Using Machine Learning Algorithms to predict CardioVascular Disease

Group Project Team 2 – Zhang Liyuan, Ma Xiaoru, Luo Kexin, Luo Lan & Koh Swee Guan

Using Machine Learning Algorithms to predict CardioVascular Disease

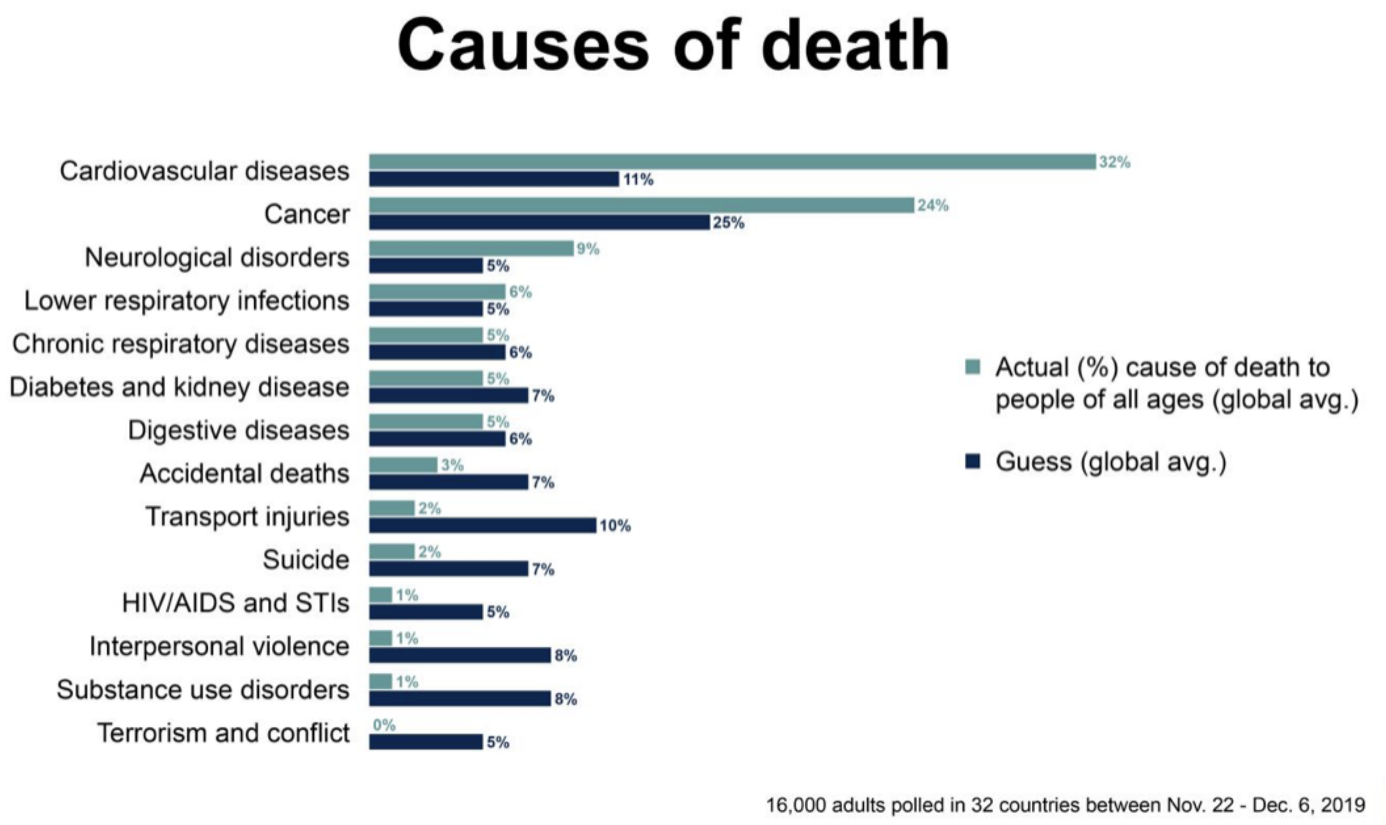
- Introduction & Data Cleaning
- Exploratory Data Analysis
- Feature Engineering
- Exploring Various Classifiers
- Hyperparameters Tuning & Cross Validation
- CardioVascular Prediction

PART 1

By : Luo Lan

Introduction & Data Cleaning

Introduction & Data Cleaning



- Singapore Statistics
- Every day, 17 people die from cardiovascular disease (heart diseases and stroke) in Singapore. Cardiovascular disease accounted for 29.2% of all deaths in 2018. This means that almost 1 out of 3 deaths in Singapore, is due to heart diseases or stroke.

- China Statistics
- A new study published in the Journal of the American Medical Association shows that, between 1990 and 2016, the proportion of Chinese people living with heart disease increased by about 15%, from 5,265 per 100,000 people to 6,037 per 100,000.

Introduction & Data Cleaning

- Data Information - taken from https://www.kaggle.com/sulianova/cardiovascular-disease-dataset

- Age: days
- gender: 1 - women, 2 - men
- Height: cm
- Weight: kg
- ap_hi: Systolic blood pressure
- ap_lo: Diastolic blood pressure
- cholesterol: 1: normal, 2: above normal, 3: well above normal
- gluc: 1:normal, 2: above normal, 3: well above normal
- smoke: 1 when patient smoke, 0 when patient don't smoke
- alco Binary feature: 1 when patient drinks alcohol, 0 when patient don't drink alcohol
- active Binary feature: 1 when patient is active, 0 when patient is not active
- cardio Target variable: 1 when patient has cardiovascular disease, 0 for healthy patient.

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

Introduction & Data Cleaning

Clean the dataset

- Check for missing data → df.isnull().values.any()
- Drop id since it is not useful for modelling
- Check for duplicate data

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
2677	22077	1	175	69.0	120	80	1	1	0	0	1	1
45748	22077	1	175	69.0	120	80	1	1	0	0	1	1
1568	21945	1	165	60.0	120	80	1	1	0	0	1	0
48917	21945	1	165	60.0	120	80	1	1	0	0	1	0

- Remove duplicates and Confirm the removal
- Convert the age from days to years
- Explore the patient's height and weight

  Min weight is 10 kg, min height is 55cm, min age is 30 years old (10kg is not normal for anyone 30 years old or more)

  Max weight is 200 kg, max height is 250cm, max age is 65 years old (250 cm is not normal for anyone)
- To improve data quality, remove outliers. Remove weights and heights, that fall below 1% or above 99% of a given range.

Introduction & Data Cleaning

Clean the dataset

- Explore the patient's blood pressure. Remove those that fall below 1.5% or above 98.5% of a given range.
- Explore the blood pressure data. Remove data where diastolic pressure is higher than systolic pressure.

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	acti
count	65147.000000	65147.000000	65147.000000	65147.000000	65147.000000	65147.000000	65147.000000	65147.000000	65147.000000	65147.000000	65147.0000
mean	53.310682	1.347645	164.471887	73.710447	125.873993	81.043440	1.357576	1.222036	0.086942	0.052527	0.8041
std	6.758899	0.476226	7.293385	12.836057	15.029385	8.809493	0.674307	0.568265	0.281752	0.223090	0.3968
min	30.000000	1.000000	147.000000	48.000000	90.000000	60.000000	1.000000	1.000000	0.000000	0.000000	0.0000
25%	48.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.0000
50%	54.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.0000
75%	58.000000	2.000000	170.000000	81.000000	140.000000	90.000000	1.000000	1.000000	0.000000	0.000000	1.0000
max	65.000000	2.000000	184.000000	117.000000	173.000000	110.000000	3.000000	3.000000	1.000000	1.000000	1.0000

Data is now cleaned.

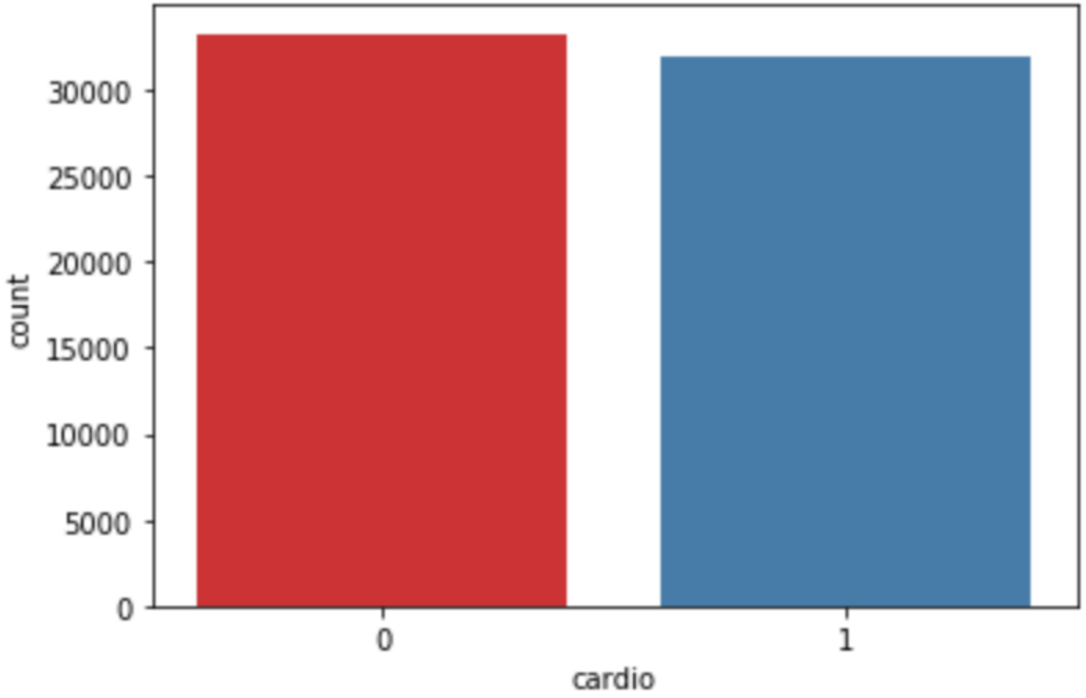
PART 2

By : Zhang Liyuan

Exploratory Data Analysis

```
# checking if target are balanced
import seaborn as sns
df.groupby("cardio").count()
#traindata[["cardio", "height"]].groupby("Outcome").count()
sns.countplot(x="cardio", data=df, palette="Set1")
print ('Number of people without and with CardioVascular Disease')
```

```
Number of people without and with CardioVascular Disease
```



Exploratory Data Analysis

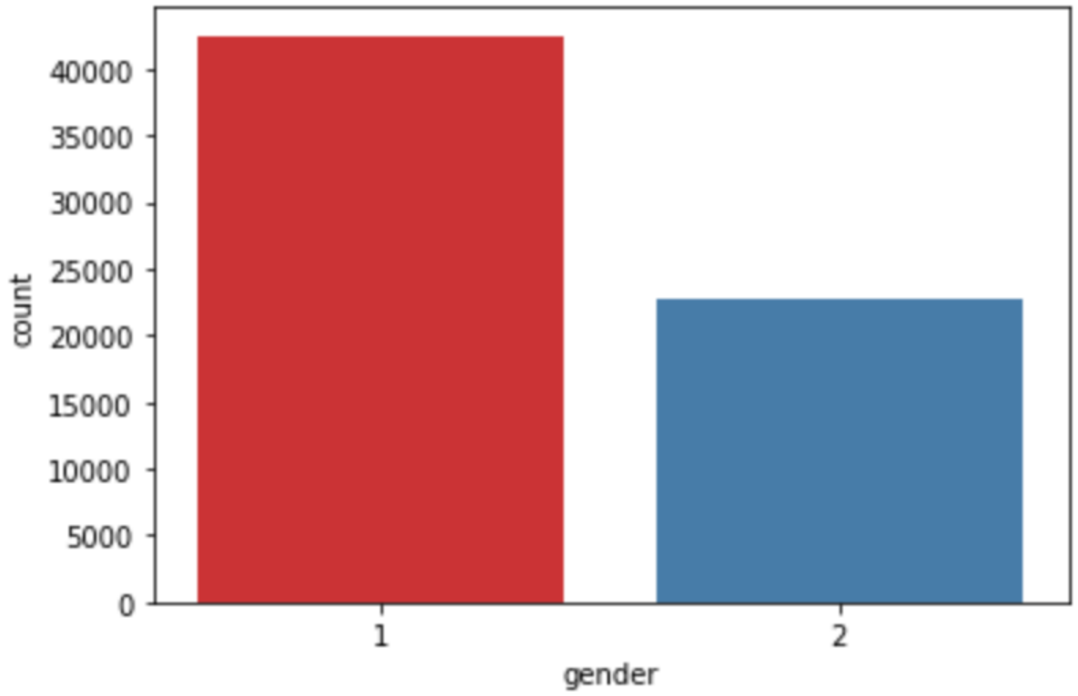
Checking if target are balanced

Exploratory Data Analysis

Checking dataset of gender

```
# checking dataset of gender
import seaborn as sns
df.groupby("gender").count()
#traindata[["cardio", "height"]].groupby("Outcome").count()
sns.countplot(x="gender", data=df, palette="Set1")
print ('Number of people based on gender')
```

```
Number of people based on gender
```

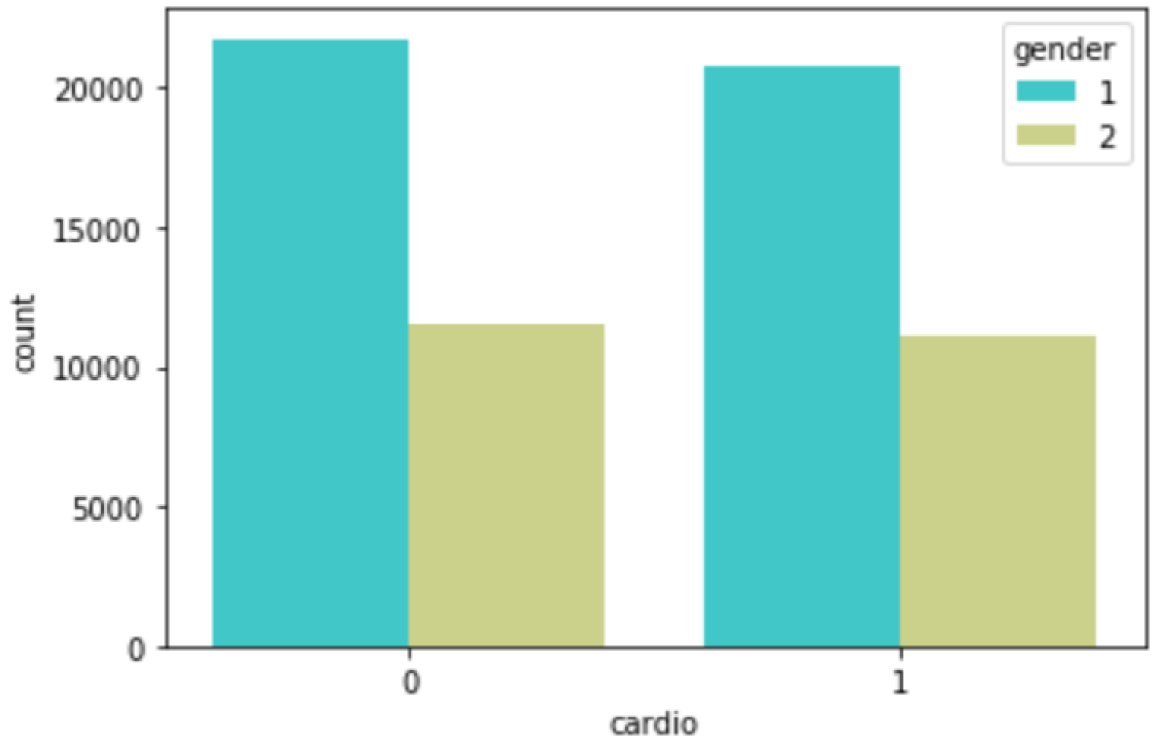


Exploratory Data Analysis

Visualize cardio with gender

```
# visualize cardio with gender
sns.countplot(x='cardio',data=df,hue='gender',palette='rainbow')
print ('Visualize gender with CardioVascular Disease')
```

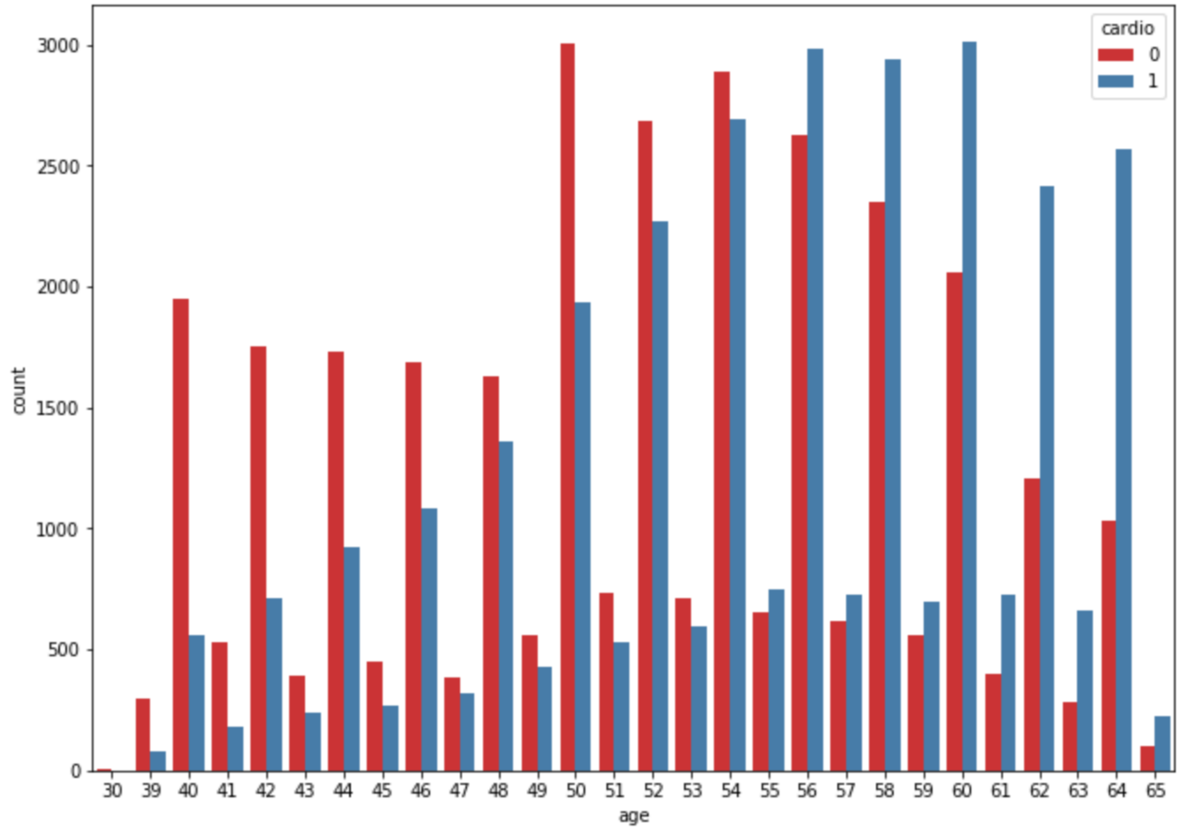
```
Visualize gender with CardioVascular Disease
```



Exploratory Data Analysis

Age boundary

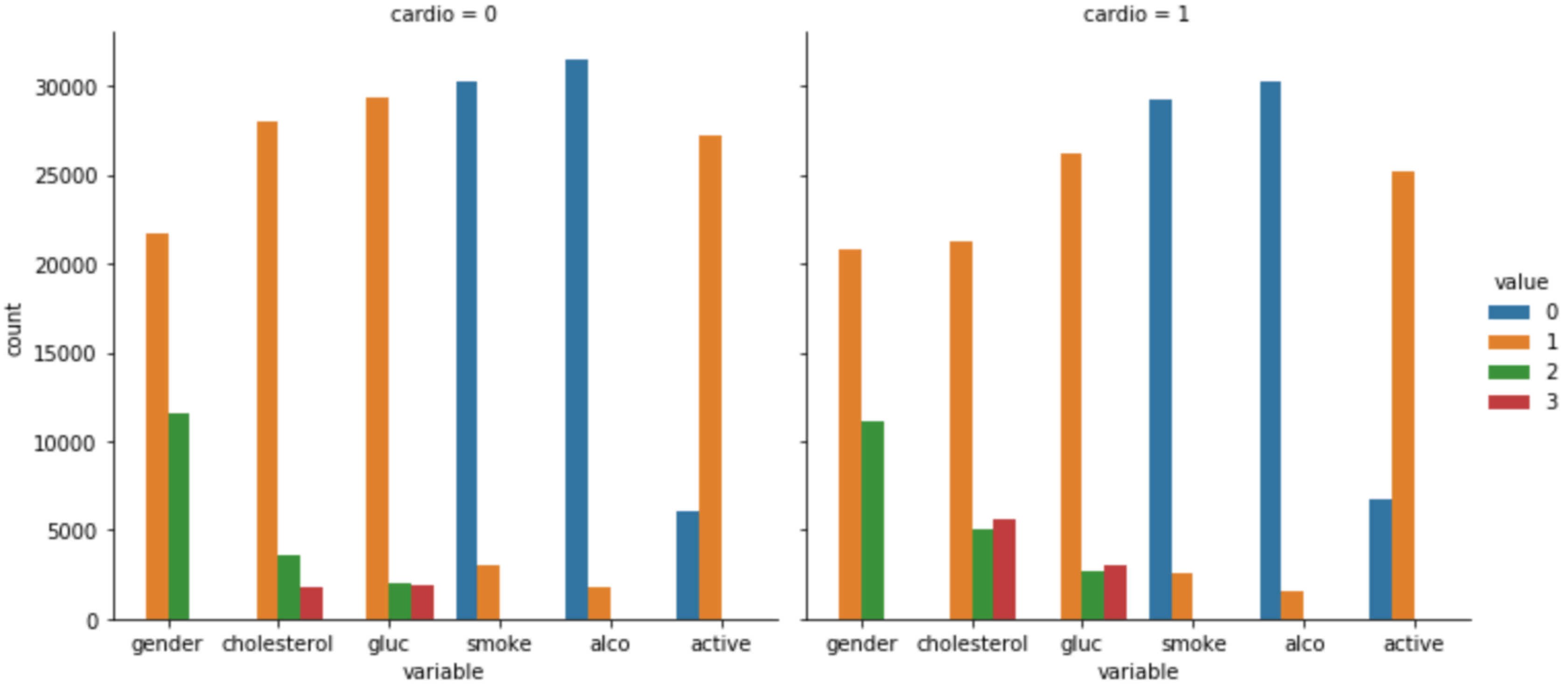
```python
# Explore at what age does the number of people with CVD exceed the number of people without CVD
from matplotlib import rcParams
rcParams['figure.figsize'] = 11, 8
sns.countplot(x='age', hue='cardio', data = df, palette="Set1");
```



Exploratory Data Analysis

Bivariate analysis

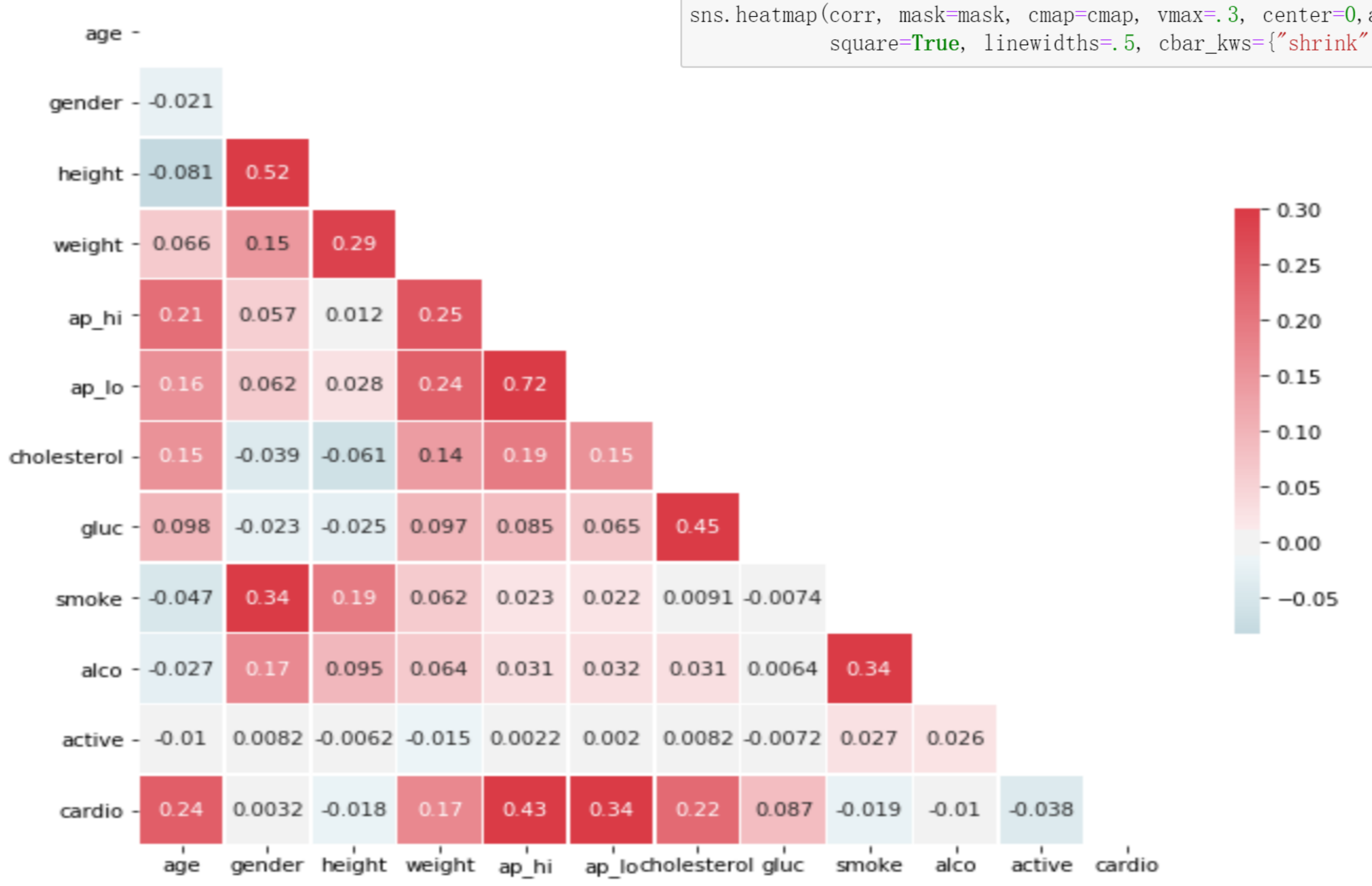
```
#Bivariate analysis - split categorical variables by target class:
df_long = pd.melt(df, id_vars=['cardio'], value_vars=['gender','cholesterol','gluc', 'smoke', 'alco', 'active'])
sns.catplot(x="variable", hue="value", col="cardio",
            data=df_long, kind="count");
```



Exploratory Data Analysis

Multivariate analysis

```python
# Multivariate analysis - It might be useful to consider correlation matrix
corr = df.corr()
cmap = sns.diverging_palette(220, 10, as_cmap=True)
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,annot = True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5});
```



PART 3

By : Ma Xiaoru

Feature Engineering

Feature Engineering

Categorizing BMI
(Body Mass Index)

NEWSLETTER MEDICALNEWSTODAY

Understanding the results

The following table shows standard weight status categories associated with BMI ranges for adults.

BMI	Weight status
Below 18.5	Underweight
18.5–24.9	Healthy
25.0–29.9	Overweight
30.0 and above	Obese

In [85]:

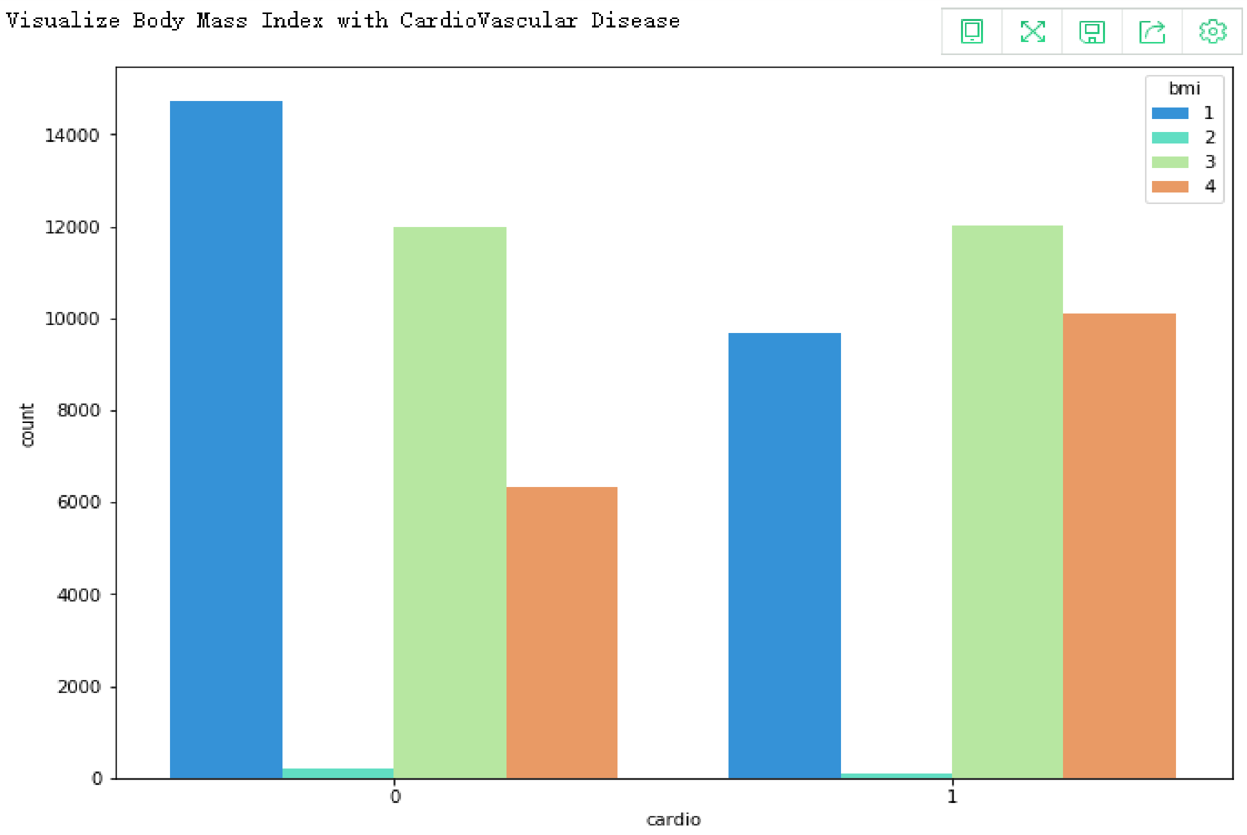
```
bmi = (df.weight/df.height/df.height)*10000

cat_bmi = []
for n in bmi:
    if n <= 18.5:                #Underweight
        val = 2
    elif n> 18.5 and n<25:        #Healthy
        val = 1
    elif n>= 25 and n<30:        #Overweight
        val = 3
    else:
        val = 4                  #Obese
    cat_bmi.append(val)

df['bmi'] = cat_bmi
```

In [86]:

```
# visualise cardio with bmi
sns.countplot(x='cardio',data=df,hue='bmi',palette='rainbow')
print ('Visualize Body Mass Index with CardioVascular Disease')
```

Feature Engineering

Categorizing BMI
(Body Mass Index)



The figure shows that obese person tends to have higher risk of cardiovascular disease

Feature Engineering

Categorizing blood pressure

How Your Numbers Translate

Blood Pressure Stages

Blood Pressure Category	Systolic mm Hg (upper #)		Diastolic mm Hg (lower #)
Normal	less than 120	and	less than 80
Elevated	120-129	and	less than 80
High Blood Pressure (Hypertension) Stage 1	130-139	or	80-89
High Blood Pressure (Hypertension) Stage 2	140 or higher	or	90 or higher
Hypertensive Crisis (Seek Emergency Care)	higher than 180	and/or	higher than 120

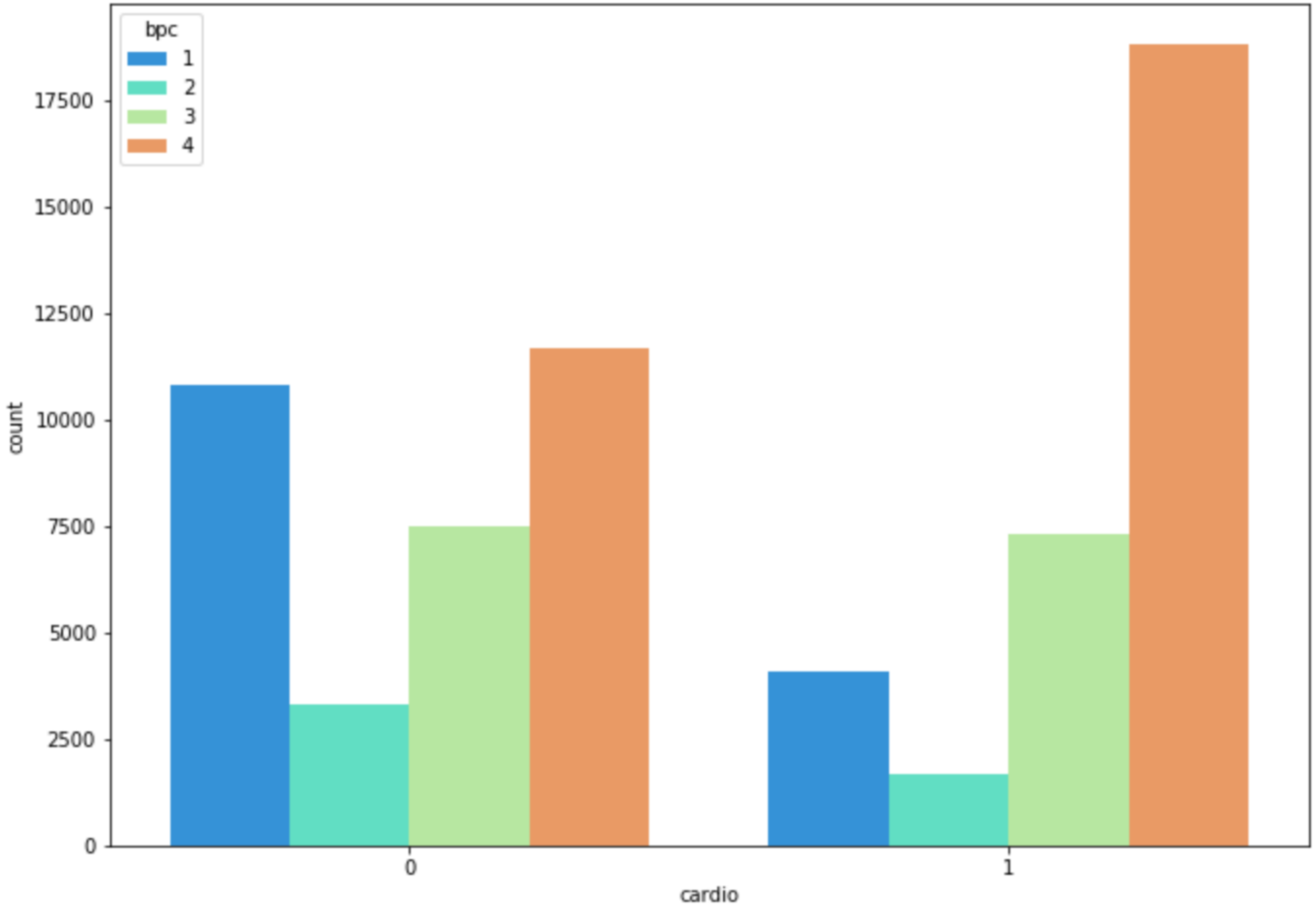
Source: American Heart Association

```
In [87]: bpc = []
for n,v in zip(df.ap_hi, df.ap_lo):
    if n < 120 and v < 80:
        val = 1                          # Normal
    elif n >= 120 and n < 130 and v < 80:
        val = 2                          # Elevated
    elif n >= 130 and n < 140 or v >= 80 and n < 90:
        val =3                           # High blood pressure stage 1(Hypertension)
    elif n >= 140 and n < 180 or v >= 90 and v < 120:
        val = 4                          # High blood pressure stage 2(Hypertension)
    elif n >= 180 or v >= 120:
        val = 5                          # Hyertensive Crisis (see doctor immediately)
    bpc.append(val)

df['bpc'] = bpc
```

```
In [88]: # visualize cardio with bmi
sns.countplot(x='cardio',data=df,hue='bmi',palette='rainbow')
print ('Visualize Blood Pressure Category with CardioVascular Disease')
```

Feature Engineering

Categorizing blood pressure



The figure shows that normal blood pressure correlate with lower CVD while hypertensive correlate with higher CVD

PART 4

By : Luo Kexin

Exploring Various Classifiers

Exploring Various Classifiers

1. Train/ Test split

Define test size=0.25

MinMaxScaler rescales the data set such that all feature values are in the range [0, 1]

After spliting:
Train data: 48860
Test data:16287

```
print (x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(48860, 13) (48860,) (16287, 13) (16287,)
```

Exploring Various Classifiers

Comparison of Different Classifiers

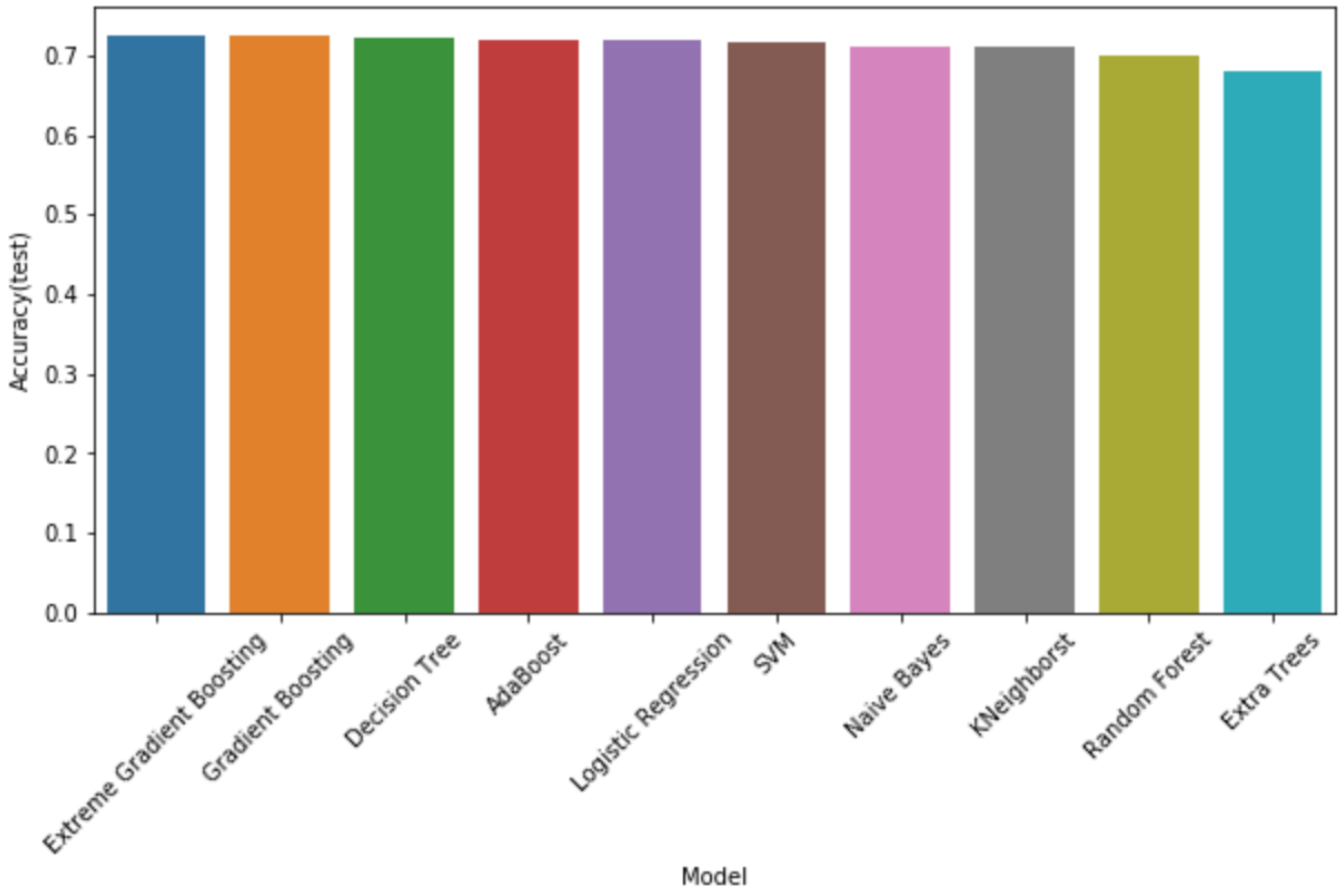
```
Processing Naive Bayes
Processing SVM
Processing Logistic Regression
Processing Decision Tree
Processing Random Forest
Processing KNeighborst
Processing AdaBoost
Processing Extra Trees
Processing Gradient Boosting
Processing Extreme Gradient Boosting
```

Comparisons of various algorithms

Summary of Results

Model	Accuracy(train)	Accuracy(test)	F1 Score	Time
Extreme Gradient Boosting	0.73842	0.72469	0.70142	29.75746
Gradient Boosting	0.73717	0.72444	0.70223	3.03555
Decision Tree	0.73878	0.72125	0.69938	0.08049
AdaBoost	0.72972	0.72008	0.68669	2.58600
Logistic Regression	0.72835	0.71824	0.69506	0.10230
SVM	0.72583	0.71665	0.67743	52.46618
Naive Bayes	0.71832	0.71106	0.68099	0.04439
KNeighborst	0.72603	0.71094	0.67811	25.17822
Random Forest	0.98721	0.70019	0.68576	5.17517
Extra Trees	0.98723	0.68146	0.66778	5.69215

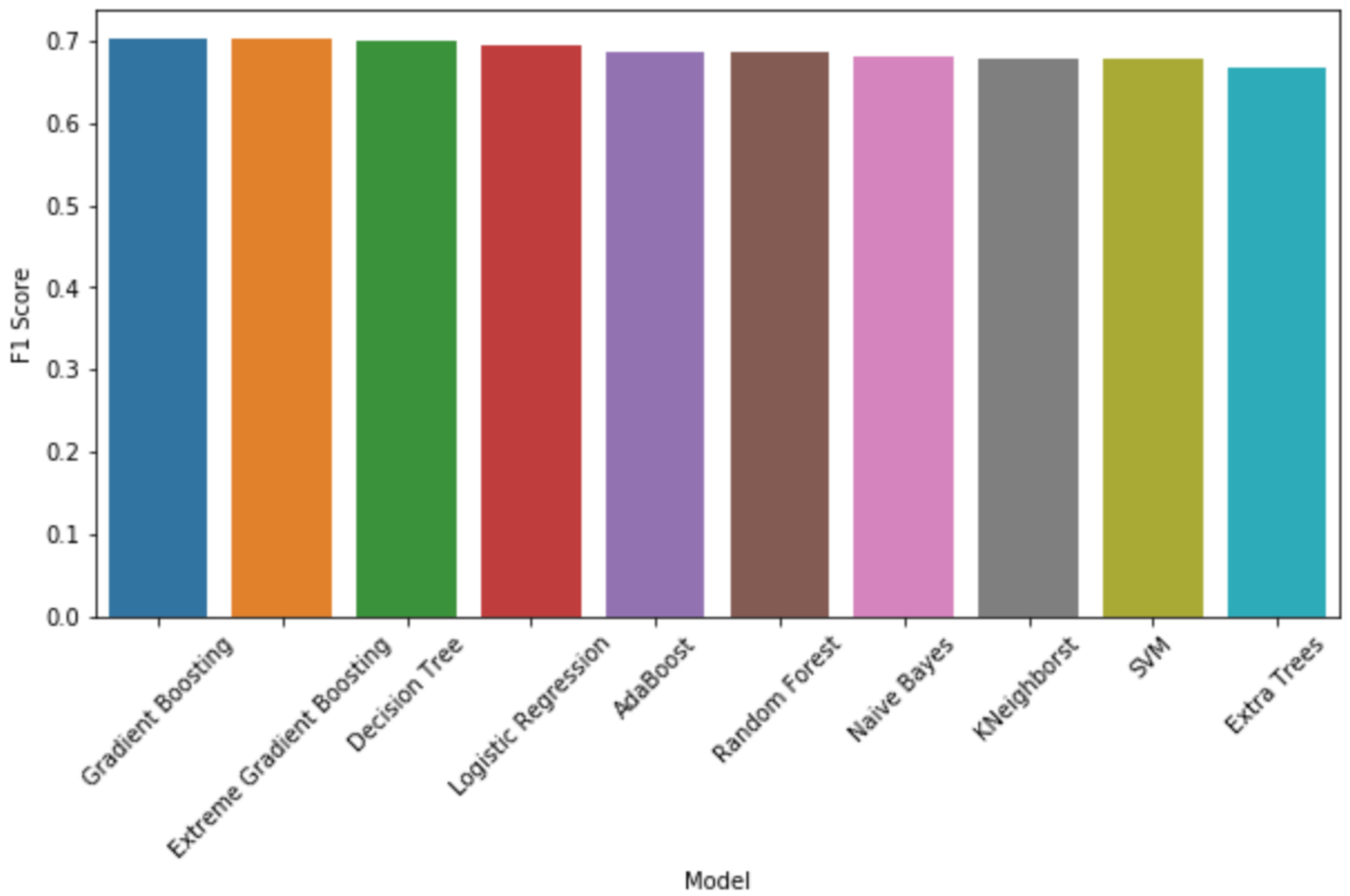
Comparisons of various algorithms



3.1 Testing Accuracy

Comparisons of various algorithms

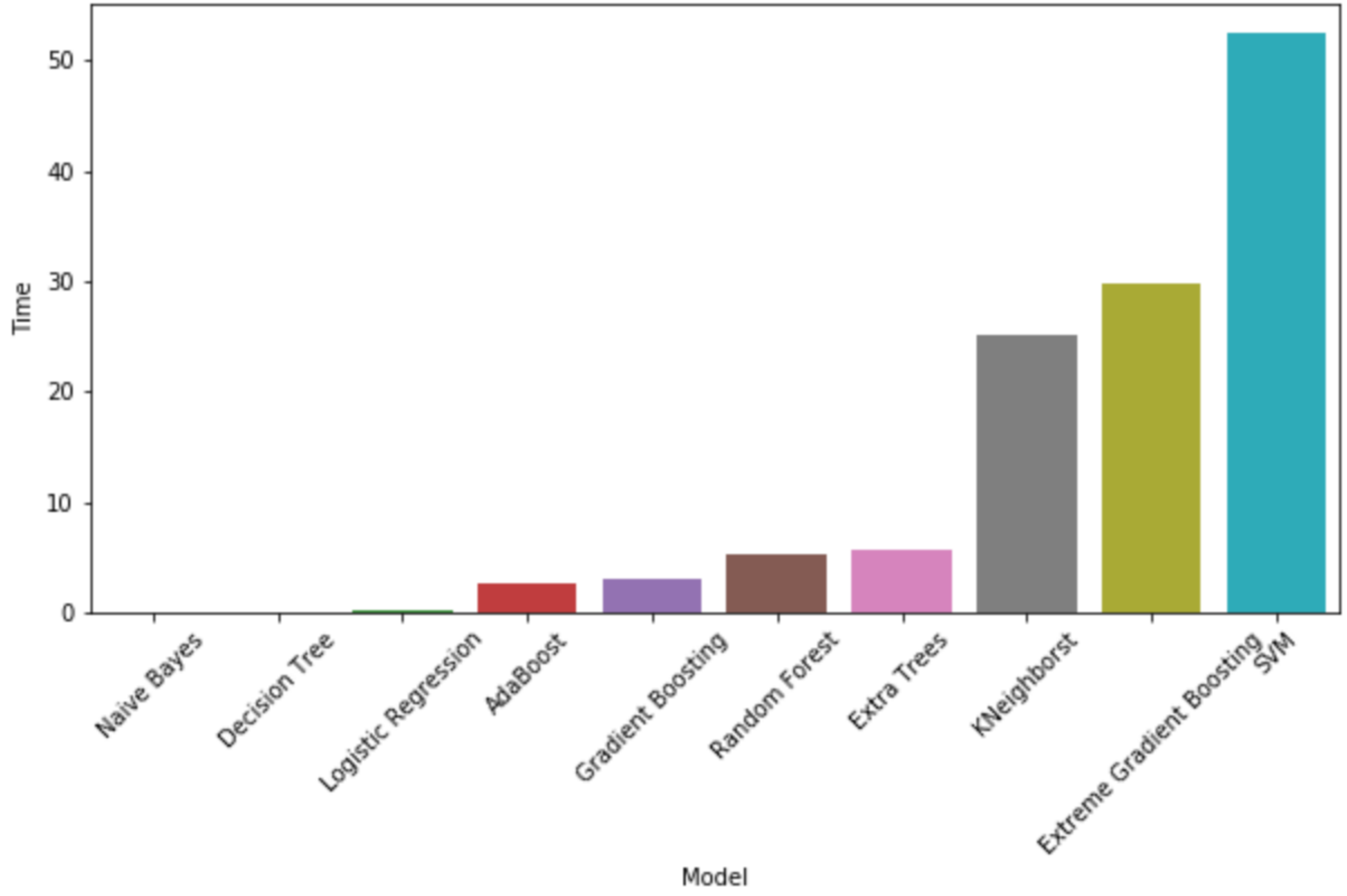
3.2 F1 score



Graph showing the F1 Score for the various models

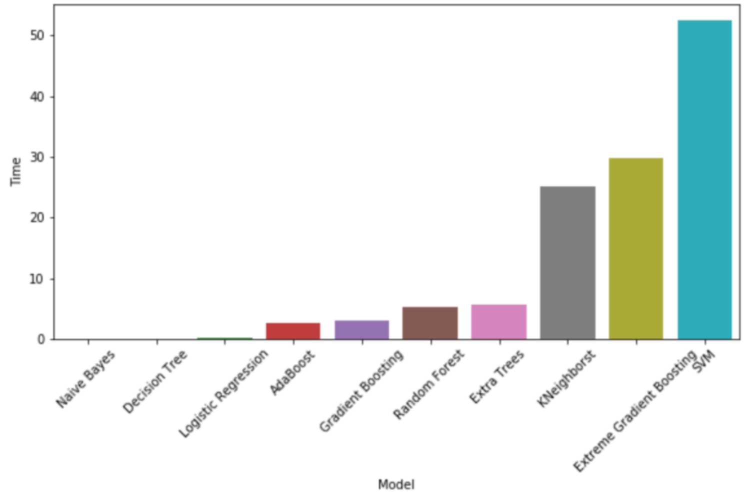
Comparisons of various algorithms

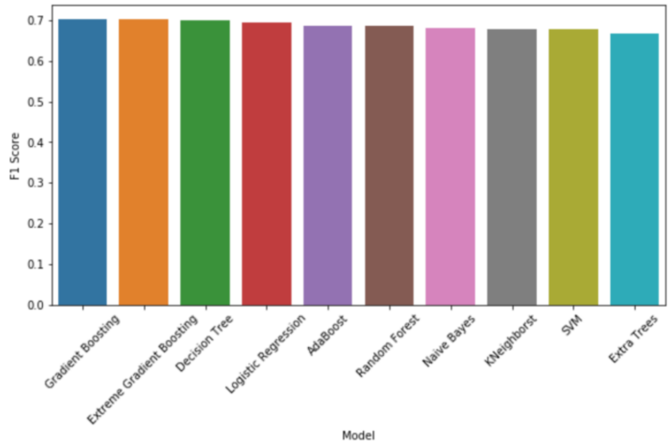
3.3 Training Time

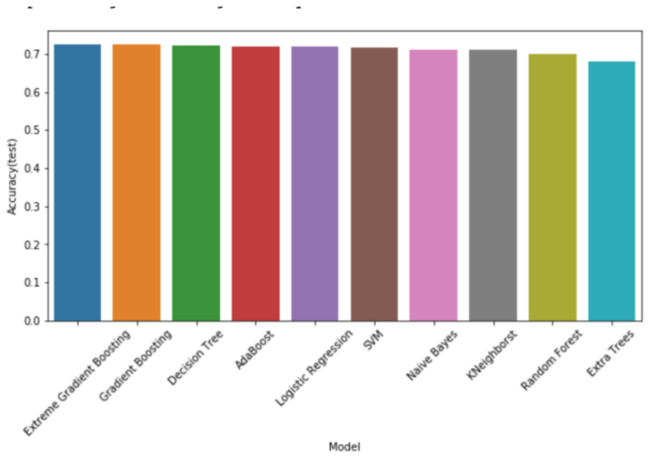


Graph showing the processing time for the various models

From the results, we will focus our model using Extreme Gradient Boosting







PART 5

By : Koh Swee Guan

Tuning of XGBoost hyperparameters & Cross Validation

Tuning of XGBoost hyperparameters

```
xgb = XGBClassifier(n_estimators= 2000, learning_rate=0.01)
print (xgb)
xgb.fit(dfx, dfy)
predict = xgb.predict(dfx)
acc = round(accuracy_score(dfy, predict), 5)
print (acc)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.01, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=2000, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

- Best Score: 0.73544

Tuning of XGBoost hyperparameters

Tuning of max_depth & min_child_weight parameters

```
# Step 1: Fix learning rate and number of estimators for tuning tree-based parameters
# Tuning of max_depth & min_child_weight
param_test1 = {
 'max_depth':range(1,8,1),
 'min_child_weight':range(1,8,1)
}
gsearch1 = GridSearchCV(estimator = XGBClassifier( learning_rate =0.1, n_estimators=2000, max_depth=5,
 min_child_weight=1, gamma=0, subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1, seed=27),
 param_grid = param_test1, scoring='roc_auc',n_jobs=4,cv=5)

gsearch1.fit(dfx, dfy)

# Print hyperparameter
print("Tuned hyperparameter: {}".format(gsearch1.best_params_))
print("Best score: {}".format(gsearch1.best_score_))
```

- Tuned hyperparameter:

{'max_depth': 2,
'min_child_weight': 4}

- Best score: 0.79743

Tuning of XGBoost hyperparameters

Tuning of gamma

```
# Step 2: Tune gamma
# Use 'max_depth': 2, 'min_child_weight': 4
param_test2 = {'gamma':[i/10.0 for i in range(0,10)]}

gsearch2 = GridSearchCV(estimator = XGBClassifier( learning_rate =0.1, n_estimators=2000, max_depth=2,
 min_child_weight=4, gamma=0, subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1,seed=27),
 param_grid = param_test2, scoring='roc_auc',n_jobs=4, cv=5)

gsearch2.fit(dfx, dfy)

# Print hyperparameter
print("Tuned hyperparameter: {}".format(gsearch2.best_params_))
print("Best score: {}".format(gsearch2.best_score_))
```

- Tuned hyperparameter: {'gamma': 0.5}
- Best score: 0.79744

Best Score: 0.79743 -> 0.79744

Tuning of XGBoost hyperparameters

Tuning of subsample & colsample_bytree

```
# Step 3: Tune subsample and colsample_bytree
# Use 'max_depth': 2, 'min_child_weight': 4, 'gamma': 0.5
param_test3 = {
 'subsample':[i/10.0 for i in range(6,12)],
 'colsample_bytree':[i/10.0 for i in range(1,8)]
}

gsearch3 = GridSearchCV(estimator = XGBClassifier( learning_rate =0.1, n_estimators=2000, max_depth=2,
 min_child_weight=4, gamma=0.5, subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1,seed=27),
 param_grid = param_test3, scoring='roc_auc',n_jobs=4,cv=5)

gsearch3.fit(dfx, dfy)

# Print hyperparameter
print("Tuned hyperparameter: {}".format(gsearch3.best_params_))
print("Best score: {}".format(gsearch3.best_score_))
```

- Tuned hyperparameter: {'colsample_bytree': 0.3, 'subsample': 1.0}
- Best score: 0.79851

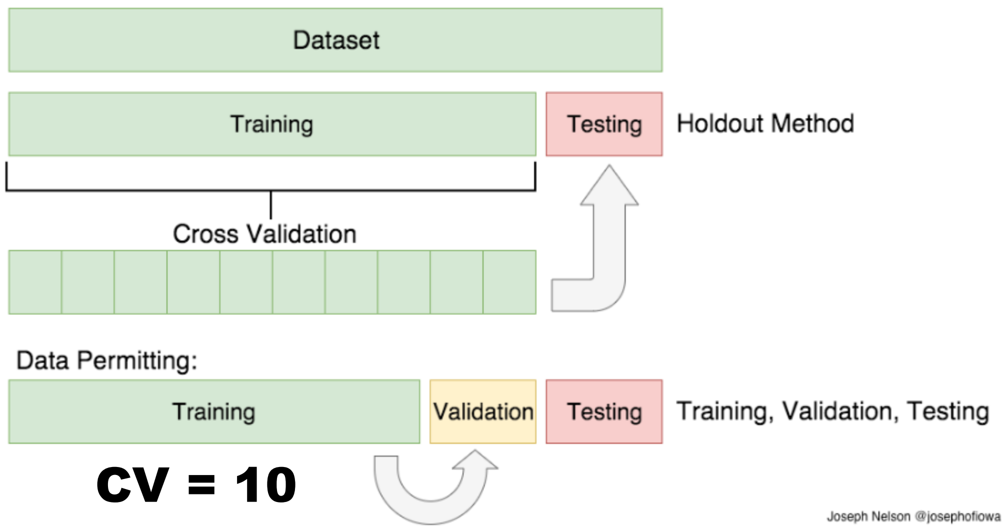
Best Score: 0.79743 -> 0.79744 -> 0.79851

Optimized Parameters ('max_depth': 2, 'min_child_weight': 4, 'gamma': 0.5, 'colsample_bytree': 0.3, 'subsample': 1.0)

Cross Validation

- **Extreme Gradient Boosting gives the best average accuracy and the standard deviation is relatively low indicating the reliability of the training algorithm.**

```python
# Cross validation for top 5 algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score

lr = LogisticRegression( class_weight="balanced", random_state=42, solver='liblinear', max_iter=10000, C = 1, penalty= 'l2'
dt = DecisionTreeClassifier(max_depth=8, min_samples_split=50, min_samples_leaf=50, random_state=13)
ada = AdaBoostClassifier(n_estimators=100)
gb = GradientBoostingClassifier(n_estimators = 100)
xgb = XGBClassifier(learning_rate =0.1, n_estimators=2000, max_depth=2,
      min_child_weight=4, gamma=0.5, subsample=1.0, colsample_bytree=0.3,
      objective= 'binary:logistic', nthread=4, scale_pos_weight=1,seed=27)
accuracies_xgb = cross_val_score(estimator=xgb, X=dfx, y=dfy, cv=10)
print("Extreme Gradient Boosting Average accuracy: ", accuracies_xgb.mean())
print("Extreme Gradient Boosting Standard Deviation: ", accuracies_xgb.std())

accuracies_gb = cross_val_score(estimator=gb, X=dfx, y=dfy, cv=10)
print("Gradient Boosting Average accuracy: ", accuracies_gb.mean())
print("Gradient Boosting Standard Deviation: ", accuracies_gb.std())

accuracies_dt = cross_val_score(estimator=dt, X=dfx, y=dfy, cv=10)
print("Decision Tree Average accuracy: ", accuracies_dt.mean())
print("Decision Tree Standard Deviation: ", accuracies_dt.std())

accuracies_ada = cross_val_score(estimator=ada, X=dfx, y=dfy, cv=10)
print("AdaBoosting Average accuracy: ", accuracies_ada.mean())
print("AdaBoosting Standard Deviation: ", accuracies_ada.std())

accuracies_lr = cross_val_score(estimator=lr, X=dfx, y=dfy, cv=10)
print("Logistic Regression Average accuracy: ", accuracies_lr.mean())
print("Logistic Regression Standard Deviation: ", accuracies_lr.std())
```




```
Extreme Gradient Boosting Average accuracy:  0.7323284920771627
Extreme Gradient Boosting Standard Deviation:  0.004677612963894862
Gradient Boosting Average accuracy:  0.7319907697477138
Gradient Boosting Standard Deviation:  0.004555104186902315
Decision Tree Average accuracy:  0.7290128493538093
Decision Tree Standard Deviation:  0.005768765772964501
AdaBoosting Average accuracy:  0.7273090440307917
AdaBoosting Standard Deviation:  0.00363061817869509
Logistic Regression Average accuracy:  0.7260656791877038
Logistic Regression Standard Deviation:  0.004372793000511551
```

Saving the Optimised Model

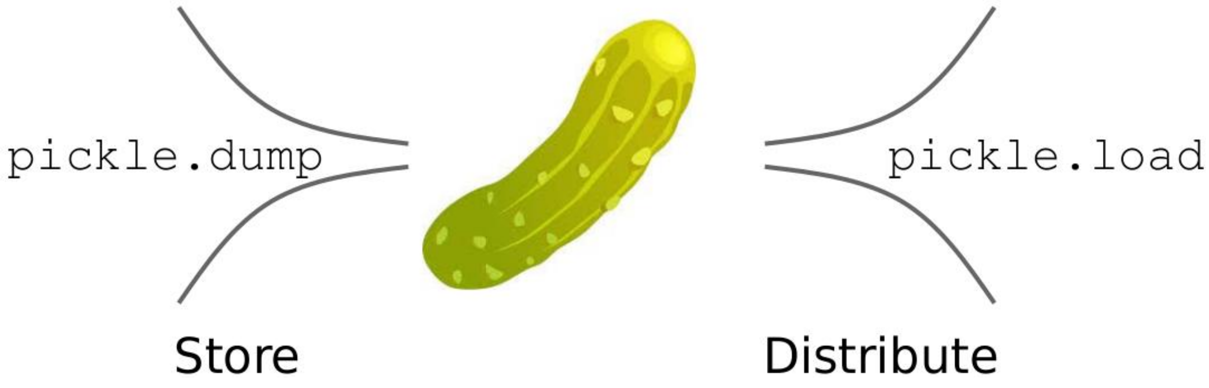
Running the XGB Algorithm

- **The hyperparameters tuning resulted in an improvement in both the testing accuracy and F1 Score.**

```
from sklearn import metrics
from sklearn.metrics import f1_score
from xgboost import XGBClassifier
import pickle

xgb = XGBClassifier(learning_rate =0.1, n_estimators=2000, max_depth=2,
      min_child_weight=4, gamma=0.5, subsample=1.0, colsample_bytree=0.3,
      objective= 'binary:logistic', nthread=4, scale_pos_weight=1,seed=27)

xgb.fit(x_train, y_train)
y_train_predict = xgb.predict(x_train)
y_test_predict = xgb.predict(x_test)

accuracy1 = round(accuracy_score(y_train, y_train_predict), 5)
accuracy2 = round(accuracy_score(y_test, y_test_predict), 5)
accuracy3 = round(f1_score(y_test, y_test_predict), 5)

print ('Training accuracy (optimized):', accuracy1, '(pre-optimized 0.73842)')
print ('Testing accuracy (optimized):', accuracy2, '(pre-optimized 0.72469)')
print ('F1 Score (optimized):', accuracy3, '(pre-optimized 0.70142)')

# save the optimised trained model
print ('\nSave the Optimised XGB Model')
model = xgb.fit(dfx, dfy)
filename = 'xgbopt_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

```
Training accuracy (optimized): 0.73813 (pre-optimized 0.73842)
Testing accuracy (optimized): 0.725 (pre-optimized 0.72469)
F1 Score (optimized): 0.70154 (pre-optimized 0.70142)

Save the Optimised XGB Model
```

SAVING THE TRAINED XGB MODEL



PART 6

By : Koh Swee Guan

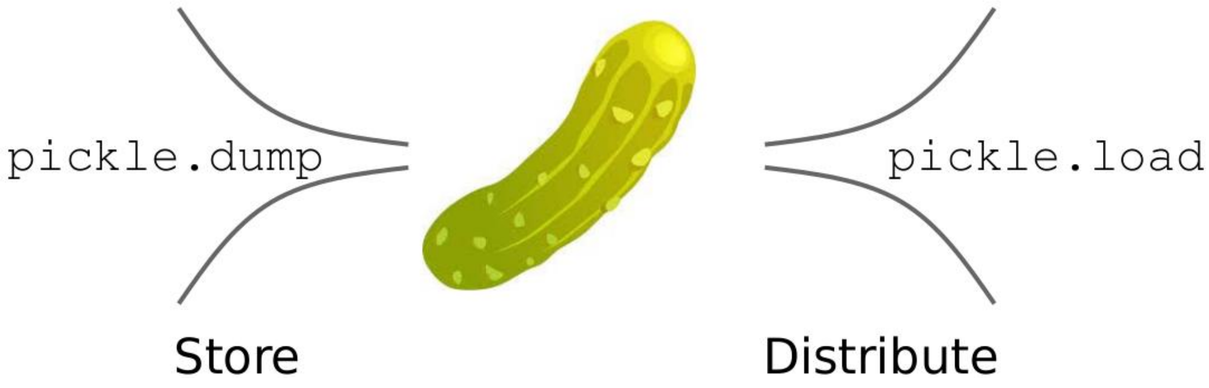
Using Optimized XGB Algorithm for Prediction

Using Optimized XGB Algorithm for Prediction

Getting Health Information of Patient & Predicting

```
This alogorithm will try to predict if you are likely to have Cardiovascular Disease.
The following information is needed for the prediction.

Please input your name: Ms Zhang
Please input your age (in years): 35
Please input 1 for female & 2 for male: 1
Please input your weight in kg: 88
Please input your height in cm: 150
Please input your Systolic Blood Pressure (the higher value) in mmHg: 165
Please input your Diastolic Blood Pressure (the lower value) in mmHg: 140
Please input 1 for normal cholesterol, 2 for above normal cholesterol & 3 for well above normal cholesterol: 2
Please input 1 for normal gluclose, 2 for above normal gluclose & 3 for well above normal glucose: 2
Please input 1 for smoker & 0 for non-smoker: 0
Please input 1 for drinker & 0 for non-drinker: 0
Please input 1 for active & 0 for non-active: 0

----------------------------------------------

Your name : Ms Zhang
Your age is 35 years old
Your weight is 88 kg
Your height is 150 cm
Your Systolic Blood Pressure is 165 mmHg
Your Diastolic Blood Pressure is 140 mmHg
Your cholesterol is above normal
Your glucose is above normal
You are female
You are a non-smoker
You are non-alcohol drinker
You are non-active

Is the information correct (Y/N)?y

----------------------------------------------
```

LOADING THE TRAINED XGB MODEL



Using Optimized XGB Algorithm for Prediction

Getting Information of Patient & Predicting

```
Is the information correct (Y/N)?y

---------------------------------------------


Your BMI is 39.11 and you are in Obese Category
Your Blood Pressure Category is Hypertension Stage 2

Hello, Ms Zhang

The probability of you having or to have a Cardiovascular Disease is high. :(
You must visit a doctor to check it.

-----------------------------------------------------------------------------------------------------

Thank you for using this tool.
This prediction of Cardiovascular Disease is carried out using Machine Learning.
It is important that the user understands this is still a prediction and not an absolute.
The authors/developers of the tools are in no way liable for outcomes following the use of the tools.

-----------------------------------------------------------------------------------------------------
```

Blood Pressure Stages

Blood Pressure Category	Systolic mm Hg (upper #)		Diastolic mm Hg (lower #)
Normal	less than 120	and	less than 80
Elevated	120-129	and	less than 80
High Blood Pressure (Hypertension) Stage 1	130-139	or	80-89
High Blood Pressure (Hypertension) Stage 2	140 or higher	or	90 or higher
Hypertensive Crisis (Seek Emergency Care)	higher than 180	and/or	higher than 120

Source: American Heart Association

BMI Classification

BMI	Category
Lower than 18.5	Underweight
18.5 up to 25	Optimal
25 up to 30	Overweight
30 upwards	Obese

https://www.prokerala.com/health/bmi.htm

Documents/MyPython_Koh_Swee | CVD Prediction Program - Jupyte

localhost:8888/notebooks/Documents/MyPython_Koh_Swee_Guan/CVD%20Prediction%20Program.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown

CardioVascular Disease Prediction using XGB Algorithm

- Please note that the model is developed using Kaggle Cardiovascular Disease dataset
- https://www.kaggle.com/sulianova/cardiovascular-disease-dataset
- The dataset consists of 70 000 records of patients data, 11 features + target.
- The accuracy of the data are not verified.
- Do note that this is an exercise to learn the various machine learning methods.
- This is purely a prediction exercise and the results may not be fair, accountable or transparent (FAT ML)
- Please note that it is a prediction and is not absolute.
- The authors/developers of this tool are in no way liable for outcomes following the use of the tools.
- Submited by Team 2 of K6312 Module

In [1]:

```
# Getting input data from users
import pandas as pd
import numpy as np
import pickle

def inputNumber(message):
  while True:
    try:
       userInput = int(input(message))
    except ValueError:
       print("Please input an integer! Try again.")
       continue
    else:
       return userInput
       break

print ('\nThis alogorithm will try to predict if you are likely to have Cardiovascular Disease.')
print ('The following information is needed for the prediction.\n')
```

Thank you!

Python
Pandas

Questions