## Machine Learning Engineer Nanodegree

# Heart Diseases Prediction: Saving lives using Machine Learning

By Nidhi Pansuriya

April 23rd, 2021

## I. Definition

### Project Overview

Heart diseases is a term covering any disorder of the heart. Heart diseases have become a major concern to deal with as studies show that the number of deaths due to heart diseases have increased significantly over the past few decades in the world, in fact it has become the leading cause of death in the world.

- One person dies every 36 seconds just in the United States alone from cardiovascular disease.
- About 655,000 Americans die from heart disease each year—that's 1 in every 4 deaths.
- Heart disease costs the United States about $219 billion each year from 2014 to 2015. This includes the cost of healthcare services, medicines, and lost productivity due to death. Data Source: Heart Disease Facts | cdc.gov
- From 1990 to 2016 the death rate due to heart diseases have increased around 34 per cent from 155.7 to 209.1 deaths per one lakh population in India.

Thus preventing Heart diseases has become more than necessary. Good data-driven systems for predicting heart diseases can improve the entire research and prevention process, making sure that more people can live healthy lives. This is where Machine Learning comes into play.

Machine learning can potentially play a significant role in helping doctors and scientists predict heart disease. A person’s chance of having a heart disease includes many factors such as diabetes, high blood pressure, high cholesterol, abnormal heart rate, and age.

## Problem Statement

This project is the capstone assignment for Udacity Machine Learning NanoDegree. It aims to model and predict the presence of heart disease in patients by using binary classification machine learning algorithms and a dataset of patient information, which can be found on Kaggle and UC Irvine's Machine Learning Repository.

In this project, we will use Logistic Regression to predict whether a person has a heart disease or not based on the various biological and physical parameters of the body.

Note: This is only a sample application and should not be considered as medical advice.

## Metrics

The model will be using various evaluation metrics such as

- Accuracy: which refers to how close a measurement is to the true value and can be calculated using the following formula

$$\textbf{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total}}$$

- Precision: which is how consistent results are when measurements are repeated and can be calculated using the following formula

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

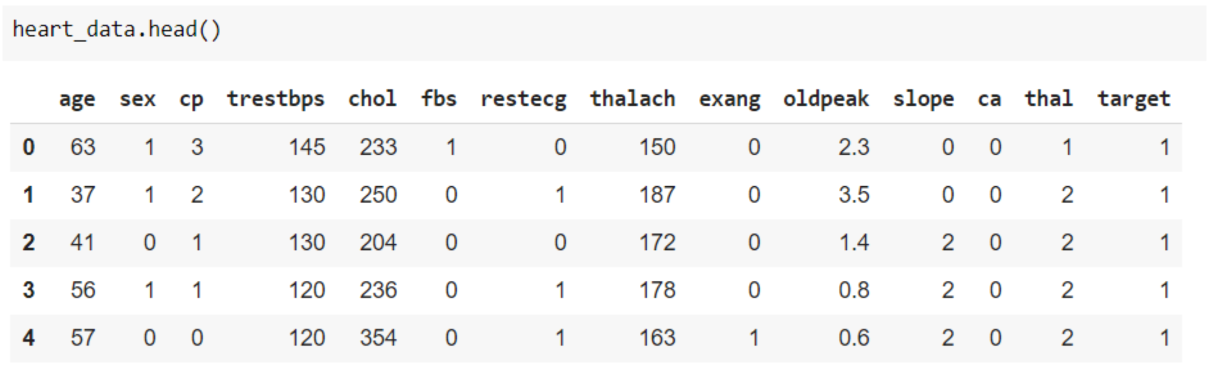
- Recall: which refers to the percentage of total relevant results correctly classified by the model and can be calculated using the formula

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

## II. Analysis

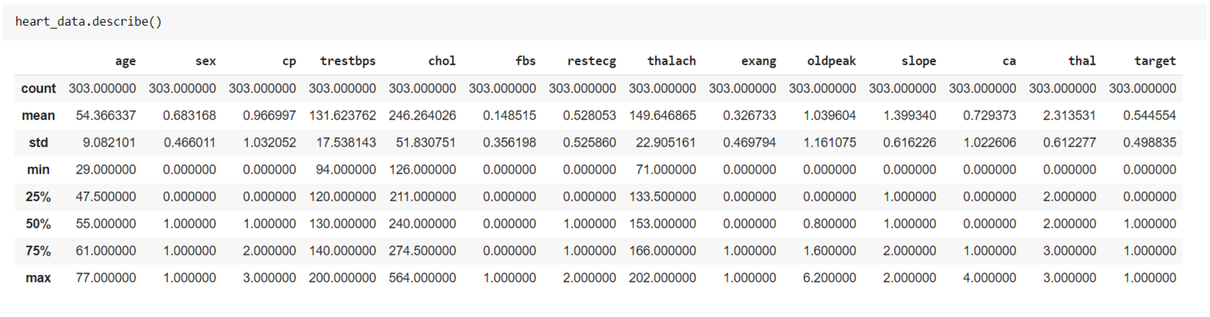
## Data Exploration

The data used for training and testing is the Heart Disease UCI downloaded from Kaggle. The dataset consists of 303 individual data. There are 76 attributes in the dataset, however all published experiments refer to using a subset of 14 of them as far as machine learning is considered so. The "goal" field refers to the presence of heart disease in the patient.



```
heart_data.head()
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | 1 |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | 1 |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | 1 |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 | 1 |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 | 1 |



```
heart_data.describe()
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 |
| **mean** | 54.366337 | 0.683168 | 0.966997 | 131.623762 | 246.264026 | 0.148515 | 0.528053 | 149.646865 | 0.326733 | 1.039604 | 1.399340 | 0.729373 | 2.313531 | 0.544554 |
| **std** | 9.082101 | 0.466011 | 1.032052 | 17.538143 | 51.830751 | 0.356198 | 0.525860 | 22.905161 | 0.469794 | 1.161075 | 0.616226 | 1.022606 | 0.612277 | 0.498835 |
| **min** | 29.000000 | 0.000000 | 0.000000 | 94.000000 | 126.000000 | 0.000000 | 0.000000 | 71.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 47.500000 | 0.000000 | 0.000000 | 120.000000 | 211.000000 | 0.000000 | 0.000000 | 133.500000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 2.000000 | 0.000000 |
| **50%** | 55.000000 | 1.000000 | 1.000000 | 130.000000 | 240.000000 | 0.000000 | 1.000000 | 153.000000 | 0.000000 | 0.800000 | 1.000000 | 0.000000 | 2.000000 | 1.000000 |
| **75%** | 61.000000 | 1.000000 | 2.000000 | 140.000000 | 274.500000 | 0.000000 | 1.000000 | 166.000000 | 1.000000 | 1.600000 | 2.000000 | 1.000000 | 3.000000 | 1.000000 |
| **max** | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 564.000000 | 1.000000 | 2.000000 | 202.000000 | 1.000000 | 6.200000 | 2.000000 | 4.000000 | 3.000000 | 1.000000 |

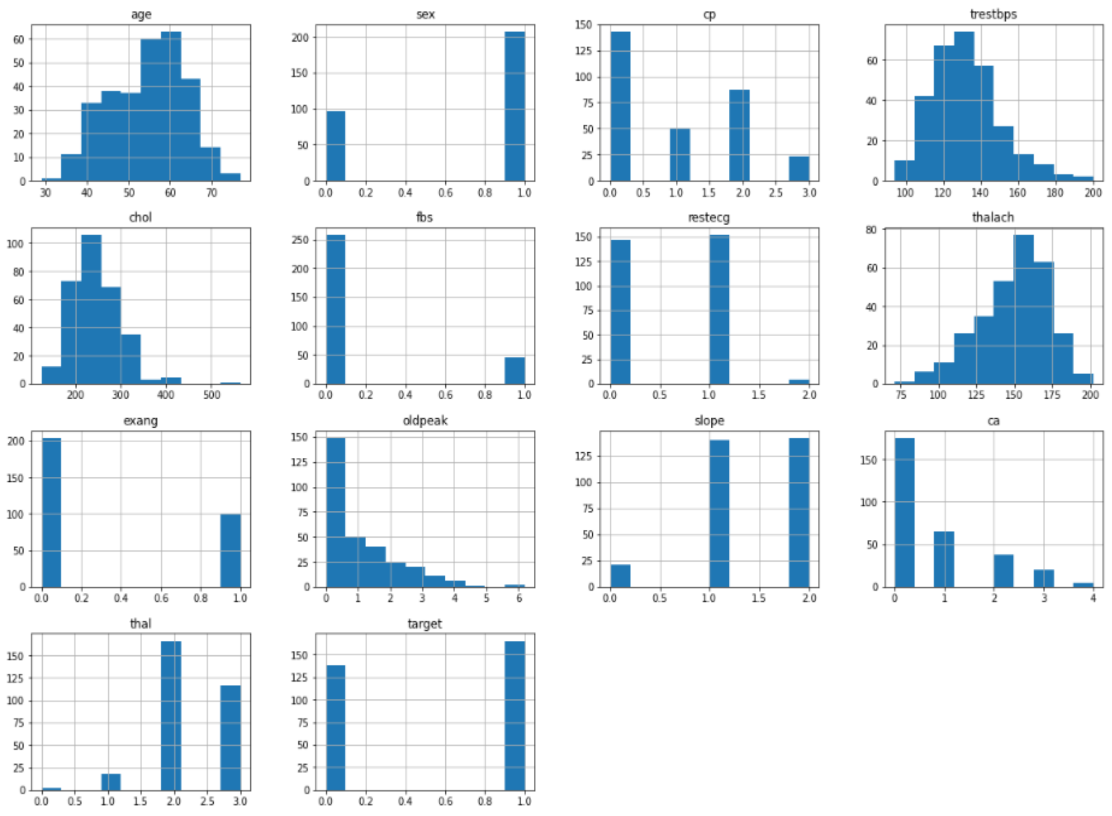
It's a clean, easy to understand set of data. However, the meaning of some of the column headers are not obvious. Here's what they mean,

- age: displays the age of the individual.
- sex: displays the gender of the individual using the following format : 1 = male 0 = female.
- cp: Chest-pain type - displays the type of chest-pain experienced by the individual using the following format : 1 = typical angina 2 = atypical angina 3 = non - anginal pain 4 = asymptotic
- trestbps: Resting Blood Pressure - displays the resting blood pressure value of an individual in mmHg (unit)
- chol: Serum Cholesterol - displays the serum cholesterol in mg/dl (unit)
- fbs: Fasting Blood Sugar - compares the fasting blood sugar value of an individual with 120mg/dl. If fasting blood sugar > 120mg/dl then : 1 (true) else : 0 (false)

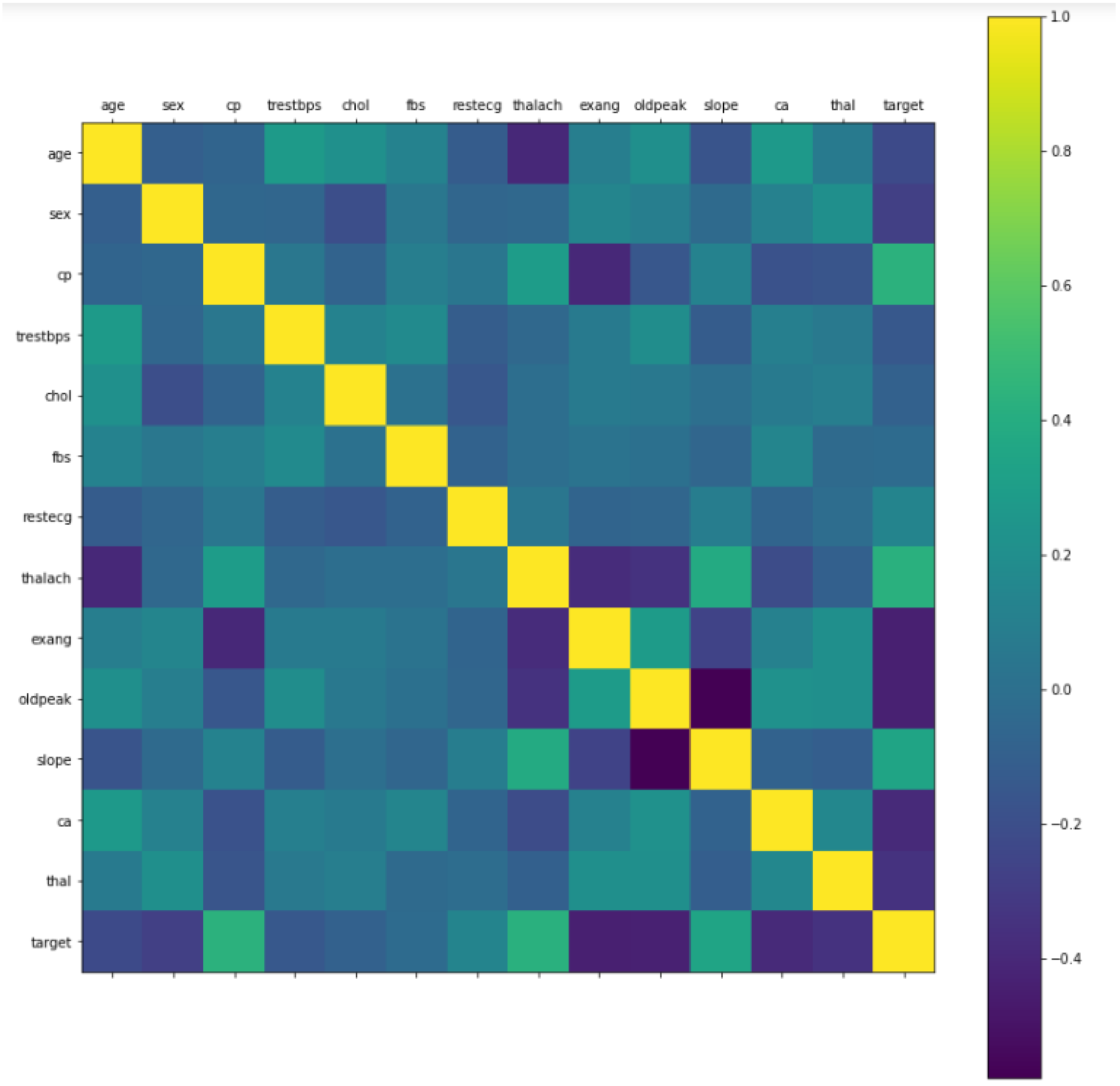
- restecg: Resting ECG measurement - (0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria)
- Thalach: Max heart rate achieved - displays the max heart rate achieved by an individual.
- exang: Exercise induced angina - (1 = yes; 0 = no)
- ldpeak: ST depression induced by exercise relative to rest ('ST' relates to positions on the ECG plot. See more here)
- slope: the slope of the peak exercise ST segment (1 = upsloping 2 = flat 3 = downsloping)
- ca: Number of major vessels (0-3) colored by fluoroscopy - displays the value as integer or float.
- thal: A blood disorder called thalassemia (3 = normal; 6 = fixed defect; 7 = reversible defect)
- target: Displays whether the individual is suffering from heart disease or not : 0 = absence 1 = present.

## Exploratory Visualization

Now let's see various visual representations of the data to understand more about various features.

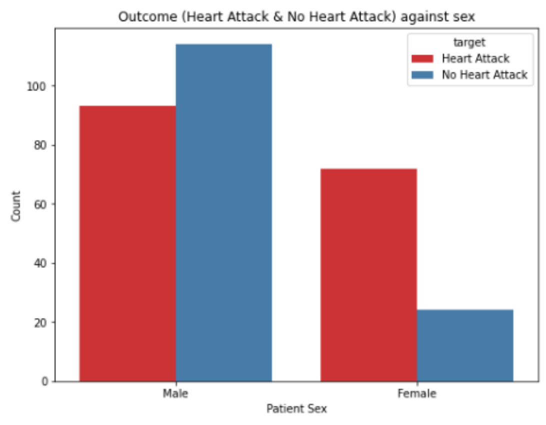


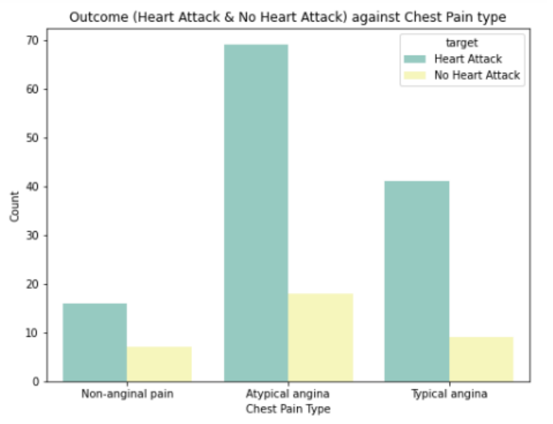
The best way to compare relationships between various features is to look at the correlation matrix between those features.

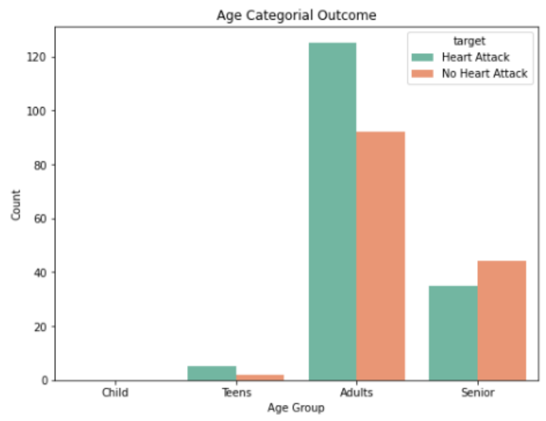


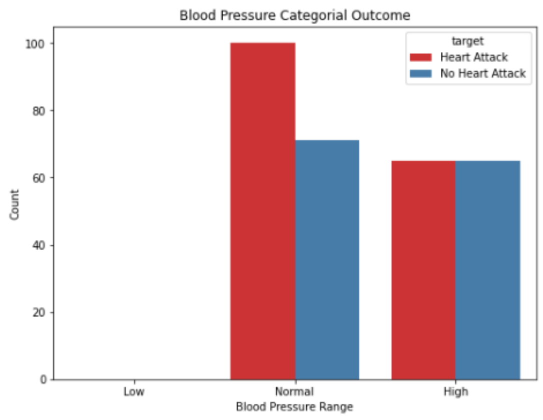
Here our Model will trained to predict whether a person has a heart disease or not based on the following common features as input:

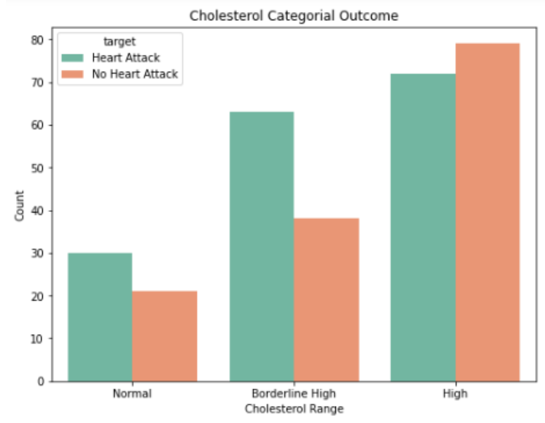
- age
- gender
- chest pain
- blood pressure
- cholesterol level
- blood sugar
- max heart rate

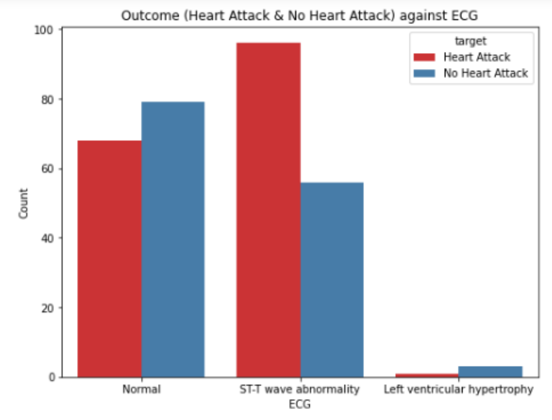

Outcome (Heart Attack & No Heart Attack) against sex
target
Heart Attack
No Heart Attack
Count
0
20
40
60
80
100
Male
Female
Patient Sex

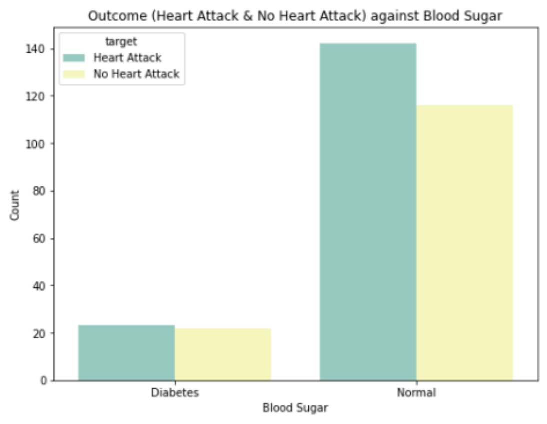

Outcome (Heart Attack & No Heart Attack) against Chest Pain type
target
Heart Attack
No Heart Attack
Count
0
10
20
30
40
50
60
70
Non-anginal pain
Atypical angina
Typical angina
Chest Pain Type

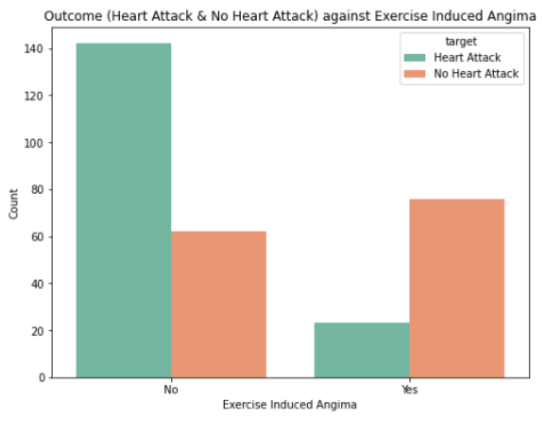

Age Categorial Outcome
target
Heart Attack
No Heart Attack
Count
0
20
40
60
80
100
120
Child
Teens
Adults
Senior
Age Group


Blood Pressure Categorial Outcome
target
Heart Attack
No Heart Attack
Count
0
20
40
60
80
100
Low
Normal
High
Blood Pressure Range


Cholesterol Categorial Outcome
target
Heart Attack
No Heart Attack
Count
0
10
20
30
40
50
60
70
80
Normal
Borderline High
High
Cholesterol Range


Outcome (Heart Attack & No Heart Attack) against ECG
target
Heart Attack
No Heart Attack
Count
0
20
40
60
80
100
Normal
ST-T wave abnormality
Left ventricular hypertrophy
ECG


Outcome (Heart Attack & No Heart Attack) against Blood Sugar
target
Heart Attack
No Heart Attack
Count
0
20
40
60
80
100
120
140
Diabetes
Normal
Blood Sugar


Outcome (Heart Attack & No Heart Attack) against Exercise Induced Angima
target
Heart Attack
No Heart Attack
Count
0
20
40
60
80
100
120
140
No
Yes
Exercise Induced Angima

## Algorithms and Techniques

**Logistic Regression Model**

What is Logistic Regression ?

- *Logistic Regression is a statistical and machine-learning technique classifying records of a dataset based on the values of the input fields . It predicts a dependent variable based on one or more sets of independent variables to predict outcomes . It can be used both for binary classification and multi-class classification.*

**AWS Sagemaker**

Linear models are supervised learning algorithms used for solving either classification or regression problems. For input, you give the model labeled examples (x, y). x is a high-dimensional vector and y is a numeric label. For binary classification problems, the label must be either 0 or 1. For multiclass classification problems, the labels must be from 0 to num_classes - 1. For regression problems, y is a real number. The algorithm learns a linear function, or, for classification problems, a linear threshold function, and maps a vector x to an approximation of the label y.

The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. For this project, I have implemented a Linear Learner for Binary classification problem.

The linear learner algorithm requires a data matrix, with rows representing the observations, and columns representing the dimensions of the features. It also requires an additional column that contains the labels that match the data points. At a minimum, Amazon SageMaker linear learner requires you to specify input and output data locations, and objective type (classification) as arguments.

In this project, I will be using the container image of 'linear-learner' for training the model with hyperparameters:

- feature dimension
- predictor type

```
from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(boto3.Session().region_name, 'linear-learner', "latest")

linear = sagemaker.estimator.Estimator(container,
                                       role,
                                       train_instance_count=1,
                                       train_instance_type='ml.c4.xlarge',
                                       output_path=output_path,
                                       sagemaker_session=session)
linear.set_hyperparameters(feature_dim=30,
                           predictor_type='binary_classifier',
                           mini_batch_size=100)
```

## Benchmarking

I have picked the XGBoost model as the benchmark baseline in this project. XGBoost works well on complicated data. Also, for xgboost, less feature engineering required (No need for scaling, normalizing, can handle missing values also).

However the biggest limitation is the blackbox nature, because of it, xgboost is most likely to be overfit if parameters are not tuned properly.

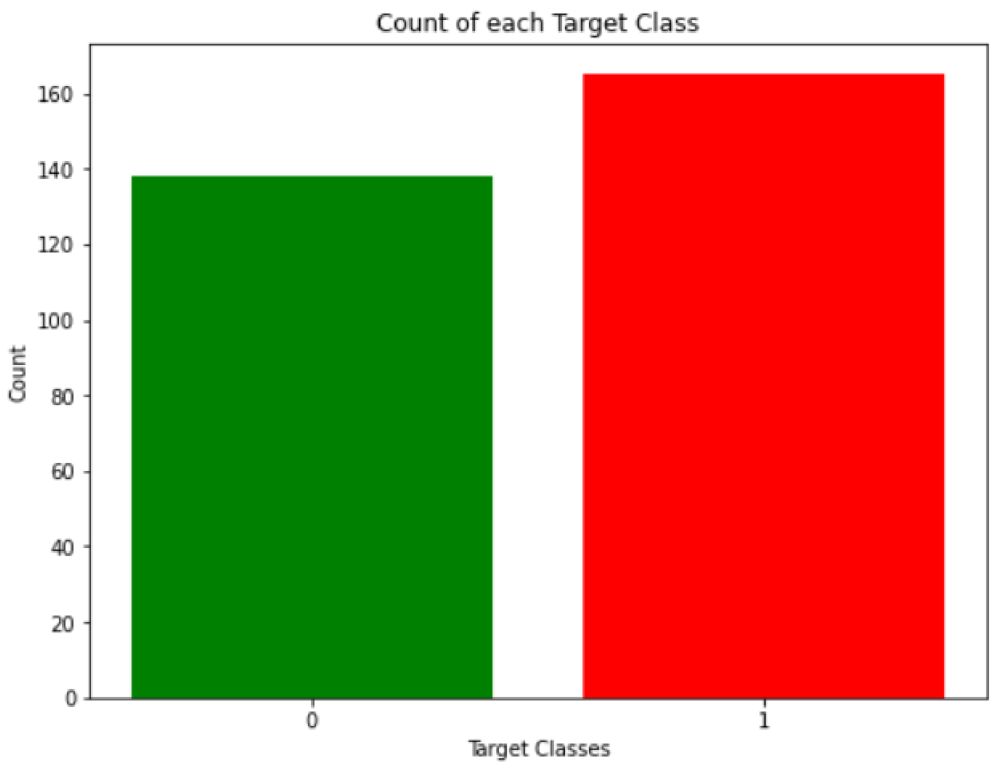
On the other hand, Logistic regression is intrinsically simple, it has low variance and so is less prone to overfitting.

## III. Methodology

## Data Preprocessing

In this project, data is split based on a ratio of 80:20 for the training set and the test set. The training set data is used in the logistic regression component for model training, while the prediction set data is used in the prediction component.

The Dataset class balance is as below:



From the above figure, it is quite clear that the dataset class is balanced (class 0 - 140, class 1 - 160). Thus class balancing is not required.

Standard classifier algorithms like Logistic Regression have a bias towards classes which have a number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class if the dataset is imbalanced.

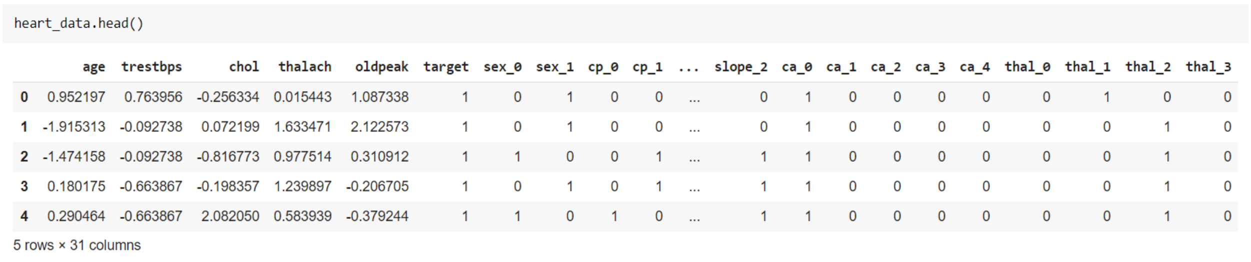
After exploring the dataset, I have observed that some categorical variables need to be converted into dummy variables and all the values must be scaled before training the Machine Learning models. With the use of *get_dummies* method,dummy columns for categorical variables are created.

```
heart_data = pd.get_dummies(heart_data, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
```

After that the dataset is scaled by using sklearn.StandardScaler.

```
standardScaler = StandardScaler()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
heart_data[columns_to_scale] = standardScaler.fit_transform(heart_data[columns_to_scale])
```

Scaled dataset:



```
heart_data.head()
```

| | age | trestbps | chol | thalach | oldpeak | target | sex_0 | sex_1 | cp_0 | cp_1 | ... | slope_2 | ca_0 | ca_1 | ca_2 | ca_3 | ca_4 | thal_0 | thal_1 | thal_2 | thal_3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.952197 | 0.763956 | -0.256334 | 0.015443 | 1.087338 | 1 | 0 | 1 | 0 | 0 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **1** | -1.915313 | -0.092738 | 0.072199 | 1.633471 | 2.122573 | 1 | 0 | 1 | 0 | 0 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **2** | -1.474158 | -0.092738 | -0.816773 | 0.977514 | 0.310912 | 1 | 1 | 0 | 0 | 1 | ... | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **3** | 0.180175 | -0.663867 | -0.198357 | 1.239897 | -0.206705 | 1 | 0 | 1 | 0 | 1 | ... | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **4** | 0.290464 | -0.663867 | 2.082050 | 0.583939 | -0.379244 | 1 | 1 | 0 | 1 | 0 | ... | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

5 rows × 31 columns

## Implementation

The first part of the implementation was to get the image for the *linear-learner* algorithm by making use of Docker containers. And then, creating the Estimator object by *sagemaker.estimator.Estimator*.

```
from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(boto3.Session().region_name, 'linear-learner', "latest")

linear = sagemaker.estimator.Estimator(container,
                                       role,
                                       train_instance_count=1,
                                       train_instance_type='ml.c4.xlarge',
                                       output_path=output_path,
                                       sagemaker_session=session)
```

After creating the linear-learner estimator object, hyperparameters must be set by the *set_hyperparameters* method.

```
linear.set_hyperparameters(feature_dim=30,
                           predictor_type='binary_classifier',
                           mini_batch_size=100)
```

Since this project is a classification problem, I have selected *predictor_type* as *'binary_classifier'* and '*feature_dim' = 30* (scaled features).

After setting up the model, I have attached the training dataset by *fit()* method.

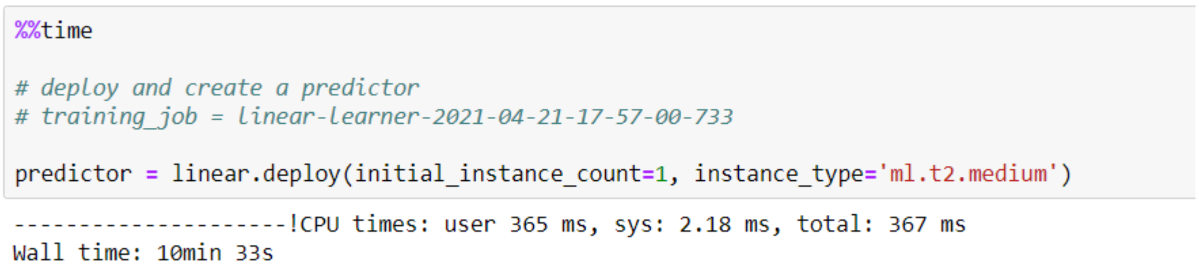
```
from sagemaker.inputs import TrainingInput

s3_input_train = TrainingInput(s3_data=train_location, content_type='text/csv')

linear.fit({'train': s3_input_train})
```

After training, we have to deploy the trained model to create a predictor object.

To deploy a trained model, we'll use the *deploy()* method.



```
%%time

# deploy and create a predictor
# training_job = linear-learner-2021-04-21-17-57-00-733

predictor = linear.deploy(initial_instance_count=1, instance_type='ml.t2.medium')
```

```
---------------------!CPU times: user 365 ms, sys: 2.18 ms, total: 367 ms
Wall time: 10min 33s
```

We can train our prediction model by analyzing existing data because we already know whether each patient has heart disease. This process is also known as supervision and learning. The trained model is then used to predict if users suffer from heart disease.

## IV. Results

## Model Evaluation and Validation

From the training logs, we can see the metrics of this model as :

- ('binary_classification_cross_entropy_objective', 0.31188186929245626)
- ('binary_classification_accuracy', 0.8884297520661157)
- ('binary_f_1.000', 0.898876404494382)
- ('precision', 0.8823529411764706)
- ('recall', 0.916030534351145)
- ('roc_auc_score', 0.939275153015611)

And best model found for hyperparameters:

- "optimizer": "adam",
- "learning_rate": 0.1,
- "wd": 0.01, "l1": 0.0,
- "lr_scheduler_step": 100,
- "lr_scheduler_factor": 0.99,
- "lr_scheduler_minimum_lr": 1e-05

The final evaluation of this model is :

| | |
|---|---|
| Recall: | 0.971 |
| Precision: | 0.868 |
| Accuracy: | 0.902 |

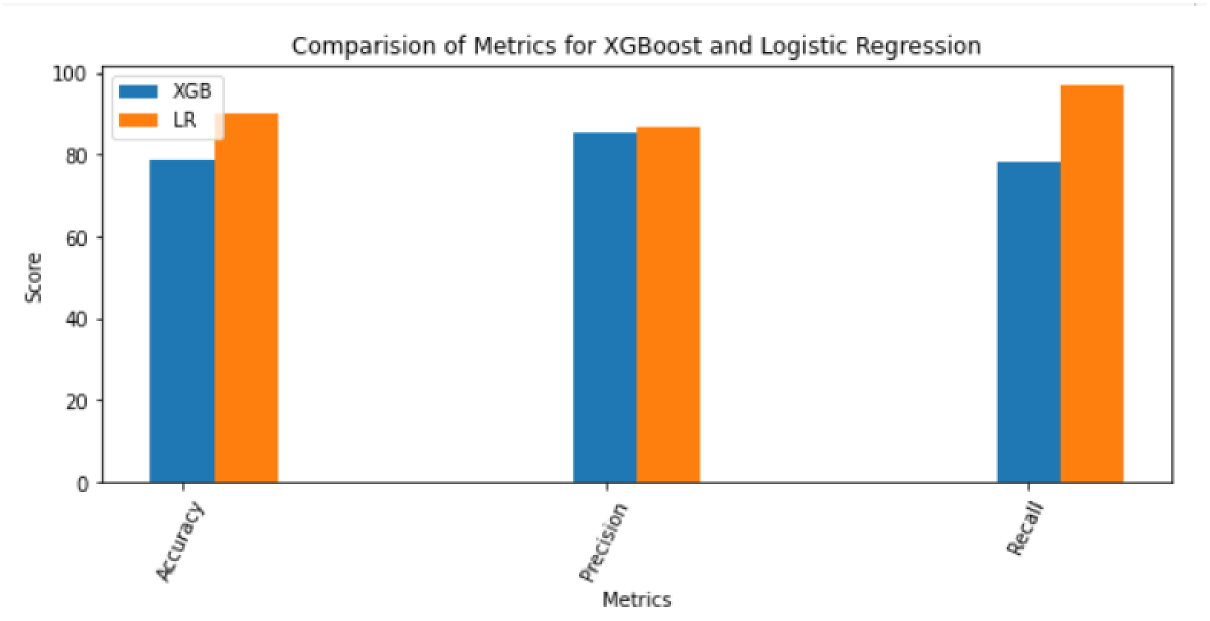
With the confusion matrix:

```
prediction (col)   0   1
actual (row)
0                 22   5
1                  1  33
```

## Conclusion

### Free Form Visualization:

As the performance of our model was benchmarked by its accuracy, I've recorded the accuracy, precision and recall of both the models for test datasets. Here's the metrics score graph of XGBoost model and Logistic Regression model.



Below are the metrics for XGBoost Model:

```
prediction (col)   0   1
actual (row)
0                 19   5
1                  8  29

Recall:     0.784
Precision:  0.853
Accuracy:   0.787
```

And Logistic Regression Model:

```
prediction (col)   0   1
actual (row)
0                 22   5
1                  1  33

Recall:     0.971
Precision:  0.868
Accuracy:   0.902
```

These results showed that our model generalized surprisingly well to the data it had not seen before. It was able to classify 55 out of 61 input data provided to it.

## Refinement

### Summary of the end-to-end solution:

The entire end-to-end problem solution can be summarized as following:

1. The input data is preprocessed such as Standard Scaling.
2. Constructing new features from current features with categorical values.
3. Transformed features are then fed into machine learning models- Logistic Regression and XGBoost model with well chosen hyperparameters.
4. Final model is ready to predict the unseen data.

### Critical steps and main challenges:

The most interesting aspect of this project is to construct the new features based on existing features. I think with the current feature engineering, there are still plenty of rooms to improve or drop out of the input data. Also, the most difficult part in this project is to deal with feature engineering. Without any domain knowledge and experience in the industry, I cannot figure a lot of possibilities to create new features. Another problem is time limitation to do some experiments with building new features.

## Improvement

As in most projects, there are way too many aspects that can be improved. But in my opinion the most important are:

- Better benchmarks - In this project, we used a simple benchmark - XGBoost. Due to time and computational cost it was not possible for me to run more experiments using different known models for this dataset. It's definitely possible that some of them would be more effective and can also improve the accuracy of the classifier. Given enough time and computational power, I'd definitely like to explore the different approaches

- Dropping irrelevant features may improve the model accuracy. Also, reducing the number of features could solve the curse of dimensionality.