Prediction of hearth disease:

The following analyses attempt to build a model able to classify potential cardiopathic patients from healthy controls taking advantage of a constellation of biomarkers. In this excercise, explainability is at a prime since once obtained a reliable classifier it would be useful to identify those biomarkers (or their interactions) that best predict the onset of hearth disease.

In [ ]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score
from lightgbm import LGBMClassifier
from rfpimp import *
```

Data:

UCI's heart disease prediction dataset (License: CC BY 4.0) This dataset contains 14 attributes. The "target" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. In this work I will simply attempt to distinguish presence (values 1,2,3,4) from absence (value 0) of heart disease.

```
-- 1.  (age)  age in years
-- 2.  (sex)  1 = male; 0 = female
-- 3.  (cp)   chest pain type
      -- Value 1: typical angina
      -- Value 2: atypical angina
      -- Value 3: non-anginal pain
      -- Value 4: asymptomatic
-- 4.  (trestbps)  resting blood pressure (in mm Hg on admission to the hospital)
-- 5.  (chol) serum cholestoral in mg/dl
-- 6.  (fbs)  fasting blood sugar > 120 mg/dl  (1 = true; 0 = false)
-- 7.  (restecg) resting electrocardiographic results
      -- Value 0: normal
      -- Value 1: having ST-T wave abnormality (T wave inversions and/or ST
                  elevation or depression of > 0.05 mV)
      -- Value 2: showing probable or definite left ventricular hypertrophy
                  by Estes' criteria
-- 8.  (thalach)   maximum heart rate achieved
-- 9.  (exang)     exercise induced angina (1 = yes; 0 = no)
-- 10.  (oldpeak)  ST depression induced by exercise relative to rest
-- 11.  (slope) slope: the slope of the peak exercise ST segment
      -- Value 1: upsloping
      -- Value 2: flat
      -- Value 3: downsloping
-- 12.  (ca)  number of major vessels (0-3) colored by flourosopy
-- 13.  (thal) thalassemia  3 = normal; 6 = fixed defect; 7 = reversable defect
-- 14.  (diagnosis - the predicted attribute) diagnosis of heart disease (angiographic disease status)
                  -- Value 0: < 50% diameter narrowing
                  -- Value 1: > 50% diameter narrowing
```

In [ ]:
```
# load data
mypath = '/Users/ale/ownCloud/4-Python/ML_course/Data/'
data = pd.read_csv(mypath+'heartdisease.cleveland.csv')
```

Data Preprocessing:

1 - Give meaningful names to the features

2 - Binarise diagnosis (1 cardiopathic 0 normal)

3 - Remove missing values (indicated by the character '?')

4 - Harmonise format of features (float)

5 - Scale continuous variables for Logistic regression and Naive Bayes

6 - Split data in dependent (y) and independent (X) features

7 - Check for collinearity of features

In [ ]:
```
# preprocess
# name columns
columns = ['age','sex', 'cp','trestbps','chol','fbs', 'restecg','thalach','exang', 'oldpeak','slope', 'ca', 'thal','diagnosis']

data.columns = columns

#distinguish presence (values >0) from absence (value = 0) of heart disease
data['diagnosis'] = data['diagnosis'].apply(lambda x: 1 if x>0 else 0)

# remove missing data
data=data.replace('?',np.nan).dropna()

data[['ca','thal']]=data[['ca','thal']].astype(float)

scaler = StandardScaler()
data[['age','trestbps','chol','thalach']] = scaler.fit_transform(data[['age','trestbps','chol','thalach']])

data.head()
```

Out [ ]:

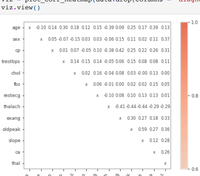
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	diagnosis
0	1.381810	1.0	4.0	1.597628	0.742464	0.0	2.0	-1.813215	1.0	1.5	2.0	3.0	3.0	1
1	1.381810	1.0	4.0	-0.656413	-0.353882	0.0	2.0	-0.897846	1.0	2.6	2.0	2.0	7.0	1
2	-1.938123	1.0	3.0	-0.092903	0.050035	0.0	0.0	1.630318	0.0	3.5	3.0	0.0	3.0	0
3	-1.495465	0.0	2.0	-0.092903	-0.834736	0.0	2.0	0.976483	0.0	1.4	1.0	0.0	3.0	0
4	0.164501	1.0	2.0	-0.656413	-0.219243	0.0	0.0	1.238017	0.0	0.8	1.0	0.0	3.0	0

In [ ]:
```
data.dtypes
```

Out [ ]:
```
age          float64
sex          float64
cp           float64
trestbps     float64
chol         float64
fbs          float64
restecg      float64
thalach      float64
exang        float64
oldpeak      float64
slope        float64
ca           float64
thal         float64
diagnosis      int64
dtype: object
```

In [ ]:
```
# Split data
X = data
y = data.pop('diagnosis')
```

In [ ]:
```
# No collinearity between the adopted variables
viz = plot_corr_heatmap(data.drop(columns = 'diagnosis'), figsize=(7,5))
viz.view()
```



In [ ]:
```
# Diagnostic categories are bilanced - accuracy can be used as metrics
y.value_counts(normalize=True)
```

Out [ ]:
```
0    0.537162
1    0.462838
Name: diagnosis, dtype: float64
```

Modelling:

1 - 5 Fold cross-validation of 3 models: Logistic regression, Random Forest, Naive-Bayes.

2 - Creation and training of ensemble model from the above models.

3 - Cross-validation grid-search to optimise components of the ensemble (soft voting)

4 - Model selection based on accuracy (target classes are rather balanced)

In [ ]:
```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier,VotingClassifier
import numpy as np

clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()

print('5-fold cross validation:\n')

labels = ['Logistic Regression', 'Random Forest', 'Naive Bayes']

for clf, label in zip([clf1, clf2, clf3], labels):

    scores = model_selection.cross_val_score(clf, X, y,
                                             cv=5,
                                             scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
5-fold cross validation:

Accuracy: 0.83 (+/- 0.03) [Logistic Regression]
Accuracy: 0.82 (+/- 0.06) [Random Forest]
Accuracy: 0.84 (+/- 0.05) [Naive Bayes]
```

In [ ]:
```
VotingClassifier(estimators=list(zip(labels, [clf1,clf2,clf3])), weights=[1,1,1])
```

Out [ ]:
```
VotingClassifier
Logistic Regression | Random Forest | Naive Bayes
LogisticRegression | RandomForestClassifier | GaussianNB
```

In [ ]:
```
eclf = VotingClassifier(estimators=list(zip(labels, [clf1,clf2,clf3])), weights=[1,1,1])


labels = ['Logistic Regression', 'Random Forest', 'Naive Bayes', 'Ensemble']
for clf, label in zip([clf1, clf2, clf3, eclf], labels):

    scores = model_selection.cross_val_score(clf, X, y,
                                             cv=5,
                                             scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.83 (+/- 0.03) [Logistic Regression]
Accuracy: 0.82 (+/- 0.06) [Random Forest]
Accuracy: 0.84 (+/- 0.05) [Naive Bayes]
Accuracy: 0.84 (+/- 0.04) [Ensemble]
```

In [ ]:
```
GridSearchCV(estimator=eclf, param_grid=params, cv=5)
```

Out [ ]:
```
GridSearchCV
estimator: VotingClassifier
Logistic Regression | Random Forest | Naive Bayes
LogisticRegression | RandomForestClassifier | GaussianNB
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV


params = {'Logistic Regression__C': [1.0, 100.0],
          'Random Forest__n_estimators': [20, 200],}

grid = GridSearchCV(estimator=eclf, param_grid=params, cv=5)
grid.fit(X, y)

cv_keys = ('mean_test_score', 'std_test_score', 'params')

for r, _ in enumerate(grid.cv_results_['mean_test_score']):
    print("%0.3f +/- %0.2f %r"
          % (grid.cv_results_[cv_keys[0]][r],
             grid.cv_results_[cv_keys[1]][r] / 2.0,
             grid.cv_results_[cv_keys[2]][r]))
```

```
0.838 +/- 0.02 {'Logistic Regression__C': 1.0, 'Random Forest__n_estimators': 20}
0.843 +/- 0.02 {'Logistic Regression__C': 1.0, 'Random Forest__n_estimators': 200}
0.838 +/- 0.02 {'Logistic Regression__C': 100.0, 'Random Forest__n_estimators': 20}
0.845 +/- 0.02 {'Logistic Regression__C': 100.0, 'Random Forest__n_estimators': 200}
```

Conclusions:

Models' performance did not differ significantly (when taking into account standard deviations the scores are all overlapping) Ensemble model did outperform the other models only after hyperparameters tuning via gridsearch. However, the magnitude is still marginal and not enough to justify the sacrifice of explainability in adopting the ensamble vs a logistic regression. To sum-up, since explainability is at a prime and difference in performance between models is marginal, logistic regression is an obvious choice.

In [ ]:
```
# Check feature importance to estimate the most predictive biomarkers
from sklearn.linear_model import LogisticRegression

m = LogisticRegression()
m.fit(X, y)

# Standardised coefficients
coeffs=list(zip(np.std(X, 0).index.values,(np.std(X, 0).values*m.coef_)[0]))
```

The standardised coefficients from the logistic regression indicate that number of major vessels ('ca') and thalassemy ('thal') are the most influential biomarkers in predicting hearth pathology.

In [ ]:
```
sorted(coeffs, key=lambda coeff: coeff[1])
```

Out [ ]:
```
[('thalach', -0.4596014333305303),
 ('fbs', -0.2298714217251798),
 ('age', -0.09899172016640025),
 ('chol', 0.20135313819900062),
 ('restecg', 0.2559794935627897),
 ('slope', 0.2898606269335845),
 ('oldpeak', 0.3418796454568085),
 ('exang', 0.3621227648378705),
 ('trestbps', 0.36548728033039185),
 ('sex', 0.4170859806195945),
 ('cp', 0.5146650848314757),
 ('thal', 0.7055500990059511),
 ('ca', 1.0533639299011794)]
```

Final thougths

Ensemble model did not perform better than its single components. One possible cause for the lack of improvement is that probabilities from the single models did not reflect the called class - i.e. the model probabilities were not calibrated.

Also, sex seems to play a rather important role but the number of male/females is biased against females. The analysis could benefit from removing such biases.