predictive capabilities. Cross-validation methods, such as stratified K-fold, and GridSearchCV were applied to ensure the quality of the results. SMOTE was also applied to help with imbalanced classes.

SMOTE SVM

In [33]:
```
sm = SMOTE(random_state=42)
X_train_smote, y_train_smote = sm.fit_resample(X_train, y_train)
y_train_smote.value_counts()
```

Out[33]:
```
0    1746
1    1746
Name: TenYearCHD, dtype: int64
```

In [34]:
```
smote = SVC(random_state=101).fit(X_train_smote,y_train_smote)
smote_preds = smote.predict(X_test)
```

In [35]:
```
print('Test Accuracy:' , accuracy_score(y_test, smote_preds))
print('Test F1:' , f1_score(y_test, smote_preds, average='weighted'))
print('Test Recall:' , recall_score(y_test, smote_preds, average='weighted'))
```

```
Test Accuracy: 0.6598407281001137
Test F1: 0.7031490003316345
Test Recall: 0.6598407281001137
```

In [36]:
```
cm = confusion_matrix(y_test, smote_preds)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[490 247]
 [ 52  90]]
```

For Support Vector Machine, I tuned regularization parameters with cross validation technique GridCV. Prior to this I applied smote as my classes were unbalanced. The test accuracy was 0.65.

Logisitic Regression GridSearchCV

In [105]:
```
X = newdf.drop('TenYearCHD', axis = 1)
y = newdf['TenYearCHD']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

In [240]:
```
lr = LogisticRegression(multi_class='multinomial', random_state = 42)
accuracies = cross_val_score(lr,X_train,y_train,cv=10)
lr.fit(X_train,y_train)
logreg_preds = lr.predict(X_test)
print("Train Score:", np.mean(accuracies))
print("Test Score:", lr.score(X_test,y_test))
```

```
Train Score: 0.849634478996181
Test Score: 0.8481228668941979
```

In [107]:
```
grid ={'penalty':['l1', 'l2'],
            'C': [0.1,1,10]
              }
estimator_cv = LogisticRegression(multi_class='multinomial', random_state = 42)

LR = GridSearchCV(estimator=estimator_cv, param_grid=grid, cv=cv_method,verbose=1, n_jobs=-1)
LR.fit(X_train,y_train)

print("Best Parameters:",LR.best_params_)
print("Train Score:",LR.best_score_)
print("Test Score:",LR.score(X_test,y_test))
print("Best Estimator", LR.best_estimator_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best Parameters: {'C': 10, 'penalty': 'l2'}
Train Score: 0.8526829268292684
Test Score: 0.8486916951080774
Best Estimator LogisticRegression(C=10, multi_class='multinomial', random_state=42)
```

In [111]:
```
#Predicting Test Set
GSCV_LR_preds = LR.best_estimator_.predict(X_test)

print('Test Accuracy:' , accuracy_score(y_test, GSCV_LR_preds))
print('Test F1:' , f1_score(y_test, GSCV_LR_preds, average='weighted'))
print('Test Recall:' , recall_score(y_test, GSCV_LR_preds, average='weighted'))
```

```
Test Accuracy: 0.8486916951080774
Test F1: 0.7900908047326117
Test Recall: 0.8486916951080774
```

SMOTE Decison Tree

In [225]:
```
dt = DecisionTreeClassifier(random_state = 42)
dt.fit(X_train_smote, y_train_smote)
dt_preds = dt.predict(X_test)

dt_f1 = metrics.f1_score(y_test, dt_preds, average='weighted')
dt_acc = metrics.accuracy_score(y_test, dt_preds)
dt_recall = metrics.recall_score(y_test, dt_preds, average='weighted')

# Checking Accuracy, F1, and Recall scores
print('Test F1:' , dt_f1)
print('Test Accuracy:' , dt_acc)
print('Test Recall:' , dt_recall)
```

```
Test F1: 0.9191755285116006
Test Accuracy: 0.9180887372013652
Test Recall: 0.9180887372013652
```

In [226]:
```
cm = confusion_matrix(y_test, dt_preds)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[470  27]
 [ 21  68]]
```

In [227]:
```
dt.tree_.node_count, dt.tree_.max_depth
```

Out[227]: (787, 18)

In [228]:
```
def measure_error(y_true, y_pred, label):
    return pd.Series({'accuracy':accuracy_score(y_true, y_pred),
                      'precision': precision_score(y_true, y_pred),
                      'recall': recall_score(y_true, y_pred),
                      'f1': f1_score(y_true, y_pred)},
                      name=label)
```

In [229]:
```
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)

train_test_full_error = pd.concat([measure_error(y_train, y_train_pred, 'train'),
                                   measure_error(y_test, y_test_pred, 'test')],
                                   axis=1)

train_test_full_error
```

Out[229]:

	train	test
accuracy	0.913285	0.918089
precision	0.691919	0.715789
recall	0.771831	0.764045
f1	0.729694	0.739130

Based on the evaluation, the Decision Tree classifier is recommended as the final model that best fits the needs regarding accuracy and explainability. The Random Forest model yielded the highest accuracy, precision, recall and F1 score. I think this is due to their ensemble nature and ability to capture complex relationships in the data. The feature importance analysis provided by Decision Trees allows for some level of explainability, helping to understand the key factors of CHD risk prediction.

Key findings and insights derived from the classifier model revealed the significant predictors of CHD risk, were age, smoking status, blood pressure, cholesterol levels, and BMI. These insights can aid in risk stratification, early detection, and planning for individuals at high risk of CHD.

Some suggestions for next steps in analyzing this data include exploring additional feature engineering techniques, such as creating polynomial features, to capture potential nonlinear relationships in the data. Additionally, incorporating external data sources, such as genetic information or lifestyle factors, could enhance the predictive power and explanation of the model.