SC1015: MINI PROJECT

Heart Disease dataset from UCI

Weng Pei He, Neoh Kai Xiang, Tan Ding Jiang

1

Practical Motivation

Why this data set?

Practical Motivation

Heart Disease is the leading cause of death in Singapore

Angiography is usually not packaged together with common health checkups, so it is difficult to correctly ascertain constricted blood vessels without ordering a specialized test.

Problem Definition

How can we accurately predict the presence of ***heart disease*** (defined as narrowing of blood vessels) with data that are commonly obtained in regular health checkups?

2

Data Extraction and Analysis

Cleaning the weeds

Data Extraction

	Age	Sex	CPainType	RestBP	Chol	FBSugar	RestECG	MaxHR	ExAng	OldPeak	STSlope	CA	Thal	HeartDisease
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
915	70.0	1.0	4.0	115	0	0	1	92	1	0	2	?	7	1
916	70.0	1.0	4.0	140	0	1	0	157	1	2	2	?	7	3
917	72.0	1.0	3.0	160	0	?	2	114	0	1.6	2	2	?	0
918	73.0	0.0	3.0	160	0	0	1	121	0	0	1	?	3	1
919	74.0	1.0	2.0	145	0	?	1	123	0	1.3	1	?	?	1

UCI Machine Learning Repository

Heart Disease Data Set

.data file format with csv

Original Data Size: 920

Data Cleaning - Drop

```
[6] df.isnull().sum()

Age               0
Sex               0
CPainType         0
RestBP           59
Chol             30
FBSugar          90
RestECG           2
MaxHR            55
ExAng            55
OldPeak          62
STSlope         309
CA              611
Thal            486
HeartDisease      0
dtype: int64
```

Some columns are filled with NULL values

```
In [ ]: df = df.drop(columns = ['CA', 'Thal'])
        df
```

Out[23]:

	Age	Sex	CPainType	RestBP	Chol	FBSugar	RestECG	MaxHR	ExAng	OldPeak	STSlope	HeartDisease
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
915	70.0	1.0	4.0	115	0	0	1	92	1	0	2	1
916	70.0	1.0	4.0	140	0	1	0	157	1	2	2	3
917	72.0	1.0	3.0	160	0	NaN	2	114	0	1.6	2	0
918	73.0	0.0	3.0	160	0	0	1	121	0	0	1	1
919	74.0	1.0	2.0	145	0	NaN	1	123	0	1.3	1	1

920 rows × 12 columns

As more than half the values in CA and Thal are missing, we have decided to drop them from the dataset.

Data Cleaning – Drop

```
df[df.isnull().sum(axis=1) >= 2]
#an example of the rows with 2 or more missing values
```

	Age	Sex	CPainType	RestBP	Chol	FBSugar	RestECG	MaxHR	ExAng	OldPeak	STSlope	HeartDisease
316	63.0	1.0	3.0	NaN	0	0	2	NaN	NaN	NaN	NaN	1
326	74.0	1.0	3.0	NaN	0	0	0	NaN	NaN	NaN	NaN	0
329	51.0	1.0	4.0	NaN	0	1	1	NaN	NaN	NaN	NaN	2
332	55.0	1.0	3.0	NaN	228	0	1	NaN	NaN	NaN	NaN	3
333	54.0	1.0	4.0	NaN	0	0	1	NaN	NaN	NaN	NaN	3
...	...	...	...	...	...	...	...	...	...	...	...	...
889	62.0	0.0	1.0	140	0	NaN	0	143	0	0	NaN	2
905	65.0	1.0	4.0	145	0	NaN	1	67	0	NaN	NaN	3
907	65.0	1.0	4.0	160	0	1	1	122	0	NaN	NaN	3
908	66.0	0.0	4.0	155	0	NaN	0	90	0	0	NaN	1
914	69.0	1.0	4.0	NaN	0	0	1	NaN	NaN	NaN	NaN	3

Drop rows with 2 or more missing values

Data Cleaning – Updating data types

```
df = df[(df != '?')] #replaces the ? with NaN
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Age           920 non-null    float64
 1   Sex           920 non-null    float64
 2   CPainType     920 non-null    float64
 3   RestBP        861 non-null    object
 4   Chol          890 non-null    object
 5   FBSugar       830 non-null    object
 6   RestECG       918 non-null    object
 7   MaxHR         865 non-null    object
 8   ExAng         865 non-null    object
 9   OldPeak       858 non-null    object
 10  STSlope       611 non-null    object
 11  CA            309 non-null    object
 12  Thal          434 non-null    object
 13  HeartDisease  920 non-null    int64
dtypes: float64(3), int64(1), object(10)
memory usage: 100.8+ KB
```

Misleading data types

```
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Age           818 non-null    float64
 1   Sex           818 non-null    object
 2   CPainType     818 non-null    object
 3   RestBP        816 non-null    float64
 4   Chol          810 non-null    float64
 5   FBSugar       750 non-null    object
 6   RestECG       818 non-null    object
 7   MaxHR         818 non-null    float64
 8   ExAng         818 non-null    object
 9   OldPeak       818 non-null    float64
 10  STSlope       609 non-null    object
 11  HeartDisease  818 non-null    object
dtypes: float64(5), object(7)
memory usage: 76.8+ KB
```

Data Cleaning – Reclassify response variable

```
#maps heartdisease to categorical values: true if value is 1,2,3 or 4, false if value is 0
cleandf.loc[cleandf['HeartDisease'] == 0 , 'HeartDisease'] = 'False'
cleandf.loc[cleandf['HeartDisease'] == 1 , 'HeartDisease'] = 'True'
cleandf.loc[cleandf['HeartDisease'] == 2 , 'HeartDisease'] = 'True'
cleandf.loc[cleandf['HeartDisease'] == 3 , 'HeartDisease'] = 'True'
cleandf.loc[cleandf['HeartDisease'] == 4 , 'HeartDisease'] = 'True'
```

Categorical data type

0 remains as is: classified as no heart disease

1 to 4 classified as 1 (high)

2

Data Extraction and Analysis

Exploratory Data Analysis

Overview

All the variables apart from *CA* and *Thal* were used in the model

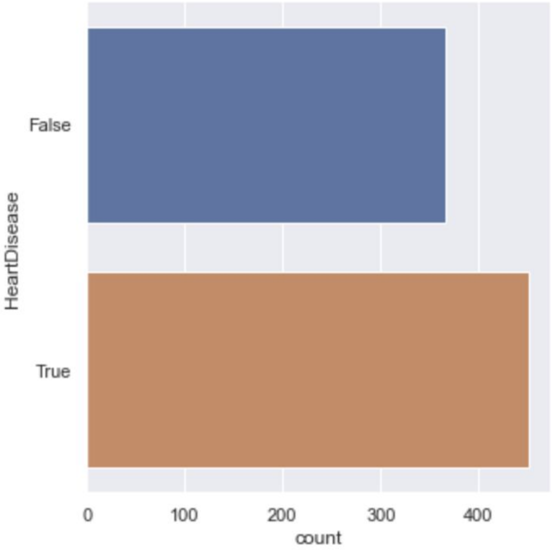
- Age
- Sex
- CPainType
- RestBP
- Chol
- FBSugar
- RestECG
- MaxHr
- ExAng
- OldPeak
- StSlope

Overview

All the variables apart from *CA* and *Thal* were used in the model

- Age
- Sex
- CPainType
- RestBP
- Chol
- FBSugar
- RestECG
- MaxHr
- ExAng
- OldPeak
- StSlope

Data Analysis – Response variable

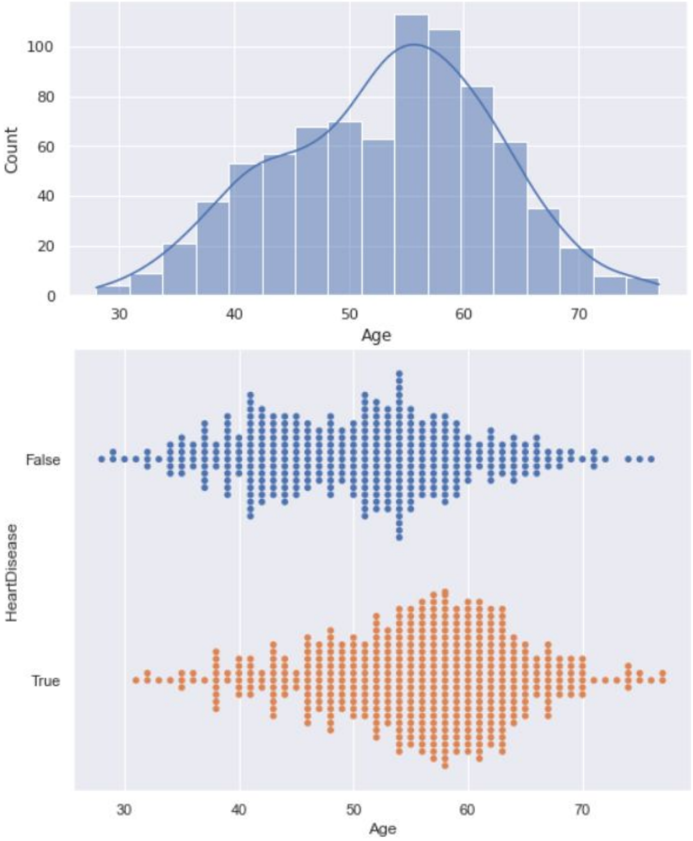


HeartDisease is defined as presence of clogged arteries

Response variable (HeartDisease) is evenly spread between true and false

→ **no** need for over/undersampling

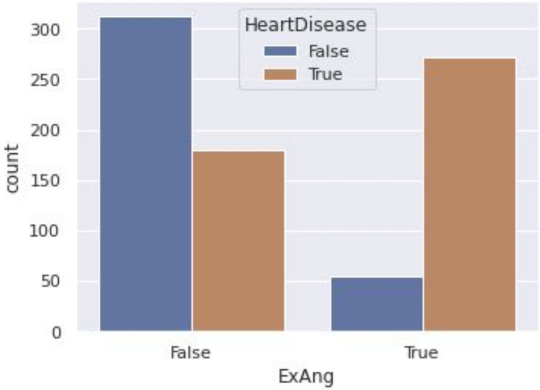
Data Analysis – Age



Age ranges from 30 to 77 years old

Patients with heart disease tended to be older

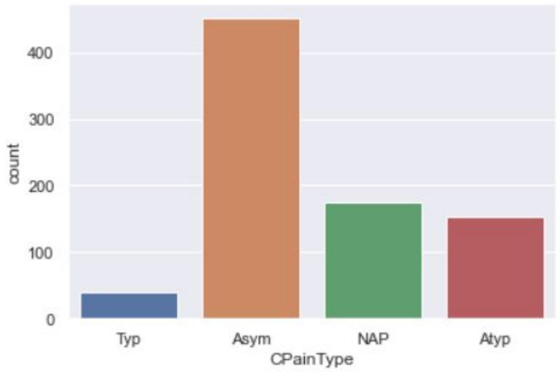
Data Analysis – ExAng

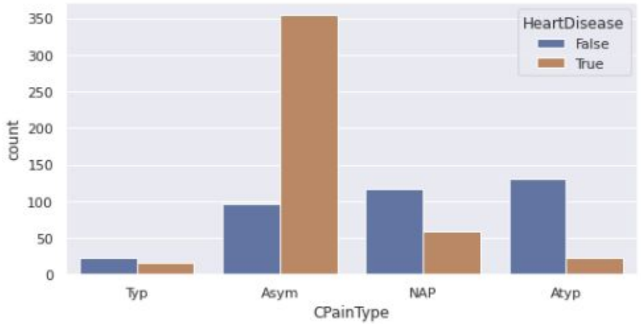


ExAng - Exercise Induced Angina

A large proportion of those with exercise-induced angina have heart disease.

Data Analysis – Chest pain type

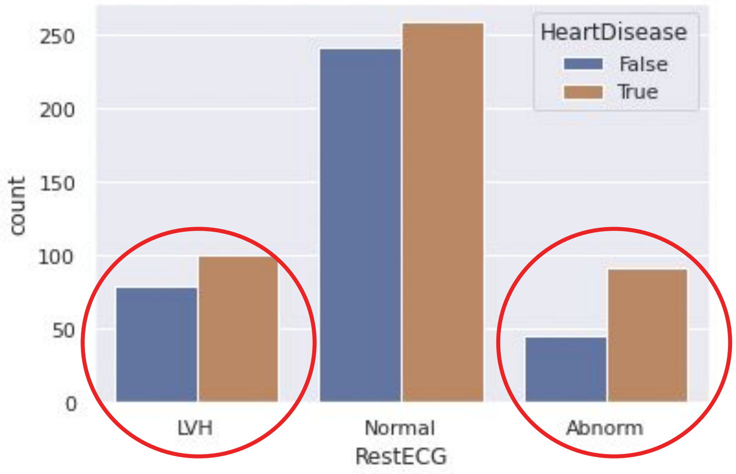




Chest Pain Type - 4 different categories:

1. asymptomatic
2. typical angina
3. atypical angina
4. non-anginal pain

Data Analysis – RestECG



For patients with ‘LVH’ and ‘Abnorm’ ECG patterns, there is a higher proportion of them afflicted with heart disease.

3

Imputation of Missing Values

With KNNImputer

Data Cleaning – Imputing Missing Values

Age	0
Sex	0
CPainType	0
RestBP	3
Chol	153
FBSugar	68
RestECG	0
MaxHR	0
ExAng	0
OldPeak	0
STSlope	209
HeartDisease	0

After our exploratory analysis, we were left with some missing or erroneous values.

KNN Imputer

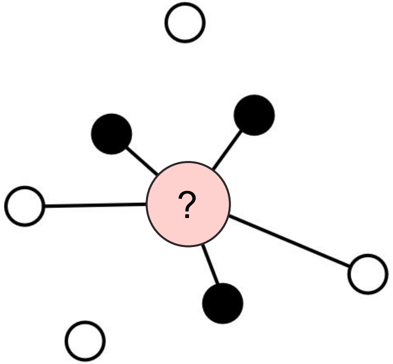


Image obtained from https://medium.datadriveninvestor.com/k-nearest-neighbours-knn-a9f8ba09cb8b?gi=516922eb55a6

- Imputes missing data based on the values of their nearest neighbours.
- Neighbours are obtained based on Euclidean Distance.
- Why we used it:
  - Preserves sample size and statistical power
  - Results in unbiased estimates, providing more accuracy and validity

KNN Imputer – Process

```
#Use minmaxscaler to scale the data first
labeldf_filled = labeldf.copy()
df_scaled = df.copy()
df_scaled = df_scaled.drop(columns = ['HeartDisease'])  #drops response variable: we will not be us

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns = df.columns)
df_scaled.head()
```

	Age	Sex	CPainType	RestBP	Chol	FBSugar	RestECG	MaxHR	ExAng	OldPeak	STSlope	HeartDisease
0	0.714286	1.0	0.000000	0.541667	0.285714	1.0	1.0	0.633803	0.0	0.556818	1.0	0.00
1	0.795918	1.0	1.000000	0.666667	0.388031	0.0	1.0	0.338028	1.0	0.465909	0.5	0.50
2	0.795918	1.0	1.000000	0.333333	0.277992	0.0	1.0	0.485915	1.0	0.590909	0.5	0.25
3	0.183673	1.0	0.666667	0.416667	0.318533	0.0	0.0	0.894366	0.0	0.693182	1.0	0.00
4	0.265306	0.0	0.333333	0.416667	0.229730	0.0	1.0	0.788732	0.0	0.454545	0.0	0.00

1. Data is normalized to reduce bias in the calculation of Euclidean Distance

KNN Imputer – Process

```
#Using KNNImputer on the normalized data:
from sklearn.impute import KNNImputer
imputer_cat = KNNImputer(n_neighbors=1)

df_cat = df_scaled.copy()

#performs imputation
df_cat = pd.DataFrame(imputer_cat.fit_transform(df_cat),columns = df_cat.columns)
```

Categorical data: Filled in with the value of the nearest neighbour.

```
imputer_num = KNNImputer(n_neighbors=3)

df_num = df_scaled.copy()

#performs imputation
df_num = pd.DataFrame(imputer_num.fit_transform(df_num),columns = df_num.columns)
```

Numeric data: Filled in with the mean of the value of the 3 nearest neighbours.

2. Imputer is used to fill in missing data in the columns.

KNN Imputer – Process

```
#transforms the data back to the original values after imputation
df_num = pd.DataFrame(scaler.inverse_transform(df_num),columns = df_num.columns)

#fills in dataframe
df['RestBP'].fillna(df_num['RestBP'], inplace=True)
df['Chol'].fillna(df_num['Chol'], inplace=True)
```

3. Data is denormalized and used to fill in the missing values in the columns.

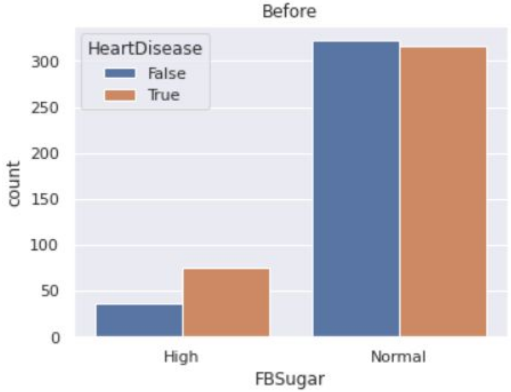
Before and After Imputation

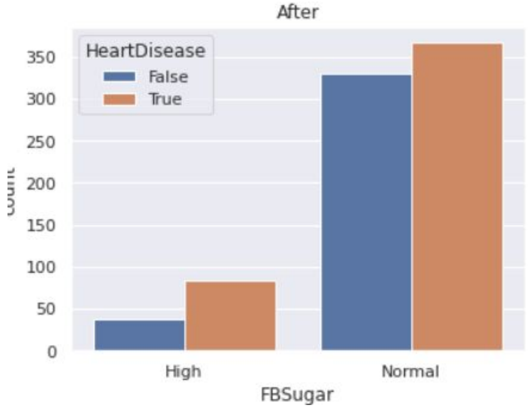
Age	0
Sex	0
CPainType	0
RestBP	3
Chol	153
FBSugar	68
RestECG	0
MaxHR	0
ExAng	0
OldPeak	0
STSlope	209
HeartDisease	0

Age	0
Sex	0
CPainType	0
RestBP	0
Chol	0
FBSugar	0
RestECG	0
MaxHR	0
ExAng	0
OldPeak	0
STSlope	0
HeartDisease	0

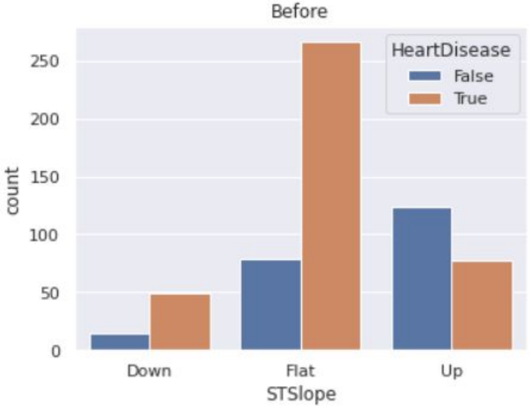
Before and After Imputation

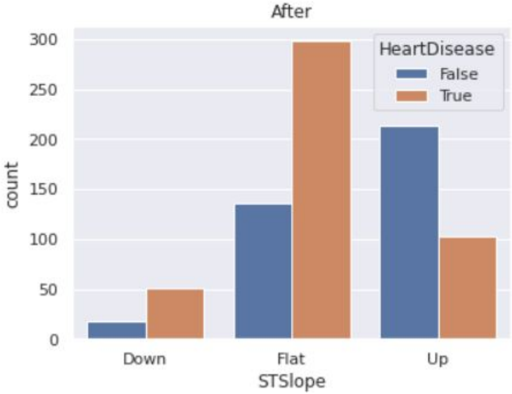
FBSugar:





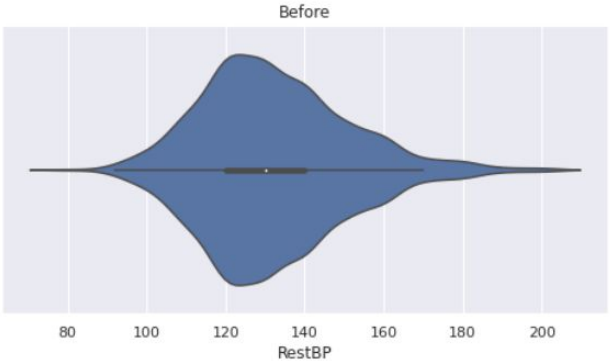
STSlope:

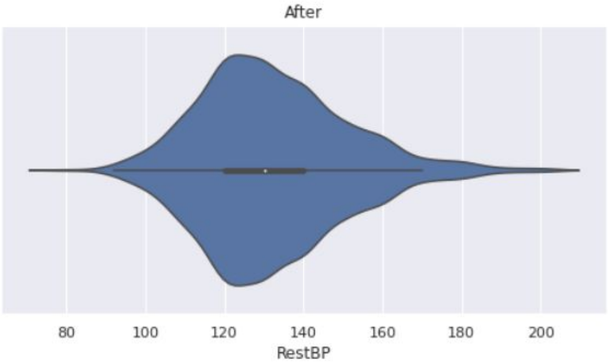




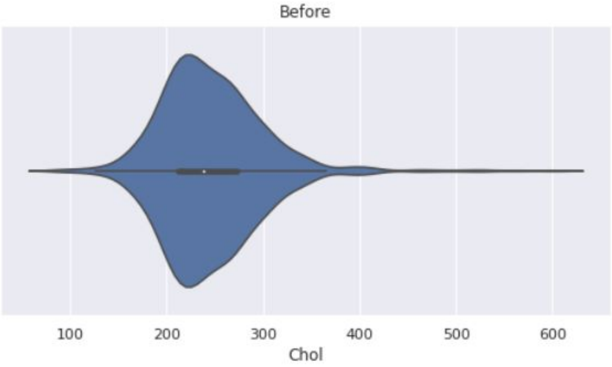
Before and After Imputation

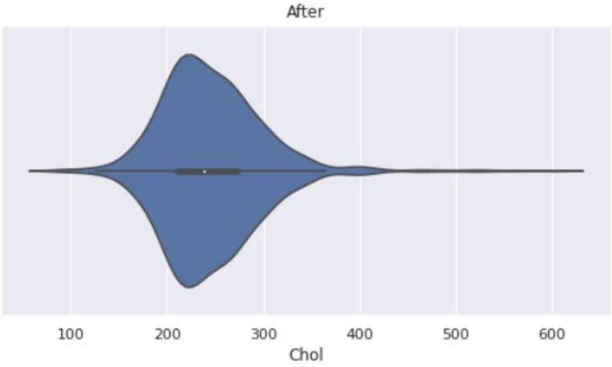
RestBP:





Chol:





4

Machine Learning

- Encoding of categorical predictors
- Decision Tree Classifier
- Random Forest Classifier
- Logistic Regression

Encoding of Categorical Predictors

0	Sex_F	818 non-null	float64
1	Sex_M	818 non-null	float64
2	CPainType_Asym	818 non-null	float64
3	CPainType_Atyp	818 non-null	float64
4	CPainType_NAP	818 non-null	float64
5	CPainType_Typ	818 non-null	float64
6	STSlope_Down	818 non-null	float64
7	STSlope_Flat	818 non-null	float64
8	STSlope_Up	818 non-null	float64

One-hot encoding of the nominal categorical variables: Sex, Chest Pain Type, and ST Slope

Encoding of Categorical Predictors

	RestECG	FBSugar	ExAng
0	2.0	1.0	0.0
1	2.0	0.0	1.0
2	2.0	0.0	1.0
3	0.0	0.0	0.0
4	2.0	0.0	0.0

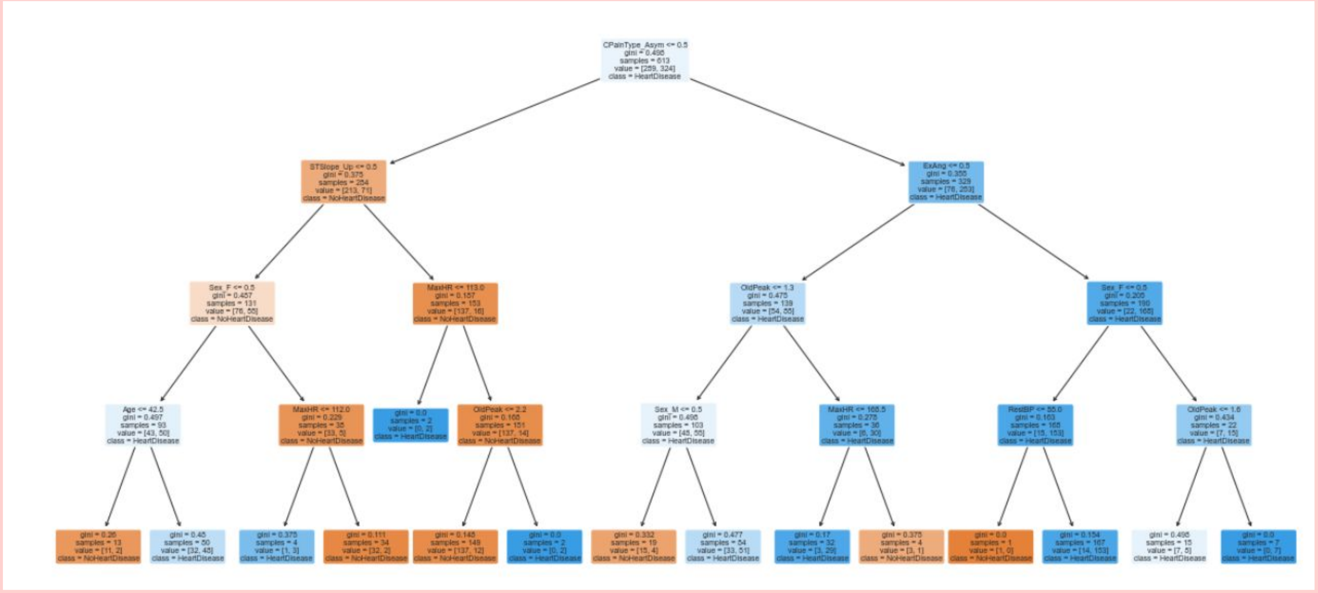
Ordinal predictors such as RestECG, FBSugar and ExAng are encoded in order.

Machine Learning Input

0	Age	818 non-null	float64
1	RestBP	818 non-null	float64
2	Chol	818 non-null	float64
3	MaxHR	818 non-null	float64
4	OldPeak	818 non-null	float64
5	RestECG	818 non-null	float64
6	FBSugar	818 non-null	float64
7	ExAng	818 non-null	float64
8	Sex_F	818 non-null	float64
9	Sex_M	818 non-null	float64
10	CPainType_Asym	818 non-null	float64
11	CPainType_Atyp	818 non-null	float64
12	CPainType_NAP	818 non-null	float64
13	CPainType_Typ	818 non-null	float64
14	STSlope_Down	818 non-null	float64
15	STSlope_Flat	818 non-null	float64
16	STSlope_Up	818 non-null	float64
17	HeartDisease	818 non-null	object

Decision Tree Classifier

- We start with this classic approach to binary classification
- Predicts presence of heart disease based on the numerically encoded predictors
- Tentative max depth of 4



Decision Tree Classifier

Train Data

Accuracy:	0.8189233278955954
F1:	0.8451882845188284
ROC_AUC:	0.8851403306420608

TPR:	0.9351851851851852
TNR:	0.6885813148788927

FPR:	0.31141868512110726
FNR:	0.06481481481481481

Test Data

Accuracy:	0.7853658536585366
F1:	0.8394160583941606
ROC_AUC:	0.830254391278013 4

TPR:	0.905511811023622
TNR:	0.5897435897435898

FPR:	0.41025641025641024
FNR:	0.09448818897637795

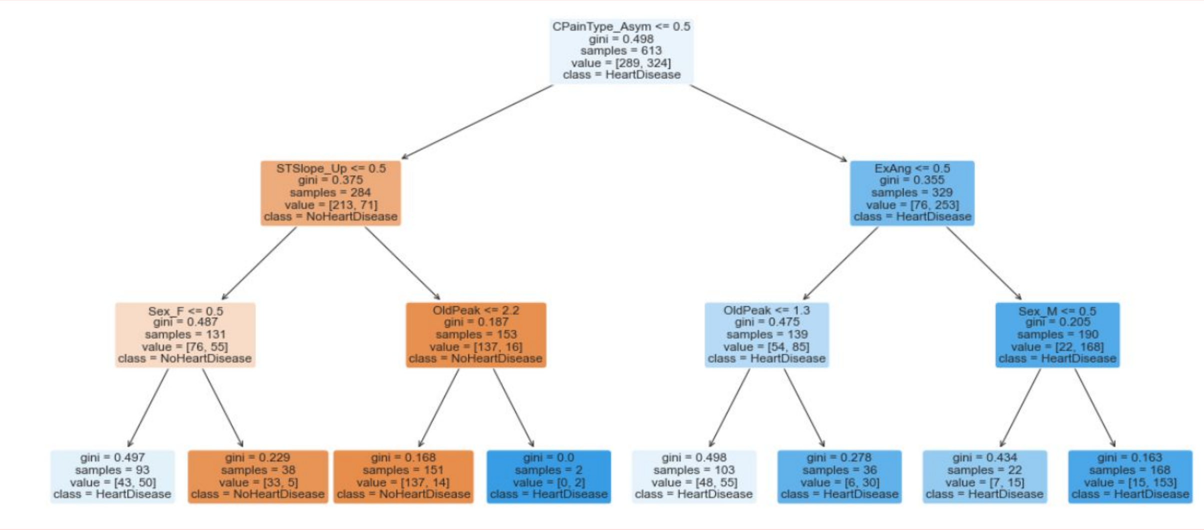
- Accuracy of around 0.8
- Seems better at predicting positive classes than negative classes.

Decision Tree Classifier with cross-validation

- To improve performance, we use cross-validation with GridSearch to find the **optimal depth of decision tree**
- Used Area Under Curve of Receiving Operator Characteristic (ROC_AUC) as evaluation metric.

Best depth:

```
{'max_depth': 3}
0.8173307519213786
```



Decision Tree Classifier with cross-validation

```
Train Data
Accuracy:        0.7748776508972267
F1:              0.8155080213903744
ROC_AUC:         0.8556164295783673

TPR:     0.941358024691358
TNR:     0.5882352941176471

FPR:     0.4117647058823529
FNR:     0.05864197530864197
```


```
Test Data
Accuracy:        0.775609756097561
F1:              0.8368794326241135
ROC_AUC:         0.8361094286291137

TPR:     0.9291338582677166
TNR:     0.5256410256410257

FPR:     0.47435897435897434
FNR:     0.07086614173228346
```

- Performance did not increase as expected for the current split.

Random Forest Classifier with Cross-Validation

- Using a **random forest classifier** might yield better results than simple decision trees
- It predicts presence of heart disease based on the **majority output** from a group of decision trees
- The **hyperparameters**: **Number of Trees**, and **Maximum depth** are tuned using GridSearch cross-validation
- Again, performance of the GridSearch is evaluated with the ROC_AUC score

```
# Fetch the best set of Hyper-parameter
print(hpGrid.best_params_)

# Print the score (accuracy) of the bes
print(np.abs(hpGrid.best_score_))
```

```
{'max_depth': 3, 'n_estimators': 200}
0.8801657600015034
```

Random Forest Classifier with Cross-Validation

```
Train Data
Accuracy:        0.8221859706362153
F1:              0.8399412628487518
ROC_AUC:         0.9039044811824511

TPR:     0.8827160493827161
TNR:     0.754325259515571

FPR:     0.2456747404844290
FNR:     0.11728395061728394
```


```
Test Data
Accuracy:        0.8146341463414634
F1:              0.8560606060606061
ROC_AUC:         0.871996769634565

TPR:     0.889763779527559
TNR:     0.6923076923076923

FPR:     0.3076923076923077
FNR:     0.11023622047244094
```

- Accuracy of around 0.82
- Improvement in performance from decision trees on all metrics.
- Able to predict positive and negative classes with more balance.

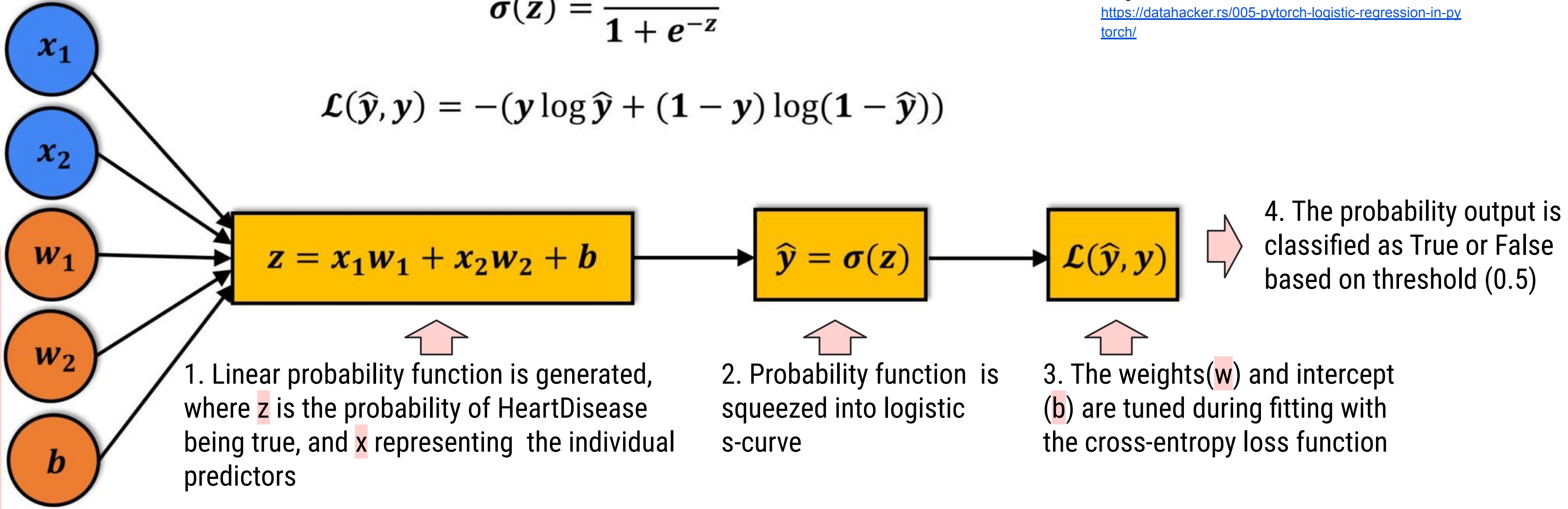
Logistic Regression with CV

- Finally, we will use **logistic regression** to check the performance of a **non tree**-based algorithm in predicting heart failure.
- It has the added advantage of **generating coefficients** or weights, allowing us to **evaluate the importance** of the predictor variables.
- The **hyperparameters** (C value and L1 ratio) are automatically tuned with cross-validation.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Image obtained from https://datahacker.rs/005-pytorch-logistic-regression-in-pytorch/

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$



Logistic Regression with CV

```
Train Data
Accuracy:         0.799347471451876
F1:               0.8133535660091047
ROC_AUC:          0.8908112264513648

TPR:      0.8271604938271605
TNR:      0.7681660899653979

FPR:      0.23183391003460208
FNR:      0.1728395061728395
```

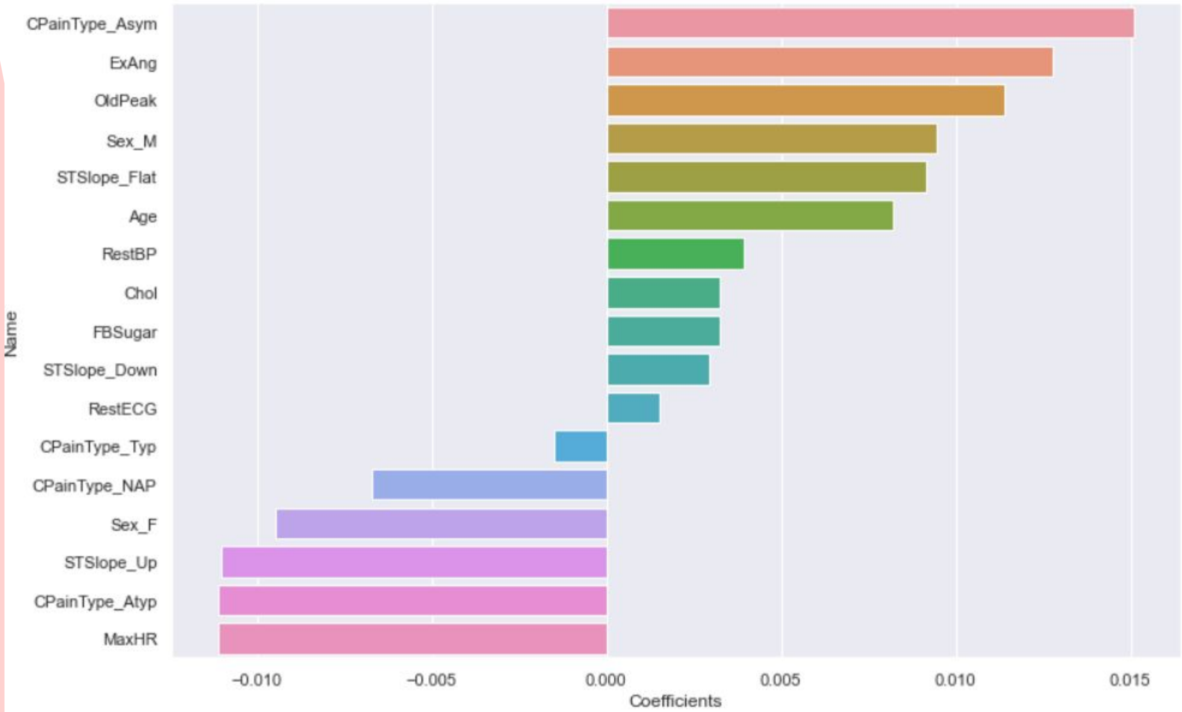

```
Test Data
Accuracy:         0.824390243902439
F1:               0.8615384615384616
ROC_AUC:          0.8718958207147184

TPR:      0.8818897637795275
TNR:      0.7307692307692307

FPR:      0.2692307692307692
FNR:      0.11811023622047244
```

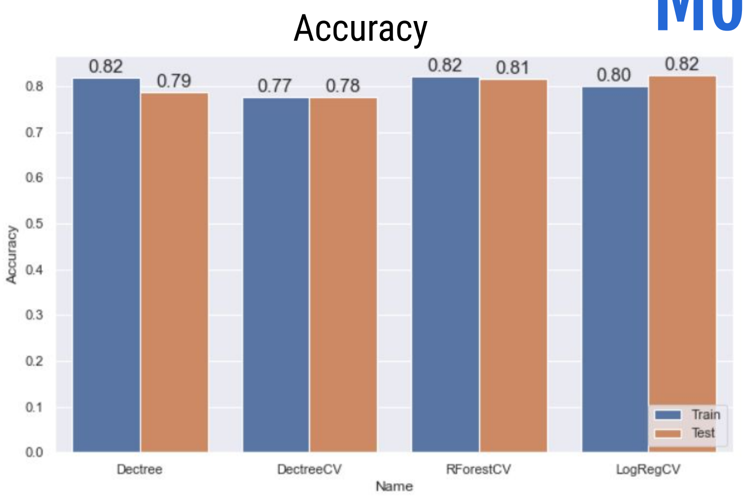
- Model performance was on par with decision tree classifier
- Smaller difference between TPR and TNR than decision tree

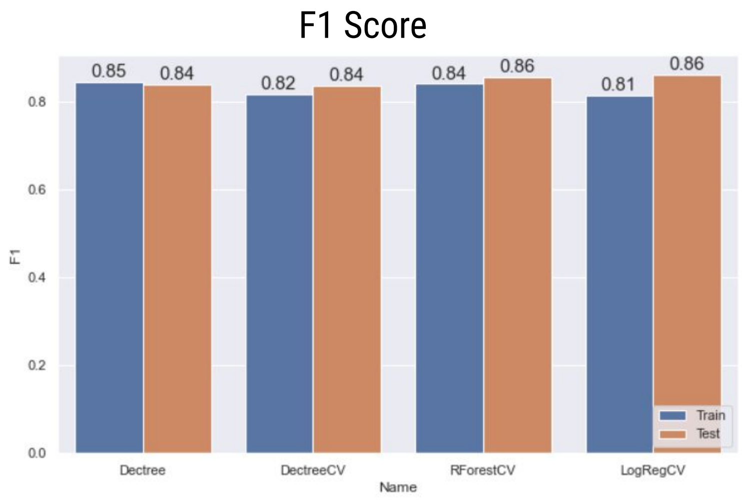
Logistic Regression with CV

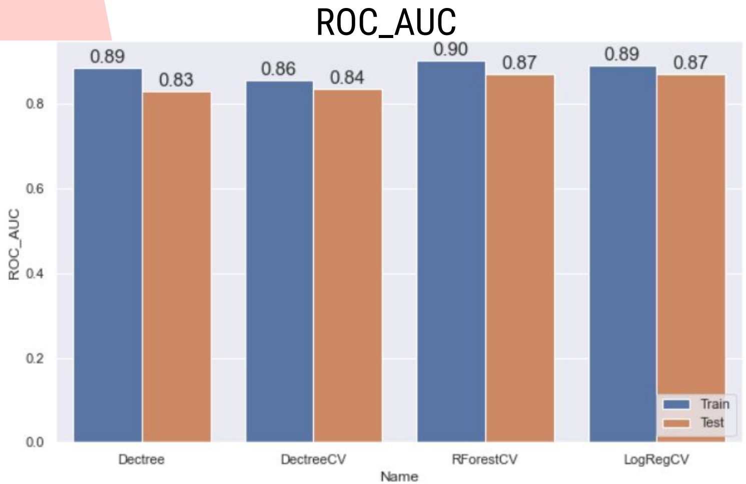


- The standardised coefficients allow us to compare the effects of each predictor variable on the probability of having heart disease.
- A positive coefficient means that the risk of having heart disease is increased when that variable is true, or when there is an increase in that variable.
- For example, presence of exercise induced angina (ExAng) would increase risk of heart disease more than having a high age will.

Model comparison





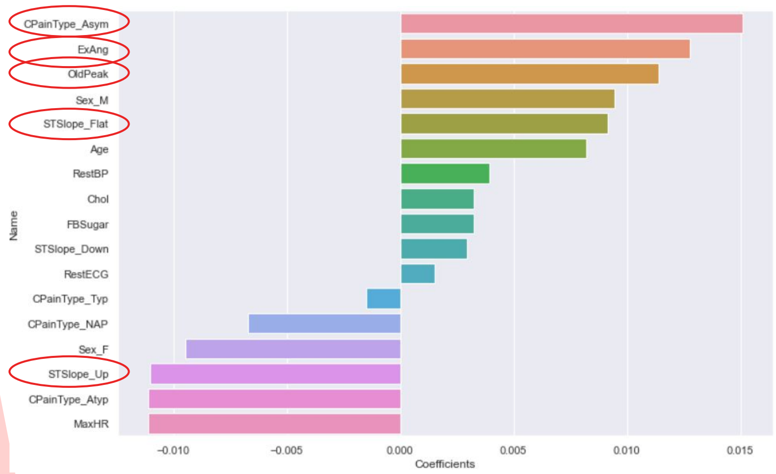


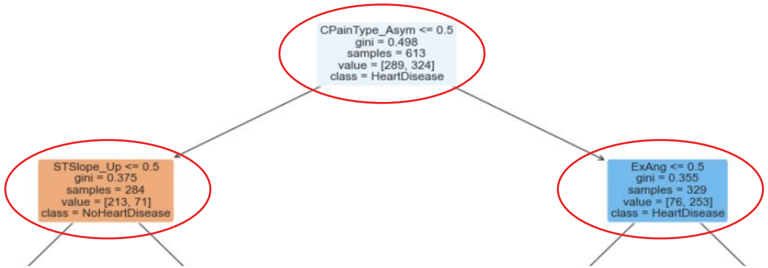
- The performance of the models are largely similar across the metrics
- The random forest classifier had the best performance for our current problem.

5

Data Driven Insights & Conclusion

Data-Driven Insights & Recommendations





- Asymptomatic Chest Pain is a leading predictor of the presence of heart disease.
  - Silent Myocardial Infarction (SMI), where atypical symptoms like indigestion, flu or a strained chest muscle are felt.
- Exercise - related variables such as exercise induced angina (ExAng), ST-Depression during exercise (OldPeak), and ST Slope during peak exercise are also important predictors.
  - Exercise stress test can be be a good preliminary test if you suspect the presence of heart disease.

Data-Driven Insights & Recommendations

- If you are older or Male, you face a higher risk of heart disease
    - To balance these risks, it is best to watch your diet and to keep other aggravating factors like Cholesterol and Fasting Blood Sugar low, by **eating healthily** and **exercising regularly!**

What we learnt

Medical terminologies to understand variables

Handling missing values
- Trade offs with deletion and imputation
- KNNImputer

Different ML models and Techniques
1. Random Forest
2. GridSearchCV
3. Logistic Regression

Outcome

- Our model was able to predict heart disease to a reasonable accuracy
- Can be used as a preliminary test to recommend if the patient is suitable for angiography

Thank You!