GROUP ASSIGNMENT PART 1

TECHNOLOGY PARK MALAYSIA

CT032-3-3-FAI

FURTHER ARTIFICIAL INTELLIGENCE

Submission Date: 10 March 2023

Lecturer Name: Ts. Nor Anis Asma Binti Sulaiman

Intake: APD3F2211CS(IS)

Assignment Tittle: Heart Disease Classification

Student Name & TP ID:

Wong Tik Soon TP054784

Lim Sheng Xue TP054683

Ng Eng Siong TP055095

Wong Horng Woei TP055241

INSTRUCTIONS TO CANDIDATES:

1 **Submit your assignment online via Moodle.**

2 **Students are advised to underpin their answers with the use of references (cited using the APA System of Referencing)**

3 **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld**

4 **Cases of plagiarism will be penalized**

5 **Ensure that all answers are presented clearly, including using appropriate font sizes and clarity of images. All important information and steps of the workings should be shown.**

6 **You must obtain 50% overall to pass this module.**

Table of Contents

List of Figures

Literature Review

According to the estimation of World Health Organization in 2020, cardiovascular disease (CVD) is the top cause of death around the world. The cases of death caused by CVD are around 17.9 million per year. The detection of CVD at an earlier stage is crucial to reduce the rate of mortality and the cost of treatment. To address this issue, machine learning (ML) techniques have been applied in the medical industry to assist doctors in diagnosing CVD. CVD could include heart disease and disorder of blood vessels. The following research focused solely on the prediction of heart disease. (WHO, n.d.)

The work of (Ali et al., 2021) could provide some insight into applying classification algorithms to detect the presence of heart disease. It used a dataset that contains the clinical information of the patients. Each of the records is labelled with 0 to indicate the absence of heart disease and 1 to indicate the presence of heart disease. The study compared the performance of 6 different classification algorithms on this task which include multilayer perceptron (MP), K-nearest neighbours (KNN), random forest (RF), decision tree (DT), logistic regression (LR) and AdaboostM1 (ABM1). Their performances are evaluated based on different metrics such as Accuracy, Precision, Recall, et cetera. Out of the six models, KNN, DT and LR achieved 100 per cent in all metrics. The feature importance values of the variables are computed based on their coefficients. From the result, it could be observed that different algorithms have ranked the importance of features differently. However, chest pain type (cp) is a very important feature that is given a high importance score by multiple models. Other important features include number of major vessels (0–3) colored by fluoroscopy (ca), age, maximum heart rate achieved (thalach), and ST depression induced by exercise relative to rest (oldpeak).

Feature Ranking	LR	ABM1	DT	RF
1st	cp	chol	cp	cp
2nd	slope	thalach	ca	oldpeak
3rd	restecg	age	age	ca
4th	thalach	oldpeak	oldpeak	thalach
5th	age	ca	thalach	thal

Figure 1: Important features ranked by different classification models (Ali et al., 2021)

By using feature selection techniques like Relief and Least Absolute Shrinkage and Selection Operator (LASSO), the performance of the prediction system could be improved. The study of (Ghosh et al., 2021) demonstrated the use of these 2 techniques to improve the performance of the classification model in predicting heart failure. It used a large dataset that combined 5 different datasets. Similar to other studies in this domain, the dataset contains clinical information to help the prediction. The task of the model is to classify the observations from 0 to 4, while 0 indicates no heart disease, and 1 to 4 indicates different stages of heart disease. Relief is a selection technique that used similar technique as the KNN algorithm to determine the feature importance. It assigns weight to features and gradually updates them according to their ability to distinguish the sample between different classes. On the other hand, LASSO works by shrinking the coefficient of the feature. The unimportant feature will have their coefficient shrink to 0, thus removing them from the model. The Relief technique has been shown to have a better effect in improving the classification model performance. The features ranked by Relief are shown in figure 2. The important features include serum cholesterol, resting electrocardiographic results, number of major vessels (0–3) colored by fluoroscopy.

Feature name	Feature code	Score
Age in years	age	0.19
Serum cholesterol	chol	0.867
fasting blood sugar	fbs	0.0233
resting electrocardiographic results	restecg	0.582
maximum heart rate	thalach	0.543
exercise induced angina	exang	0.0089
number of major vessels (0-3) colored by fluoroscopy	ca	0.581

Figure 2: Features ranked by Relief (Ghosh et al., 2021)

Based on a study by (Gárate-Escamila et al., 2020), chi-square test (CHI) is applied to categorize heart disease features for the Cleveland, Hungarian, and CH (Cleveland-Hungarian) datasets. Chi-square test (CHI) is a technique for determining the independence of two variables or features. A variable or feature that has a higher Chi-Square value is more dependent and this indicates that the variable or feature is compatible to be chosen for model training. Hence, CHI is suitable for feature filtering so that only the top features that relies on class label are chosen. In the study, the chi-square acquires the features of physical and physiological relevance that are associated with heart diseases. The author concluded that CHI-PCA has the highest accuracy and is more compatible for feature abstraction in prediction of heart diseases.

Dataset	Method	DT	GBT	LOG	MPC	RF	Average
Cleveland	Raw data %	0.3	0.0	1.2	1.1	0.1	0.54
	CHI %	0.5	2.1	1.0	0.1	0.1	0.76
	CHI-PCA %	1.3	0.4	0.6	0.1	0.5	0.58
Hungarian	Raw data %	1.3	1.1	2.5	2.5	1.6	1.8
	CHI %	1.6	0.6	1.7	1.0	0.8	1.14
	CHI-PCA %	2.2	0.3	0.6	2.6	0.6	1.26
CH	Raw data %	0.3	0.0	1.4	2.7	1.3	1.14
	CHI %	0.1	0.1	0.3	2.0	0.4	0.58
	CHI-PCA %	0.3	0.3	0.1	3.3	0.1	0.82
Average		0.87	0.54	1.04	1.71	0.61	0.96

In a study on hybrid technique for heart disease diagnosis, (Abdollahi & Nouri-Moghaddam, 2022) applied the FCBF feature selection method along with genetic algorithms (GA) to extract relevant features for their research. 13 features of heart disease diagnosis are listed out at the start of the study. GA's methodology is based on the fundamental of natural transformation. It is inspired by the theory of human growth, and it works by mutating the human population with a random "parent" in order to achieve the aim of reproduction of the best offspring. In feature selection, the best features are obtained through this concept.

Research done by Mansur Huang et al. (2021) provides a comprehensive review on the existing literature on the use of machine learning techniques for heart failure prediction. The authors found that various machine learning techniques have been used for this purpose, including random forest (RF), naive bayes (NB), logistic regression (LR) and support vector machines. They mentioned that the performance of these techniques may vary depending on the specific machine learning technique applied. By utilizing the 13 features in the dataset such as age, sex, cp (chest pain), trestbps (resting blood pressure), chol (cholesterol), fbs (fasting blood sugar), restecg (resting electrocardiographic), thalach (maximum heart rate), exang (exercise-induced angina), oldpeak, slope, ca (number of major vessels), and thal (thalassemia), they serve as input variables and the output variable consist of values of 0 and 1, indicates the absence and presence of heart failure.

ML model/result	Random Forest	Support Vector Machine	Naïve Bayes	Logistic Regression
Accuracy	0.88	0.85	0.86	0.88
Validation accuracy	0.83	0.90	0.79	0.72
Precision	0.84	0.81	0.85	0.87
Recall	0.97	0.86	0.90	0.94
F1-Score	0.90	0.82	0.88	0.90
Sensitivity	0.97	0.85	0.90	0.94
Specificity	0.76	0.75	0.80	0.81

ML model/results	Average Performance Score
Random Forest	0.88
Support Vector Machine	0.83
Naive Bayes	0.85
Logistic Regression	0.87

Figure 3: Different Classification Models Performance Score and Average Performance Score (Mansur Huang et al., 2021).

Each algorithm shows different performance for its accuracy, precision, recall, F1-score, sensitivity, and specificity. From figure 3, it can be observed that RF has achieve the best performance compared to SVM, NB, and LR. The authors have also done a research experiment with RF to find out the most significant feature to predict heart failure in figure 4.

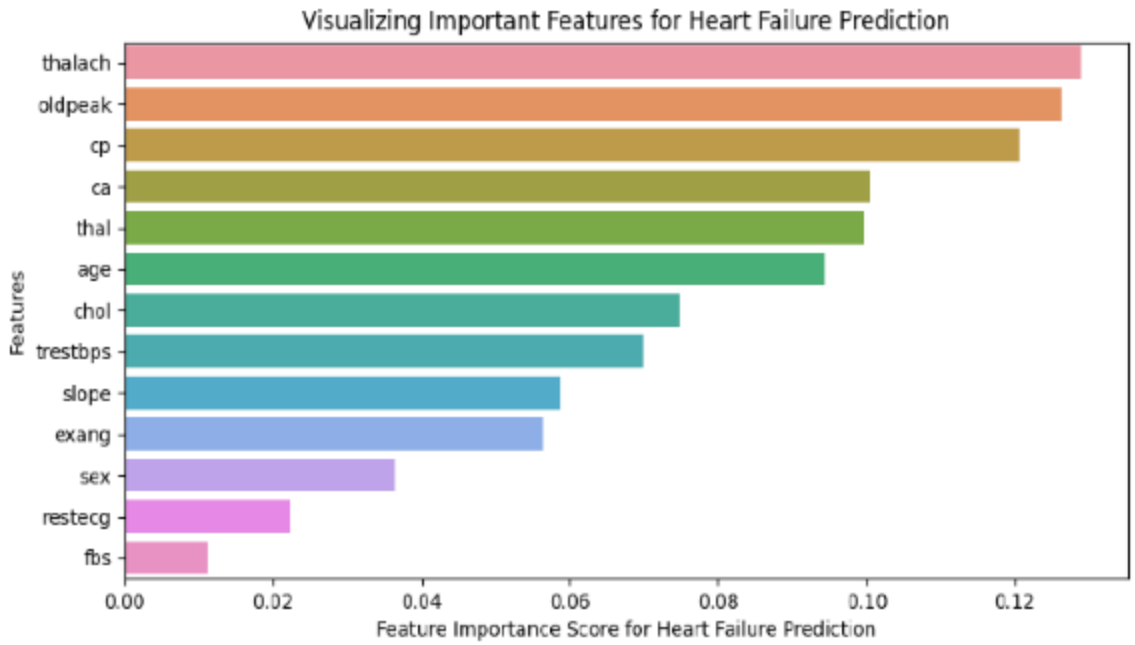


Figure 4: Random Forest model feature ranking (Mansur Huang et al., 2021).

From the RF model feature ranking, the most significant feature is maximum heart rate (thalach) and follows by ST depression induced by exercise relative to rest (oldpeak), then, chest pain (cp), these features show importance when it comes to heart failure prediction. The least significant feature is fasting blood sugar (fbs). The authors concluded that while machine learning techniques have shown potential in predicting heart failure, further research is needed to determine the most effective technique and to validate the findings.

Another work by Khalid Hossen (2022) aims to evaluate the effectiveness of machine learning techniques for heart disease prediction. The study used a dataset of patients with heart disease and healthy individuals to train and test various machine learning models, such as logistic regression, support vector machines, K-Nearest-Neighbors, Random Forest Classifier, and Gradient Boosting Classifier. Before the prediction starts, the author has to undergo data collection and select relevant attributes. Then, relevant attributes are chosen for data pre-processing before applying to the different classification models.

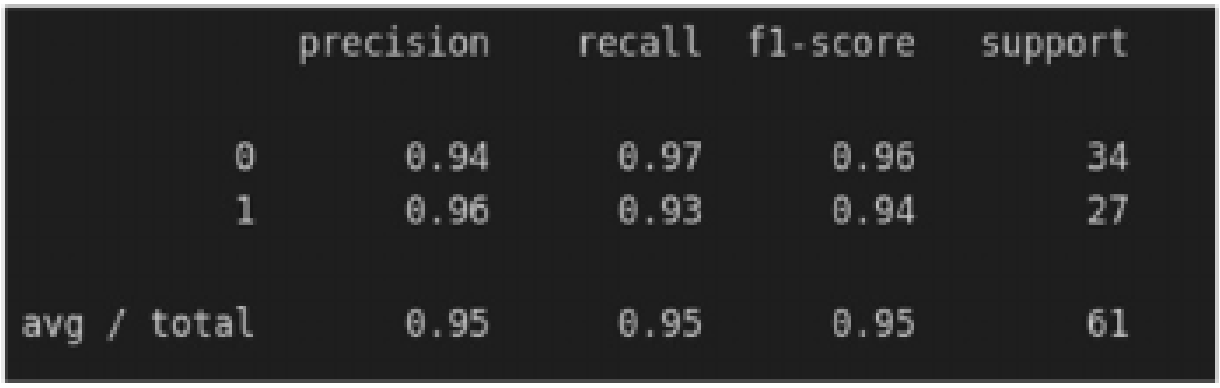
Figure 5: Correlation matrix of the features (Khalid Hossen, 2022).

The study also performs the correlation of features to define to positive and negative correlation between the features. In figure 5, cp (chest pain), thalach (maximum heart rate), slope positively correlate with target feature.

Algorithms	Accuracy
Logistic Regression (LR)	0.95
Support vector machine (SVM)	0.90
K-Nearest-Neighbors (KNN)	0.87
Random Forest Classifier (RF)	0.79
Gradient Boosting Classifier (GBC)	0.80

Figure 6: Accuracy comparison of different classification models (Khalid Hossen, 2022).

The author then proceeds to use different classification model to test the accuracy of the prediction. The results showed that the logistic regression model performed the best, with an accuracy of 95%. By achieving the best performance model, Khalid Hossen (2022) decided to use it and proceed to calculate the precision value, recall value, f1-score, and support value.



	precision	recall	f1-score	support
0	0.94	0.97	0.96	34
1	0.96	0.93	0.94	27
avg / total	0.95	0.95	0.95	61

Figure 7: precision, recall, f1-score, and support value of LR algorithm (Khalid Hossen, 2022).

The author concluded that machine learning techniques can be an effective tool for predicting heart disease and recommended further research to validate the findings and stated that further applications of this study can be implemented in other diseases.

The research article by Mansur Huang et al. (2021) and Khalid Hossen (2022) both highlights the potential of machine learning techniques for predicting heart disease and heart failure. However, it is crucial to remember that the accuracy of these methods can change based on the dataset used and the machine learning model applied. Future enhancements could include the use of larger and more diverse datasets, the integration of additional features such as demographic information and lifestyle factors, and the application of advanced machine learning techniques.

To further support this, a journal article by (Amin et al., 2019) was aimed to identify the important features in a cardiovascular disease model as well as which of those features can improve the accuracy of predicting heart disease. By utilizing RapidMiner Studio and the Cleveland database of patients which contains 303 records and 76 total attributes (by which 13 attributes are used for heart disease prediction), the researchers were able to conclude that there are nine significant attributes, namely “sex”, “cp”, “fbs”, “restecg”, “exang”, “oldpeak”, “slope”, “ca” and “thal”.

Comparison between Attributes resulting in the highest performance.

Features	Occurrence												
	Age	Sex	cp	Trestbps	Chol	Fbs	Restecg	Thalach	Exang	Oldpeak	Slope	ca	Thal
Occurrence in the Highest Accuracy	2	7	7	1	2	5	4	3	4	6	4	7	5
Occurrence in the Highest F-measure	2	7	7	1	2	5	4	3	4	6	4	7	5
Occurrence in the Highest Precision	0	6	4	2	1	2	2	2	4	2	4	5	4
Total Number of Occurrence	4	20	18	4	5	12	10	8	12	14	12	19	14

Figure 8: Comparison between attributes resulting in highest performance (Amin et al., 2019)

Besides that, this research also aimed to identify the highest performing data mining technique to improve the accuracy of the prediction, the researchers were able to conslude that the highest performing techniques were (in order of highest to low): Vote, Naïve Bayes, Support Vector Machine (SVM), Logistic Regression, Neural Network, k-NN and Decision Tree.

The highest accuracy achieved by each data mining technique.

Technique	Accuracy	Combination
Support Vector Machine (SVM)	86.87%	Age, sex, cp, chol, fbs, exang, oldpeak, slope, ca
Vote	86.20%	Sex, cp, fbs, thalach, exang, slope, ca, thal
Naïve Bayes	85.86%	Sex, cp, thalach, exang, oldpeak, ca
Logistic Regression	85.86%	Age, sex, cp, chol, restecg, oldpeak, slope, ca, thal
Neural Network	84.85%	Sex, cp, trestbps, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal
k-NN	82.49%	Sex, cp, fbs, restecg, oldpeak, ca, thal
Decision Tree	82.49%	Sex, cp, fbs, restecg, oldpeak, ca, thal

Figure 9: highest accuracy achieved by each data mining technique (Amin et al., 2019)

Average accuracy achieved by each data mining technique.

Technique	Average Accuracy Achieved
Vote	78.20%
Naïve Bayes	78.20%
Support Vector Machine (SVM)	78.15%
Logistic Regression (LR)	78.03%
Neural Network	75.18%
k-NN	63.50%
Decision Tree	63.50%

Figure 10: average accuracy achieved by each data mining technique (Amin et al., 2019)

Finally, research conducted by (Garg et al., 2021) uses machine learning to detect whether a person has heart disease or not. The researchers used classification algorithms based on supervised learning such as K-NN and Random Forest, before the researchers begin the predictions, several relevant attributes were selected and then chosen for data pre-processing before applying into the classification algorithms.

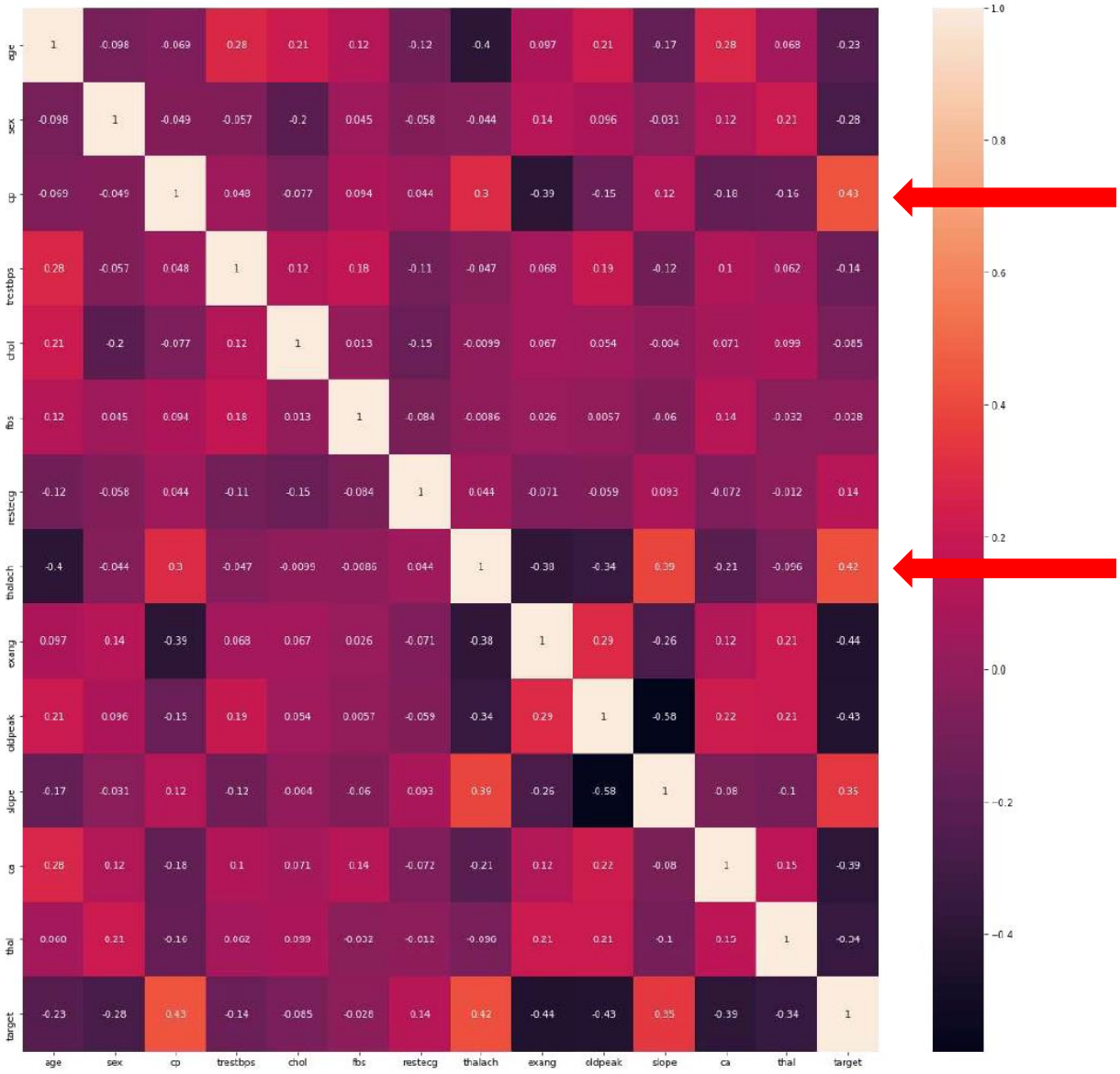


Figure 11: correlation between attributes (Garg et al., 2021)

The chosen attributes were plotted into a heat map to find the correlation between the data. According to the results of the heat map, attributes like cp (chest pain) and thalach (max heart rate achieved) have positive correlation against the target attribute. The results after applying the data into the algorithms clearly shows that using machine learning can be valuable in predicting hear disease whereby the accuracy obtained by K-NN is 86.885% and Random Forest is 81.967%.

Algorithm Used	TP	FP	TN	FN	Accuracy
K-NN	23	4	30	4	86.885%
Random Forest	22	5	28	6	81.967%

Figure 12: results after applying dataset into supervised machine learning algorithms (Garg et al., 2021)

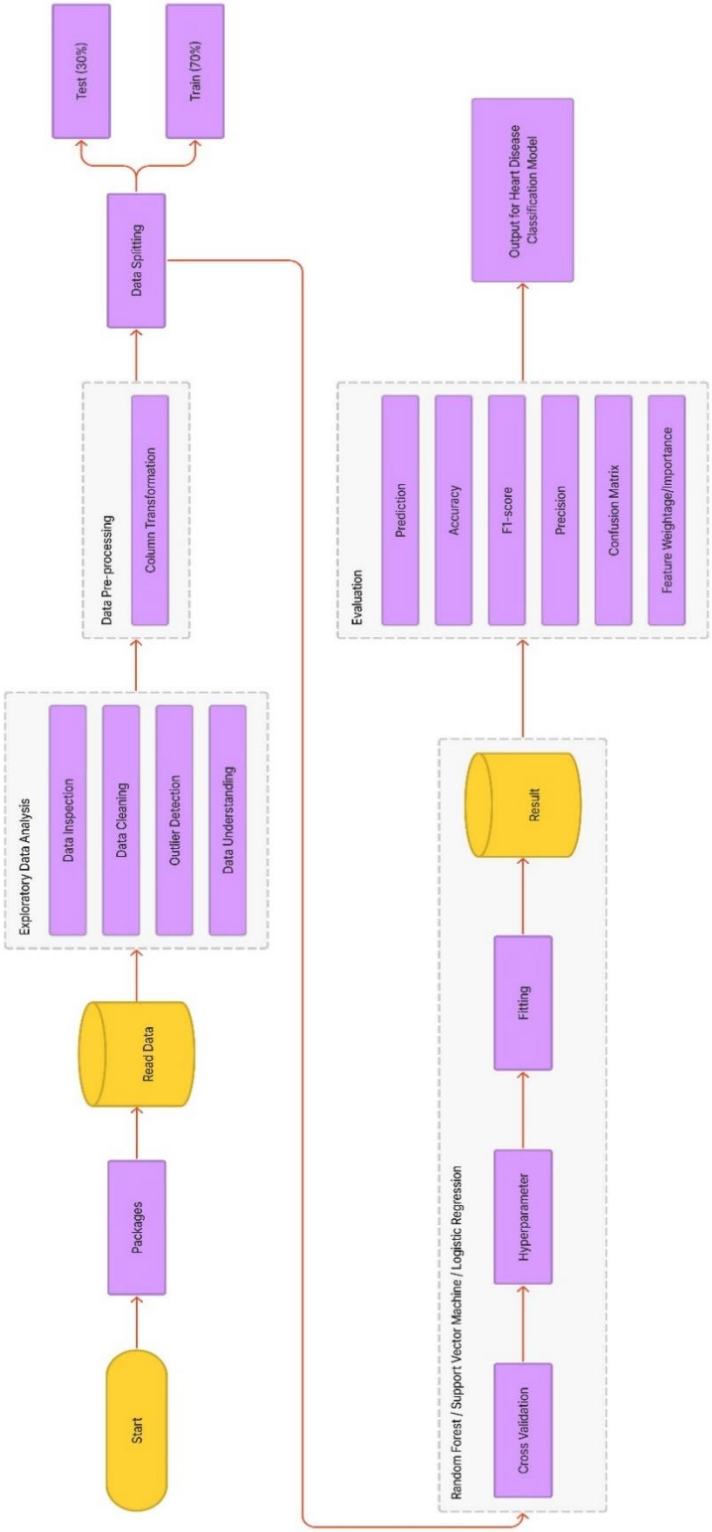
In the end, (Garg et al., 2021) concluded that using ML is proving to be extremely valuable in heart disease prediction since heart disease is one of the most prominent diseases in today's world.

Based on the review conducted on the existing literatures, the dataset from Kaggle (fedesoriano, 2021) has been selected. The dataset has the following variables:

Variable	Description
Age	age of the patient (years)
Sex	sex of the patient (M: Male, F: Female)
ChestPainType	chest pain type (TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic)
RestingBP	resting blood pressure (mm Hg)
Cholesterol	serum cholesterol (mm/dl)
FastingBS	fasting blood sugar (1: if FastingBS > 120 mg/dl, 0: otherwise)
RestingECG	resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
MaxHR	maximum heart rate achieved (Numeric value between 60 and 202)
ExerciseAngina	exercise-induced angina (Y: Yes, N: No)
Oldpeak	ST (Numeric value measured in depression)
ST_Slope	the slope of the peak exercise ST segment (Up: upsloping, Flat: flat, Down: downsloping)
HeartDisease	output class (1: heart disease, 0: Normal)

The dataset is selected as all its features are useful clinical features that have been used in previous research works. All the features will be used in the implementation stage with some preprocessing and transformation to develop the models for Heart Disease Classification.

Architecture Framework of Proposed Model



Algorithm Justification

Based on the literature review above, Random Forest (RF) classifier is notably the most appropriate algorithm to perform the classification task. Random Forest falls under the class of Supervised Machine Learning Algorithm. RF is basically an assemblage of decision trees and the result is generated according to the mean ranking. It is commonly chosen as a classifier due to its diverseness. Each individual tree in the RF differs from each other. This is because it used 2 mechanisms known as bootstrapping and feature sampling. These 2 mechanisms will allow the tree to be constructed with different subsets of the observations and features (Raj, 2020). RF also supports parallelization. CPU can be fully utilized to boost the random forest creation. RF is a more stable algorithm as the output generated is based on average raking. (R, 2021) Another solid reason for RF algorithm selection is it can handle collinearity. Collinearity is a special case whereby there is correlation between predictor variables. Collinearity in the model might have no effect on Type I error but might increase Type II error (Lavery et al., 2019). However, RF is a model that is immune to the effect of collinearity since the decision trees are built with a subset of the features instead of all of them (Lindner et al., 2022). This immunity will be useful as some features such as cholesterol and age are suspected to have collinearity.

Support Vector Machine (SVM) is commonly applied to solve classification issues, but it can handle regression tasks as well. SVM works by plotting the data or features as a point coordinate in a n-dimensional space, where n is the total number of features on hand. These point coordinates holding the features are also known as Support Vectors. The SVM algorithm will act as a boundary by finding the hyper-plane to sort out different classes. (Sunil, 2017) If the classes could not be classified linearly, SVM will perform a method known as kernel method to transform the Support Vectors to higher dimensional space where the classes are linearly separatable (Pisner & Schnyer, 2020). SVM is memory efficient, and it has the ability to classify even in high dimensional spaces. It can perform classifications whereby the number of dimensions is larger than the sample number. SVM's high accuracy in high dimensional space is important as the dataset chosen could have high dimensions especially after the categorical features are encoded.

Logistic Regression (LR) is applied in datasets to predict the possibility of the occurrence of an event according to a group of specific independent variables. It is used to evaluate the connection between a dependent variable and one or more independent variables. The final result of LR will be delimited between 0 and 1 as it is also considered a probability outcome. (IBM, 2016) According to the research earlier, it can be concluded that LR is a more accurate approach compared to other algorithms. This is credited to its high efficiency and accuracy. LR is also easy to be applied and trained, it is very flexible in terms of expanding to multinomial regression when there are more than 2 dependent variables. (AmiyaRanjanRout, 2023)

References

Mansur Huang, N. S., Ibrahim, Z., & Mat Diah, N. (2021). Machine Learning Techniques for Heart Failure Prediction. *MALAYSIAN JOURNAL OF COMPUTING*, *6*(2), 872. https://doi.org/10.24191/mjoc.v6i2.13708

Khalid Hossen, M. (2022). Heart Disease Prediction Using Machine Learning Techniques. *American Journal of Computer Science and Technology*, *5*(3), 146–154. https://doi.org/10.11648/j.ajcst.20220503.11

Amin, M. S., Chiam, Y. K., & Varathan, K. D. (2019). Identification of significant features and data mining techniques in. Telematics and Informatics, 83-93. https://doi.org/10.1016/j.tele.2018.11.007

Garg, A., Sharma, B., & Khan, R. (2021). Heart disease prediction using machine learning techniques. IOP Conference Series: Materials Science and Engineering, 1022, 1-10. https://doi.org/10.1088/1757-899X/1022/1/012046

Abdollahi, J., & Nouri-Moghaddam, B. (2022). A hybrid method for heart disease diagnosis utilizing feature selection based ensemble classifier model generation. *Iran Journal of Computer Science*, *5*(3), 229–246. https://doi.org/10.1007/s42044-022-00104-x

Ali, M. M., Paul, B. K., Ahmed, K., Bui, F. M., Quinn, J. M. W., & Moni, M. A. (2021). Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison. *Computers in Biology and Medicine*, *136*, 104672. https://doi.org/10.1016/j.compbiomed.2021.104672

AmiyaRanjanRout. (2023, January 10). *Advantages and Disadvantages of Logistic Regression—GeeksforGeeks*. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression/

fedesoriano. (2021, September). *Heart Failure Prediction Dataset*. https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction

Gárate-Escamila, A. K., Hajjam El Hassani, A., & Andrès, E. (2020). Classification models for heart disease prediction using feature selection and PCA. *Informatics in Medicine Unlocked*, *19*, 100330. https://doi.org/10.1016/j.imu.2020.100330

Ghosh, P., Azam, S., Jonkman, M., Karim, A., Shamrat, F. M. J. M., Ignatious, E., Shultana, S., Beeravolu, A. R., & De Boer, F. (2021). Efficient Prediction of Cardiovascular Disease Using Machine Learning Algorithms With Relief and LASSO Feature Selection Techniques. *IEEE Access*, *9*, 19304–19326. https://doi.org/10.1109/ACCESS.2021.3053759

IBM. (2016). *What is Logistic regression? | IBM*. https://www.ibm.com/my-en/topics/logistic-regression

Lavery, M. R., Acharya, P., Sivo, S. A., & Xu, L. (2019). Number of predictors and multicollinearity: What are their effects on error and bias in regression? *Communications in Statistics - Simulation and Computation*, *48*(1), 27–38. https://doi.org/10.1080/03610918.2017.1371750

Lindner, T., Puck, J., & Verbeke, A. (2022). Beyond addressing multicollinearity: Robust quantitative analysis and machine learning in international business research. *Journal of International Business Studies*, *53*(7), 1307–1314. https://doi.org/10.1057/s41267-022-00549-z

Pisner, D. A., & Schnyer, D. M. (2020). Chapter 6—Support vector machine. In A. Mechelli & S. Vieira (Eds.), *Machine Learning* (pp. 101–121). Academic Press. https://doi.org/10.1016/B978-0-12-815739-8.00006-7

R, S. E. (2021, June 17). Random Forest | Introduction to Random Forest Algorithm. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Raj, S. (2020, June 18). *Effects of Multi-collinearity in Logistic Regression, SVM, RF*. Medium. https://medium.com/@raj5287/effects-of-multi-collinearity-in-logistic-regression-svm-rf-af6766d91f1b

Sunil, R. (2017, September 13). *SVM | Support Vector Machine Algorithm in Machine Learning*. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

WHO. (n.d.). *Cardiovascular diseases*. Retrieved January 31, 2023, from https://www.who.int/health-topics/cardiovascular-disease

GROUP ASSIGNMENT PART 2

TECHNOLOGY PARK MALAYSIA

CT032-3-3-FAI

FURTHER ARTIFICIAL INTELLIGENCE

Submission Date: 10 March 2023

Lecturer Name: Ts. Nor Anis Asma Binti Sulaiman

Intake: APD3F2211CS(IS)

Assignment Tittle: Heart Disease Classification

Student Name & TP ID:

Wong Tik Soon TP054784

Lim Sheng Xue TP054683

Ng Eng Siong TP055095

Wong Horng Woei TP055241

INSTRUCTIONS TO CANDIDATES:

1 **Submit your assignment online via Moodle.**

2 **Students are advised to underpin their answers with the use of references (cited using the APA System of Referencing)**

3 **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld**

4 **Cases of plagiarism will be penalized**

5 **Ensure that all answers are presented clearly, including using appropriate font sizes and clarity of images. All important information and steps of the workings should be shown.**

6 **You must obtain 50% overall to pass this module.**

Table of Contents

List of Figures

Implementation of Software/Libraries

The software artifact of this project is developed using Python on Jupyter Lab. Below documented the code and functions written to build the artifact. The software artifacts developed are Heart Disease Classification Models which could classify if a patient has heart disease or is normal.

Library Import and Data Reading

```
# Import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
```

Figure 1: Import Libraries

The import statements import the libraries that are required by the project.

Firstly, there is “pandas” for data processing purpose. Pandas is a fast and powerful data analysis and manipulation library (pandas, n.d.). Pandas will be used to read the data from the csv file. It will produce a DataFrame object which is very crucial, especially in the Exploratory Data Analysis (EDA) stage. Different functions such as group by, subsetting, and indexing could be performed on the DataFrame object.

The matplotlib and seaborn libraries are used for Data Visualization purposes. Matplotlib provides basic data visualization function to create different types of plots. With the API provided by Matplotlib, different customizations such as adjusting the title, the axis label, the figure size could be performed easily. On the other hand, seaborn is another data visualization library which has a simpler syntax when working with DataFrame object. For example, it could easily create a clustered bar chart by mapping the hue of the chart to one of the columns of the DataFrame object. The plots of this project will be mostly created by seaborn which has built-in theme providing a better appearance (Agrawal, 2021).

The numpy library is for numerical processing (NumPy, n.d.). It will only be used on some minor function of this project since numpy functions are considered as lower level functions.

Sklearn, also known as Scikit-learn, is a machine learning library that provide supervised and unsupervised learning algorithms together with a wide range of functions to support developing and evaluating the machine learning models (ScikitLearn, n.d.). The functions and objects that it provides are well organized into different modules. For this project, the train_test_split function from model_selection module is used to split the data to train and test sets for training and evaluating the model. The cross_val_score is used to support cross validation for hyperparameter tuning. The metrics module provides various functions to measure the performance of the model. Lastly, the 3 models selected are imported from different modules such as ensemble, SVM, and linear_model.

Data Reading

```
# Read data from the downloaded csv file
heart = pd.read_csv("heart.csv")
```

Figure 2: Data Reading

The dataset downloaded is read. The dataset is in the form of Comma-Separated Values (CSV) file. Thus, the read_csv function of Pandas is used to read the data into a DataFrame object. The dataset is downloaded from Kaggle (fedesoriano, 2021). This dataset contains all the features that have been frequently used in previous research works. All the features will be used in the modelling phase as they are all important clinical features. However, preprocessing steps will be performed on them.

Exploratory Data Analysis

In this phase, 4 processes will be carried out. The processes are Data Inspection, Data Cleaning, Outlier Detection, and Data Understanding.

Data Inspection

```
[3]: # View the top records of the data
     heart.head()
```

Figure 3: Check the top records of the data

Pandas' head method allows checking the top records of the DataFrame.

```
[4]: # View the dimensions of the data
     heart.shape
```

Figure 4: Check the dimension of the data

Pandas' shape method allows checking the dimension of the data. The rows and columns of the DataFrame could be shown.

```
[5]: # View the features of the data
     heart.columns
```

Figure 5: Checking the columns of the data

Pandas' columns method allows checking the columns of the data. The features of the dataset will be shown.

```
[6]: # View the information of the data
     heart.info()
```

Figure 6: Checking the information of the data

Pandas' info method allows checking the basic information of the DataFrame. Some important information such as the columns' data types, the columns' non null count will be provided.

Data Cleaning

```
[7]: # Check if there is column with null value
     heart.isnull().sum()
```

Figure 7: Check if there is null value

Pandas' isnull method allow checking if there is null value. Calling the sum method after it allows summing up the number of null values for each column if they present.

Outlier Detection

```
:  # Create a box plot for Resting BP
   sns.boxplot(data = heart, x="RestingBP")
   plt.title('RestingBP Boxplot')
```

Figure 8: Creating boxplot for RestingBP column

To detect outlier, boxplot is a good tool. Seaborn boxplot function is used to generate a boxplot for the RestingBP column of the data. This column is the resting blood pressure of the patient.

```
# View statistical properties of Resting BP
heart["RestingBP"].describe()
```

Figure 9: View statistical property of RestingBP

As an outlier is detected, the describe method is used to inspect the statistical properties of the column.

```
# Filter record with RestingBP equals to 0
heart[heart["RestingBP"] == 0]
```

Figure 10: Filter DataFrame to get records with RestingBP equals to 0

The code above is to view records with RestingBP equals to 0. The records are outliers.

```
[11]: # Remove the record that has 0 Resting BP
      heart = heart[heart["RestingBP"] != 0]
```

Figure 11: Remove record with Resting BP equals to 0

The code above removes the record with RestingBP equals to 0 by only selecting the records with RestingBP not equals to 0.

```
# Create a box plot for Resting BP
sns.boxplot(data = heart, x="RestingBP")
plt.title('RestingBP Boxplot')
```

Figure 12: Create Boxplot of RestingBP

The code above is to create the boxplot for RestingBP again, after removing the outlier. This is to view the effect of the outlier removal on the distribution of the data.

Data Understanding

```
# View number of records for each label
print(heart["HeartDisease"].value_counts())

# Compute the percentage of records for each label
perc_of_classes = heart["HeartDisease"].value_counts().values / len(heart) * 100
print("Percentage of class with heart diseases : {:.2f} %".format(perc_of_classes[0]))
print("Percentage of class normal : {:.2f} %".format(perc_of_classes[1]))
```

Figure 13: View the percentages of number of observations group by label

The code above first prints the value_counts of Heart Diseases column. It will shows the possible values in that column and the number of observations holding those values. The number of observations for each label which stored in the “values” of value_counts are divided by the number

of rows to produce the percentage of observations for each label. This is to show the proportion of observation for each class/label.

```
# Generate a bar plot showing No of records per label
target_counts = heart['HeartDisease'].value_counts().values

# Create the labels for the bars
bc = sns.countplot(x="HeartDisease", data=heart, order=heart['HeartDisease'].value_counts().index)
bc.bar_label(container=bc.containers[0], labels=target_counts)
plt.title('Number of observations per class')
```

Figure 14: Plot the barchart to show number of observations per class

The code above is to create a bar chart showing the number of observation per class. The number of observations is computed using the value_counts function. The seaborn countplot is used to generate the bar chart. The bar_label function attached the count, which is the number of observations for each class to the bars of the bar chart. Lastly, the title of the chart is added.

```
# Create the histogram for Age
sns.histplot(heart, x = "Age",  bins=[0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
plt.title('Age Histogram')

# Compute the mean age
print("Average age : {:.2f}".format(heart["Age"].mean()))
```

Figure 15: Histogram and Mean of Age

The code above plots the Histogram of age with specific bins. The mean of Age is also computed and displayed. The mean is formatted to 2 decimal places.

```
#  Create the histogram of age for records without heart disease
plt.figure(figsize=(13,5))
plt.subplot(1, 2, 1)
sns.histplot(heart[heart['HeartDisease'] == 0], x="Age", bins=[0, 10, 20, 30, 40,
                                                               50, 60, 70, 80, 90])

plt.title('Age Histogram for Observations without Heart Disease')

#  Create the histogram of age for records with heart disease
plt.subplot(1, 2, 2)
sns.histplot(heart[heart['HeartDisease'] == 1], x="Age", bins=[0, 10, 20, 30, 40,
                                                               50, 60, 70, 80, 90])
plt.title('Age Histogram for Observations with Heart Disease')

# Compute the mean of age for records without and with heart disease
print("Average age for observations without heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 0]["Age"].mean()))
print("Average age for observations with heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 1]["Age"].mean()))
```

Figure 16: Plot 2 histograms for Age of Heart Disease patients and normal people

The code above plots 2 histograms. Firstly, it uses the figure function with figsize to adjust the plot size. It creates the first age histogram with data that has been filtered by column HeartDisease equals to 0. This data is the records of normal people. It creates the second age histogram with data that has been filtered by column HeartDisease equals to 1. This data is the records of heart disease patient. Finally, the mean ages of these 2 groups are computed using the same filtering mechanism.

```
# Generate a bar plot showing No of records per chest pain
bc = sns.countplot(x="ChestPainType", data=heart)
plt.xlabel("Chest Pain Types")
plt.title('Number of Observations per Chest Pain')
```

Figure 17: Create bar plot showing number of observations per chest pain type

The code above generates a bar plot showing the number of observations grouped by chest pain type.

```
# Sum the value group by ChestPainType, this allow the total Heart Disease (with label 1)
# to be computed
heart_gb_cp_sum = heart.groupby("ChestPainType").sum(numeric_only = True)

# Generate a bar plot showing No of Heart Diseases per chest pain
sns.barplot(x=heart_gb_cp_sum.index, y = "HeartDisease", data=heart_gb_cp_sum)
plt.xlabel("Chest Pain Types")
plt.ylabel("Number of Heart Diseases")
plt.title('Number of Heart Disease Presences per Chest Pain')
```

Figure 18: Create bar plot showing the number of Heart Diseases grouped by chest pain type

The code above generates a bar plot showing the number of heart diseases grouped by chest pain type. It used the groupby function of Pandas to group the observation by ChestPainType column. It then performs the sum function to sum up the values of other columns. The numeric only parameter ensures the sum only performed on numeric columns. In HeartDisease column, 1 represents a heart disease, while 0 represents normal. Thus, the sums in column HeartDisease is the total number of heart diseases grouped by chest pain type. These sums are then passed to the barplot function of seaborn to create the bar plot.

```
#  Create the histogram of cholesterol for records without heart disease
plt.figure(figsize=(13,5))
plt.subplot(1, 2, 1)
sns.histplot(heart[heart['HeartDisease'] == 0], x="Cholesterol", bins=[0, 100, 200,
                                                                       300, 400, 500, 600, 700])

plt.title('Cholesterol Histogram for Observations without Heart Disease')

#  Create the histogram of cholesterol for records with heart disease
plt.subplot(1, 2, 2)
sns.histplot(heart[heart['HeartDisease'] == 1], x="Cholesterol", bins=[0, 100, 200,
                                                                       300, 400, 500, 600, 700])
plt.title('Cholesterol Histogram for Observations with Heart Disease')

#  Compute the mean of cholesterol for records without and with heart disease
print("Average Cholesterol for observations without heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 0]["Cholesterol"].mean()))
print("Average Cholesterol for observations with heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 1]["Cholesterol"].mean()))
```

Figure 19: Plot 2 histograms for Cholesterol of Heart Disease patients and normal people

The code above works similarly to the plotting of 2 age histograms mentioned above. The code plots 2 histograms. Firstly, it uses the figure function with figsize to adjust the plot size. It creates the first cholesterol histogram with data that has been filtered by column HeartDisease equals to 0. This data is the records of normal people. It creates the second cholesterol histogram with data that has been filtered by column HeartDisease equals to 1. This data is the records of heart disease

patient. Finally, the means cholesterol of these 2 groups are computed using the same filtering mechanism.

```
# Create bar plot showing no of observations per Exercise Angina
ax = sns.countplot(x="ExerciseAngina", data=heart)
plt.xlabel("Exercise Induced Angina")
plt.title('Number of Observations per Exercise Angina')
ax.set_xticklabels(['No', 'Yes'])
```

Figure 20: Create bar plot showing number of observations per ExerciseAngina

The code above generates a bar plot showing the number of observations grouped by exercise angina. The column only has 2 values which are "N" and "Y". The set_xticklabels is to map the "N" and "Y" to "No" and "Yes" to enhance the readability of the plot.

```
[66]: # Create bar plot showing no of observations per Exercise Angina and Heart Disease
      ax = sns.countplot(x="HeartDisease", hue="ExerciseAngina" , data=heart)
      plt.title('Number of observations per Heart Disease and Exercise Angina')
      ax.legend(title='ExerciseAngina', labels=['No', 'Yes'])
```

Figure 21:Create bar plot showing observation grouped by Heart Disease and ExerciseAngina

The code above will create a clustered bar chart. The clusters represent Heart Disease. There will be 2 clusters for since Heart Disease has the value of 0 and 1. Each cluster will have 2 bars, which have different hue. The hue is mapped to the ExerciseAngina. The legend is to provide the labels for the hue. The bar chart generated visualizes the number of observations grouped by Heart Disease and Exercise Angina.

```
# Create boxplot of Oldpeak seperated by Heart Disease
sns.boxplot(x ='HeartDisease', y = "Oldpeak" , data =heart)
plt.title('Boxplot of Oldpeak seperated by Heart Disease')

#  Compute the mean of Oldpeak for records without and with heart disease
print("Average Oldpeak for observations without heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 0]["Oldpeak"].mean()))
print("Average Oldpeak for observations with heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 1]["Oldpeak"].mean()))
```

Figure 22: Create boxplot of Oldpeak separated by HeartDisease

The code above generates boxplot of Oldpeak separated by Heart Disease. It visualizes the distribution of Oldpeak for normal people and heart disease patient. It also computes the mean of Oldpeak for normal people and heart disease patient.

```
# Create boxplot of MaxHR seperated by Heart Disease
sns.boxplot(x ='HeartDisease', y = "MaxHR" , data =heart)
plt.title('Boxplot of MaxHR seperated by Heart Disease')

#  Compute the mean of MaxHR for records without and with heart disease
print("Average MaxHR for observations without heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 0]["MaxHR"].mean()))
print("Average MaxHR for observations with heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 1]["MaxHR"].mean()))
```

Figure 23:Create boxplot of MaxHR separated by HeartDisease

The code generates boxplot of MaxHR separated by Heart Disease. It visualized the distribution of MaxHR for normal people and heart disease patient. It also computes the mean of MaxHR for normal people and heart disease patient.

```
# Create heatmap showing correlation of MaxHR, Oldpeak and Heart Disease
sns.heatmap(heart[["HeartDisease","MaxHR","Oldpeak"]].corr(), annot=True)
plt.title('Heatmap for Correlation of Heart Disease, MaxHR, and Oldpeak')
```

Figure 24: Create heatmap to visualize correlation of Heart Disease, MaxHR, Oldpeak

The code generates a heatmap to visualize the correlation of Heart Disease, MaxHR, and OldPeak. The correlation is obtained using corr method of Pandas. Setting the annot parameter to True will allow the value of correlation to be shown on the heatmap.

```
# Create bar plot showing no of observations per ST_Slope and Heart Disease
ax = sns.countplot(x="HeartDisease", hue="ST_Slope" , data=heart)
plt.title('Number of Observations per Heart Disease and ST_Slope')
```

Figure 25: Create Bar Plot showing observations grouped by Heart Disease and STSlope

The code above will create a clustered bar chart. The clusters represent Heart Disease. Each cluster will have 2 bars, which have different hue. The hue is mapped to the STSlope. The legend is to provide the labels for the hue. The bar chart generated visualizes the number of observations grouped by Heart Disease and STSlope.

Data Preparation

This phase prepares the data for modelling. It consists of 2 processes which are Column Transformation and Train Test Split.

Column Transformation

```
# View information of the data frame
heart.info()
```

Figure 26: Inspecting column data types

The info method is used to inspect the data type of the columns. Columns with "object" data type need to be transformed to be accepted by the model.

```
# view top records
heart.head()
```

Figure 27: Inspecting the top rows of the data

Using head to inspect the top rows provide some insight to help with the column transformation.

```
# View possible values and its counts of sex
heart['Sex'].value_counts()
```

Figure 28: Calling value_counts on Sex column

The method value count is to inspect the possible values of Sex column. It should only have 2 possible values which are "M" and "F".

```
# View possible values and its counts of ExerciseAngina
heart['ExerciseAngina'].value_counts()
```

Figure 29: Calling value_counts on ExerciseAngina column

The method value count is to inspect the possible values of Exercise Angina column. It should only have 2 possible values which are "N" and "Y".

```
# Create label encoder
le = preprocessing.LabelEncoder()

# Encode sex and exerciseangina columns' values
heart['Sex']= le.fit_transform(heart['Sex'])
heart['ExerciseAngina']= le.fit_transform(heart['ExerciseAngina'])
```

Figure 30: Using Label Encoder to encode column with 2 possible values

An instance of Label Encoder is created. The label encoder is used to encode the Sex and Exercise Angina columns of the data. Label Encoder is used for columns that have only 2 possible values.

```
# Check encoding result
heart.head()
```

Figure 31: Inspect the top records to confirm the result of encoding

The head method is used to inspect the top records. This is to confirm the result of encoding.

```
# Confirm encoding result
heart.info()
```

Figure 32: Check the data type of the columns to confirm the result of encoding

The info method is used to inspect the data type of the columns. This is to confirm the result of encoding.

```
# View possible values and its counts of ChestPainType
heart['ChestPainType'].value_counts()
```

Figure 33: View the possible values of Chest Pain Type

The method value count is to inspect the possible values of Chest Pain Types column. It should have 4 possible values.

```
# View possible values and its counts of RestingECG
heart['RestingECG'].value_counts()
```

Figure 34: View the possible values of Resting ECG

The method value count is to inspect the possible values of Resting ECG column. It should have 3 possible values.

```
# View possible values and its counts of ST_Slope
heart['ST_Slope'].value_counts()
```

Figure 35: View the possible values of ST Slope

The method value count is to inspect the possible values of ST Slope column. It should have 3 possible values.

```
# Create Label Binarizer to implement one hot encoding
lb = preprocessing.LabelBinarizer()

# Encode ChestPainType column with Label Binarizer
heart = heart.join(pd.DataFrame(lb.fit_transform(heart["ChestPainType"]),
                          columns=lb.classes_,
                          index=heart.index))

# Encode RestingECG column with Label Binarizer
heart = heart.join(pd.DataFrame(lb.fit_transform(heart["RestingECG"]),
                          columns=lb.classes_,
                          index=heart.index))

# Encode STSlope column with Label Binarizer
heart = heart.join(pd.DataFrame(lb.fit_transform(heart["ST_Slope"]),
                          columns=lb.classes_,
                          index=heart.index))
```

Figure 36: Using Label Binarizer to encode columns Chest Pain Type, Resting ECG, ST Slope

An instance of Label Binarizer is created. The label binarizer is used to encode the Chest Pain Type, Resting ECG, and ST Slope columns of the data. Dummy columns will be created as the result of the encoding. The dummy columns are joined to the original data frame.

Label Binarizer is used for columns that have more than 2 possible values.

```
# Confirm encoding result
heart.info()
```

Figure 37: Inspect the data frame information to confirm the result of encoding

The info method is to check the columns produced and their data types to confirm the result of encoding.

```
# Drop the original columns that has been encode by LabelBinarizer
heart = heart.drop(columns=["ChestPainType", "RestingECG","ST_Slope"])
```

Figure 38: Drop the Original columns

The original columns that have been encoded by Label Binarizer are dropped. Their information has being conveyed by the dummy columns created.

```
# Confirm dropping result
heart.info()
```

Figure 39: Inspect the data frame information to confirm the result of dropping

The info method is to check the columns of the data frame to confirm the result of dropping.

```
# Drop the columns with redundant information
# ASY ATA NAP TA - one of them should be dropped
# LVH Normal ST - one of them should be dropped
# Down Flat Up - one of them should be dropped

heart = heart.drop(columns=["ASY","ST","Down"])
```

Figure 40: Drop some dummy columns

From the dummy columns created by same column, one of them should be dropped. For example, the column Chest Pain Type, when encoded by Label Binarizer, will resulted in 4 dummy columns: ASY, ATA, NAP, TA. The ASY column will have a value of 1 if the original column has a value of "ASY". ATA will be 1 if the original column has a value of "ATA". The logic repeats for NAP and TA. Thus, one of the columns will be conveying redundant information. To illustrate, if ASY, ATA, and NAP is 0. The record must have TA equals to 1, this information is known without having to look at the value of TA.

Thus, the redundant columns have to be dropped. An arbitrary column for each group is selected and dropped using the drop method.

Train Test Split

```
# Split the predictors and target variable
heart_X = heart.iloc[:, heart.columns != "HeartDisease"]
heart_y = heart["HeartDisease"]
```

Figure 41: Split the predictor variables and target variable

The code above splits the predictor variable and target variable. The predictor variables are obtained by indexing all columns except the Heart Disease column. The target variable is obtained by indexing the Heart Disease column.

```
# Perform 70 : 30 train test split
heart_X_train, heart_X_test, heart_y_train, heart_y_test = train_test_split(
    heart_X, heart_y, test_size=0.30, random_state=123)
```

Figure 42: Perform train test split

Train test split is performed using the train_test_split function of sklearn. The test size is specify as 0.3 to have a 70:30 split. This is the usual split used by the existing research works. It is also suitable for this dataset as the dataset is not too large to have 40:60 or 50:50 split. The random state parameter allows setting a seed for the random shuffling behaviour that executes before the splitting.

```
# View dimensions of traing data X
heart_X_train.shape
```

Figure 43: Inspect the dimension of training data predictor variables

Pandas' shape method is used to inspect the dimension of training data predictor variables.

```
# View dimensions of testing data X
heart_X_test.shape
```

Figure 44: Inspect the dimension of testing data predictor variables

Pandas' shape method is used to inspect the dimension of testing data predictor variables.

Modelling

Random Forest Classifier

```
# using cross validation to find the optimal hyperparameter
scores = []
hyperparams = []

for n_estimator in [500,1000,2000]:
    for max_depth in [3,5,7]:
        for max_features in [2,3,4]:
            rf_clf = RandomForestClassifier(n_estimators=n_estimator, max_depth=max_depth,
                                            random_state=0, max_features = max_features)
            cv_scores = cross_val_score(rf_clf, heart_X_train, heart_y_train, cv=5)
            scores.append(cv_scores.mean())
            hyperparams.append((n_estimator,max_depth,max_features))
```

Figure 45: Using cross validation to find the optimal hyperparameter for random forest classifier

The code above finds the optimal hyperparameter for Random Forest Classifier. It iterates through a list of possible values for n_estimator, max_depth, and max_features. The n_estimator is the number of decision tress that the random forest could consists of. The max_depth specifies the

maximum depth of the tree. And maximum features specify the maximum features that could be considered when building a branch of the tree. The values are used to construct the model. The model is then evaluated using 5 fold cross validation. The scores will be a list of 5 as 5 fold cross validation is used. The mean of the scores is then appended to a list. The hyperparameters are also appended to a list.

```
# Show the best hyperparameter
best_hyperparam = hyperparams[np.argmax(scores)]
print("Optimal n_estimator : " + str(best_hyperparam[0]))
print("Optimal max_depth : " + str(best_hyperparam[1]))
print("Optimal max_features : " + str(best_hyperparam[2]))
```

Figure 46: Showing the optimal hyperparameter

The code above displayed the optimal hyperparameter. The numpy's argmax function will return the index of the maximum value. In this case, it will return the index of the maximum score. This index is also the index of the optimal hyperparameter. After indexing the optimal hyperparameters, the hyperparameters are then printed sequentially.

```
# Creating Random Forest Model
rf_clf = RandomForestClassifier(n_estimators=500, max_depth=7, random_state=0, max_features = 2)

# Fitting the model to training data
rf_clf.fit(heart_X_train, heart_y_train)
```

Figure 47: Building and training the Random Forest Classifier

The code above constructs a Random Forest Classifier. The hyperparameter n_estimators, max_depth, max_features are chosen based on the previous cross validation steps. The model is trained with all the training data.

Support Vector Machine Classifier

```
# using cross validation to find the optimal hyperparameter
scores = []
hyperparams = []

for C in [100,200,500,1000]:
    svm_clf = SVC(C= C)
    cv_scores = cross_val_score(svm_clf, heart_X_train, heart_y_train, cv=5)
    scores.append(cv_scores.mean())
    hyperparams.append((C))
```

Figure 48: Using cross validation to find optimal hyperparameter for Support Vector Machine Classifier

The code above finds the optimal hyperparameter for Support Vector Machine Classifier. It iterates through a list of possible values for C. C is a hyperparameter for regularization purpose. The larger

the value of C, the weaker the strength of regularization. Regularization is a technique to remove redundant features. The value is then used to construct the model. The model is then evaluated using 5 fold cross validation. The scores will be a list of 5 as 5 fold cross validation is used. The mean of the scores is then appended to a list. The hyperparameters are also appended to a list.

```
# Show the best hyperparameter
best_hyperparam = hyperparams[np.argmax(scores)]
print("Optimal C : " + str(best_hyperparam))
```

Figure 49: Show the optimal hyperparameter

The optimal hyperparameter is shown. The logic of this code is similar to the display of optimal hyperparameters of Random Forest shown above.

```
# Creating SVM model
svm_clf = SVC(C= 1000)

# Fitting the model to training data
svm_clf.fit(heart_X_train, heart_y_train)
```

Figure 50: Constructing and Training the Support Vector Machine Classifier

The code above constructs a Support Vector Machine Classifier. The hyperparameter C is chosen based on the previous cross validation steps. The model is trained with all the training data.

Logistic Regression Classifier

```
# using cross validation to find the optimal hyperparameter
scores = []
hyperparams = []

for C in [1,5,50,100]:
    for solver in ['lbfgs', 'liblinear', 'newton-cholesky']:
        lr_clf = LogisticRegression(C = C, random_state=0, solver=solver, max_iter = 5000)
        cv_scores = cross_val_score(lr_clf, heart_X_train, heart_y_train, cv=5)
        scores.append(cv_scores.mean())
        hyperparams.append((C, solver))
```

Figure 51: Using cross validation to find the optional hyperparameter for Logistic Regression Classifier

The code above finds the optimal hyperparameter for Logistic Regression Classifier. It iterates through a list of possible values for C and solver. Similar to Support Vector Machine, C is a hyperparameter for regularization purpose. The larger the value of C, the weaker the strength of regularization. The solver is the algorithm that is used to find the solution for the model. The values are then used to construct the model. The max_iter is set to a higher value of 500. This guarantees the model could converge within the iteration. The model is then evaluated using 5 fold cross

validation. The scores will be a list of 5 as 5 fold cross validation is used. The mean of the scores is then appended to a list. The hyperparameters are also appended to a list.

```
# Show the best hyperparameter
best_hyperparam = hyperparams[np.argmax(scores)]
print("Optimal C : " + str(best_hyperparam[0]))
print("Optimal solver : " + str(best_hyperparam[1]))
```

Figure 52: Display the optimal hyperparameters

The optimal hyperparameters are shown. The logic of this code is similar to the display of optimal hyperparameters of Random Forest mentioned above.

```
# Creating Logistic Regression model
lr_clf = LogisticRegression(random_state=0, solver="newton-cholesky", C = 1)

# Fitting the model to training data
lr_clf.fit(heart_X_train, heart_y_train)
```

Figure 53: Constructing and training the Logistic Regression Classifier

The code above constructs a Logistic Regession Classifier. The hyperparameters C and solver are chosen based on the previous cross validation steps. The model is trained with all the training data.

Evaluation

Prediction

```
# Obtain the predictions of the models
rf_pred_y = rf_clf.predict(heart_X_test)
svm_pred_y = svm_clf.predict(heart_X_test)
lr_pred_y = lr_clf.predict(heart_X_test)
```

Figure 54: Stored the predicted value of the 3 models

The 3 models perform prediction on the testing data by calling their predict method. The predictions are stored in 3 variables respectively.

```
# Creating barchart that shows number of records per true label and prediction
df = pd.DataFrame({"Predicted" : rf_pred_y,"True" : heart_y_test})
ax = sns.countplot(x="True", hue="Predicted", data=df)
plt.title('Barchart of the predicted value per true label (Random Forest)')
```

Figure 55: Create a clustered Bar chart showing true value and predicted value of Random Forest

The code above plots a bar chart to visualize the prediction done by Random Forest Classifier. Firstly, a Data Frame is created with the predicted value and true value. The seaborn countplot function is used to create a clustered bar chart. There will be 2 clusters representing a true 0 and a

true 1. There will be 2 bars in cluster true 0 showing the count of true 0 predicted as 0 and true 0 predicted as 1. There will also be 2 bars in cluster true 1 showing the count of true 1 predicted as 0 and true 1 predicted as 1.

```
# Creating barchart that shows number of records per true label and prediction
df = pd.DataFrame({"Predicted" : svm_pred_y,"True" : heart_y_test})
ax = sns.countplot(x="True", hue="Predicted", data=df)
plt.title('Barchart of the predicted value per true label (SVM)')
```

Figure 56: Create a clustered Bar chart showing true value and predicted value of Support Vector Machine

This code works the same as above. It is just that the predicted value is obtained from the support vector machine classifier.

```
# Creating barchart that shows number of records per true label and prediction
df = pd.DataFrame({"Predicted" : lr_pred_y,"True" : heart_y_test})
ax = sns.countplot(x="True", hue="Predicted", data=df)
plt.title('Barchart of the predicted value per true label (Logistic Regression)')
```

Figure 57: Create a clustered Bar chart showing true value and predicted value of Logistic Regression

This code works the same as above. It is just that the predicted value is obtained from the logistic regression classifier.

Accuracy

```
# Compute the accuracy score of 3 models
rf_acc = accuracy_score(heart_y_test, rf_pred_y)
svm_acc = accuracy_score(heart_y_test, svm_pred_y)
lr_acc = accuracy_score(heart_y_test, lr_pred_y)
```

Figure 58: Compute the accuracy score of the 3 models

The code above used the accuracy_score function of sklearn to compute the accuracy scores for the 3 models respectively. The scores are then stored in 3 variables.

```
# Shows the accuracy for the 3 algorithms rounded off to 2 decimal place
print("The accuracy for Random Forest : " + str(round(rf_acc, 2)))
print("The accuracy for Support Vector Machine : " + str(round(svm_acc, 2)))
print("The accuracy for Logistic Regression : " + str(round(lr_acc, 2)))
```

Figure 59: Display the accuracy scores for 3 models

The code above displays the accuracy scores for the 3 models respectively. The values have been rounded off to 2 decimal places.

```
# Creating bar plot to show the accuracy score
algorithms = ["Random Forest", "Support Vector Machine", "Logistic Regression"]
plt.figure(figsize=(10,5))
bp = sns.barplot(x = algorithms,y = [rf_acc, svm_acc, lr_acc])
bp.bar_label(container=bp.containers[0], labels=[round(rf_acc, 2),
                                                 round(svm_acc, 2), round(lr_acc, 2)])

plt.xlabel("Types of Algorithms")
plt.ylabel("Accuracy")
plt.title('Accuracy of Algorithms')
```

Figure 60: Create bar chart to show accuracy scores

The code above creates a bar chart to show the accuracy scores of the 3 models. The rounded off accuracy scores have been included as the label of the bars with the bar_label function.

F1 Score

```
# Compute the F1 score of 3 models
rf_f1 = f1_score(heart_y_test, rf_pred_y)
svm_f1 = f1_score(heart_y_test, svm_pred_y)
lr_f1 = f1_score(heart_y_test, lr_pred_y)
```

Figure 61: Compute the F1 Scores of the 3 models

The code above used the f1_score function of sklearn to compute the F1 scores for the 3 models respectively. The scores are then stored in 3 variables.

```
# Shows the F1 Score for the 3 algorithms rounded off to 2 decimal place
print("The F1 Score for Random Forest : " + str(round(rf_f1, 2)))
print("The F1 Score for Support Vector Machine : " + str(round(svm_f1, 2)))
print("The F1 Score for Logistic Regression : " + str(round(lr_f1, 2)))
```

Figure 62: Display the F1 Scores of the 3 models

The code above displays the F1 scores for the 3 models respectively. The values have been rounded off to 2 decimal places.

```
# Creating bar plot to show the F1 score
plt.figure(figsize=(10,5))
bp = sns.barplot(x = algorithms,y = [rf_f1, svm_f1, lr_f1])
bp.bar_label(container=bp.containers[0], labels=[round(rf_f1, 2),
                                                 round(svm_f1, 2), round(lr_f1, 2)])

plt.xlabel("Types of Algorithms")
plt.ylabel("F1 Score")
plt.title('F1 Score of Algorithms')
```

Figure 63: Creating bar plot of F1 scores of the 3 models

The code above creates a bar chart to show the F1 scores of the 3 models. The rounded off F1 scores have been included as the label of the bars with the bar_label function.

Precision

```
# Compute the Precision score of 3 models
rf_prec = precision_score(heart_y_test, rf_pred_y)
svm_prec= precision_score(heart_y_test, svm_pred_y)
lr_prec = precision_score(heart_y_test, lr_pred_y)
```

Figure 64: Compute the precision scores of 3 models

The code above used the precision_score function of sklearn to compute the Precision scores for the 3 models respectively. The scores are then stored in 3 variables.

```
# Shows the Precision Score for the 3 algorithms rounded off to 2 decimal place
print("The Precision Score for Random Forest : " + str(round(rf_prec, 2)))
print("The Precision Score for Support Vector Machine : " + str(round(svm_prec, 2)))
print("The Precision Score for Logistic Regression : " + str(round(lr_prec, 2)))
```

Figure 65: Display the Precision Score of the 3 models

The code above displays the precision scores for the 3 models respectively. The values have been rounded off to 2 decimal places.

```
# Creating bar plot to show the Precision score
plt.figure(figsize=(10,5))
bp = sns.barplot(x = algorithms,y = [rf_prec, svm_prec, lr_prec])
bp.bar_label(container=bp.containers[0], labels=[round(rf_prec, 2),
                                                 round(svm_prec, 2), round(lr_prec, 2)])

plt.xlabel("Types of Algorithms")
plt.ylabel("Precision Score")
plt.title('Precision Score of Algorithms')
```

Figure 66: Creating bar plot showing Precision scores of the 3 models

The code above creates a bar chart to show the Precision scores of the 3 models. The rounded off Precision scores have been included as the label of the bars with the bar_label function.

Confusion Matrix

```
cm = confusion_matrix(heart_y_test, rf_pred_y)
ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = rf_clf.classes_).plot()
plt.title("Confusion Matrix of Random Forest Classifier")
```

Figure 67: Creating Confusion Matrix for Random Forest Classifier

The code above creates a confusion matrix for the Random Forest Classifier. It used the confusion_matrix function of sklearn. The arguments passed are the true label and the predicted

values of Random Forest. The function ConfusionMatrixDisplay is to visualize the matrix. The label used the class names of the model.

```
cm = confusion_matrix(heart_y_test, svm_pred_y)
ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = svm_clf.classes_).plot()
plt.title("Confusion Matrix of SVM Classifier")
```

Figure 68: Creating confusion matrix for Support Vector Machine Classifier

The code above creates a confusion matrix for the Support Vector Machine Classifier. It used the confusion_matrix function of sklearn. The arguments passed are the true label and the predicted values of Support Vector Machine classifier. The function ConfusionMatrixDisplay is to visualize the matrix. The label used the class names of the model.

```
cm = confusion_matrix(heart_y_test, lr_pred_y)
ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = lr_clf.classes_).plot()
plt.title("Confusion Matrix of Logistic Regression Classifier")
```

Figure 69: Creating the confusion matrix for Logistic Regression

The code above creates a confusion matrix for the Logistic Regression Classifier. It used the confusion_matrix function of sklearn. The arguments passed are the true label and the predicted values of Logistic Regression classifier. The function ConfusionMatrixDisplay is to visualize the matrix. The label used the class names of the model.

Feature Importance/Weightage

```
# Creating bar plot to show the Feature Importance in Random Forest
sorted_index = rf_clf.feature_importances_.argsort()
sns.barplot(y = heart_X_train.columns[sorted_index],
            x = rf_clf.feature_importances_[sorted_index])
plt.ylabel("Features")
plt.title("Random Forest Feature Importances")
plt.xlabel("Importances")
```

Figure 70: Create a bar plot to show feature importance in Random Forest model

The code above creates a bar plot to show the feature importances in Random Forest Classifier model. It sorts the feature importances and obtains their index. It used the index to rearrange the columns and feature importances. The rearranged columns and feature importances are then passed to the barplot function to create the bar plot.

Plotting Experimental Data

Exploratory Data Analysis

Outlier Detection

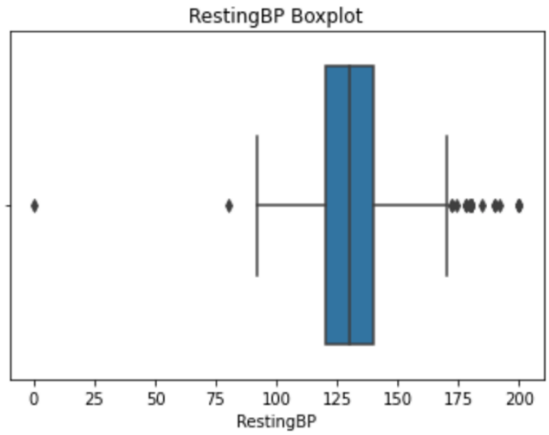


Figure 71 Resting Blood Pressure Boxplot

From the boxplot above, an outlier with Resting Blood Pressure of 0 is detected. It should be removed as it is impossible to have a 0 Resting Blood Pressure.

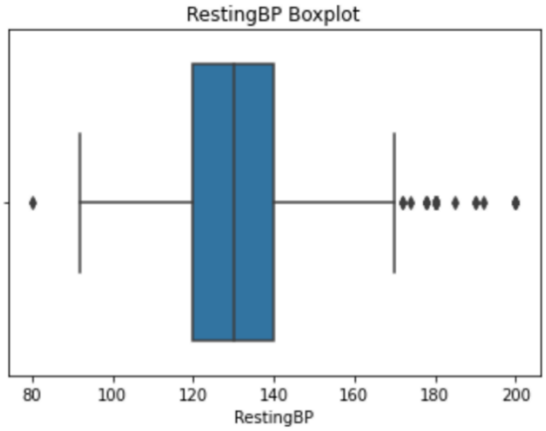


Figure 72 Enhanced Resting Blood Pressure Boxplot

The distribution changed significantly after the removal of outlier. This has justified the removal. The outlier may be caused by measurement error or other systematic error.

Data Understanding

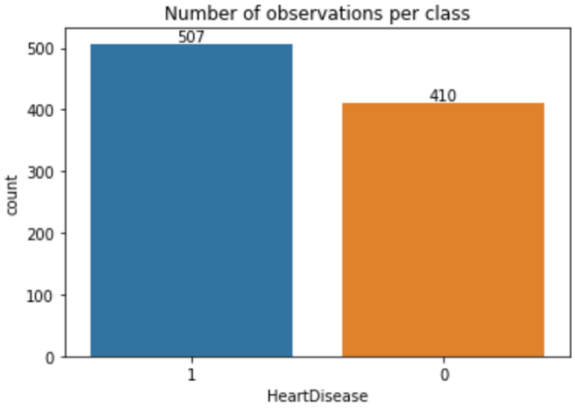


Figure 73 Number of Observations per Class

The bar graph shows the number of observations per class. Labels 0 and 1 are almost equal (44.71 % and 55.29 %) hence proving that the dataset is a balanced dataset.

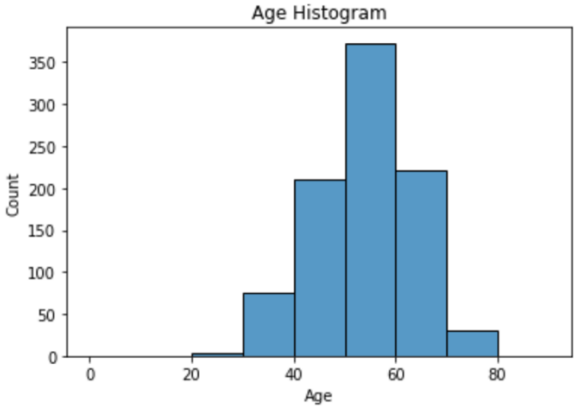


Figure 74 Age Histogram of Dataset

The age histogram shows the number of patients within a range of age. From the histogram, the dataset has the most patients within the age of 50 to 60, and the least patients within the age of 20 to 30. The average age of patients is 53 years old.

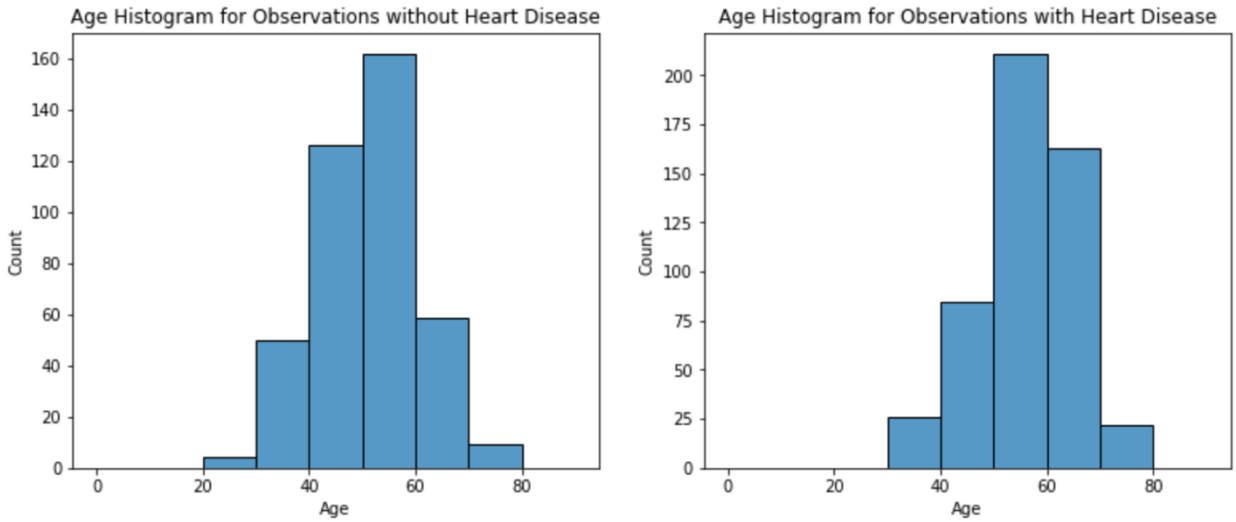


Figure 75 Age Histogram for People with and without Heart Disease

Both graphs above present the histograms of age for records with and without heart disease. From the graphs above, we can conclude that most patients below the age of 40 do not suffer from heart disease and patients within the age of 50 to 70 are more likely to be diagnosed with heart disease. The average age of patients who have heart disease is slightly older than those without.

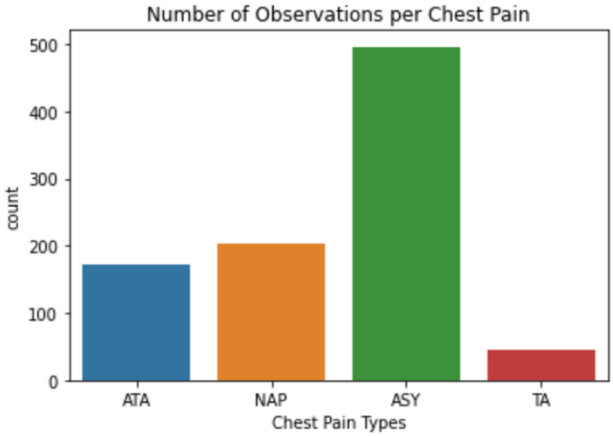


Figure 76 Number of Patients per Type of Chest Pain

According to the graph, it is noted that Asymptomatic (ASY) chest pain is the most common type of chest pain whereas Typical Angina (TA) chest pain is the least common type of chest pain.

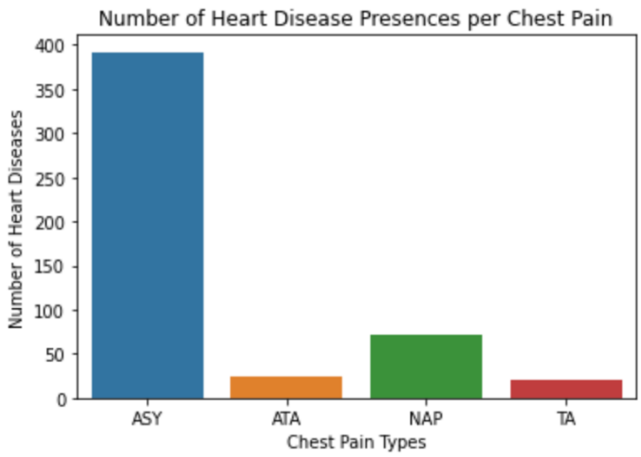


Figure 77 Number of Heart Diseases per Type of Chest Pain

Asymptomatic (ASY) chest pain is the most common chest pain for patients with heart disease, the proportion is significantly larger than other types of chest pain.

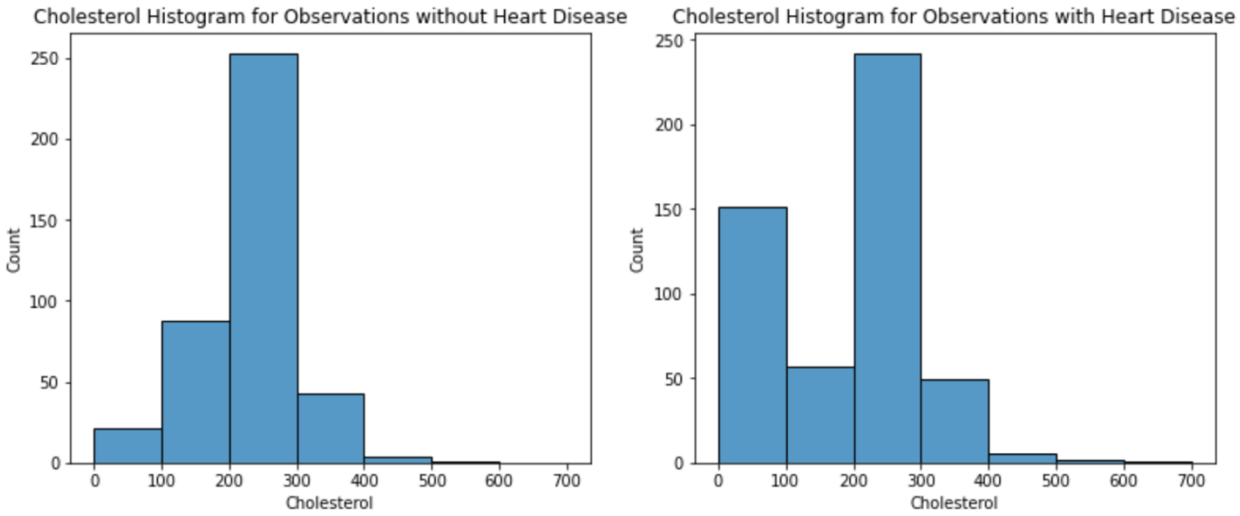


Figure 78 Cholesterol Histogram for Observations with and without Heart Disease

Patients with heart disease has a mean serum cholesterol lower than normal people. From the histograms, it can be observed that people with a relatively lower serum cholesterol (0-100 mm Hg), have a higher risk to have heart diseases.

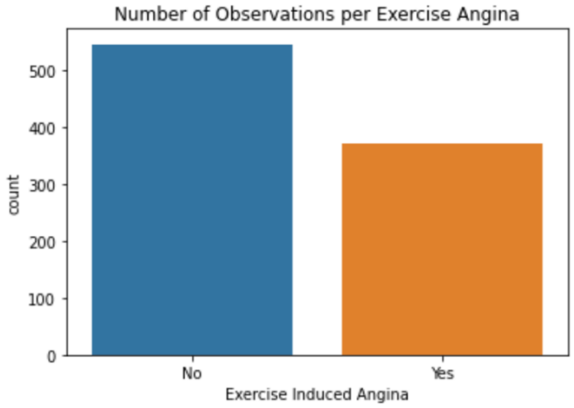


Figure 79 Number of Observations per Exercise Induced Angina

The number of observations which have no exercise induced angina (a chest pain induced by exercise) is higher.

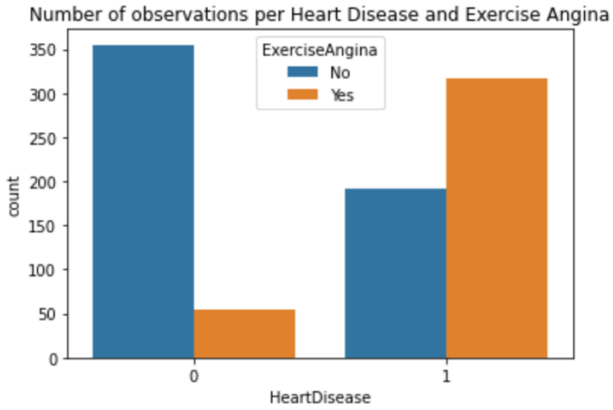


Figure 80 Number of Observations per Heart Disease and Exercise Induced Angina

People without heart disease mostly does not have exercise induced angina. While among people who have heart disease, the number of people who have exercise induced angina is higher.

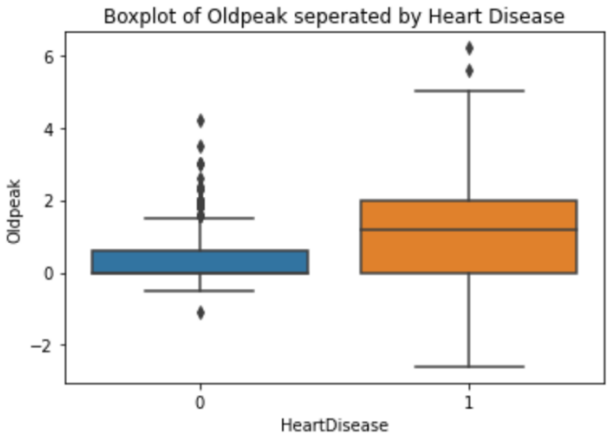


Figure 81 Boxplot of Oldpeak separated by Heart Disease

The average old peak for people without heart disease is lower.

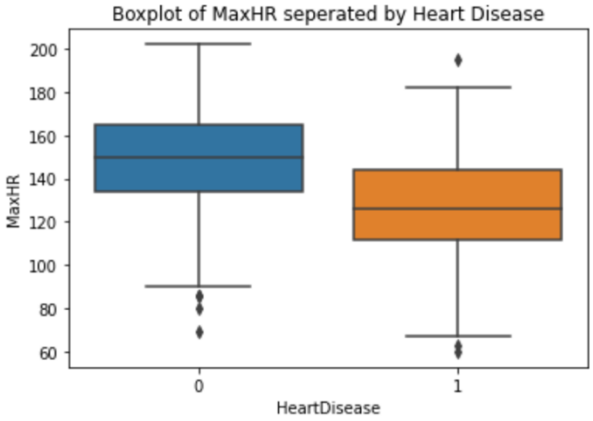


Figure 82 Boxplot of Max Heart Rate separated by Heart Disease

The average maximum heart rate achieved for people without heart disease is higher.

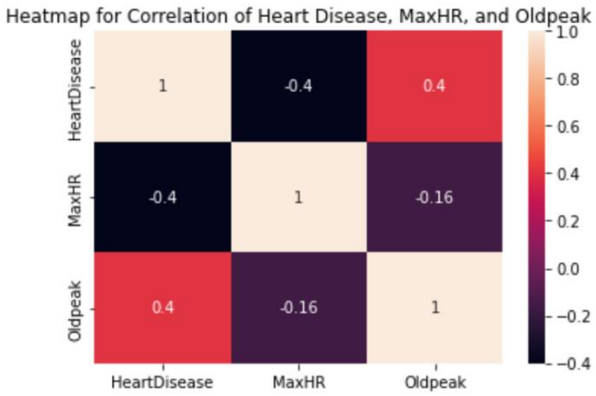


Figure 83 Heatmap for Correlation of Heart Disease, Maximum Heart Rate and Oldpeak

The heatmap above summarizes the information conveyed by the 2 boxplots above. Both Oldpeak and MaxHR (maximum heart rate achieved) shows a moderate correlation (0.4) with heart disease, whereby the MaxHR is negative correlation and Oldpeak is positive correlation.

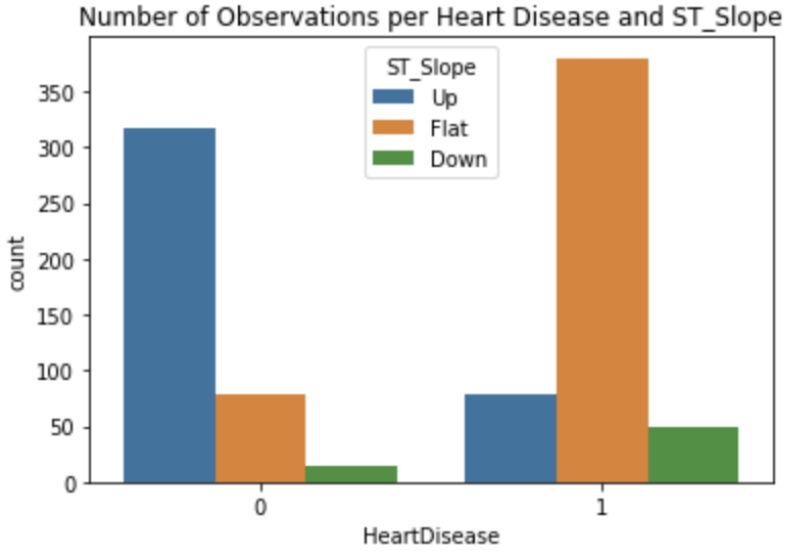


Figure 84 Number of Observations per Heart Disease and ST_Slope

It could be observed that for people who do not suffer from heart disease, most of them have Up ST_Slope. As for patients with heart disease, most of them have Flat ST_Slope.

Evaluation

Prediction

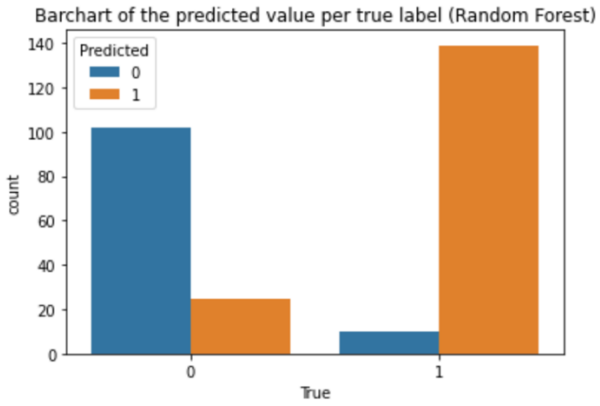


Figure 85 Bar Chart of Predicted Value per True Label (Random Forest)

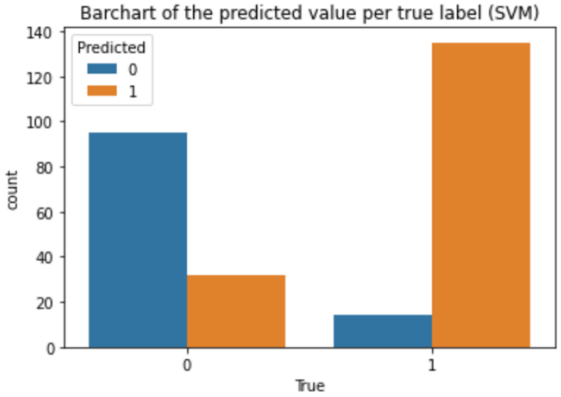


Figure 86 Bar Chart of Predicted Value per True Label (SVM)

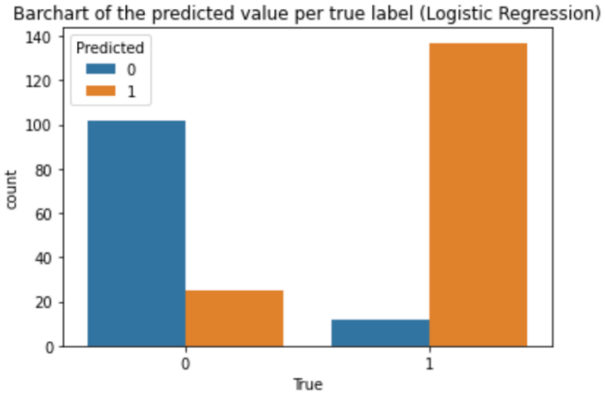


Figure 87 Bar Chart of Predicted Value per True Label (Logistic Regression)

Based on the 3 bar charts, Random Forest algorithm and Logistic Regression have similar predicted values, whereas Support Vector Machine has a fairly inaccurate predicted value compared to the other algorithms.

Accuracy

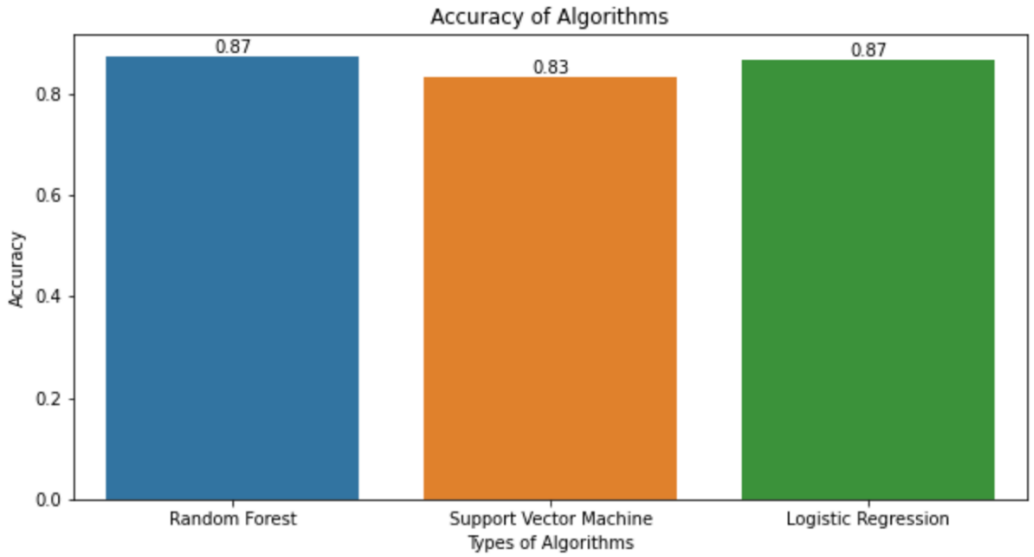


Figure 88 Accuracy Chart of Random Forest, SVM and Logistic Regression

Random Forest algorithm and Logistic Regression achieved the best accuracy, followed by Support Vector Machine.

F1 Score

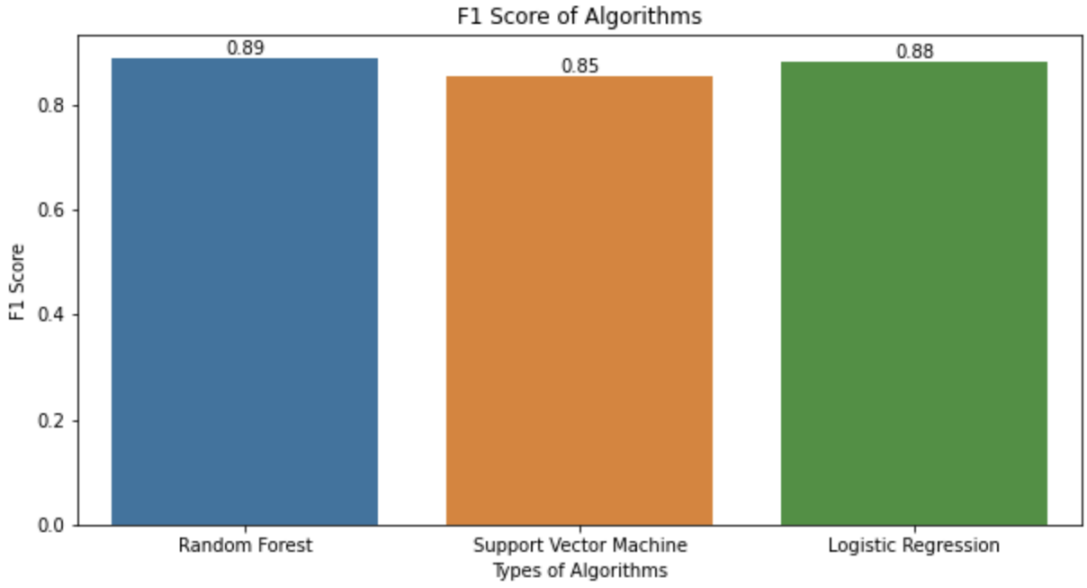


Figure 89 F1 Score Chart of Random Forest, SVM and Logistic Regression

Random Forest algorithm achieved the best F1 Score, followed by Logistic Regression and Support Vector Machine.

Precision

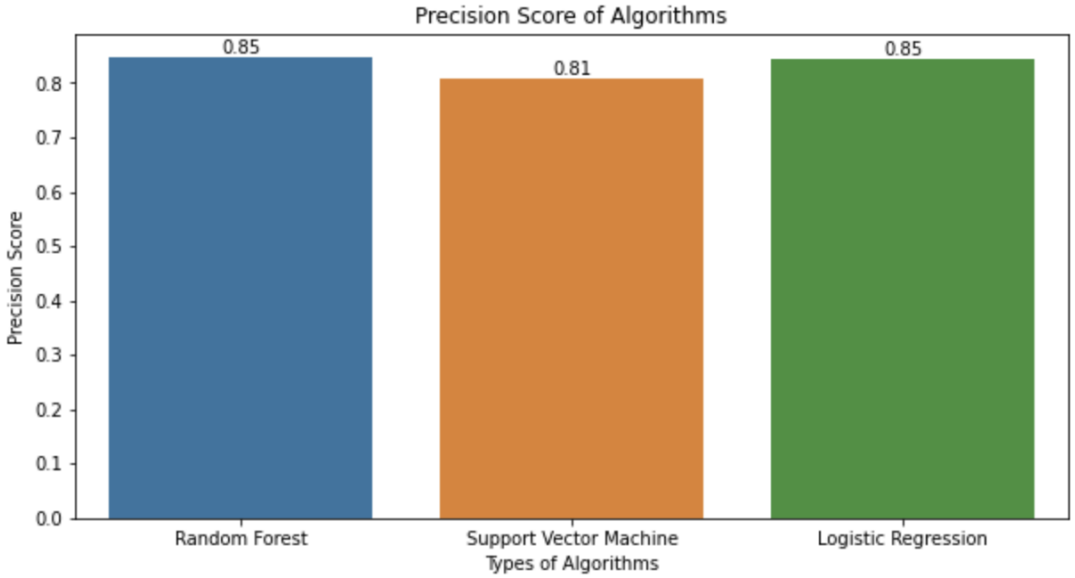


Figure 90 Precision Score Chart of Random Forest, SVM and Logistic Regression

Random Forest and SVM achieved the best Precision Score, followed by Logistic Regression.

Confusion Matrix

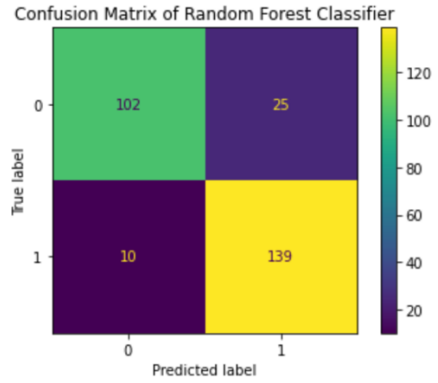


Figure 91 Confusion Matrix of Random Forest Classifier

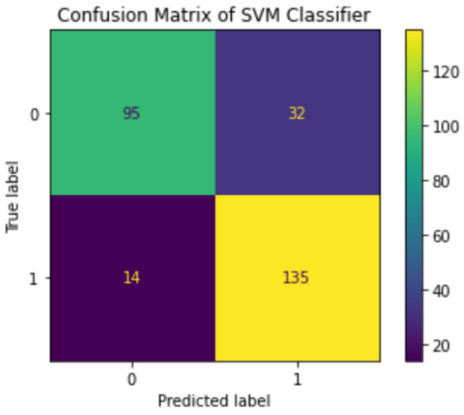


Figure 92 Confusion Matrix of SVM Classifier

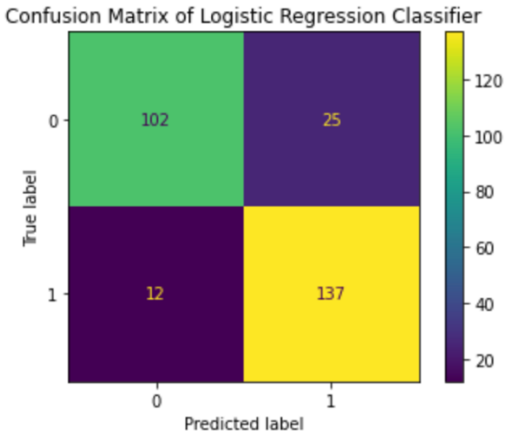


Figure 93 Confusion Matrix of Logistic Regression Classifier

From the confusion matrix, we could see SVM perform worst at it has the lowest value on the diagonal. Logistic Regression has same number of mistake (25) with Random Forest in predicting normal people as people who have heart disease. Logistic Regression make slightly more mistake in predicting heart disease patient as normal compared to Random Forest.

Feature Importance/Weightage

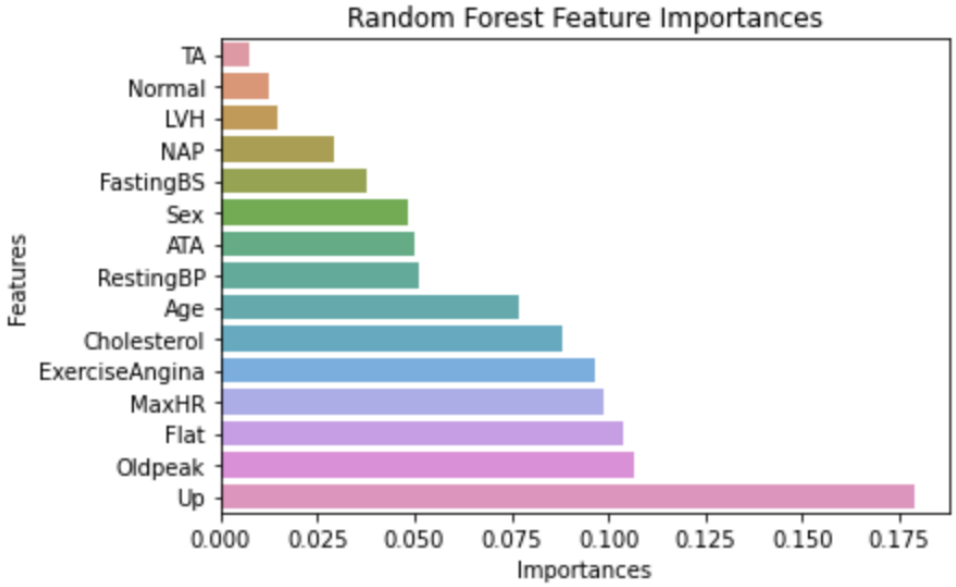


Figure 94 Random Forest Feature Importance Graph

According to the feature importance graph, Up feature is notably the most important feature, followed by Oldpeak, Flat, MaxHR, Exercise Angina, Cholesterol and Age. Other features including Resting Blood Pressure, Sex, Fasting Blood Sugar are less importance as they fall within a lower importance level.

Simulating Artificial Intelligence System / Tools

Justification on how operations perform.

The heart disease classification model starts with importing the necessary packages and reading the data into the system. The next step is exploratory data analysis, where an initial inspection of the data is performed to clean and understand it. For instance, this dataset has 918 rows and 12 columns, it shows that it has total of 12 features. The next step is crucial as it helps to improve the quality of the data, which is data cleaning and outlier detection, this is important for the model's accuracy.

Data cleaning phase is to check if any column has null values. Having missing or null values in the data can have a negative impact on the analysis and modeling process by causing issues in exploratory data analysis, data preprocessing, and modeling (Sonal, 2021). It can lead to skewed statistics, incorrect normalization, biased predictions, and skewed evaluation metrics (Sonal, 2021). It is important to handle missing or null values properly to ensure a valid and accurate analysis and modeling process.

The same also applies for outlier detection step, outlier detection is important in exploratory data analysis to identify and handle data points that are significantly different from the rest of the data (Enterprise Big Data Framework, 2020). Outliers can affect the accuracy of the summary statistics and visualizations, leading to incorrect conclusions (Sonal, 2021). In the modeling stage, outliers can also affect the performance of the model, leading to biased or incorrect predictions and overfitting (DataMites AI Team, 2020). Outlier detection helps to improve the accuracy of the analysis and modeling process.

Next, data understanding is a crucial step in the exploratory data analysis process as it helps to gain insights into the structure and properties of the data, informing the decisions made in the next steps of the analysis and modeling process.

After cleaning the data, data preprocessing is performed to transform the columns, making the data suitable for the machine learning models. In column transformation, information needs to be vied from the data frame to identify variables that need to be encoded before the modelling phase. While identifying possible values and counts, the Sex and ExerciseAngina attributes both have 2 possible values only. These types of variables will be used by LabelEncoder. On the other hand,

ChestPainType, RestingECG, and ST_Slope has 3 possible values. One hot encoding will be used when there are more than two possible values for a category variable.

LabelEncoder is used to encode categorical variables with only two possible values, such as a binary classification problem (Brownlee, 2020b). It works by assigning a unique integer to each category in the column (Brownlee, 2020b). For example, in a binary classification problem, two categories may be encoded as 0 and 1, respectively. On the other hand, LabelBinarizer, also known as one-hot encoding, is used to encode categorical variables with more than two possible values (Brownlee, 2020b). One-hot encoding works by converting each unique category in the column into a separate binary column, with a 1 in the row corresponding to the category and 0 in all other rows (Brownlee, 2020b). For example, if a column has 3 unique categories, one-hot encoding would result in 3 separate binary columns, each representing one of the categories. The reason why LabelEncoder is used for columns with only 2 possible values and LabelBinarizer for columns with more than 2 possible values is due to the nature of the algorithms used in the modeling process (Brownlee, 2020b). Some algorithms, such as logistic regression, can handle categorical variables with a single integer encoding, while others, such as decision trees and neural networks, work better with one-hot encoded categorical variables (Brownlee, 2020b). One-hot encoding helps to ensure that the categorical variables are represented in a way that is suitable for the algorithm used in the modeling process, helping to improve the performance and accuracy of the model (Brownlee, 2020b).

After performing One-hot encoding, dummy variable encoding and removing some dummy columns after LabelBinarizer is done to avoid the phenomenon of multicollinearity, which occurs when two or more independent variables are highly correlated (Brownlee, 2020b). In this situation, the coefficients in the model become unstable and difficult to interpret, and the model's ability to make accurate predictions is impacted. One common approach to avoiding multicollinearity is to remove one of the columns for each pair of highly correlated columns (Cope, 2023). This is known as "dummy variable trap". The dummy variable trap refers to the idea that one of the columns created from one-hot encoding must be removed to avoid perfect multicollinearity (Cope, 2023). This is because the information from all the columns is redundant, and one column can be used to represent the information from the others (Cope, 2023). In summary, removing some of the dummy columns after one-hot encoding is done to avoid the phenomenon of multicollinearity and to ensure

that the model is stable and accurate. By removing one column for each pair of highly correlated columns, the information represented by the columns is not lost, and the model's ability to make accurate predictions is not impacted. The data is then split into 70% for training and 30% for testing, the reason for this split is to avoid overfitting the model to the training data and to get a good estimate of how well the model will perform on new, unseen data. Then, 3 different machine learning models such as Random Forest, Support Vector Machine (SVM), Logistic Regression model are selected and fitted on the training data.

During the modelling phase, the 3 models have to undergo cross validation, hyperparameter tuning, and fitting step before evaluating the performance results. Cross-validation is a technique used to evaluate and tune the performance of machine learning models (Shulga, 2018). It involves dividing the dataset into multiple folds or subsets and training the model on different combinations of these folds while evaluating its performance on the remaining fold (Shulga, 2018).

In the case of hyperparameter tuning, cross-validation is used to find the best hyperparameters for a given machine learning model. Hyperparameters are parameters that are not learned from the data during model training but must be set before training begins (Brownlee, 2019). Examples of hyperparameters in Random Forest, Support Vector Machine (SVM), and Logistic Regression models include the number of trees in the forest for Random Forest, the regularization parameter for SVM, and the inverse of regularization strength for Logistic Regression (Brownlee, 2019).

The process of hyperparameter tuning using cross-validation works as follows:

1. Define a range of possible values for each hyperparameter
2. Train the model for each combination of hyperparameters using cross-validation
3. Evaluate the performance of the model for each combination using a performance metric such as accuracy, F1-score, precision, etc.
4. Choose the hyperparameters that result in the best performance

By using cross-validation to tune hyperparameters, we can avoid overfitting to the training data and get a more accurate estimate of the model's performance on unseen data (Baheti, 2023).

These selected models are evaluated using various metrics such as accuracy, F1-score, precision, and recall, along with a confusion matrix and feature weightage. The feature weightage helps to understand the contribution of each feature to the model's prediction. Finally, the output for the

heart disease classification model is generated, providing insights into the model's performance and the important features.

Output of the Health Disease Classification Model

Exploratory Data Analysis (EDA)

In machine learning, it is considered crucial to perform EDA before using the dataset. This is because EDA helps in getting a clear understanding of the data set that is being used for model training. EDA also helps detect the quality of data to seek out any potential missing values, outliers, and inconsistencies in the data so that it can be resolved before the model training phase since any inconsistencies may lead to incorrect model predictions or biasness. Additionally, EDA provides visualizations from bar charts, scatter plots, histograms, and heat maps. With visualization, patterns, trends, and correlations can be identified because it may not be apparent when looking at the raw data. Below shows the various output of the EDA in the Health Disease Classification Model.

Data Inspection

```
In [3]: # View the top records of the data
        heart.head()
```

Out[3]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

Figure 95: Output of the top records of the data set

Figure 95 shows the output of the top records of the data set.

```
In [4]: # View the dimensions of the data
        heart.shape

Out[4]: (918, 12)
```

Figure 96: Output of the dimensions of the data

Upon further analysis, the data has 918 rows and 12 columns as shown in figure 96.

```
In [5]: # View the features of the data
        heart.columns

Out[5]: Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
               'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
               'HeartDisease'],
              dtype='object')

In [6]: # View the information of the data
        heart.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 918 entries, 0 to 917
        Data columns (total 12 columns):
         #   Column          Non-Null Count  Dtype
        ---  ------          --------------  -----
         0   Age             918 non-null    int64
         1   Sex             918 non-null    object
         2   ChestPainType   918 non-null    object
         3   RestingBP       918 non-null    int64
         4   Cholesterol     918 non-null    int64
         5   FastingBS       918 non-null    int64
         6   RestingECG      918 non-null    object
         7   MaxHR           918 non-null    int64
         8   ExerciseAngina  918 non-null    object
         9   Oldpeak         918 non-null    float64
         10  ST_Slope        918 non-null    object
         11  HeartDisease    918 non-null    int64
        dtypes: float64(1), int64(6), object(5)
        memory usage: 86.2+ KB
```

Figure 97: Output of the column names and data information

In figure 97, the columns names and the data information were inspected. According to the description in Kaggle, the variables (columns) carry the following meaning:

1. Age: age of the patient [years]
2. Sex: sex of the patient [M: Male, F: Female]
3. ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
4. RestingBP: resting blood pressure [mm Hg]
5. Cholesterol: serum cholesterol [mm/dl]
6. FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
7. RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
8. MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
9. ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
10. Oldpeak: oldpeak = ST [Numeric value measured in depression]

11. ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
12. HeartDisease: output class [1: heart disease, 0: Normal]

The variables/features mentioned above are considered important information because it was widely used in previous research, evident in Part One. Furthermore, the description in Kaggle highlighted that some categorical variables must be encoded before the modelling phase.

Data Cleaning

```
In [7]: # Check if there is column with null value
        heart.isnull().sum()

Out[7]: Age              0
        Sex              0
        ChestPainType    0
        RestingBP        0
        Cholesterol      0
        FastingBS        0
        RestingECG       0
        MaxHR            0
        ExerciseAngina   0
        Oldpeak          0
        ST_Slope         0
        HeartDisease     0
        dtype: int64
```

Figure 98: Output of data cleaning step to check whether there are any columns with null values.

The figure above shows the output of isnull() sum to check whether the data set has any null value. The output shows that there is no missing value in the dataset. Thus, the dataset is clean.

Outlier Detection

```
In [8]: # Create a box plot for Resting BP
        sns.boxplot(data = heart, x="RestingBP")
        plt.title('RestingBP Boxplot')

Out[8]: Text(0.5, 1.0, 'RestingBP Boxplot')
```

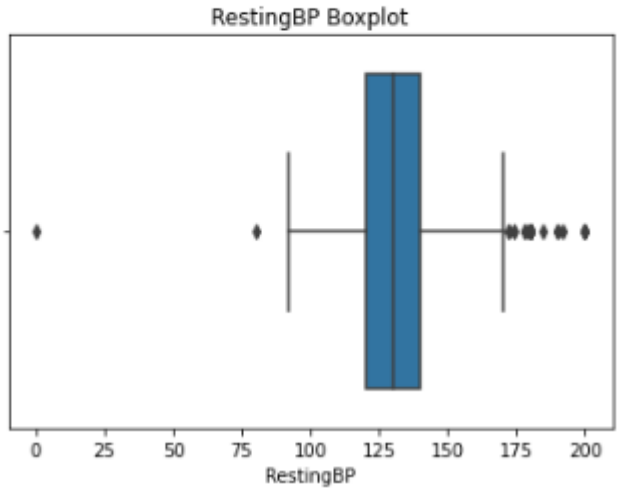


Figure 99: Output of the outlier detection step

Figure 99 showcases the potential outlier in the dataset. From the boxplot in figure 99, outlier is detected, and it should be further inspection.

```
In [9]: # View statistical properties of Resting BP
        heart["RestingBP"].describe()

Out[9]: count    918.000000
        mean     132.396514
        std       18.514154
        min        0.000000
        25%      120.000000
        50%      130.000000
        75%      140.000000
        max      200.000000
        Name: RestingBP, dtype: float64
```

Figure 100: Output of the statistical properies of "Resing BP" column

```
In [10]: # Filter record with RestingBP equals to 0
         heart[heart["RestingBP"] == 0]
```

Out[10]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
449	55	M	NAP	0	0	0	Normal	155	N	1.5	Flat	1

Figure 101: Output of filter where "RestingBP" = 0

```
In [11]: # Remove the record that has 0 Resting BP
         heart = heart[heart["RestingBP"] != 0]

In [12]: # Create a box plot for Resting BP
         sns.boxplot(data = heart, x="RestingBP")
         plt.title('RestingBP Boxplot')

Out[12]: Text(0.5, 1.0, 'RestingBP Boxplot')
```

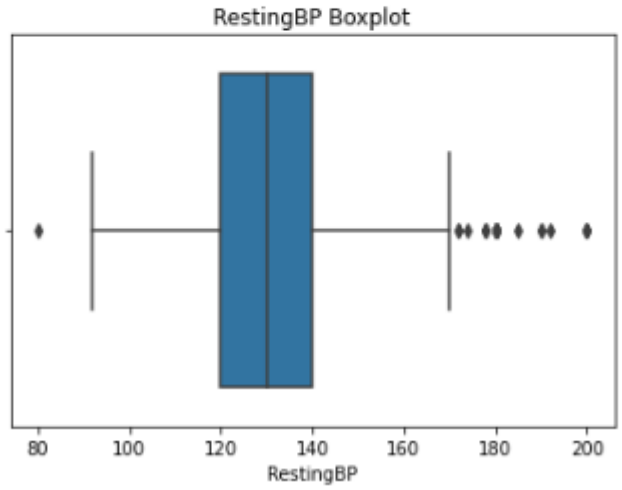


Figure 102: Removal of outlier

The figure above showcases the boxplot result of the outlier after being removed. Outlier was further explained in the previous section here.

Data Understanding

In [13]:
```
# View number of records for each label
print(heart["HeartDisease"].value_counts())

# Compute the percentage of records for each label
perc_of_classes = heart["HeartDisease"].value_counts().values / len(heart) * 100
print("Percentage of class with heart diseases : {:.2f} %".format(perc_of_classes[0]))
print("Percentage of class normal : {:.2f} %".format(perc_of_classes[1]))
```

```
1    507
0    410
Name: HeartDisease, dtype: int64
Percentage of class with heart diseases : 55.29 %
Percentage of class normal : 44.71 %
```

In [14]:
```
# Generate a bar plot showing No of records per label
target_counts = heart['HeartDisease'].value_counts().values

# Create the labels for the bars
bc = sns.countplot(x="HeartDisease", data=heart, order=heart['HeartDisease'].value_counts().index)
bc.bar_label(container=bc.containers[0], labels=target_counts)
plt.title('Number of observations per class')
```

Out[14]:
```
Text(0.5, 1.0, 'Number of observations per class')
```

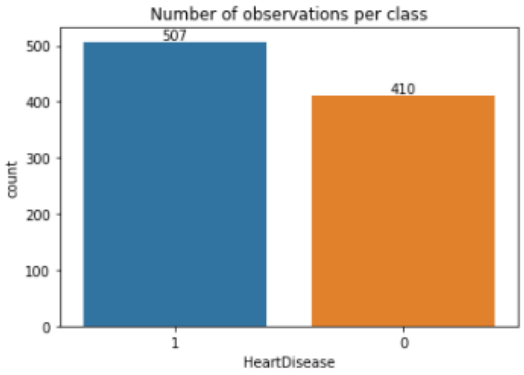


Figure 103: Output of the percentage of record for each label and the bar plot of the number of observations per class

In [15]:
```
# Create the histogram for Age
sns.histplot(heart, x = "Age",  bins=[0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
plt.title('Age Histogram')

# Compute the mean age
print("Average age : {:.2f}".format(heart["Age"].mean()))
```

```
Average age : 53.51
```

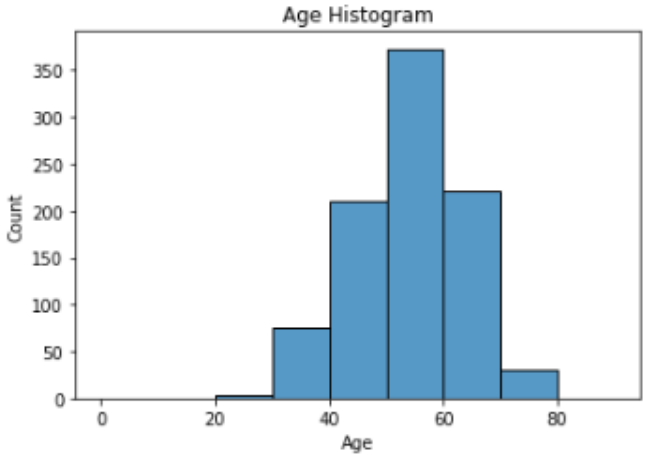


Figure 104: Histogram of "Age" and its mean

In [16]:
```
#  Create the histogram of age for records without heart disease
plt.figure(figsize=(13,5))
plt.subplot(1, 2, 1)
sns.histplot(heart[heart['HeartDisease'] == 0], x="Age", bins=[0, 10, 20, 30, 40,
                                                               50, 60, 70, 80, 90])

plt.title('Age Histogram for Observations without Heart Disease')

#  Create the histogram of age for records with heart disease
plt.subplot(1, 2, 2)
sns.histplot(heart[heart['HeartDisease'] == 1], x="Age", bins=[0, 10, 20, 30, 40,
                                                               50, 60, 70, 80, 90])
plt.title('Age Histogram for Observations with Heart Disease')

# Compute the mean of age for records without and with heart disease
print("Average age for observations without heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 0]["Age"].mean()))
print("Average age for observations with heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 1]["Age"].mean()))
```

```
Average age for observations without heart disease: 50.55
Average age for observations with heart disease: 55.90
```

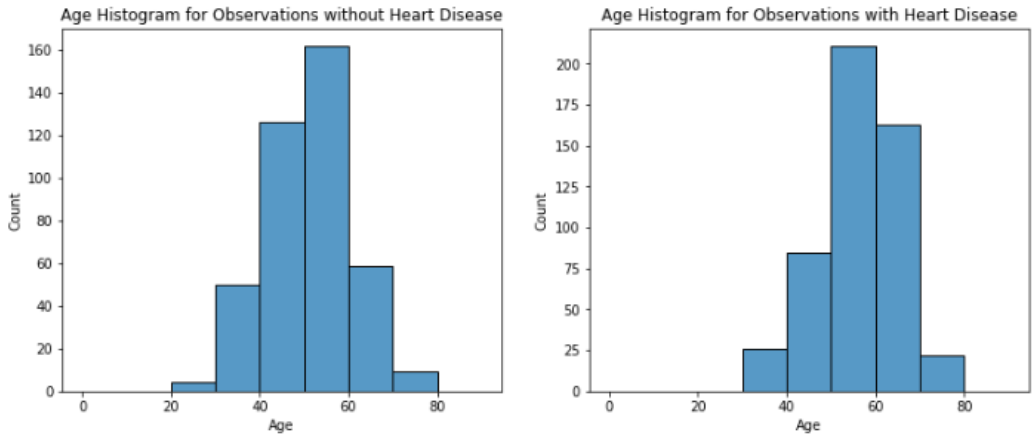


Figure 105: Histogram for "Age" with and without Heart Disease

```
In [17]: # Generate a bar plot showing No of records per chest pain
         bc = sns.countplot(x="ChestPainType", data=heart)
         plt.xlabel("Chest Pain Types")
         plt.title('Number of Observations per Chest Pain')
```

```
Out[17]: Text(0.5, 1.0, 'Number of Observations per Chest Pain')
```

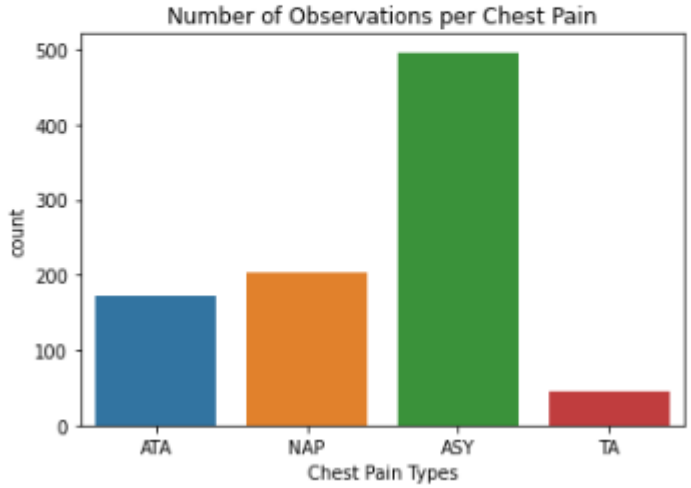


Figure 106: Bar chart for the number of observations per chest pain

```
In [18]: # Sum the value group by ChestPainType, this allow the total Heart Disease (with label 1)
         # to be computed
         heart_gb_cp_sum = heart.groupby("ChestPainType").sum(numeric_only = True)

         # Generate a bar plot showing No of Heart Diseases per chest pain
         sns.barplot(x=heart_gb_cp_sum.index, y = "HeartDisease", data=heart_gb_cp_sum)
         plt.xlabel("Chest Pain Types")
         plt.ylabel("Number of Heart Diseases")
         plt.title('Number of Heart Disease Presences per Chest Pain')
```

```
Out[18]: Text(0.5, 1.0, 'Number of Heart Disease Presences per Chest Pain')
```

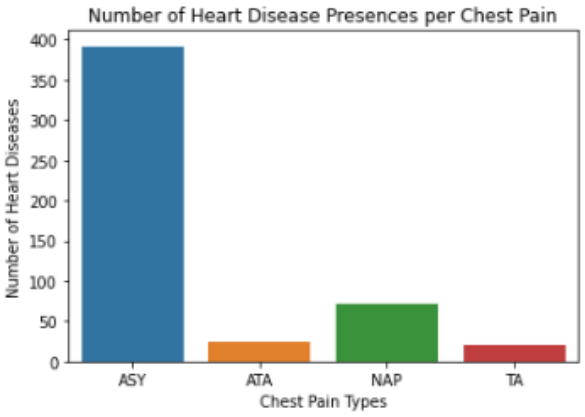


Figure 107: Bar chart of the number of heart disease presences per chest pain

In [19]:
```
#  Create the histogram of cholesterol for records without heart disease
plt.figure(figsize=(13,5))
plt.subplot(1, 2, 1)
sns.histplot(heart[heart['HeartDisease'] == 0], x="Cholesterol", bins=[0, 100, 200,
                                                                       300, 400, 500, 600, 700])

plt.title('Cholesterol Histogram for Observations without Heart Disease')

#  Create the histogram of cholesterol for records with heart disease
plt.subplot(1, 2, 2)
sns.histplot(heart[heart['HeartDisease'] == 1], x="Cholesterol", bins=[0, 100, 200,
                                                                       300, 400, 500, 600, 700])
plt.title('Cholesterol Histogram for Observations with Heart Disease')

#  Compute the mean of cholesterol for records without and with heart disease
print("Average Cholesterol for observations without heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 0]["Cholesterol"].mean()))
print("Average Cholesterol for observations with heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 1]["Cholesterol"].mean()))
```

```
Average Cholesterol for observations without heart disease: 227.12
Average Cholesterol for observations with heart disease: 176.29
```

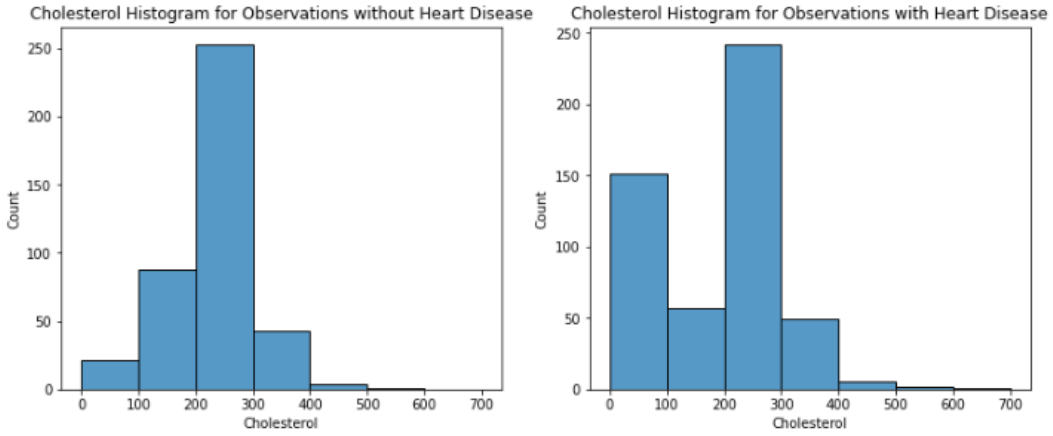


Figure 108: Cholesterol histogram for observations with and without heart disease

```
In [20]: # Create bar plot showing no of observations per Exercise Angina
         ax = sns.countplot(x="ExerciseAngina", data=heart)
         plt.xlabel("Exercise Induced Angina")
         plt.title('Number of Observations per Exercise Angina')
         ax.set_xticklabels(['No', 'Yes'])

Out[20]: [Text(0, 0, 'No'), Text(1, 0, 'Yes')]
```

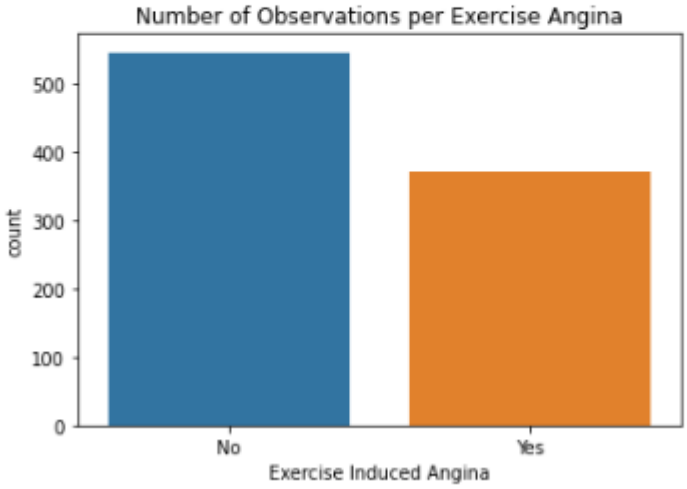


Figure 109: Bar chart of the number of observation per Exercise Angina

```
In [21]: # Create bar plot showing no of observations per Exercise Angina and Heart Disease
         ax = sns.countplot(x="HeartDisease", hue="ExerciseAngina" , data=heart)
         plt.title('Number of observations per Heart Disease and Exercise Angina')
         ax.legend(title='ExerciseAngina', labels=['No', 'Yes'])

Out[21]: <matplotlib.legend.Legend at 0x1b7569a6790>
```

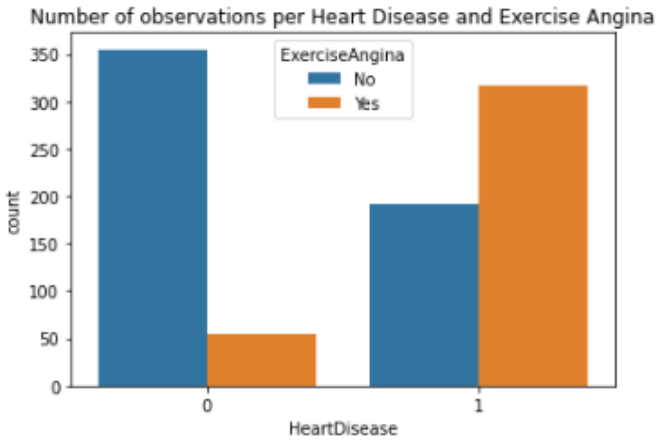


Figure 110: Bar chart of the number of observation per heart disease and Exercise Angina

In [22]:
```
# Create boxplot of Oldpeak seperated by Heart Disease
sns.boxplot(x ='HeartDisease', y = "Oldpeak" , data =heart)
plt.title('Boxplot of Oldpeak seperated by Heart Disease')

#  Compute the mean of Oldpeak for records without and with heart disease
print("Average Oldpeak for observations without heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 0]["Oldpeak"].mean()))
print("Average Oldpeak for observations with heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 1]["Oldpeak"].mean()))
```

```
Average Oldpeak for observations without heart disease: 0.41
Average Oldpeak for observations with heart disease: 1.27
```

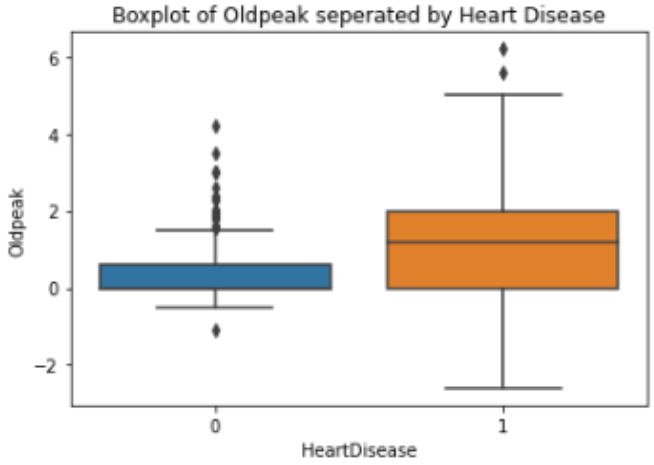


Figure 111: Boxplot of Oldpeak separated by heart disease.

In [23]:
```
# Create boxplot of MaxHR seperated by Heart Disease
sns.boxplot(x ='HeartDisease', y = "MaxHR" , data =heart)
plt.title('Boxplot of MaxHR seperated by Heart Disease')

#  Compute the mean of MaxHR for records without and with heart disease
print("Average MaxHR for observations without heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 0]["MaxHR"].mean()))
print("Average MaxHR for observations with heart disease: {:.2f}".format(
    heart[heart['HeartDisease'] == 1]["MaxHR"].mean()))
```

```
Average MaxHR for observations without heart disease: 148.15
Average MaxHR for observations with heart disease: 127.60
```

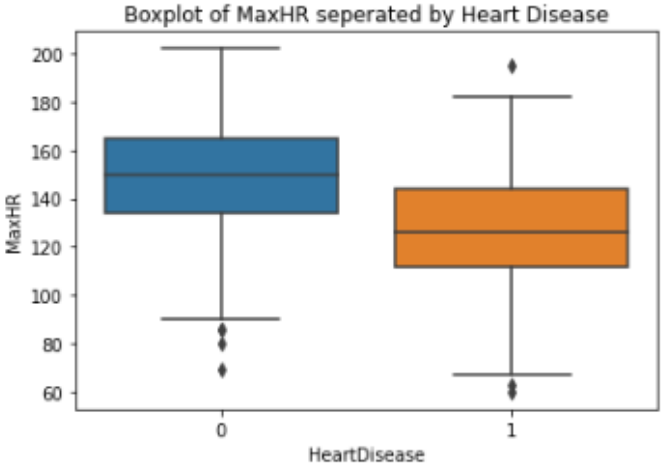


Figure 112: Boxplot of MaxHR separated by heart disease.

In [24]:
```
# Create heatmap showing correlation of MaxHR, Oldpeak and Heart Disease
sns.heatmap(heart[["HeartDisease","MaxHR","Oldpeak"]].corr(), annot=True)
plt.title('Heatmap for Correlation of Heart Disease, MaxHR, and Oldpeak')
```

Out[24]:
```
Text(0.5, 1.0, 'Heatmap for Correlation of Heart Disease, MaxHR, and Oldpeak')
```

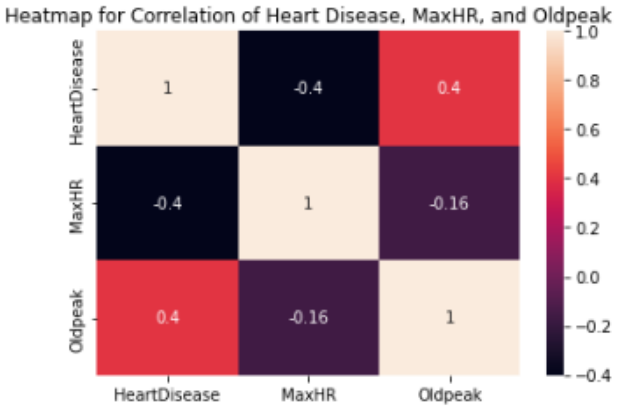


Figure 113: Heatmap shing the correlation of MaxHR, Oldpeak and Heart Disease

```
In [25]: # Create bar plot showing no of observations per ST_Slope and Heart Disease
         ax = sns.countplot(x="HeartDisease", hue="ST_Slope" , data=heart)
         plt.title('Number of Observations per Heart Disease and ST_Slope')

Out[25]: Text(0.5, 1.0, 'Number of Observations per Heart Disease and ST_Slope')
```

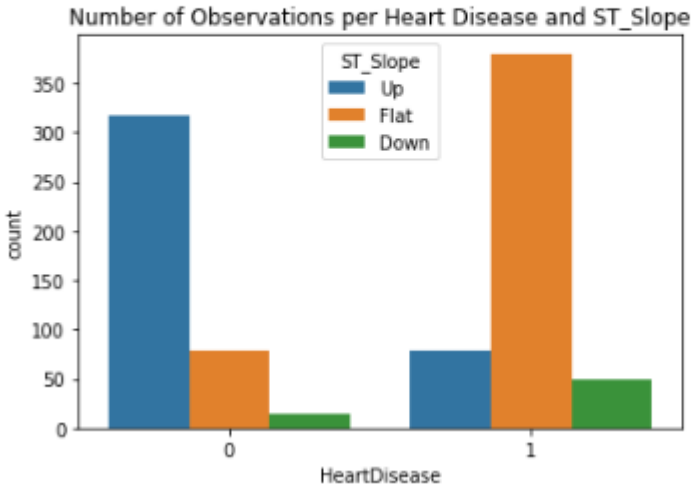


Figure 114: Bar chart of the number of observations per heart disease and ST_Slope

Data understanding is further discussed in previous sections here.

Data Preparation
Data preparation is essential because it affects the quality and performance of the classification model. This is because in some cases such as this dataset, there may have varying scales or units making it difficult for the model to learn the patterns, normalization of column transformation and standardization can help brin data into a similar scale to reduce the biasness in certain features. Additionally, the model can be simplified and improve its interpretability. For example transforming a column of categorical data into a set of binary columns can help to capture the relationship between the categories in a more straightforward way.
Column Transformation

```
In [26]: # View information of the data frame
         heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 917 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             917 non-null    int64
 1   Sex             917 non-null    object
 2   ChestPainType   917 non-null    object
 3   RestingBP       917 non-null    int64
 4   Cholesterol     917 non-null    int64
 5   FastingBS       917 non-null    int64
 6   RestingECG      917 non-null    object
 7   MaxHR           917 non-null    int64
 8   ExerciseAngina  917 non-null    object
 9   Oldpeak         917 non-null    float64
 10  ST_Slope        917 non-null    object
 11  HeartDisease    917 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 125.4+ KB
```

Figure 115: Information of the data frame

```
In [27]: # view top records
         heart.head()
```

Out[27]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

Figure 116: Top records of the dataset

```
In [28]: # View possible values and its counts of sex
         heart['Sex'].value_counts()
```

```
Out[28]: M    724
         F    193
         Name: Sex, dtype: int64
```

```
In [29]: # View possible values and its counts of ExerciseAngina
         heart['ExerciseAngina'].value_counts()
```

```
Out[29]: N    546
         Y    371
         Name: ExerciseAngina, dtype: int64
```

Figure 117: Value of the count of "Sex" and Exercise Angina

```
In [30]: # Create label encoder
         le = preprocessing.LabelEncoder()

         # Encode sex and exerciseangina columns' values
         heart['Sex']= le.fit_transform(heart['Sex'])
         heart['ExerciseAngina']= le.fit_transform(heart['ExerciseAngina'])

In [31]: # Check encoding result
         heart.head()
```

Out[31]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	1	ATA	140	289	0	Normal	172	0	0.0	Up	0
1	49	0	NAP	160	180	0	Normal	156	0	1.0	Flat	1
2	37	1	ATA	130	283	0	ST	98	0	0.0	Up	0
3	48	0	ASY	138	214	0	Normal	108	1	1.5	Flat	1
4	54	1	NAP	150	195	0	Normal	122	0	0.0	Up	0

Figure 118: Creating an encoder and its output

```
In [32]: # Confirm encoding result
         heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 917 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             917 non-null    int64
 1   Sex             917 non-null    int32
 2   ChestPainType   917 non-null    object
 3   RestingBP       917 non-null    int64
 4   Cholesterol     917 non-null    int64
 5   FastingBS       917 non-null    int64
 6   RestingECG      917 non-null    object
 7   MaxHR           917 non-null    int64
 8   ExerciseAngina  917 non-null    int32
 9   Oldpeak         917 non-null    float64
 10  ST_Slope        917 non-null    object
 11  HeartDisease    917 non-null    int64
dtypes: float64(1), int32(2), int64(6), object(3)
memory usage: 118.3+ KB
```

Figure 119: Output of the resulting encoding

```
In [33]: # View possible values and its counts of ChestPainType
         heart['ChestPainType'].value_counts()

Out[33]: ASY    496
         NAP    202
         ATA    173
         TA      46
         Name: ChestPainType, dtype: int64

In [34]: # View possible values and its counts of RestingECG
         heart['RestingECG'].value_counts()

Out[34]: Normal    551
         LVH       188
         ST        178
         Name: RestingECG, dtype: int64

In [35]: # View possible values and its counts of ST_Slope
         heart['ST_Slope'].value_counts()

Out[35]: Flat    459
         Up      395
         Down     63
         Name: ST_Slope, dtype: int64
```

Figure 120: Count of ChestPainType, RestingECG and ST_Slope

```
In [36]: # Create Label Binarizer to implement one hot encoding
         lb = preprocessing.LabelBinarizer()

         # Encode ChestPainType column with Label Binarizer
         heart = heart.join(pd.DataFrame(lb.fit_transform(heart["ChestPainType"]),
                                         columns=lb.classes_,
                                         index=heart.index))

         # Encode RestingECG column with Label Binarizer
         heart = heart.join(pd.DataFrame(lb.fit_transform(heart["RestingECG"]),
                                         columns=lb.classes_,
                                         index=heart.index))

         # Encode STSlope column with Label Binarizer
         heart = heart.join(pd.DataFrame(lb.fit_transform(heart["ST_Slope"]),
                                         columns=lb.classes_,
                                         index=heart.index))
```

Figure 121: One hot encoding (binary columns) using label binarizer for ChestPainType, RestingECG, STSlope

```
In [37]: # Confirm encoding result
         heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 917 entries, 0 to 917
Data columns (total 22 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             917 non-null    int64
 1   Sex             917 non-null    int32
 2   ChestPainType   917 non-null    object
 3   RestingBP       917 non-null    int64
 4   Cholesterol     917 non-null    int64
 5   FastingBS       917 non-null    int64
 6   RestingECG      917 non-null    object
 7   MaxHR           917 non-null    int64
 8   ExerciseAngina  917 non-null    int32
 9   Oldpeak         917 non-null    float64
 10  ST_Slope        917 non-null    object
 11  HeartDisease    917 non-null    int64
 12  ASY             917 non-null    int32
 13  ATA             917 non-null    int32
 14  NAP             917 non-null    int32
 15  TA              917 non-null    int32
 16  LVH             917 non-null    int32
 17  Normal          917 non-null    int32
 18  ST              917 non-null    int32
 19  Down            917 non-null    int32
 20  Flat            917 non-null    int32
 21  Up              917 non-null    int32
dtypes: float64(1), int32(12), int64(6), object(3)
memory usage: 154.1+ KB
```

Figure 122: Result of encoding

```
In [38]: # Drop the original columns that has been encode by LabelBinarizer
         heart = heart.drop(columns=["ChestPainType", "RestingECG","ST_Slope"])
```

```
In [39]: # Confirm dropping result
         heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 917 entries, 0 to 917
Data columns (total 19 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             917 non-null    int64  
 1   Sex             917 non-null    int32  
 2   RestingBP       917 non-null    int64  
 3   Cholesterol     917 non-null    int64  
 4   FastingBS       917 non-null    int64  
 5   MaxHR           917 non-null    int64  
 6   ExerciseAngina  917 non-null    int32  
 7   Oldpeak         917 non-null    float64
 8   HeartDisease    917 non-null    int64  
 9   ASY             917 non-null    int32  
 10  ATA             917 non-null    int32  
 11  NAP             917 non-null    int32  
 12  TA              917 non-null    int32  
 13  LVH             917 non-null    int32  
 14  Normal          917 non-null    int32  
 15  ST              917 non-null    int32  
 16  Down            917 non-null    int32  
 17  Flat            917 non-null    int32  
 18  Up              917 non-null    int32  
dtypes: float64(1), int32(12), int64(6)
memory usage: 132.6 KB
```

Figure 123: Removing unnecessary columns.

```
In [40]: # Drop the columns with redundant information
         # ASY ATA NAP TA - one of them should be dropped
         # LVH Normal ST - one of them should be dropped
         # Down Flat Up - one of them should be dropped

         heart = heart.drop(columns=["ASY","ST","Down"])
```

Figure 124: Dropping the columns.

Columns transformation the one hot encoding was thoroughly discussed in the previous section in justification on how operations perform.

Train Test Split

```
In [41]: # Split the predictors and target variable
         heart_X = heart.iloc[:, heart.columns != "HeartDisease"]
         heart_y = heart["HeartDisease"]
```

Figure 125: Dataset split.

```
In [42]: # Perform 70 : 30 train test split
         heart_X_train, heart_X_test, heart_y_train, heart_y_test = train_test_split(
             heart_X, heart_y, test_size=0.30, random_state=123)

In [43]: # View dimensions of traing data X
         heart_X_train.shape

Out[43]: (641, 15)
```

Figure 126: Dataset is split into 70 30 and dimension output of data X

The figure above shows that training data X has 641 rows and15 columns.

```
In [44]: # View dimensions of testing data X
         heart_X_test.shape

Out[44]: (276, 15)
```

Figure 127: Dimension of testing data X

The figure above shows that testing data X has 276 rows and15 columns.

Modelling
Random Forest

```
In [45]: # using cross validation to find the optimal hyperparameter
scores = []
hyperparams = []

for n_estimator in [500,1000,2000]:
    for max_depth in [3,5,7]:
        for max_features in [2,3,4]:
            rf_clf = RandomForestClassifier(n_estimators=n_estimator, max_depth=max_depth,
                                            random_state=0, max_features = max_features)
            cv_scores = cross_val_score(rf_clf, heart_X_train, heart_y_train, cv=5)
            scores.append(cv_scores.mean())
            hyperparams.append((n_estimator,max_depth,max_features))

In [46]: # Show the best hyperparameter
best_hyperparam = hyperparams[np.argmax(scores)]
print("Optimal n_estimator : " + str(best_hyperparam[0]))
print("Optimal max_depth : " + str(best_hyperparam[1]))
print("Optimal max_features : " + str(best_hyperparam[2]))

Optimal n_estimator : 500
Optimal max_depth : 7
Optimal max_features : 2

In [47]: # Creating Random Forest Model
rf_clf = RandomForestClassifier(n_estimators=500, max_depth=7, random_state=0, max_features = 2)

# Fitting the model to training data
rf_clf.fit(heart_X_train, heart_y_train)

Out[47]: RandomForestClassifier
RandomForestClassifier(max_depth=7, max_features=2, n_estimators=500,
                       random_state=0)
```

Figure 128: RandomForest output

Figure 128 above shows the output of the Random Forest Model. In the cross validation to find the optimal hyperparameter, n_estimator, max_depth and max_feature is used. These 3 key terms are used in Random Forest classification.

"n_estimators" determines the number of trees that are included in Random Forest. It represents the number of decision trees that are created and combined to make the final prediction. To further evaluate, the more trees included, the better Random Forest will be at avoiding overfitting but adding more trees also increases training time and slow down the process of prediction.

For max_depth however, is to control the maximum depth of each decision tree in Random Forest. Max_depth limits the number of splits that each can have that helps prevent overfitting. To further evaluate, if max_depth is too high, the model will be too complex and will overfit the data. Whereas if it too low, the model will be too simple and might not capture the complexity of data.

In max_features, the parameter controls the maximum number of features that are considered for splitting each node of the decision tree. Furthermore, it helps prevent Random Forest from focusing too much on small number of features that can result in overfitting. To evaluate further, if max_feature is too high, Random Forest will overfit the data whereas if is too low, it may not capture the complexity of data which is like max_depth.

Support Vector Machine (SVM)

```
In [48]: # using cross validation to find the optimal hyperparameter
         scores = []
         hyperparams = []

         for C in [100,200,500,1000]:
             svm_clf = SVC(C= C)
             cv_scores = cross_val_score(svm_clf, heart_X_train, heart_y_train, cv=5)
             scores.append(cv_scores.mean())
             hyperparams.append((C))

In [49]: # Show the best hyperparameter
         best_hyperparam = hyperparams[np.argmax(scores)]
         print("Optimal C : " + str(best_hyperparam))

         Optimal C : 1000

In [50]: # Creating SVM model
         svm_clf = SVC(C= 1000)

         # Fitting the model to training data
         svm_clf.fit(heart_X_train, heart_y_train)

Out[50]: SVC
         SVC(C=1000)
```

Figure 129: SVM output

Figure 129 shows the output of SVM. While finding the optimal hyperparameter, some terms were used such as C and cv_scores.

C is a hyperparameter that controls the trade-off between achieving a low training error and a low testing error in Support Vector Machine (SVM) algorithm. A smaller value of C creates a wider margin that allows some misclassifications, while a larger value of C creates a narrow margin that reduces misclassifications but can lead to overfitting.

cv_scores refer to cross-validation scores, which are used to evaluate the performance of the machine learning model. The cv_scores represent the average score of the model across all the splits.

Logistic Regression

```
In [51]: # using cross validation to find the optimal hyperparameter
         scores = []
         hyperparams = []

         for C in [1,5,50,100]:
             for solver in ['lbfgs', 'liblinear', 'newton-cholesky']:
                 lr_clf = LogisticRegression(C = C, random_state=0, solver=solver, max_iter = 5000)
                 cv_scores = cross_val_score(lr_clf, heart_X_train, heart_y_train, cv=5)
                 scores.append(cv_scores.mean())
                 hyperparams.append((C, solver))
```

```
In [52]: # Show the best hyperparameter
         best_hyperparam = hyperparams[np.argmax(scores)]
         print("Optimal C : " + str(best_hyperparam[0]))
         print("Optimal solver : " + str(best_hyperparam[1]))

         Optimal C : 1
         Optimal solver : newton-cholesky
```

```
In [53]: # Creating Logistic Regression model
         lr_clf = LogisticRegression(random_state=0, solver="newton-cholesky", C = 1)

         # Fitting the model to training data
         lr_clf.fit(heart_X_train, heart_y_train)

Out[53]: LogisticRegression
         LogisticRegression(C=1, random_state=0, solver='newton-cholesky')
```

Figure 130: Logistic Regression output

Figure 130 illustrates the output of logistic regression.

Evaluation

Prediction

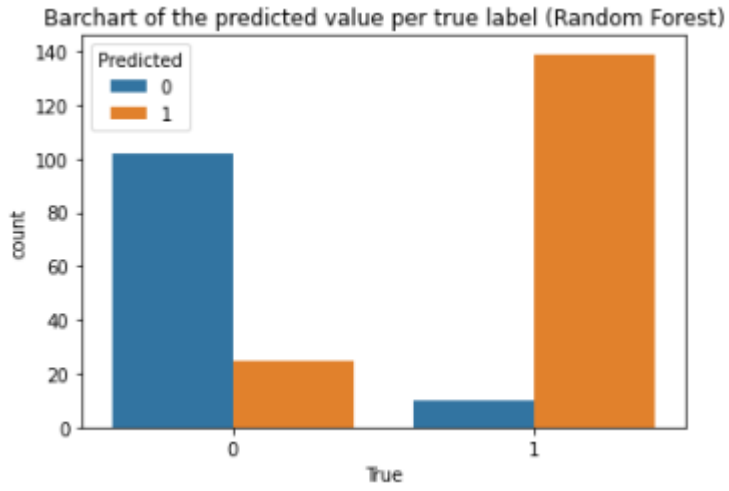


Figure 131: Bar chart of the predicted value per true label of Random Forest

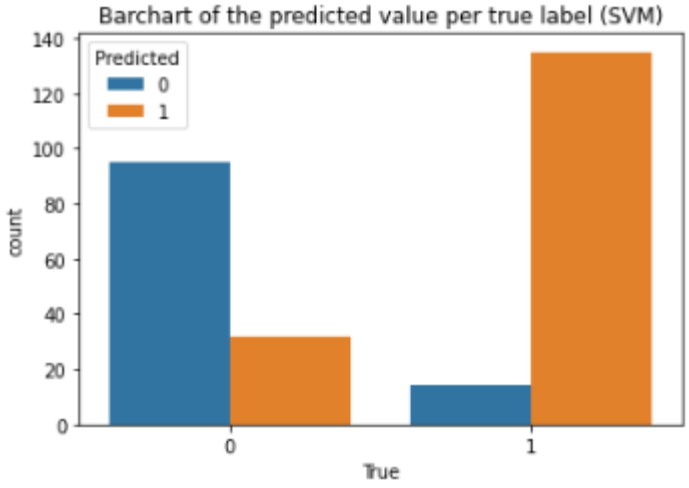


Figure 132: Barchart of the predicted value per true label for SVM

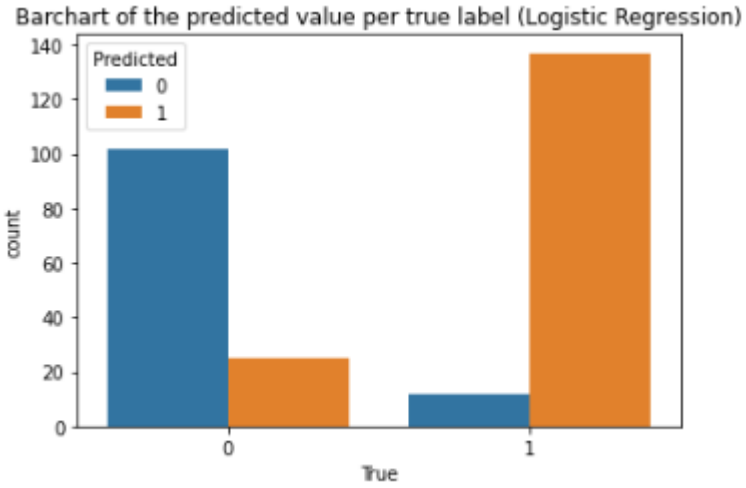


Figure 133: Barchart of the predicted value per true label for Logistic Regression

The bar charts above showcase the results/output of the Health Disease Classification Model.

Accuracy

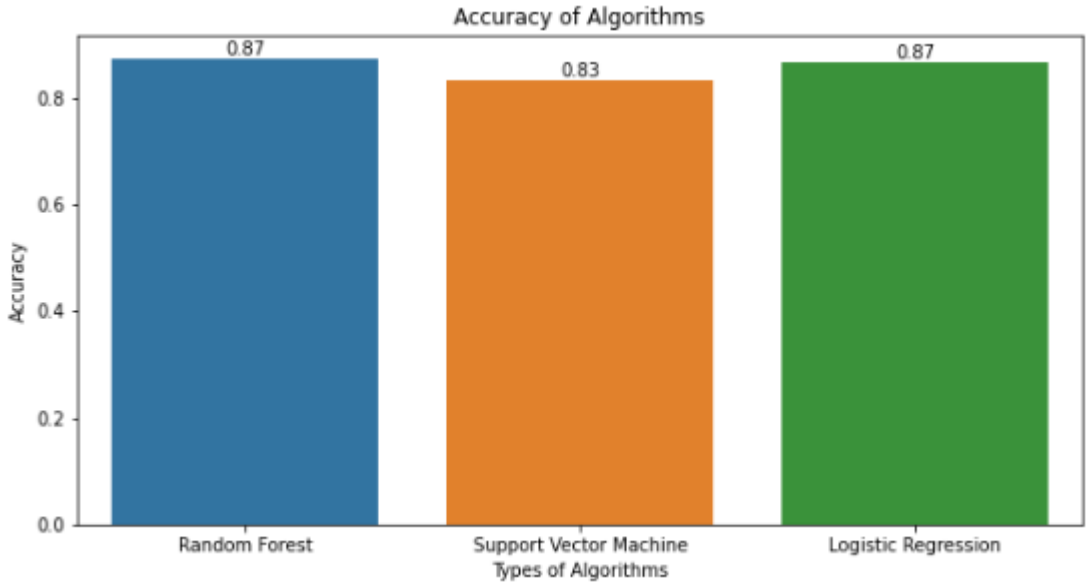


Figure 134: Barchart of the accuracy of algorithms

The accuracy is further discussed in the next section in Results and critical explanations (Accuracy).

F1-score

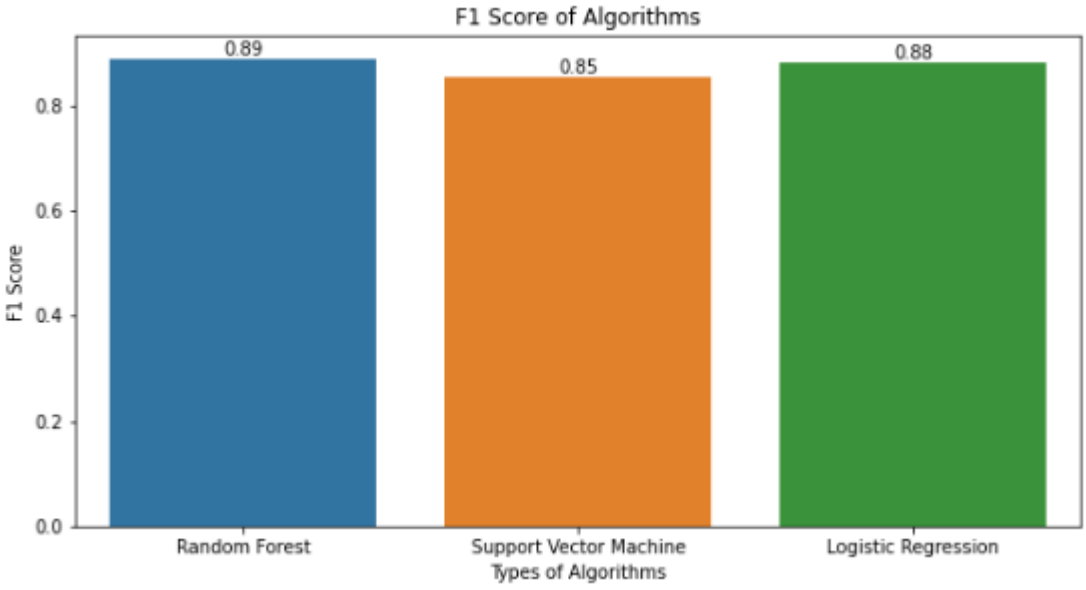


Figure 135: The F1-Score of the algorithms

The F1 Score is further discussed in the next section in Results and critical explanations (F1 Score).

Precision

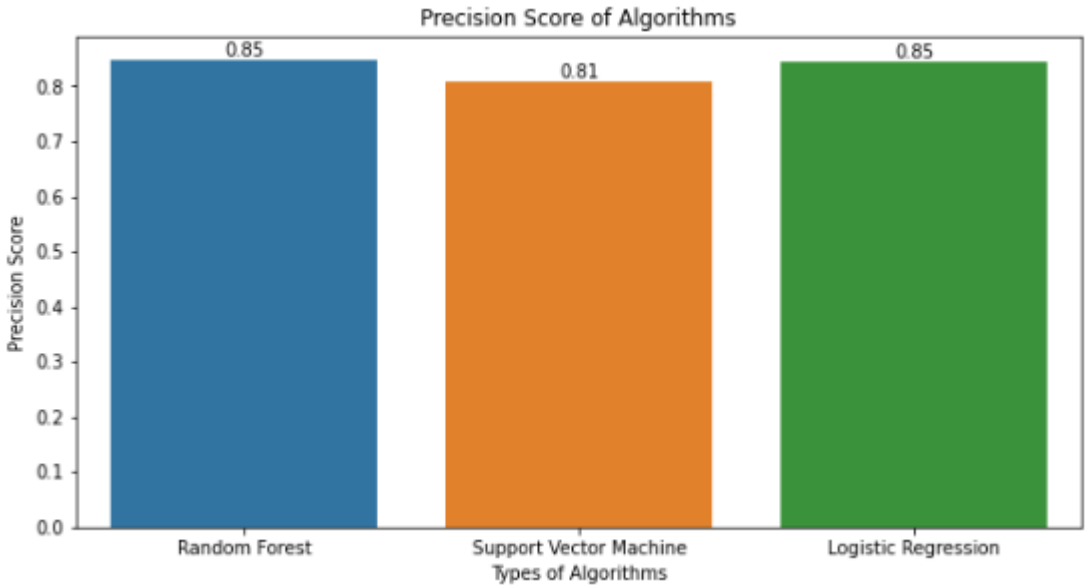


Figure 136: Precision score of algorithms

The Precision is further discussed in the next section in Results and critical explanations (Precision).

Confusion Matrix

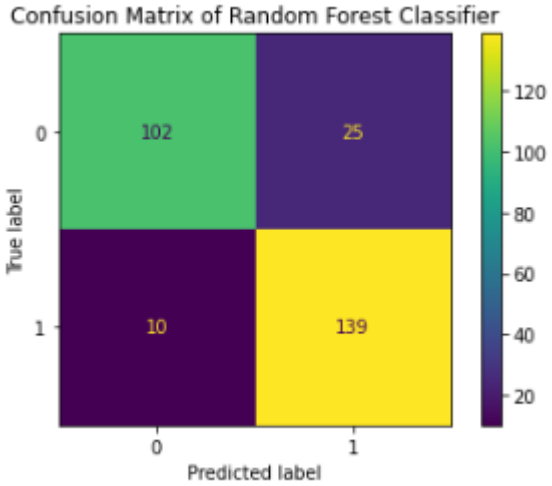


Figure 137: Confusion Matric of Randon Forest Classifier

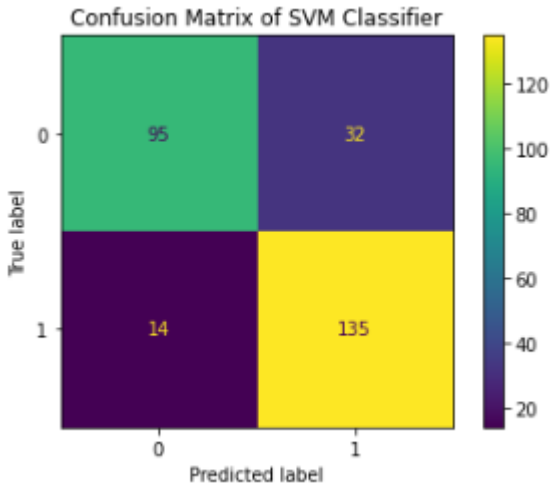


Figure 138: Confusion Matric of the SVM Classifier

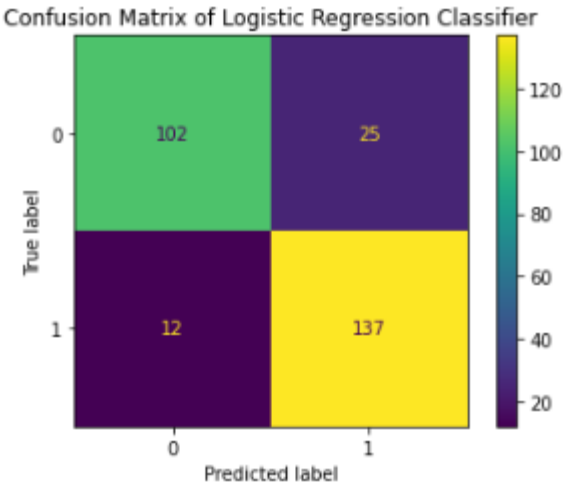


Figure 139: Confusion Matrix of Logistic Regression Classifier

The Confusion Matrix is further discussed in the next section in Results and critical explanations (Confusion Matrix).

Feature Weightage/Importance

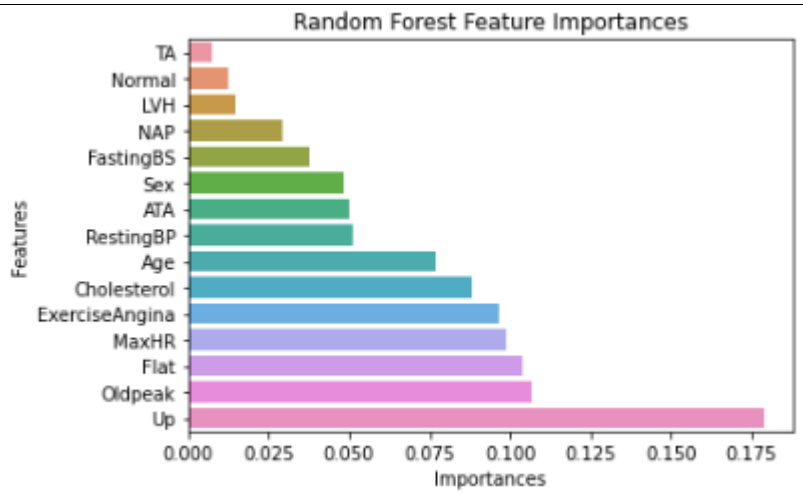


Figure 140: Feature Importances of Random Forest

The feature importance/weightage is further discussed in the next section in Results and critical explanations (Feature Imporance/Weightage).

Justification on finding

Results and critical explanations

Based on the predictions on 3 different classification models, it shows that each models displayed different accuracy, f1-score, precision, weightage, and confusion matrix.

Accuracy

According to figure 88, the accuracy for Random Forest and Logistic Regression for Heart Disease Classification are 87%, Support Vector Machine (SVM) has an accuracy of 83%. The accuracy is a commonly used metric in evaluating the performance of machine learning models (Iguazio Ltd., 2022). It measures the proportion of correct predictions made by the model to the total number of predictions. In a classification problem, accuracy is the ratio of correctly classified samples to the total number of samples (Iguazio Ltd., 2022). It is a quick and simple way to measure the performance of a model and is often used as the primary metric when the target variable is balanced, meaning the number of samples for each class is roughly the same (Iguazio Ltd., 2022). However, when the target variable is imbalanced, accuracy alone may not provide an accurate estimate of the model's performance, and other metrics such as precision, recall, and F1-score may be more appropriate (Iguazio Ltd., 2022).

F1 Score

In figure 89, the F1-score for Random Forest classification is 89% which is the best in between the other 2 classification models, followed by Logistic Regression getting 88% and SVM getting 85%. The F1-score is a commonly used performance metric in evaluating the performance of machine learning models, especially in classification problems. It is a balanced metric that takes into account both precision and recall, which are two important metrics in evaluating the performance of a model (Kanstrén, 2022). Precision is a measure of how many of the positive predictions made by the model are actually correct, while recall is a measure of how many of the actual positive cases were correctly predicted by the model (Kundu, 2023). The F1-score is the harmonic mean of precision and recall, and it provides a single, concise metric to evaluate the overall performance of a model (Kanstrén, 2022). It ranges from 0 to 1, where 1 represents perfect precision and recall, and 0 represents the worst possible performance (Kanstrén, 2022). The F1-score is particularly useful when the target variable is imbalanced, meaning that there are fewer samples for one class compared to the other (Kundu, 2023). In such cases, accuracy alone may not provide an accurate estimate of the model's performance, and the F1-score can provide a better overall assessment of the model's performance (Kanstrén, 2022).

Precision

According to figure 90, the precision for Random Forest and Logistic Regression are 85%, followed by SVM has a precision of 81%. Precision is used in evaluating the performance of machine learning models, especially in classification problems (C3.ai, 2023). It is a measure of how many of the positive predictions made by the model are actually correct. Precision is calculated as the number of true positive predictions divided by the sum of true positive and false positive predictions (C3.ai, 2023). Precision is particularly important in applications where false positive predictions are costly or have serious consequences. For example, in a medical diagnosis application, a false positive prediction could lead to unnecessary tests and treatments, while a false negative prediction could lead to delayed or missed diagnoses. In such cases, it is important to have a model that has high precision, even if it has lower recall, as the goal is to minimize false positive predictions.

Confusion Matrix

Figure 91-93 shows the confusion matrix of 3 different classification models. The results show that when SVM has the lowest value on the diagonal, it performs the poorest. When predicting healthy individuals against those with heart disease, both Logistic Regression and Random Forest make the same number of errors (25) as those methods. Compared to Random Forest, Logistic Regression makes a tiny bit more errors in classifying heart disease patients as normal. The confusion matrix is important because it provides a comprehensive view of the model's performance. By examining the confusion matrix, one can see how many cases the model got correct and how many cases it got wrong (JavaTpoint, 2023). Additionally, the confusion matrix provides insights into the types of errors made by the model, such as false positive predictions and false negative predictions, which can be used to guide further improvements in the model (JavaTpoint, 2023). The confusion matrix for Health Disease Classification model is considered good as the false negative is less than false positive in terms of error mistakes, this also indicates that false negative will cause more impact than false positive. Thus, having minimal false negative in the confusion matrix is fine for the Health Disease Classification model.

Feature Importance/Weightage

In figure 94, Up feature shows the highest weightage, it means that this feature is highly important in making accurate predictions. On the other hand, TA shows the lowest weightage, it might indicate that the feature is not important and can be removed from the model to simplify it. Another example is that Up feature is the top 1 while Flat feature is in top 3, this shows that the difference was impacted from figure 84 in the bar plot which separate the data by class. Weightage, also known as feature importance, refers to the contribution of each feature in a machine learning model to the model's overall accuracy (Brownlee, 2020a). Weightage is important during the evaluation step because it provides insight into which features are the most important for making accurate predictions (Brownlee, 2020a). This information can be used to improve the model's performance by focusing on the most important features, reducing the number of features used in the model, or even removing features that have little to no impact on the model's accuracy (Brownlee, 2020a).

Performance of the Classification Model

The predictions of 3 different classification models shows that Support Vector Machine (SVM) has the lowest performance compared to Random Forest and Logistic Regression. SVM models are known to be more sensitive to the choice of hyperparameters than other classification models, such as random forest and logistic regression (Mantovani et al., 2019). In the implementation of the model used for this project, the hyperparameters may not have been optimized to their best values and may fine tune for the optimized results in the future. Next, SVM models are particularly sensitive to the scaling of the features (Tokuç, 2022). It is possible the current SVM model did not use feature scaling causing poor performance, because feature scaling is a process that involves transforming the values of features in a dataset to a common range (Tokuç, 2022). Finally, it is possible that the data is not perfectly balanced for the SVM model (Cervantes et al., 2020). This is because SVMs are designed to maximize the margin between two classes, which may result in a bias towards the majority class (Cervantes et al., 2020).

Additionally, there are several reasons why the Heart Disease Classification model may not have a 100% F1-score or accuracy. One reason for the model's performance may be the limited search space during hyperparameter tuning due to the limited computing power. Hyperparameters can significantly impact the performance of the model, and it is important to tune them to achieve the best possible performance. If the search space for hyperparameters is too limited, the model may not be able to find the optimal set of hyperparameters, which can lead to lower performance. In the future, a larger search space with more combinations of hyperparameters values could be perform. Another reason for the model's performance may be the limited number of features used. Adding more features can increase the predictive power of the model and help it capture more complex relationships in the data. It is important to select relevant features that have a significant impact on the target variable to avoid overfitting.

References

Baheti, P. (2023, February 2). *What is Overfitting in Deep Learning [+10 Ways to Avoid It]*. V7. https://www.v7labs.com/blog/overfitting

Brownlee, J. (2019, December 13). *Tune Hyperparameters for Classification Machine Learning Algorithms*. Machine Learning Mastery. https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/

Brownlee, J. (2020a, March 30). *How to Calculate Feature Importance With Python*. Machine Learning Mastery. https://machinelearningmastery.com/calculate-feature-importance-with-python/

Brownlee, J. (2020b, June 12). *Ordinal and One-Hot Encodings for Categorical Data*. Machine Learning Mastery. https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/

C3.ai. (2023). *Precision*. C3 AI. https://c3.ai/glossary/machine-learning/precision/

Cervantes, J., García-Lamont, F., Rodríguez-Mazahua, L., & López, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, *408*, 189–215. https://doi.org/10.1016/j.neucom.2019.10.118

Cope, G. (2023). *Dummy Variable Trap in Regression Models*. Algosome Software Design. https://www.algosome.com/articles/dummy-variable-trap-regression.html

DataMites AI Team. (2020, December 31). *Outlier Detection and Its importance in Machine learning*. DataMites Offical Blog | Resources for Data Science. https://datamites.com/blog/outlier-detection-and-its-importance-in-machine-learning/

Enterprise Big Data Framework. (2020, June 25). *The Importance of Outlier Detection in Big Data*. Enterprise Big Data Framework©. https://www.bigdataframework.org/the-importance-of-outlier-detection-in-big-data/

Iguazio Ltd. (2022, October 12). *What is Model Accuracy in Machine Learning*. Iguazio. https://www.iguazio.com/glossary/model-accuracy-in-ml/

JavaTpoint. (2023). *Confusion Matrix in Machine Learning - Javatpoint*. www.javatpoint.com. https://www.javatpoint.com/confusion-matrix-in-machine-learning

Kanstrén, T. (2022, March 30). *A Look at Precision, Recall, and F1-Score - Towards Data Science*. Medium. https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec

Kundu, R. (2023, February 2). *F1 Score in Machine Learning: Intro & Calculation*. V7. https://www.v7labs.com/blog/f1-score-guide

Mantovani, R. G., Rossi, A. L., Alcobaça, E., Vanschoren, J., & De Carvalho, A. C. (2019). A meta-learning recommender system for hyperparameter tuning: Predicting when tuning improves SVM classifiers. *Information Sciences*, *501*, 193–221. https://doi.org/10.1016/j.ins.2019.06.005

Shulga, D. (2018, September 27). *5 Reasons why you should use Cross-Validation in your Data Science Projects*. Medium. https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79

Sonal. (2021, September 20). *Importance Of Exploratory Data Analysis Before ML Modelling*. Eduonix Blog. https://blog.eduonix.com/bigdata-and-hadoop/importance-exploratory-data-analysis-ml-modelling/

Tokuç, A. (2022, November 10). *Why Feature Scaling in SVM?* Baeldung. https://www.baeldung.com/cs/svm-feature-scaling

Appendix: Python Code

Heart Disease Classification

```
# Import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
```

Data Reading

```
# Read data from the downloaded csv file
heart = pd.read_csv("heart.csv")
```

1. Exploratory Data Analysis

1.1 Data Inspection

```
# View the top records of the data
heart.head()
```

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG
MaxHR  \
0   40   M           ATA        140          289          0     Normal
172
1   49   F           NAP        160          180          0     Normal
156
2   37   M           ATA        130          283          0         ST
98
3   48   F           ASY        138          214          0     Normal
108
4   54   M           NAP        150          195          0     Normal
122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

```
# View the dimensions of the data
heart.shape
```

```
(918, 12)
```

The data has 918 rows and 12 columns

```
# View the features of the data
heart.columns
```

```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol',
'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

```
# View the information of the data
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

According to description on kaggle the variables carries the following meaning

1. Age: age of the patient [years]
2. Sex: sex of the patient [M: Male, F: Female]
3. ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
4. RestingBP: resting blood pressure [mm Hg]
5. Cholesterol: serum cholesterol [mm/dl]
6. FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
7. RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05

mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]

8. MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
9. ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
10. Oldpeak: oldpeak = ST [Numeric value measured in depression]
11. ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
12. HeartDisease: output class [1: heart disease, 0: Normal]

All the features are important clinical information that used in previous research works

From the description, it could be identified that there are some categorical variables that need to be encoded before modelling phase

1.2 Data Cleaning

```
# Check if there is column with null value
heart.isnull().sum()
```

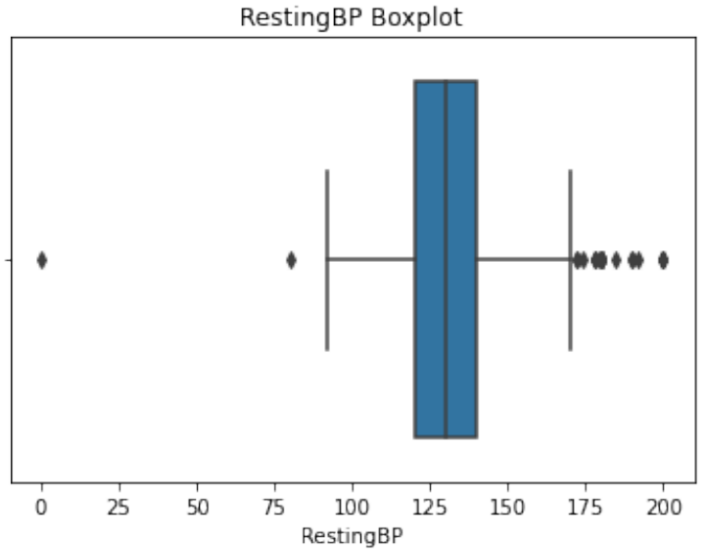
```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

There is no missing value in the dataset. The dataset is clean

1.3 Outlier Detection

```
# Create a box plot for Resting BP
sns.boxplot(data = heart, x="RestingBP")
plt.title('RestingBP Boxplot')
```

```
Text(0.5, 1.0, 'RestingBP Boxplot')
```



From the boxplot above, outlier is detected. The variable should be further inspected

```
# View statistical properties of Resting BP
heart["RestingBP"].describe()
```

```
count    918.000000
mean     132.396514
std       18.514154
min        0.000000
25%      120.000000
50%      130.000000
75%      140.000000
max      200.000000
Name: RestingBP, dtype: float64
```

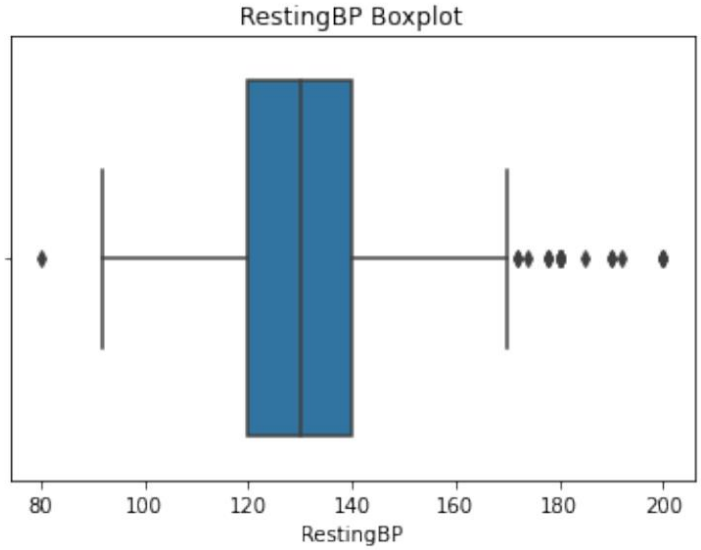
The min value for the Resting Blood Pressure is 0. This should be impossible. Observations with 0 RestingBP is inspected.

```
# Filter record with RestingBP equals to 0
heart[heart["RestingBP"] == 0]
```

```
     Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS
RestingECG  \
449   55   M           NAP          0            0          0
Normal

     MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
449    155              N      1.5     Flat             1
```

There is only one observation. This observation will be removed

```
# Remove the record that has 0 Resting BP
heart = heart[heart["RestingBP"] != 0]

# Create a box plot for Resting BP
sns.boxplot(data = heart, x="RestingBP")
plt.title('RestingBP Boxplot')

Text(0.5, 1.0, 'RestingBP Boxplot')
```



The distribution changed significantly. This has justified the removal. The reason for that outlier presence could be measurement error or other systematic error.

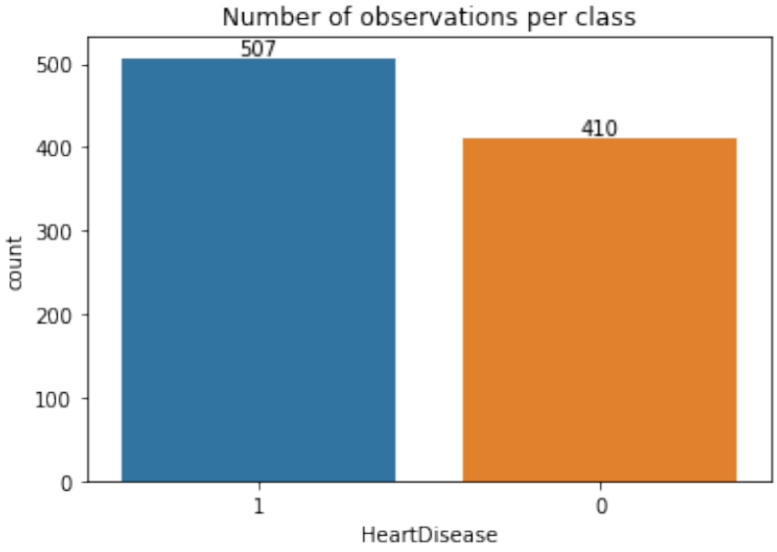
1.4 Data Understanding

```
# View number of records for each label
print(heart["HeartDisease"].value_counts())

# Compute the percentage of records for each label
perc_of_classes = heart["HeartDisease"].value_counts().values /
len(heart) * 100
print("Percentage of class with heart diseases : {:.2f}
%".format(perc_of_classes[0]))
print("Percentage of class normal : {:.2f}
%".format(perc_of_classes[1]))

1    507
0    410
```

```
Name: HeartDisease, dtype: int64
Percentage of class with heart diseases : 55.29 %
Percentage of class normal : 44.71 %
```

```
# Generate a bar plot showing No of records per label
target_counts = heart['HeartDisease'].value_counts().values

# Create the labels for the bars
bc = sns.countplot(x="HeartDisease", data=heart,
order=heart['HeartDisease'].value_counts().index)
bc.bar_label(container=bc.containers[0], labels=target_counts)
plt.title('Number of observations per class')
```

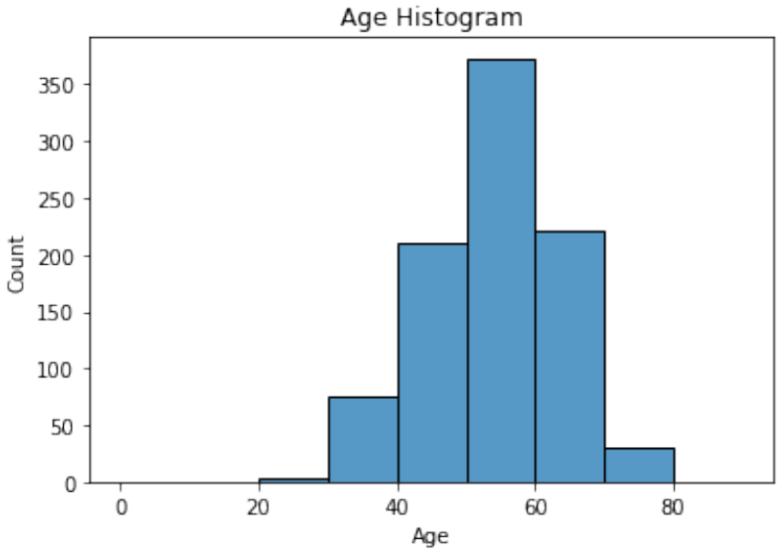
```
Text(0.5, 1.0, 'Number of observations per class')
```



The number of observations that has label 0 and 1 are almost equal (44.71 % and 55.29 %). The dataset is a balanced dataset.

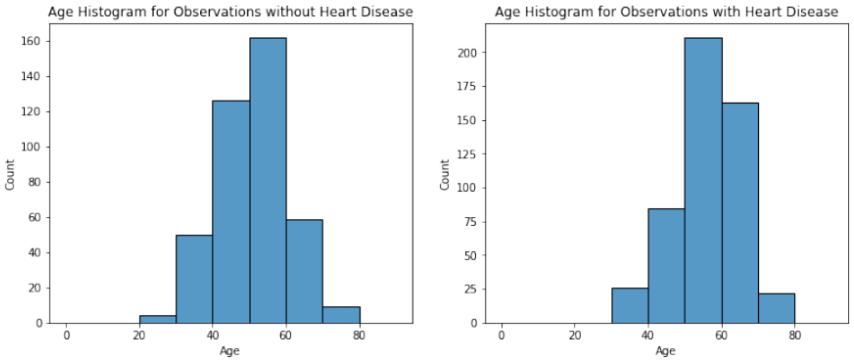
```
# Create the histogram for Age
sns.histplot(heart, x = "Age",  bins=[0, 10, 20, 30, 40, 50, 60, 70,
80, 90])
plt.title('Age Histogram')

# Compute the mean age
print("Average age : {:.2f}".format(heart["Age"].mean()))
```

```
Average age : 53.51
```




```
#  Create the histogram of age for records without heart disease
plt.figure(figsize=(13,5))
plt.subplot(1, 2, 1)
sns.histplot(heart[heart['HeartDisease'] == 0], x="Age", bins=[0, 10,
20, 30, 40,
                                                               50, 60,
70, 80, 90])

plt.title('Age Histogram for Observations without Heart Disease')

#  Create the histogram of age for records with heart disease
plt.subplot(1, 2, 2)
sns.histplot(heart[heart['HeartDisease'] == 1], x="Age", bins=[0, 10,
20, 30, 40,
                                                               50, 60,
70, 80, 90])
plt.title('Age Histogram for Observations with Heart Disease')

# Compute the mean of age for records without and with heart disease
print("Average age for observations without heart disease:
{:.2f}".format(
    heart[heart['HeartDisease'] == 0]["Age"].mean()))
print("Average age for observations with heart disease:
{:.2f}".format(
    heart[heart['HeartDisease'] == 1]["Age"].mean()))
```

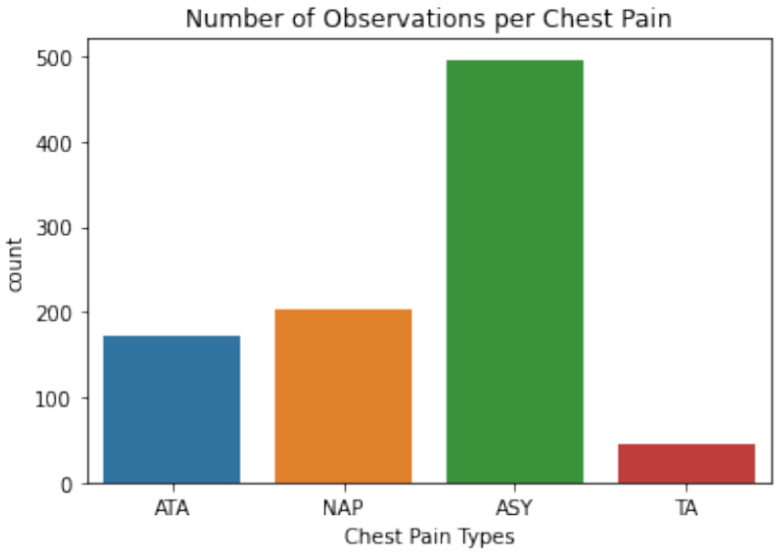
```
Average age for observations without heart disease: 50.55
Average age for observations with heart disease: 55.90
```



From the histograms, there is not a clear pattern that could be observed. But at least it could be observed that for a person between 50-60 years old, the proportion of getting a heart disease and not getting is about equal. It could be observed that 60-70 years old have higher risk for getting heart diseases. The average age of people who have heart disease is slightly older than those without.

```
# Generate a bar plot showing No of records per chest pain
bc = sns.countplot(x="ChestPainType", data=heart)
plt.xlabel("Chest Pain Types")
plt.title('Number of Observations per Chest Pain')
```

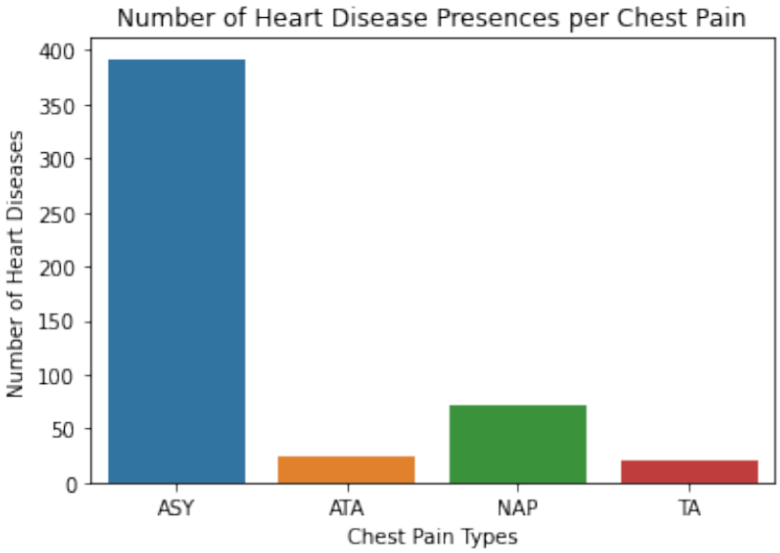
```
Text(0.5, 1.0, 'Number of Observations per Chest Pain')
```



Asymptomatic (ASY) chest pain is the most common type of chest pain

```
# Sum the value group by ChestPainType, this allow the total Heart
Disease (with label 1)
# to be computed
heart_gb_cp_sum = heart.groupby("ChestPainType").sum(numeric_only =
True)

# Generate a bar plot showing No of Heart Diseases per chest pain
sns.barplot(x=heart_gb_cp_sum.index, y = "HeartDisease",
data=heart_gb_cp_sum)
plt.xlabel("Chest Pain Types")
plt.ylabel("Number of Heart Diseases")
plt.title('Number of Heart Disease Presences per Chest Pain')
```

```
Text(0.5, 1.0, 'Number of Heart Disease Presences per Chest Pain')
```

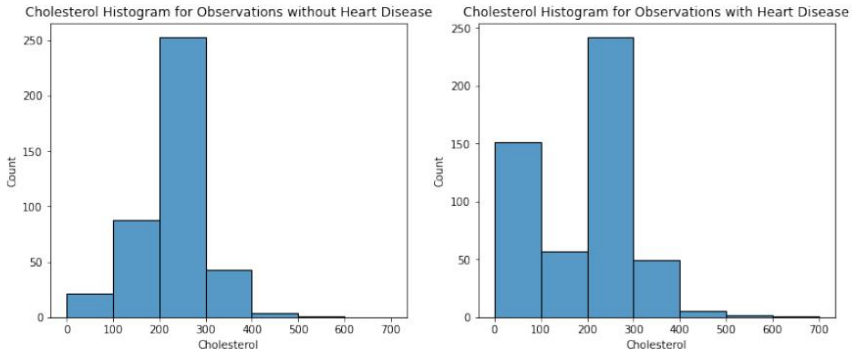


Asymptomatic (ASY) chest pain is the most common chest pain for people with heart disease, the proportion is significantly larger than other types of chest pain

```
#  Create the histogram of cholesterol for records without heart
disease
plt.figure(figsize=(13,5))
plt.subplot(1, 2, 1)
sns.histplot(heart[heart['HeartDisease'] == 0], x="Cholesterol",
bins=[0, 100, 200,

300, 400, 500, 600, 700])
```

```
plt.title('Cholesterol Histogram for Observations without Heart
Disease')

#  Create the histogram of cholesterol for records with heart disease
plt.subplot(1, 2, 2)
sns.histplot(heart[heart['HeartDisease'] == 1], x="Cholesterol",
bins=[0, 100, 200,

300, 400, 500, 600, 700])
plt.title('Cholesterol Histogram for Observations with Heart Disease')

#  Compute the mean of cholesterol for records without and with heart
disease
print("Average Cholesterol for observations without heart disease:
{:.2f}".format(
    heart[heart['HeartDisease'] == 0]["Cholesterol"].mean()))
print("Average Cholesterol for observations with heart disease:
{:.2f}".format(
    heart[heart['HeartDisease'] == 1]["Cholesterol"].mean()))
```

```
Average Cholesterol for observations without heart disease: 227.12
Average Cholesterol for observations with heart disease: 176.29
```

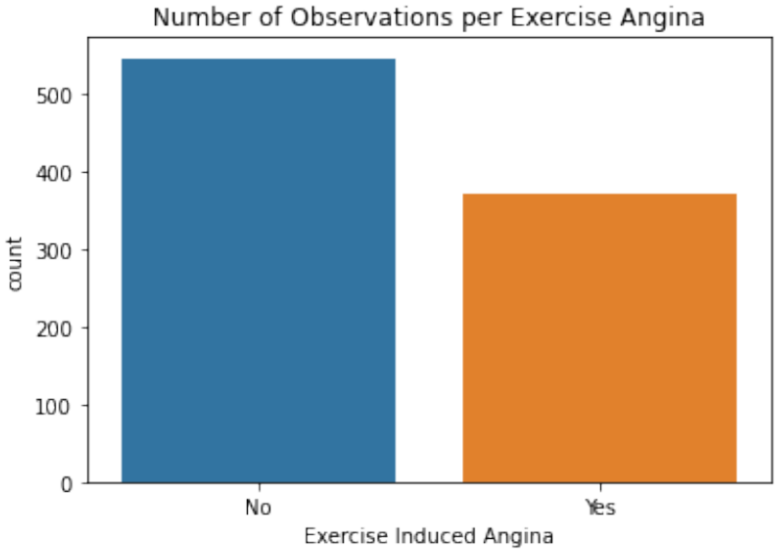


People with heart disease has a mean serum cholesterol lower than normal people.

From the histogram, it could be observed that people with too low serum cholesterol (0-100 mm Hg), have a higher risk to have heart diseases.

```
# Create bar plot showing no of observations per Exercise Angina
ax = sns.countplot(x="ExerciseAngina", data=heart)
plt.xlabel("Exercise Induced Angina")
plt.title('Number of Observations per Exercise Angina')
ax.set_xticklabels(['No', 'Yes'])
```

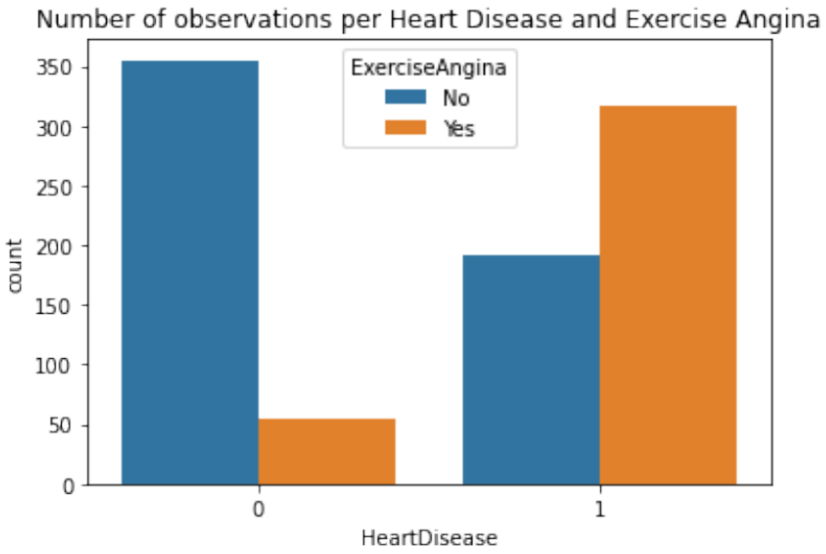
```
[Text(0, 0, 'No'), Text(1, 0, 'Yes')]
```



The number observations which has no exercise induced angine (a chest pain induced by exercise) is higher

```
# Create bar plot showing no of observations per Exercise Angina and
Heart Disease
ax = sns.countplot(x="HeartDisease", hue="ExerciseAngina" ,
data=heart)
plt.title('Number of observations per Heart Disease and Exercise
Angina')
ax.legend(title='ExerciseAngina', labels=['No', 'Yes'])
```

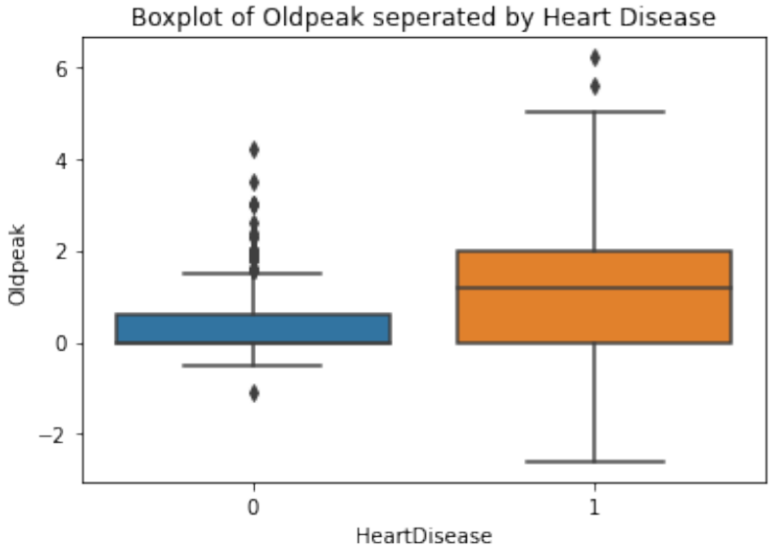
```
<matplotlib.legend.Legend at 0x1b7569a6790>
```



People without heart disease mostly does not have exercise induced angina. While among people who have heart disease, the number of people who have exercise induced angina is higher

```
# Create boxplot of Oldpeak seperated by Heart Disease
sns.boxplot(x ='HeartDisease', y = "Oldpeak" , data =heart)
plt.title('Boxplot of Oldpeak seperated by Heart Disease')

#  Compute the mean of Oldpeak for records without and with heart
disease
print("Average Oldpeak for observations without heart disease:
{:.2f}".format(
    heart[heart['HeartDisease'] == 0]["Oldpeak"].mean()))
print("Average Oldpeak for observations with heart disease:
{:.2f}".format(
    heart[heart['HeartDisease'] == 1]["Oldpeak"].mean()))
```

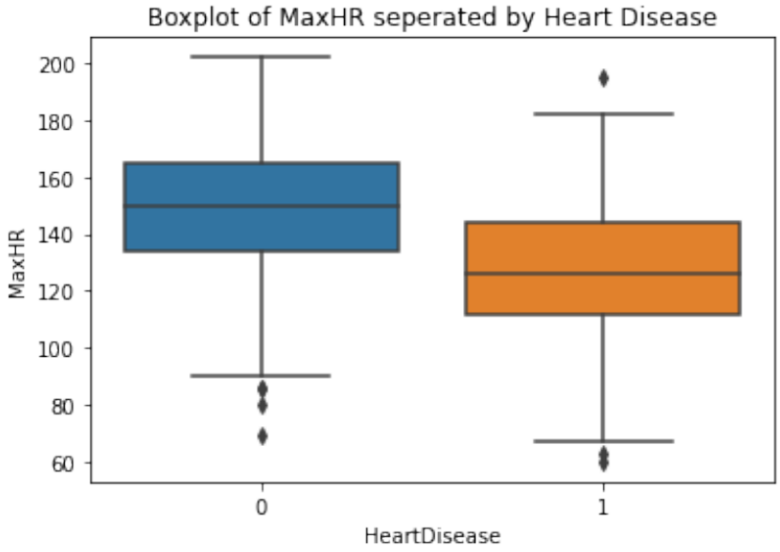
```
Average Oldpeak for observations without heart disease: 0.41
Average Oldpeak for observations with heart disease: 1.27
```



The average oldpeak for person without heart disease is lower

```
# Create boxplot of MaxHR seperated by Heart Disease
sns.boxplot(x ='HeartDisease', y = "MaxHR" , data =heart)
plt.title('Boxplot of MaxHR seperated by Heart Disease')

#  Compute the mean of MaxHR for records without and with heart
disease
print("Average MaxHR for observations without heart disease:
{:.2f}".format(
    heart[heart['HeartDisease'] == 0]["MaxHR"].mean()))
print("Average MaxHR for observations with heart disease:
{:.2f}".format(
    heart[heart['HeartDisease'] == 1]["MaxHR"].mean()))
```

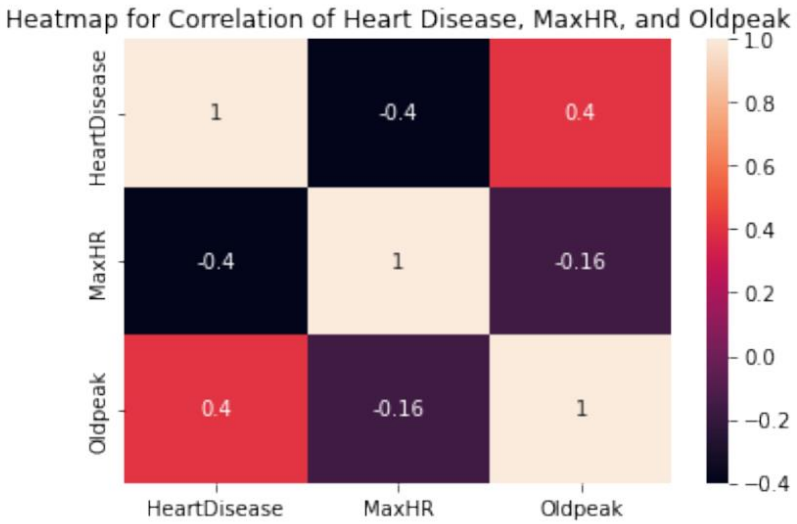
```
Average MaxHR for observations without heart disease: 148.15
Average MaxHR for observations with heart disease: 127.60
```



The average maximum heart rate achieved for person without heart disease is higher

```
# Create heatmap showing correlation of MaxHR, Oldpeak and Heart
Disease
sns.heatmap(heart[["HeartDisease","MaxHR","Oldpeak"]].corr(),
annot=True)
plt.title('Heatmap for Correlation of Heart Disease, MaxHR, and
Oldpeak')
```

```
Text(0.5, 1.0, 'Heatmap for Correlation of Heart Disease, MaxHR, and
Oldpeak')
```



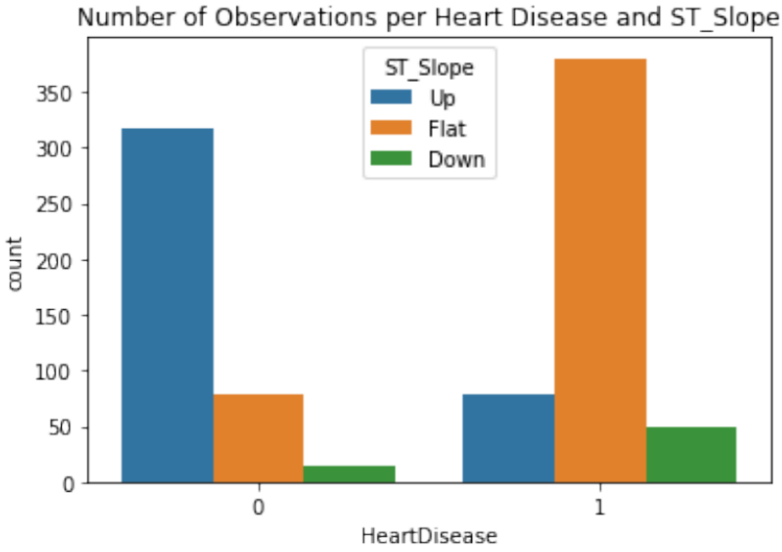
The heatmap above very good summarise the information conveyed by the 2 boxplots above.

Both Oldpeak and MaxHR (maximum heart rate achieved) shows a moderate correlation (0.4) with heart disease

While the MaxHR is negative correlation and Oldpeak is positive correlation

```
# Create bar plot showing no of observations per ST_Slope and Heart
Disease
ax = sns.countplot(x="HeartDisease", hue="ST_Slope" , data=heart)
plt.title('Number of Observations per Heart Disease and ST_Slope')
```

```
Text(0.5, 1.0, 'Number of Observations per Heart Disease and
ST_Slope')
```



It could be observed that for normal people, most of them have Up ST_Slope

While for people with heart disease, most of them have Flat ST_Slope

2. Data Preparation

2.1 Column Transformation

```
# View information of the data frame
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 917 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             917 non-null    int64  
 1   Sex             917 non-null    object 
 2   ChestPainType   917 non-null    object 
 3   RestingBP       917 non-null    int64  
 4   Cholesterol     917 non-null    int64  
 5   FastingBS       917 non-null    int64  
 6   RestingECG      917 non-null    object 
 7   MaxHR           917 non-null    int64  
 8   ExerciseAngina  917 non-null    object 
 9   Oldpeak         917 non-null    float64
 10  ST_Slope        917 non-null    object 
 11  HeartDisease    917 non-null    int64  
```

```
dtypes: float64(1), int64(6), object(5)
memory usage: 125.4+ KB
```

Some variable need to be encoded before the modelling phase

```
# view top records
heart.head()
```

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG
MaxHR  \
0   40   M           ATA        140          289          0     Normal
172
1   49   F           NAP        160          180          0     Normal
156
2   37   M           ATA        130          283          0         ST
98
3   48   F           ASY        138          214          0     Normal
108
4   54   M           NAP        150          195          0     Normal
122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

```
# View possible values and its counts of sex
heart['Sex'].value_counts()
```

```
M    724
F    193
Name: Sex, dtype: int64
```

```
# View possible values and its counts of ExerciseAngina
heart['ExerciseAngina'].value_counts()
```

```
N    546
Y    371
Name: ExerciseAngina, dtype: int64
```

It has been confirmed that Sex and ExerciseAngina has 2 possible values only. For these kind of variables, LabelEncoder will be used

```
# Create label encoder
le = preprocessing.LabelEncoder()

# Encode sex and exerciseangina columns' values
heart['Sex']= le.fit_transform(heart['Sex'])
heart['ExerciseAngina']= le.fit_transform(heart['ExerciseAngina'])
```

```
# Check encoding result
heart.head()
```

```
   Age  Sex ChestPainType  RestingBP  Cholesterol  FastingBS
RestingECG  \
0   40    1           ATA        140          289          0
Normal
1   49    0           NAP        160          180          0
Normal
2   37    1           ATA        130          283          0
ST
3   48    0           ASY        138          214          0
Normal
4   54    1           NAP        150          195          0
Normal

   MaxHR  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0    172               0      0.0       Up             0
1    156               0      1.0     Flat             1
2     98               0      0.0       Up             0
3    108               1      1.5     Flat             1
4    122               0      0.0       Up             0
```

```
# Confirm encoding result
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 917 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             917 non-null    int64
 1   Sex             917 non-null    int32
 2   ChestPainType   917 non-null    object
 3   RestingBP       917 non-null    int64
 4   Cholesterol     917 non-null    int64
 5   FastingBS       917 non-null    int64
 6   RestingECG      917 non-null    object
 7   MaxHR           917 non-null    int64
 8   ExerciseAngina  917 non-null    int32
 9   Oldpeak         917 non-null    float64
 10  ST_Slope        917 non-null    object
 11  HeartDisease    917 non-null    int64
dtypes: float64(1), int32(2), int64(6), object(3)
memory usage: 118.3+ KB
```

The transformation has been done successfully.

```
# View possible values and its counts of ChestPainType
heart['ChestPainType'].value_counts()
```

```
ASY    496
NAP    202
ATA    173
TA      46
Name: ChestPainType, dtype: int64
```

```
# View possible values and its counts of RestingECG
heart['RestingECG'].value_counts()
```

```
Normal    551
LVH       188
ST        178
Name: RestingECG, dtype: int64
```

```
# View possible values and its counts of ST_Slope
heart['ST_Slope'].value_counts()
```

```
Flat    459
Up      395
Down     63
Name: ST_Slope, dtype: int64
```

It has been confimed that ChestPainType, RestingECG, and ST_Slope has 3 possible values. For categorical variables with more than 2 possible value. One hot encoding will be used. One hot encoding is preferred over Label Encoder as the latter will impose an order to the values which does not exist in real world domain.
https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/

```
# Create Label Binarizer to implement one hot encoding
lb = preprocessing.LabelBinarizer()

# Encode ChestPainType column with Label Binarizer
heart =
heart.join(pd.DataFrame(lb.fit_transform(heart["ChestPainType"]),
                        columns=lb.classes_,
                        index=heart.index))

# Encode RestingECG column with Label Binarizer
heart = heart.join(pd.DataFrame(lb.fit_transform(heart["RestingECG"]),
                        columns=lb.classes_,
                        index=heart.index))

# Encode STSlope column with Label Binarizer
heart = heart.join(pd.DataFrame(lb.fit_transform(heart["ST_Slope"]),
                        columns=lb.classes_,
                        index=heart.index))

# Confirm encoding result
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 917 entries, 0 to 917
Data columns (total 22 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             917 non-null    int64
 1   Sex             917 non-null    int32
 2   ChestPainType   917 non-null    object
 3   RestingBP       917 non-null    int64
 4   Cholesterol     917 non-null    int64
 5   FastingBS       917 non-null    int64
 6   RestingECG      917 non-null    object
 7   MaxHR           917 non-null    int64
 8   ExerciseAngina  917 non-null    int32
 9   Oldpeak         917 non-null    float64
 10  ST_Slope        917 non-null    object
 11  HeartDisease    917 non-null    int64
 12  ASY             917 non-null    int32
 13  ATA             917 non-null    int32
 14  NAP             917 non-null    int32
 15  TA              917 non-null    int32
 16  LVH             917 non-null    int32
 17  Normal          917 non-null    int32
 18  ST              917 non-null    int32
 19  Down            917 non-null    int32
 20  Flat            917 non-null    int32
 21  Up              917 non-null    int32
dtypes: float64(1), int32(12), int64(6), object(3)
memory usage: 154.1+ KB
```

The transformation is successful. The original columns could be dropped.

```
# Drop the original columns that has been encode by LabelBinarizer
heart = heart.drop(columns=["ChestPainType", "RestingECG","ST_Slope"])

# Confirm dropping result
heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 917 entries, 0 to 917
Data columns (total 19 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             917 non-null    int64
 1   Sex             917 non-null    int32
 2   RestingBP       917 non-null    int64
 3   Cholesterol     917 non-null    int64
 4   FastingBS       917 non-null    int64
 5   MaxHR           917 non-null    int64
 6   ExerciseAngina  917 non-null    int32
 7   Oldpeak         917 non-null    float64
```

```
 8   HeartDisease    917 non-null    int64
 9   ASY             917 non-null    int32
 10  ATA             917 non-null    int32
 11  NAP             917 non-null    int32
 12  TA              917 non-null    int32
 13  LVH             917 non-null    int32
 14  Normal          917 non-null    int32
 15  ST              917 non-null    int32
 16  Down            917 non-null    int32
 17  Flat            917 non-null    int32
 18  Up              917 non-null    int32
dtypes: float64(1), int32(12), int64(6)
memory usage: 132.6 KB
```

All the columns now could be accepted by the model

Some of the columns created by One Hot Encoding need to be dropped. For each group of columns created with same feature, one of them should be dropped. This is because they carry redundant information. For example, if the observation has 0 on ATA, NAP and TA. It signify it must have a 1 in ASY. https://www.geeksforgeeks.org/ml-dummy-variable-trap-in-regression-models/

```
# Drop the columns with redundant information
# ASY ATA NAP TA - one of them should be dropped
# LVH Normal ST - one of them should be dropped
# Down Flat Up - one of them should be dropped

heart = heart.drop(columns=["ASY","ST","Down"])
```

2.2 Train Test Split

```
# Split the predictors and target variable
heart_X = heart.iloc[:, heart.columns != "HeartDisease"]
heart_y = heart["HeartDisease"]
```

The dataframe is splitted into X (independent variable) and y (target variable)

```
# Perform 70 : 30 train test split
heart_X_train, heart_X_test, heart_y_train, heart_y_test =
train_test_split(
    heart_X, heart_y, test_size=0.30, random_state=123)

# View dimensions of traing data X
heart_X_train.shape
```

```
(641, 15)
```

The training data has 641 rows

```
# View dimensions of testing data X
heart_X_test.shape
```

```
(276, 15)
```

The testing data has 276 rows

3. Modelling

3.1 Random Forest

```
# using cross validation to find the optimal hyperparameter
scores = []
hyperparams = []

for n_estimator in [500,1000,2000]:
    for max_depth in [3,5,7]:
        for max_features in [2,3,4]:
            rf_clf = RandomForestClassifier(n_estimators=n_estimator,
max_depth=max_depth,
                                            random_state=0,
max_features = max_features)
            cv_scores = cross_val_score(rf_clf, heart_X_train,
heart_y_train, cv=5)
            scores.append(cv_scores.mean())
            hyperparams.append((n_estimator,max_depth,max_features))

# Show the best hyperparameter
best_hyperparam = hyperparams[np.argmax(scores)]
print("Optimal n_estimator : " + str(best_hyperparam[0]))
print("Optimal max_depth : " + str(best_hyperparam[1]))
print("Optimal max_features : " + str(best_hyperparam[2]))
```

```
Optimal n_estimator : 500
Optimal max_depth : 7
Optimal max_features : 2
```

```
# Creating Random Forest Model
rf_clf = RandomForestClassifier(n_estimators=500, max_depth=7,
random_state=0, max_features = 2)

# Fitting the model to training data
rf_clf.fit(heart_X_train, heart_y_train)
```

```
RandomForestClassifier(max_depth=7, max_features=2, n_estimators=500,
                       random_state=0)
```

3.2 Support Vector Machine (SVM)

```
# using cross validation to find the optimal hyperparameter
scores = []
hyperparams = []

for C in [100,200,500,1000]:
    svm_clf = SVC(C= C)
    cv_scores = cross_val_score(svm_clf, heart_X_train, heart_y_train,
```

```
cv=5)
    scores.append(cv_scores.mean())
    hyperparams.append((C))

# Show the best hyperparameter
best_hyperparam = hyperparams[np.argmax(scores)]
print("Optimal C : " + str(best_hyperparam))
```

```
Optimal C : 1000
```

```
# Creating SVM model
svm_clf = SVC(C= 1000)

# Fitting the model to training data
svm_clf.fit(heart_X_train, heart_y_train)
```

```
SVC(C=1000)
```

3.3 Logistic Regression

```
# using cross validation to find the optimal hyperparameter
scores = []
hyperparams = []

for C in [1,5,50,100]:
    for solver in ['lbfgs', 'liblinear', 'newton-cholesky']:
        lr_clf = LogisticRegression(C = C, random_state=0,
solver=solver, max_iter = 5000)
        cv_scores = cross_val_score(lr_clf, heart_X_train,
heart_y_train, cv=5)
        scores.append(cv_scores.mean())
        hyperparams.append((C, solver))

# Show the best hyperparameter
best_hyperparam = hyperparams[np.argmax(scores)]
print("Optimal C : " + str(best_hyperparam[0]))
print("Optimal solver : " + str(best_hyperparam[1]))
```

```
Optimal C : 1
Optimal solver : newton-cholesky
```

```
# Creating Logistic Regression model
lr_clf = LogisticRegression(random_state=0, solver="newton-cholesky",
C = 1)

# Fitting the model to training data
lr_clf.fit(heart_X_train, heart_y_train)
```

```
LogisticRegression(C=1, random_state=0, solver='newton-cholesky')
```

4.0 Evaluation

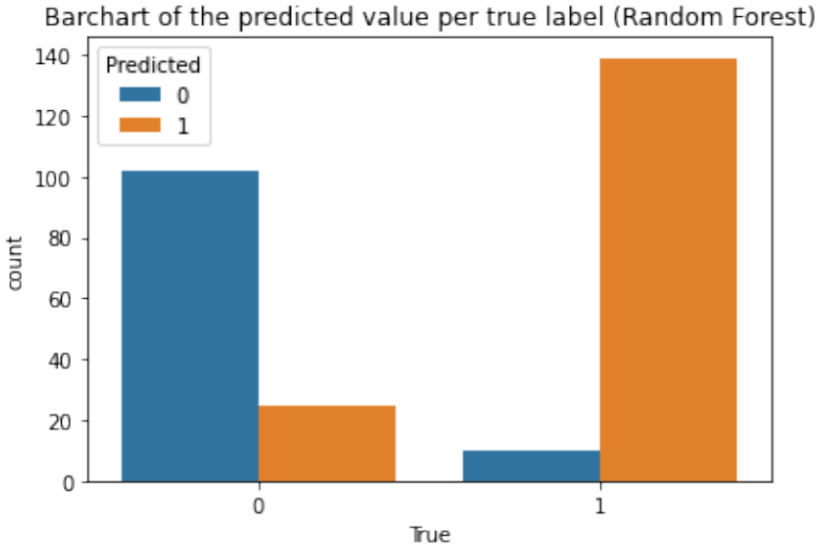
4.1 Prediction

```
# Obtain the predictions of the models
rf_pred_y = rf_clf.predict(heart_X_test)
svm_pred_y = svm_clf.predict(heart_X_test)
lr_pred_y = lr_clf.predict(heart_X_test)
```

The predictions of the models are stored in the variables

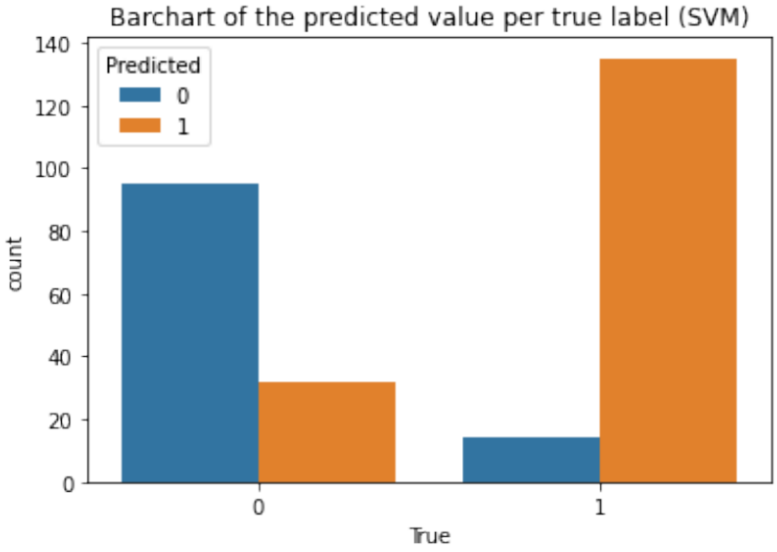
```
# Creating barchart that shows number of records per true label and
prediction
df = pd.DataFrame({"Predicted" : rf_pred_y,"True" : heart_y_test})
ax = sns.countplot(x="True", hue="Predicted", data=df)
plt.title('Barchart of the predicted value per true label (Random
Forest)')
```

```
Text(0.5, 1.0, 'Barchart of the predicted value per true label (Random
Forest)')
```



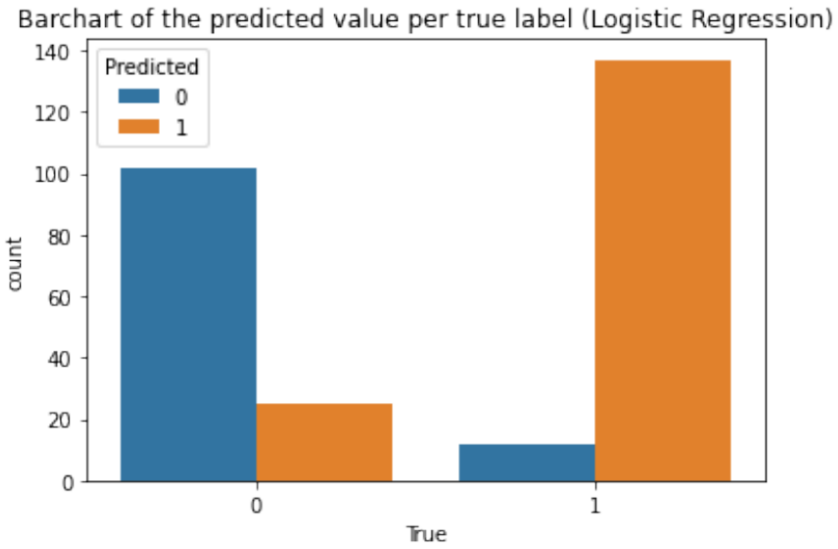

```
# Creating barchart that shows number of records per true label and
prediction
df = pd.DataFrame({"Predicted" : svm_pred_y,"True" : heart_y_test})
ax = sns.countplot(x="True", hue="Predicted", data=df)
plt.title('Barchart of the predicted value per true label (SVM)')
```

```
Text(0.5, 1.0, 'Barchart of the predicted value per true label (SVM)')
```




```
# Creating barchart that shows number of records per true label and
prediction
df = pd.DataFrame({"Predicted" : lr_pred_y,"True" : heart_y_test})
ax = sns.countplot(x="True", hue="Predicted", data=df)
plt.title('Barchart of the predicted value per true label (Logistic
Regression)')
```

```
Text(0.5, 1.0, 'Barchart of the predicted value per true label
(Logistic Regression)')
```



4.2 Accuracy

```
# Compute the accuracy score of 3 models
rf_acc = accuracy_score(heart_y_test, rf_pred_y)
svm_acc = accuracy_score(heart_y_test, svm_pred_y)
lr_acc = accuracy_score(heart_y_test, lr_pred_y)

# Shows the accuracy for the 3 algorithms rounded off to 2 decimal
place
print("The accuracy for Random Forest : " + str(round(rf_acc, 2)))
print("The accuracy for Support Vector Machine : " +
str(round(svm_acc, 2)))
print("The accuracy for Logistic Regression : " + str(round(lr_acc,
2)))
```

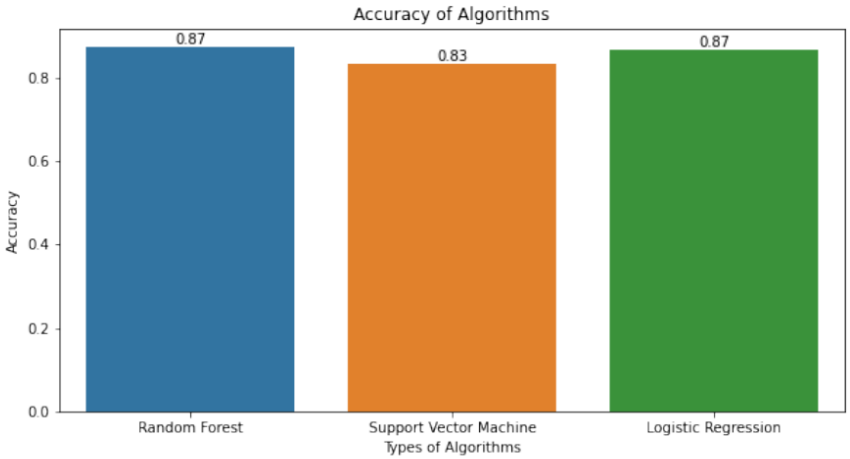
```
The accuracy for Random Forest : 0.87
The accuracy for Support Vector Machine : 0.83
The accuracy for Logistic Regression : 0.87
```

Random Forest algorithm and Logistic Regression achieved the best accuracy, followed by Support Vector Machine

```
# Creating bar plot to show the accuracy score
algorithms = ["Random Forest", "Support Vector Machine", "Logistic
Regression"]
plt.figure(figsize=(10,5))
bp = sns.barplot(x = algorithms,y = [rf_acc, svm_acc, lr_acc])
bp.bar_label(container=bp.containers[0], labels=[round(rf_acc, 2),
                                                 round(svm_acc, 2),
```

```
round(lr_acc, 2)])

plt.xlabel("Types of Algorithms")
plt.ylabel("Accuracy")
plt.title('Accuracy of Algorithms')
```

```
Text(0.5, 1.0, 'Accuracy of Algorithms')
```



4.3 F1-score

```
# Compute the F1 score of 3 models
rf_f1 = f1_score(heart_y_test, rf_pred_y)
svm_f1 = f1_score(heart_y_test, svm_pred_y)
lr_f1 = f1_score(heart_y_test, lr_pred_y)

# Shows the F1 Score for the 3 algorithms rounded off to 2 decimal
place
print("The F1 Score for Random Forest : " + str(round(rf_f1, 2)))
print("The F1 Score for Support Vector Machine : " + str(round(svm_f1,
2)))
print("The F1 Score for Logistic Regression : " + str(round(lr_f1,
2)))
```

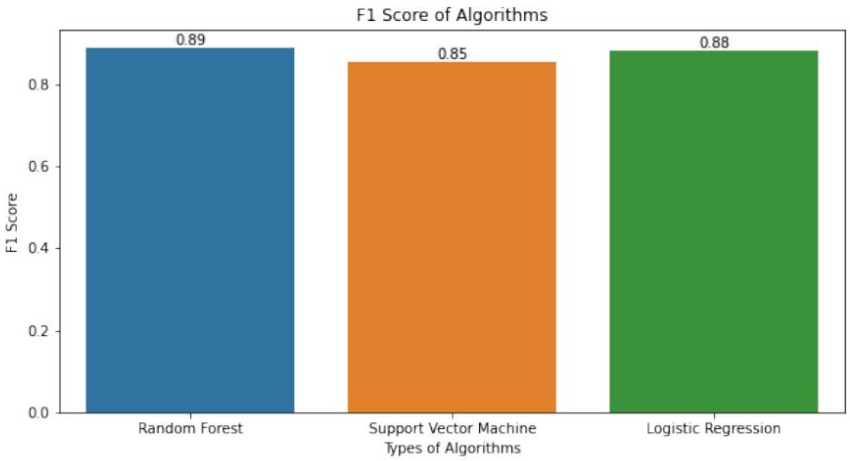
```
The F1 Score for Random Forest : 0.89
The F1 Score for Support Vector Machine : 0.85
The F1 Score for Logistic Regression : 0.88
```

Random Forest algorithm achieved the best F1 Score, followed by Logistic Regression and Support Vector Machine

```
# Creating bar plot to show the F1 score
plt.figure(figsize=(10,5))
bp = sns.barplot(x = algorithms,y = [rf_f1, svm_f1, lr_f1])
```

```
bp.bar_label(container=bp.containers[0], labels=[round(rf_f1, 2),
                                                 round(svm_f1, 2),
round(lr_f1, 2)])

plt.xlabel("Types of Algorithms")
plt.ylabel("F1 Score")
plt.title('F1 Score of Algorithms')
```

```
Text(0.5, 1.0, 'F1 Score of Algorithms')
```



4.4 Precision

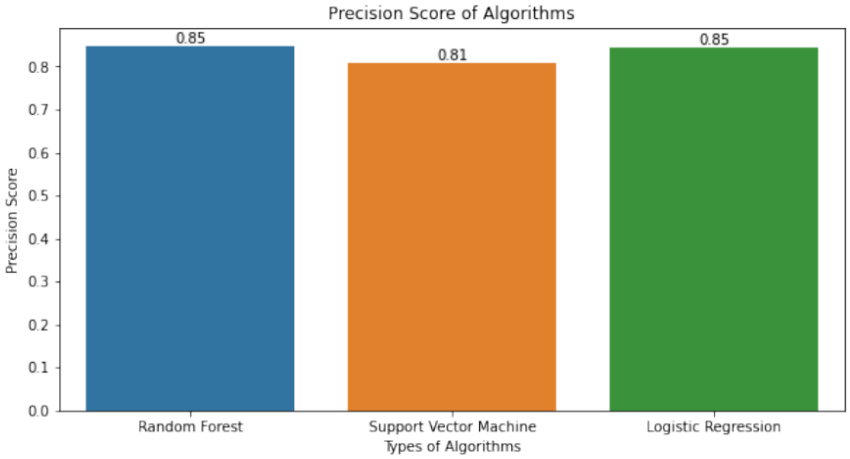
```
# Compute the Precision score of 3 models
rf_prec = precision_score(heart_y_test, rf_pred_y)
svm_prec= precision_score(heart_y_test, svm_pred_y)
lr_prec = precision_score(heart_y_test, lr_pred_y)

# Shows the Precision Score for the 3 algorithms rounded off to 2
decimal place
print("The Precision Score for Random Forest : " + str(round(rf_prec,
2)))
print("The Precision Score for Support Vector Machine : " +
str(round(svm_prec, 2)))
print("The Precision Score for Logistic Regression : " +
str(round(lr_prec, 2)))
```

```
The Precision Score for Random Forest : 0.85
The Precision Score for Support Vector Machine : 0.81
The Precision Score for Logistic Regression : 0.85
```

Random Forest and SVM achieved the best Precision Score, followed by Logistic Regression

```
# Creating bar plot to show the Precision score
plt.figure(figsize=(10,5))
bp = sns.barplot(x = algorithms,y = [rf_prec, svm_prec, lr_prec])
bp.bar_label(container=bp.containers[0], labels=[round(rf_prec, 2),
                                                 round(svm_prec, 2),
round(lr_prec, 2)])

plt.xlabel("Types of Algorithms")
plt.ylabel("Precision Score")
plt.title('Precision Score of Algorithms')
```

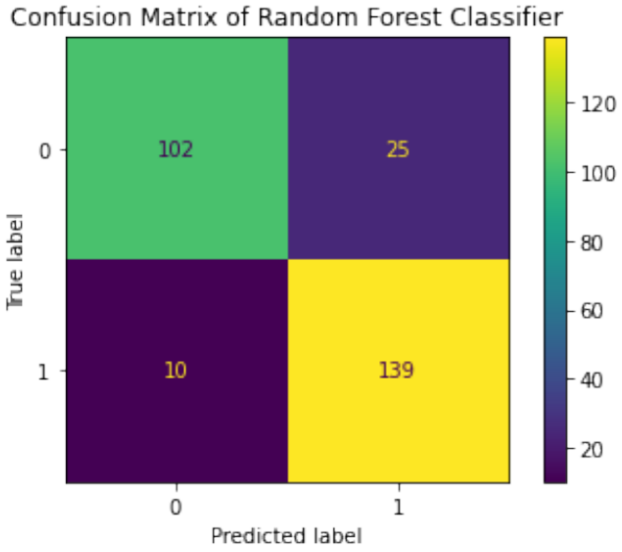
```
Text(0.5, 1.0, 'Precision Score of Algorithms')
```



4.5 Confusion Matrix

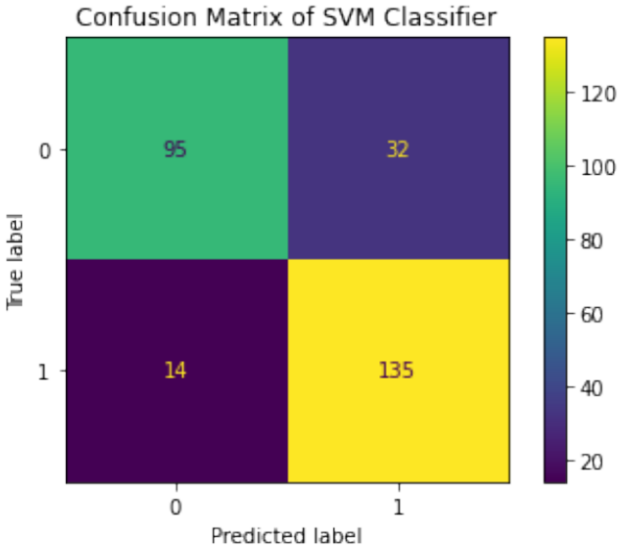
```
cm = confusion_matrix(heart_y_test, rf_pred_y)
ConfusionMatrixDisplay(confusion_matrix = cm, display_labels =
rf_clf.classes_).plot()
plt.title("Confusion Matrix of Random Forest Classifier")
```

```
Text(0.5, 1.0, 'Confusion Matrix of Random Forest Classifier')
```



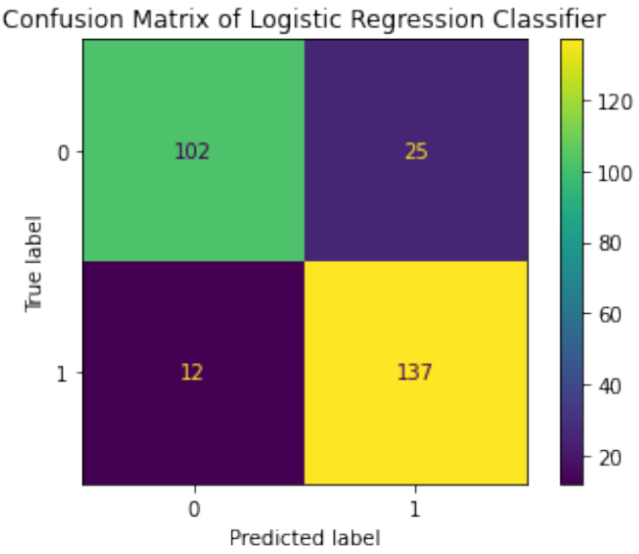

```
cm = confusion_matrix(heart_y_test, svm_pred_y)
ConfusionMatrixDisplay(confusion_matrix = cm, display_labels =
svm_clf.classes_).plot()
plt.title("Confusion Matrix of SVM Classifier")
```

```
Text(0.5, 1.0, 'Confusion Matrix of SVM Classifier')
```




```
cm = confusion_matrix(heart_y_test, lr_pred_y)
ConfusionMatrixDisplay(confusion_matrix = cm, display_labels =
lr_clf.classes_).plot()
plt.title("Confusion Matrix of Logistic Regression Classifier")
```

```
Text(0.5, 1.0, 'Confusion Matrix of Logistic Regression Classifier')
```

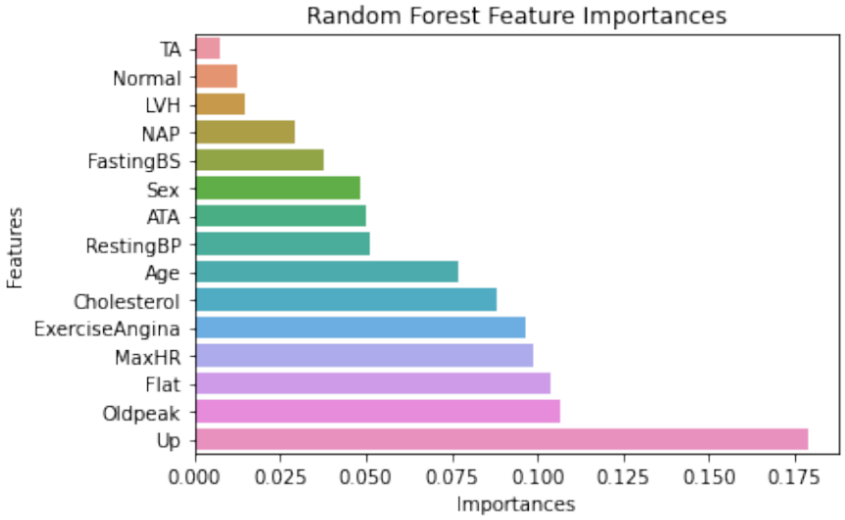


From the confusion matrix, we could see SVM perform worst at it has the lowest value on the diagonal. Logistic Regression has same number of mistake (25) with Random Forest in predicting normal people as people who have heart disease. Logistic Regression make slightly more mistake in predicting heart disease patient as normal compared to Random Forest.

4.6 Feature Weightage/Importance

```
# Creating bar plot to show the Feature Importance in Random Forest
sorted_index = rf_clf.feature_importances_.argsort()
sns.barplot(y = heart_X_train.columns[sorted_index],
            x = rf_clf.feature_importances_[sorted_index])
plt.ylabel("Features")
plt.title("Random Forest Feature Importances")
plt.xlabel("Importances")
```

```
Text(0.5, 0, 'Importances')
```


Random Forest Feature Importances
TA
Normal
LVH
NAP
FastingBS
Sex
ATA
RestingBP
Age
Cholesterol
ExerciseAngina
MaxHR
Flat
Oldpeak
Up
Features
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Importances