IE0005

Cardiovascular Disease Analysis

Team 2: Chen Yi Bin Jonathan, Gu Shiyuan, Mendos

Content

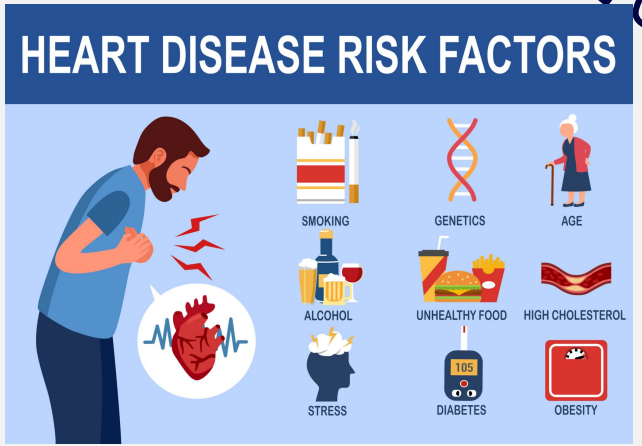
01

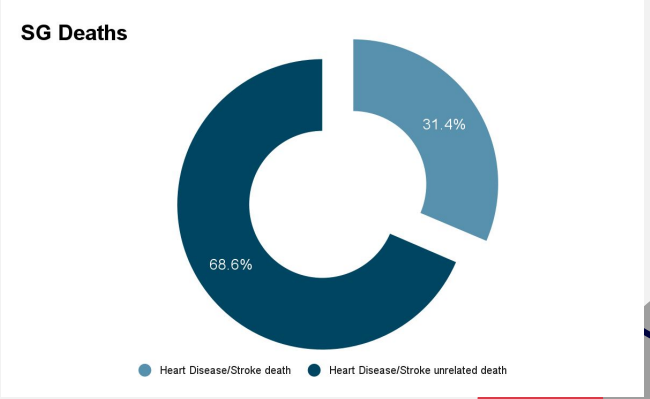
Introduction

Cleaning and structuring dataset

Background

- Group of disorders of the heart and blood vessels and include coronary heart disease, cerebrovascular disease, rheumatic heart disease and other conditions (WHO)
- Takes approximately 17,9 million lives annually around the globe
  - leading cause of death globally
- In singapore, nearly 1/3 of the deaths are due to heart disease or stroke alone





Objective

Our main objective for this dataset analysis is to determine the most indicative variable in predicting the presence of cardiovascular diseases (CVDs).

02

Methodology

Observations and findings from dataset

Methodology

Step 1

Data Preparation

Step 2

Exploratory Data Analysis

Step 3

Machine Learning

Step 1

Data Preparation

Data preprocessing

Cardiovascular Disease dataset

Consisting of **70,000** records of patients data
11 attributes of a person
NO Duplicates, **NO** missing values

The presence of **cardiovascular disease** is one of the attributes.

Data preprocessing

The published dataset has

- **12** features
- **3** types of data (Quantitative, Categorical, Binary)

Column name	Data description
age	Age (int)
height	Height (int)
weight	Weight (float)
gender	Gender (categorical code)
ap_hi	Systolic blood pressure (int)
ap_lo	Diastolic blood pressure (int)
cholesterol	Cholesterol (1:normal, 2: above normal, 3: well above normal)
gluc	Glucose (1:normal, 2: above normal, 3: well above normal)
smoke	Smoking (binary)
alco	Alcohol intake (binary)
active	Physical activity (binary)
cardio	Presence or absence of cardiovascular disease (binary)

Variables

Base on the dataset:

	Numerical	Categorical
1	Age	Gender
2	Weight	Cholesterol
3	Height	Glucose level
4	ap_hi	Smoke
5	ap_lo	Alcohol intake
6		Active

→ **convert** the "categorical" type to "category" data types

Observation on data quality

- **'Age'** - values are unrealistic
- **'Weight' and "Height"** - min/max values are unrealistic
- "**ap_hi**" and "**ap_lo**" - cannot be negative, max value is unrealistic

	count	mean	std	min	25%	50%	75%	max
age	70000.0	19468.865814	2467.251667	10798.0	17664.0	19703.0	21327.0	23713.0
height	70000.0	164.359229	8.210126	55.0	159.0	165.0	170.0	250.0
weight	70000.0	74.205690	14.395757	10.0	65.0	72.0	82.0	200.0
ap_hi	70000.0	128.817286	154.011419	-150.0	120.0	120.0	140.0	16020.0
ap_lo	70000.0	96.630414	188.472530	-70.0	80.0	80.0	90.0	11000.0
cardio	70000.0	0.499700	0.500003	0.0	0.0	0.0	1.0	1.0

Data cleaning

- **'Age'** - Needs recalculation

```
# Convert the age into years
CVdata['age'] = CVdata['age'] // 365.25
```

- **'Weight' and "Height"** - Needs filtering that fall below 2.5% or above 97.5%

```
# filter unrealistic weight and height data
CVdata.drop(CVdata[(CVdata['height'] > CVdata['height'].quantile(0.975)) |
                   (CVdata['height'] < CVdata['height'].quantile(0.025))].index,inplace=True)
CVdata.drop(CVdata[(CVdata['weight'] > CVdata['weight'].quantile(0.975)) |
                   (CVdata['weight'] < CVdata['weight'].quantile(0.025))].index,inplace=True)
```

For Blood Pressure Low (bp_lo) and Blood Pressure High (bp_hi)

- Outliers identified through boxplot (1373 anomaly data points (1.96%) removed)

```
CardioData_backup1 = CardioData_backup.copy()

CardioData_backup1
= CardioData_backup1.drop(CardioData_backup.loc[(ap_hi < 50) | (ap_hi > 180) | (ap_lo < 34) | (ap_lo > 120)].index)
fig, axes = plt.subplots(nrows=2, figsize=(10,10))
sb.boxplot(data=CardioData_backup1, x='ap_hi', ax=axes[0], orient='h')
sb.boxplot(data=CardioData_backup1, x='ap_lo', ax=axes[1], orient='h')
```

Feature engineering

- Used the weight and height data to create a new column “BMI” to carry out further analysis
- BMI = weight(kg)/height^2(m)

```
CVdata_v1 = CVdata.copy()

CVdata_v1['BMI'] = CVdata['weight']/((CVdata['height']/100)**2)
BMI = pd.DataFrame(CVdata_v1[['BMI']])
BMI.describe()
```

	BMI
count	58685.000000
mean	27.117707
std	4.119901
min	17.358919
25%	23.951227
50%	26.291724
75%	29.687500
max	43.282548

Convert Blood Pressure Low (bp_lo) and Blood Pressure High (bp_hi) from a numerical variable to a categorical one for easier analysis

Category	Systolic (mm Hg)	Diastolic (mm Hg)	Management
Dangerously low	≤50	≤33	A critical condition that requires emergency medical attention with IV fluids
Very low	≤60	≤40	Lifestyle modifications with medications
Low	Less than 90	Less than 60	Lifestyle modifications and regular checkups
Normal	Less than 120	Less than 80	Active lifestyle
Elevated	120-129	80 or more	Doctors may recommend lifestyle changes at this stage
Hypertension stage I	130-139	80-89	Doctors may prescribe blood pressure medications and some lifestyle changes to reduce the risk of heart disease and stroke.
Hypertension stage II	140-159	90-99	Doctors may prescribe a combination of medications and lifestyle changes; they may treat complications that may have increased due to high blood pressure.
Hypertensive crisis	180 or higher	120 or higher	A critical condition that requires emergency medical attention

- **6 categories** from *very low* to *High blood pressure (Hypertension) Stage 2*

```
# Rank Systolic blood pressure
CVdata_v1.loc[ap_hi <= 60, 'ap_hi'] = 1
CVdata_v1.loc[(ap_hi > 60) & (ap_hi < 90), 'ap_hi'] = 2
CVdata_v1.loc[(ap_hi >= 90) & (ap_hi < 120), 'ap_hi'] = 3
CVdata_v1.loc[(ap_hi >= 120) & (ap_hi <= 129), 'ap_hi'] = 4
CVdata_v1.loc[(ap_hi >= 130) & (ap_hi <= 139), 'ap_hi'] = 5
CVdata_v1.loc[ap_hi >= 140, 'ap_hi'] = 6

# Rank Diastolic blood pressure
CVdata_v1.loc[ap_lo <= 40, 'ap_lo'] = 1
CVdata_v1.loc[(ap_lo > 40) & (ap_lo < 60), 'ap_lo'] = 2
CVdata_v1.loc[(ap_lo >= 60) & (ap_lo < 80), 'ap_lo'] = 3
CVdata_v1.loc[(ap_lo >= 80) & (ap_lo <= 89), 'ap_lo'] = 5
CVdata_v1.loc[ap_lo >= 90, 'ap_lo'] = 6
```

- *Sort them into new column "**bp**"*

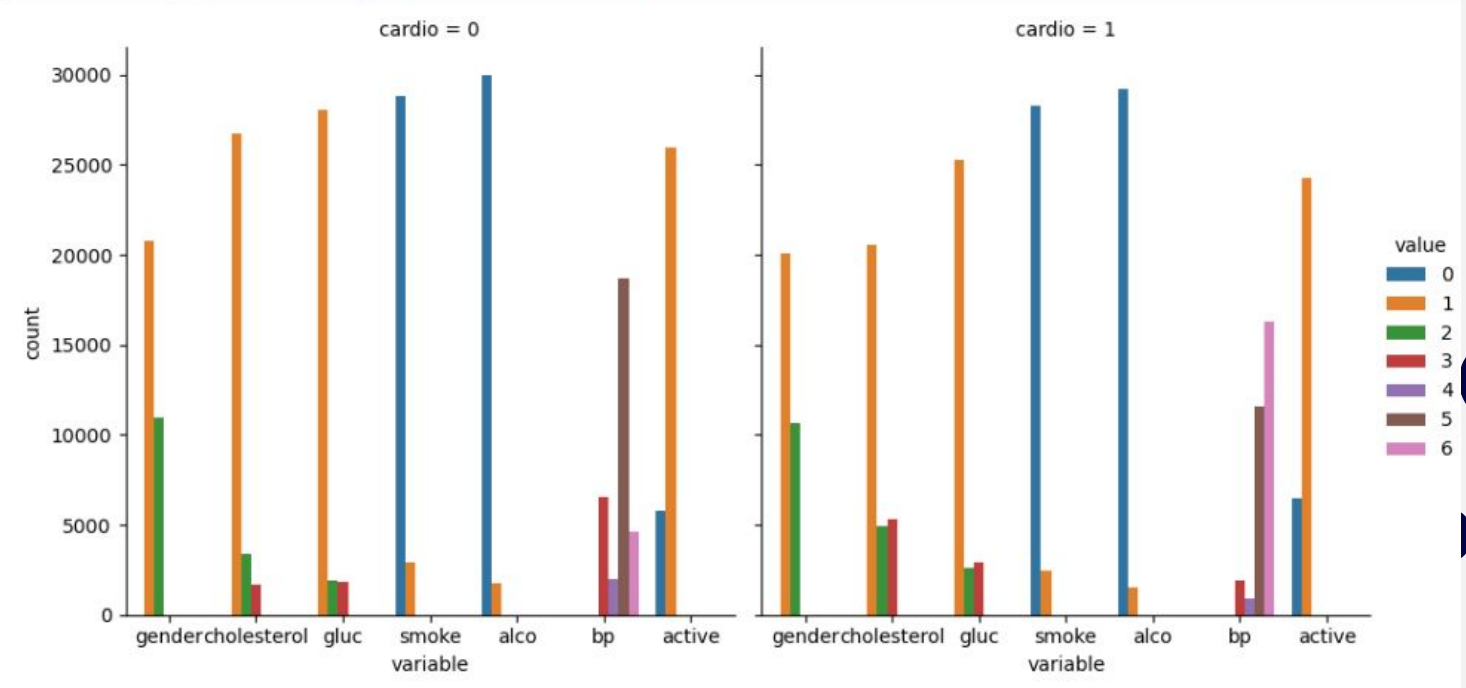
```
# Rank blood pressure and append it to the dataframe
bp = pd.DataFrame(CVdata_v1[['ap_hi', 'ap_lo']])
CVdata_v1['bp'] = bp.max(axis=1).astype('category')
```

Step 2

Exploratory Analysis

Categorical Analysis

```
#Categorical Analysis overview
CVdata_cat = pd.melt(CVdata_v1, id_vars=['cardio'],
                     value_vars=['gender', 'cholesterol', 'gluc', 'smoke', 'alco','bp', 'active'])
sb.catplot(x="variable", hue="value", col="cardio", data=CVdata_cat, kind="count");
```



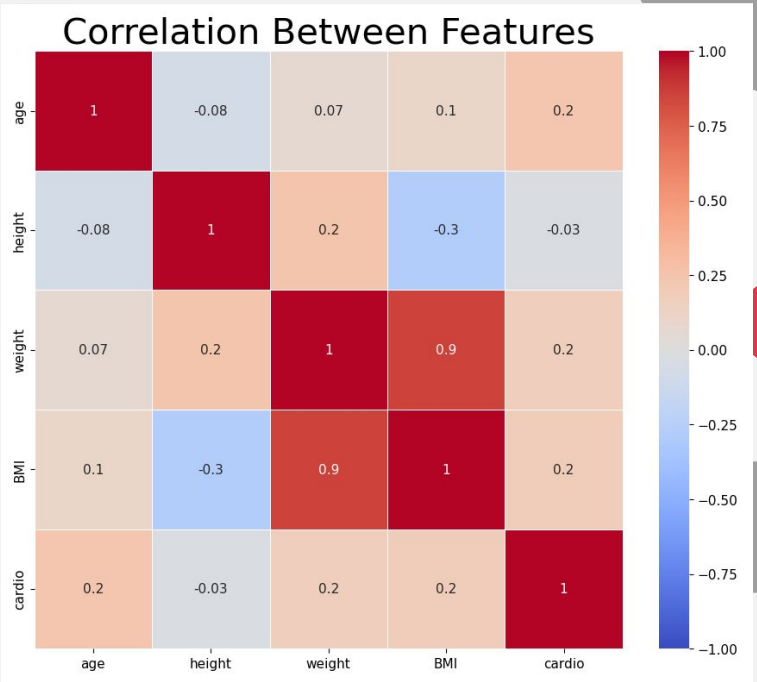
Observations

- Compared categorical data to cardio to see which contributes to CVDs the most
- It can be clearly seen that **patients with CVD have higher cholesterol and blood glucose level and are generally less active**
- **The relation between CVDs - lifestyle choices (smoking, alcohol consumption), gender** are **not obvious**
- **High relation between CVDs and blood pressure**

Correlation Between Numerical Features

```
#plotting correlation heatmap for 'age', 'height', 'weight', 'BMI', 'cardio'
plt.rcParams.update({'font.size': 11})
fig, ax = plt.subplots(figsize=(11, 9))
CVD_corr = ['age', 'height', 'weight', 'BMI', 'cardio']
sb.heatmap(CVD_clean[CVD_corr].corr(), annot = True, vmin=-1, vmax=1, center= 0,
           cmap= 'coolwarm', ax=ax, fmt='.1g', linewidths=.5);
plt.title('Correlation Between Features', fontsize = 30)
plt.show()
```

- BMI, Weight, Height and age doesn't have good correlation with cardio at glance



Step 3

Machine Learning

Selected Attributes for simple Decision tree

Predictor
BMI
bp
cholesterol
gluc
active

- **BMI** represents the attributes height and weight
- **bp** represents ap_low and ap_high

Simple Decision Tree

Since correlation may not offer the same results as a decision tree.

We decided to use a simple decision tree to check if it offers better results.

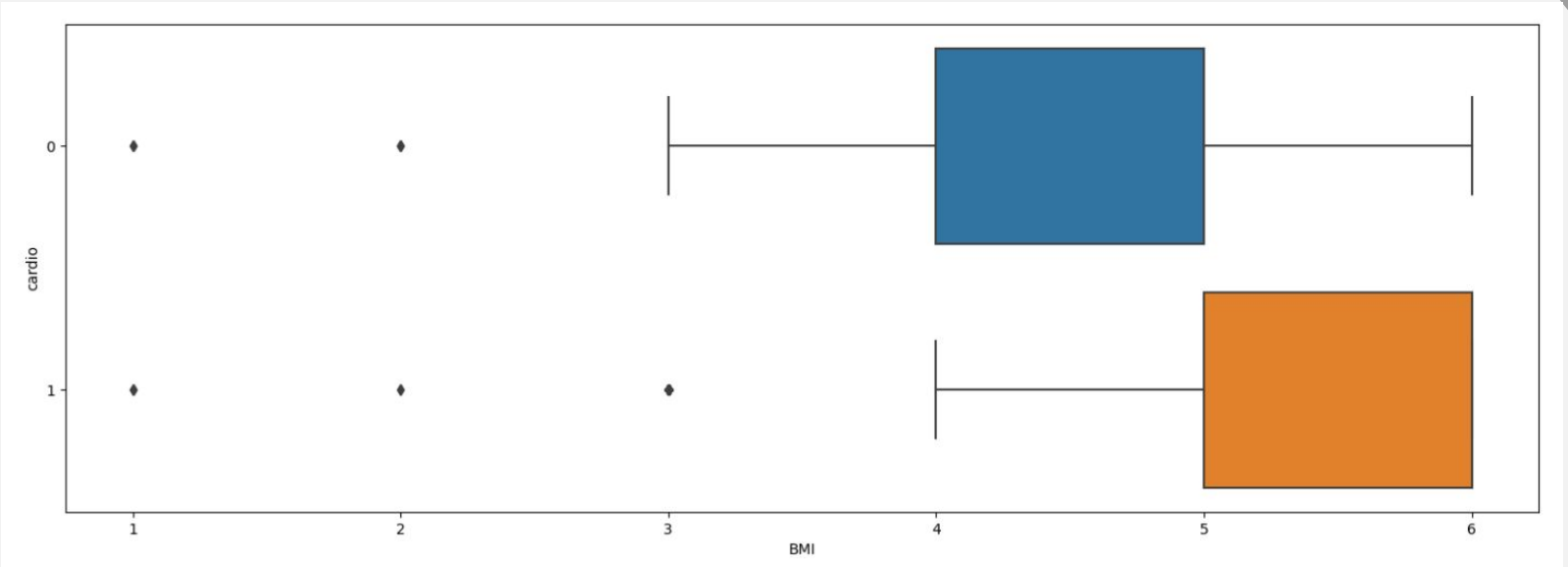
Response Variable : cardio

Predictor Feature : BMI

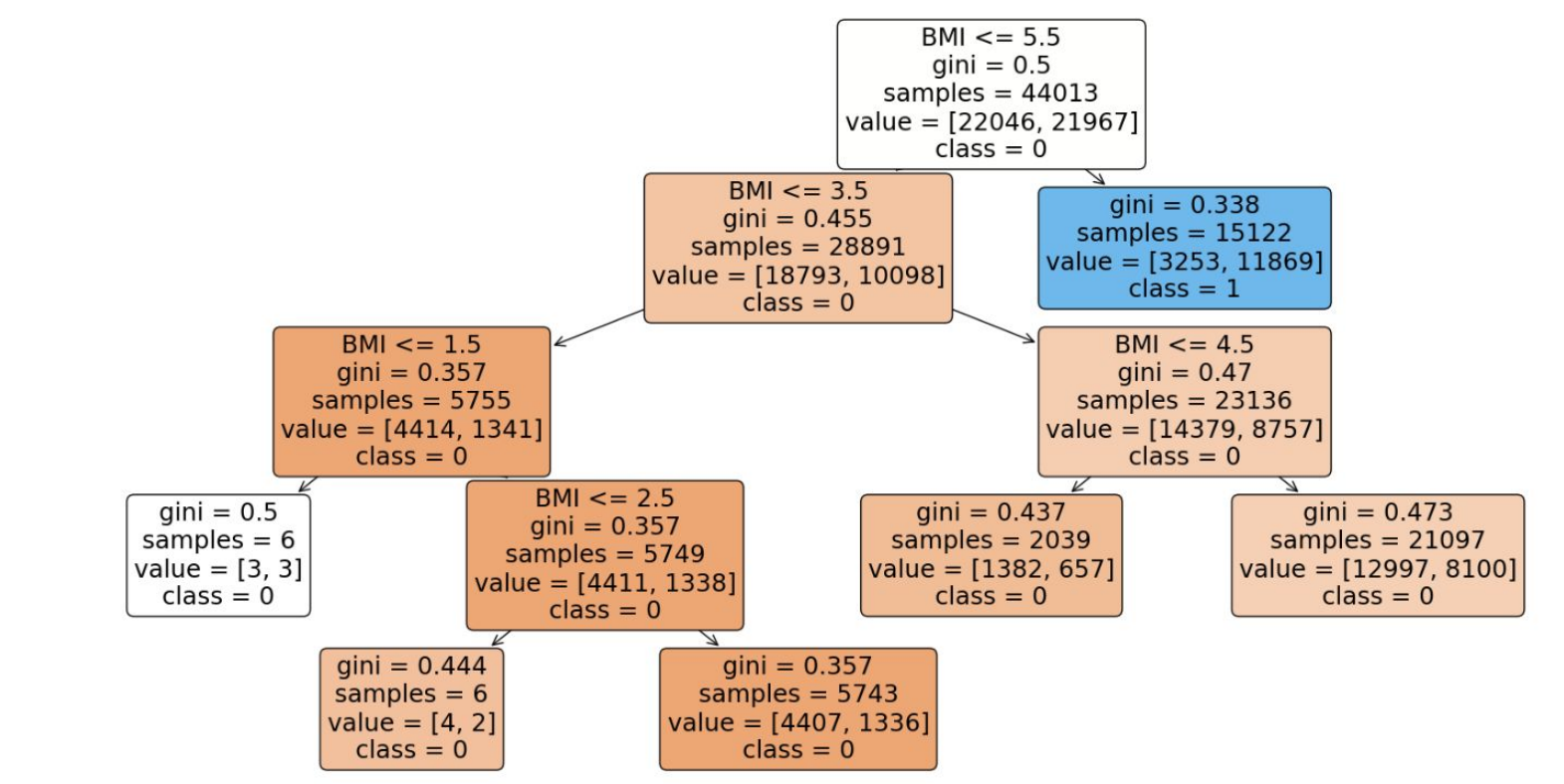
Divided the BMI and cardio data set to train and test sets

75% to train, 25% to test

Boxplot Train set

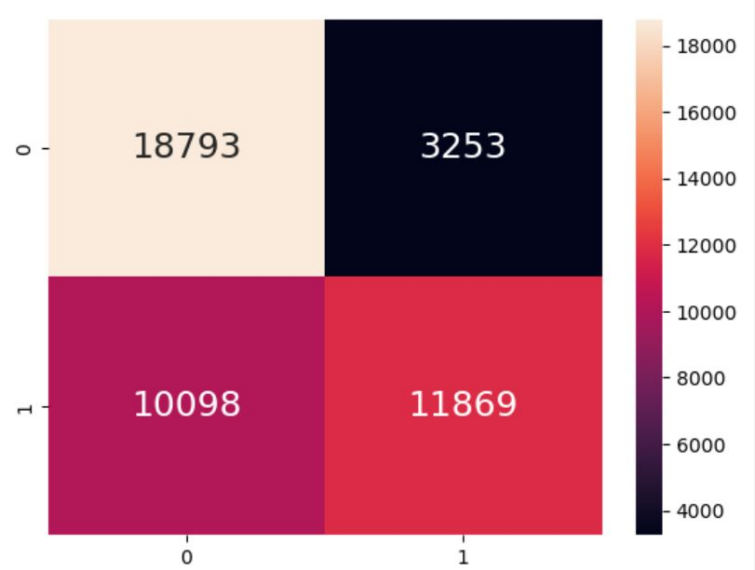


- **Develop a decision tree with the test set**



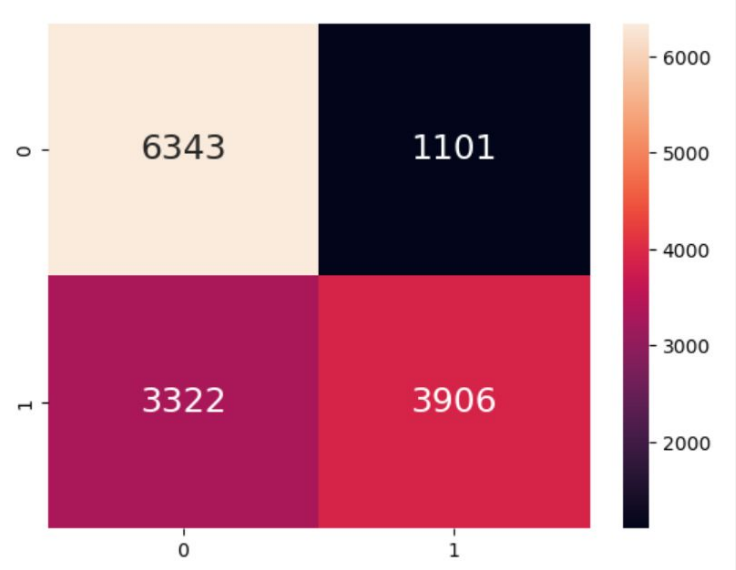
Train data

```
Classification Accuracy          : 0.6966578056483311
```



Test data

```
Classification Accuracy          : 0.698541439476554
```



- Accuracy in Train Data = (TP+TN)/Total
  = (18973 + 11869)/44013
  = 0.700
- Accuracy in Test Data = (TP+TN)/Total
  = (6343 + 3906)/14672
  = 0.699
- Train *fnr* = FN/(TP+FN) = 10098/(10098 + 11869)
  = 0.460
- Test *fnr* = FN/(TP+FN) = 3322/(3322 + 3906)
  = 0.460

Response Variable : Cardio
Predictor Feature : Blood Pressure

```
Goodness of Fit of Model          Train Dataset
Classification Accuracy           : 0.55881286137959

Goodness of Fit of Model          Test Dataset
Classification Accuracy           : 0.5620199692780338

<Axes: >
```

Simple Decision Tree

Repeated ML analysis for interested predictors:

Predictor	Train Accuracy	Test Accuracy
BMI	0.6966578056483311	0.6985414394765540
bp	0.5599650088544667	0.5585637480798771
cholesterol	**0.755531374682626**	**0.7574244751664106**
gluc	**0.8528878363097143**	**0.8552867383512545**
active	**0.8044763063005398**	**0.8002432155657963**

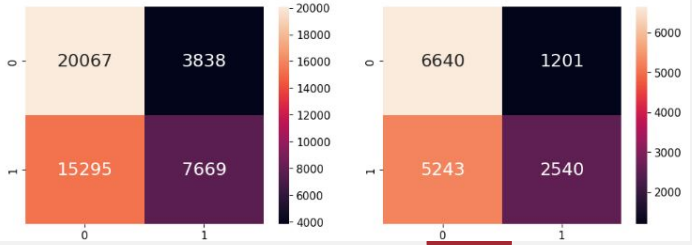
Multi-Variate Classification Tree

choosing the most accurate predictors
Multi-Variate Classification is set up.

Response Variable : **cardio**
Predictor Feature :
active, gluc, cholesterol
active, gluc
gluc, cholesterol
Active, cholesterol

```python
# Import essential models and functions from sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

# Extract Response and Predictors
y = pd.DataFrame(CVD_clean['cardio'])
X = pd.DataFrame(CVD_clean[["active", "cholesterol"]])

# Split the Dataset into Train and Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

# Decision Tree using Train Data
dectree = DecisionTreeClassifier(max_depth = 2)  # create the decision tree object
dectree.fit(X_train, y_train)                    # train the decision tree model

# Predict Response corresponding to Predictors
y_train_pred = dectree.predict(X_train)
y_test_pred = dectree.predict(X_test)

# Check the Goodness of Fit (on Train Data)
print("Goodness of Fit of Model \tTrain Dataset")
print("Classification Accuracy \t:", dectree.score(X_train, y_train))
print()

# Check the Goodness of Fit (on Test Data)
print("Goodness of Fit of Model \tTest Dataset")
print("Classification Accuracy \t:", dectree.score(X_test, y_test))
print()

# Plot the Confusion Matrix for Train and Test
f, axes = plt.subplots(1, 2, figsize=(12, 4))
sb.heatmap(confusion_matrix(y_train, y_train_pred),
           annot = True, fmt=".0f", annot_kws={"size": 18}, ax = axes[0])
sb.heatmap(confusion_matrix(y_test, y_test_pred),
           annot = True, fmt=".0f", annot_kws={"size": 18}, ax = axes[1])
```

```
Goodness of Fit of Model        Train Dataset
Classification Accuracy         : 0.5917770807996757

Goodness of Fit of Model        Test Dataset
Classification Accuracy         : 0.5875576036866359

<Axes: >
```



Multi-Variate Classification Tree

Predictor	Train Accuracy	Test Accuracy
Active, Gluc, Cholesterol	**0.5914997119631313**	**0.588389656938044**
Active, Gluc	0.5406772066824553	0.5428827444956478
Gluc, Cholesterol	**0.5924384987945124**	**0.5867895545314901**
Active, Cholesterol	**0.5917770807996757**	**0.5875576036866359**

Some single predictor has better accuracy

03

New learning

Cleaning and structuring dataset

Feature scaling

Normalizing dataset with different scales

Transform the data in such a manner that it has mean as 0 and standard deviation as 1

Prevent features with wider ranges from dominating the distance metric.

```
#we perform some Standardization
from sklearn.preprocessing import MinMaxScaler
CVD_scale=CVD_clean.copy()

columns_to_scale = ['age', 'weight', 'cholesterol','gender','BMI','height', 'bp']

scaler = MinMaxScaler()
CVD_scale[columns_to_scale] = scaler.fit_transform(CVD_clean[columns_to_scale])

CVD_scale.head()
```

	age	gender	height	weight	cholesterol	gluc	smoke	alco	active	cardio	BMI	bp
0	0.600000	1.0	0.600000	0.185185	0.0	1	0	0	1	0	0.193281	0.8
1	0.742857	0.0	0.200000	0.611111	1.0	1	0	0	1	1	0.616590	1.0
2	0.628571	0.0	0.500000	0.222222	1.0	1	0	0	0	1	0.243802	0.8
3	0.542857	1.0	0.633333	0.555556	0.0	1	0	0	1	1	0.413528	1.0
4	0.514286	0.0	0.200000	0.074074	0.0	1	0	0	0	0	0.227381	0.4

Logistic regression

Usually used for **Binary classification** problems.

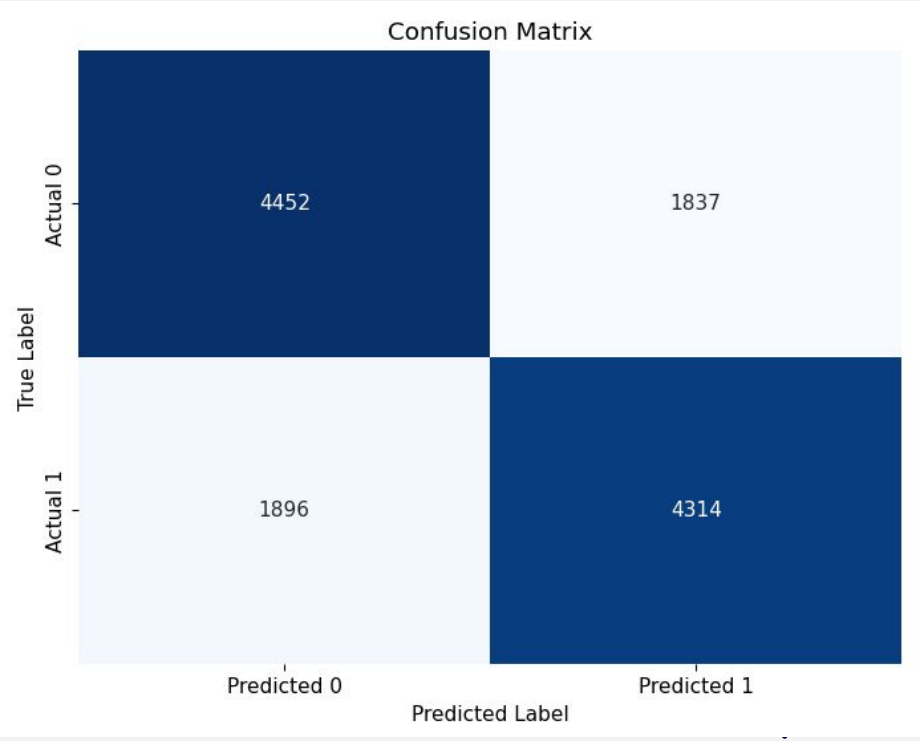
Binary Classification refers to predicting the output variable that is discrete in **two classes**.

With or without CVD in this case.

Performed 3 models:

- Logistic regression with default parameters
- Forward feature subset
- Hyperparameter tuning

Basic Model

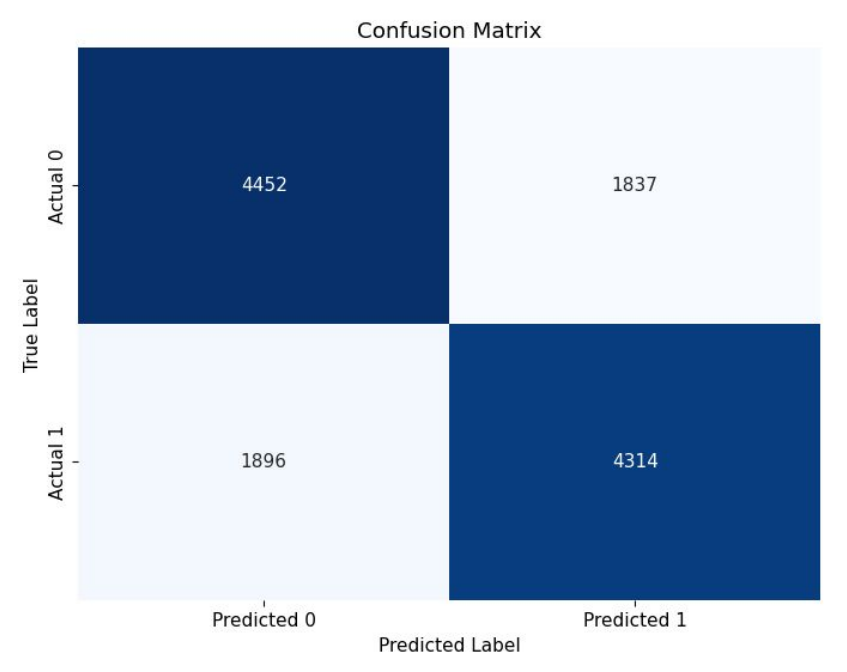



```
The accuracy score is: 0.7013361068885511
Sensitivity (TPR) = 0.6946859903381642

 Confusion matrix

              precision    recall  f1-score   support

           0       0.70      0.71      0.70      6289
           1       0.70      0.69      0.70      6210

    accuracy                           0.70     12499
   macro avg       0.70      0.70      0.70     12499
weighted avg       0.70      0.70      0.70     12499
```

```python
# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train_scaled, y_train_scaled)
y_pred = logreg.predict(X_test_scaled)

# Evaluation: Confusion matrix#
logreg_acc = accuracy_score(y_test_scaled, y_pred)
cm = confusion_matrix(y_test_scaled, y_pred) # Confusion matrix
tpr_logreg = cm[1][1] /(cm[1][0] + cm[1][1])

print('The accuracy score is:', logreg_acc) # accuracy score
print('Sensitivity (TPR) =', tpr_logreg)

print('\n Confusion matrix \n \n')
print(classification_report(y_test_scaled, y_pred ))

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sb.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Feature Subset Selection

Intends to select a subset of attributes or features that makes the **most meaningful contribution** to a machine learning activity




```
The accuracy score is: 0.7040563245059605
Sensitivity (TPR) = 0.6325281803542673

 Confusion matrix


              precision    recall  f1-score   support

           0       0.68      0.77      0.72      6289
           1       0.73      0.63      0.68      6210

    accuracy                           0.70     12499
   macro avg       0.71      0.70      0.70     12499
weighted avg       0.71      0.70      0.70     12499
```

```python
# Sequential Forward Selection(sfs)
sfs = SequentialFeatureSelector(LogisticRegression(),
          direction='forward',
          scoring = 'accuracy',
          cv = 5,
          n_jobs=-1)

sfs.fit(X_train_scaled, y_train_scaled)
print("Features selected by forward sequential selection: " f"{sfs.get_feature_names_out()}")
```

```
C:\Users\gushi\anaconda3\lib\site-packages\sklearn\feature_selection\_sequential.py:206: FutureWa
select` to None is deprecated in 1.0 and will become 'auto' in 1.3. To keep the same behaviour as
f the features) and avoid this warning, you should manually set `n_features_to_select='auto'` and
instance.
  warnings.warn(
```

```
Features selected by forward sequential selection: ['gender' 'cholesterol' 'alco' 'active' 'bp']
```

```python
#train the model after subset selection with selected features
X_train_subset = X_train_scaled[sfs.get_feature_names_out()]
X_test_subset = X_test_scaled[sfs.get_feature_names_out()]

logreg_subset = LogisticRegression()
logreg_subset.fit(X_train_subset, y_train_scaled)
y_pred = logreg_subset.predict(X_test_subset)

# Evaluation: Confusion matrix#
logreg_acc_subset = accuracy_score(y_test_scaled, y_pred)
cm_subset = confusion_matrix(y_test_scaled, y_pred) # Confusion matrix
tpr_logreg_subset = cm_subset[1][1] /(cm_subset[1][0] + cm_subset[1][1])

print('The accuracy score is:', logreg_acc_subset) # accuracy score
print('Sensitivity (TPR) =', tpr_logreg_subset)

print('\n Confusion matrix \n \n')
print(classification_report(y_test_scaled, y_pred ))

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sb.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
           xticklabels=['Predicted 0', 'Predicted 1'],
           yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Tuned Logistic regression

GridSearchCV can save effort in optimizing machine learning model.
It is computationally expensive.

This approach find out the best hyperparameters for the dataset.

```
parameters = {
     'penalty' : ['l1','l2'],  #l1 lasso l2 ridge
     'C' : [0.001,0.01,0.1,1,10,100],
     }

tun_logreg = LogisticRegression()
clf_tun1 = GridSearchCV(tun_logreg,                    # model
                   param_grid = parameters,   # hyperparameters
                   scoring='accuracy',        # metric for scoring
                   cv=5,                      # number of folds GridSearchCV does an internal 5-fold cross validation
                   verbose=3,
                   n_jobs=-1)

clf_tun1.fit(X_train_scaled,y_train_scaled)
print("Tuned Hyperparameters :", clf_tun1.best_params_)
print("Accuracy :",clf_tun1.best_score_)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Tuned Hyperparameters : {'C': 100, 'penalty': 'l2'}
Accuracy : 0.7098051353445006
```

Result

Model	accuracy score
Basic Model	**0.7302617190347743**
Feature Subset Selection	0.7269811183203324
Tuned Logistic regression	0.72577324719251

Logistic Regression model trained with default, forward subset selection and hyperparameter tuning gives us similar results, in comparison, the **basic model** is significantly better.

04

Outcome

Conclusion of analysis and areas for exploration

How ML solved our objective

Simple Decision Tree

Glucose level is the most indicative variable. Followed by Active and Cholesterol

Multi-Variate Classification Tree

Some Single predictor has better accuracy for this dataset

Logistic regression

By making use of all available variables in the better suited model, the accuracy was closer to the most indicative variable

Interesting findings

Categorical Analysis

The observation seen in categorical analysis was largely supported by the Decision Tree ML model

Modelling Multi-variable

Combining variables with high accuracy as predictor in model may not result it better accuracy

Optimizing ML

The more ML model is optimized, it is more computationally expensive.

The accuracy is more stable but does not necessarily lead to better accuracy.

Conclusions

By analysing the dataset, we have determined that Glucose level is the most indicative variable in predicting the presence of cardiovascular diseases (CVDs)

Team contribution

Chen Yi Bin Jonathan

Code: Data preparation
+slides

Gu Shiyuan

Code: Machine learning
(Simple Decision Tree) and
New learning
+slides

Mendos

Code: Exploratory Analysis
and Machine learning
(Simple Decision Tree)
+ slides

Thank You!

References

- https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset/data
- https://www.kaggle.com/code/mertoezcan/comparison-of-different-machine-learning-models
- https://www.medicinenet.com/blood_pressure_chart_reading_by_age/article.htm