CARDIOVASCULAR DISEASE PREDICTION WITH EXPLAINABLE AI (XAI)

FARIS HAIKAL FIRDAUS

Table of Contents

Background

Heart disease encompasses a spectrum of medical conditions that impact the functioning of the heart. The incidence of heart disease is on the rise, primarily due to escalating stress levels.

As per data from the World Health Organisation (WHO), cardiovascular diseases (CVDs) have emerged as the leading global cause of mortality. These diseases claim a staggering 17.9 million lives annually, accounting for approximately 31% of all worldwide fatalities.

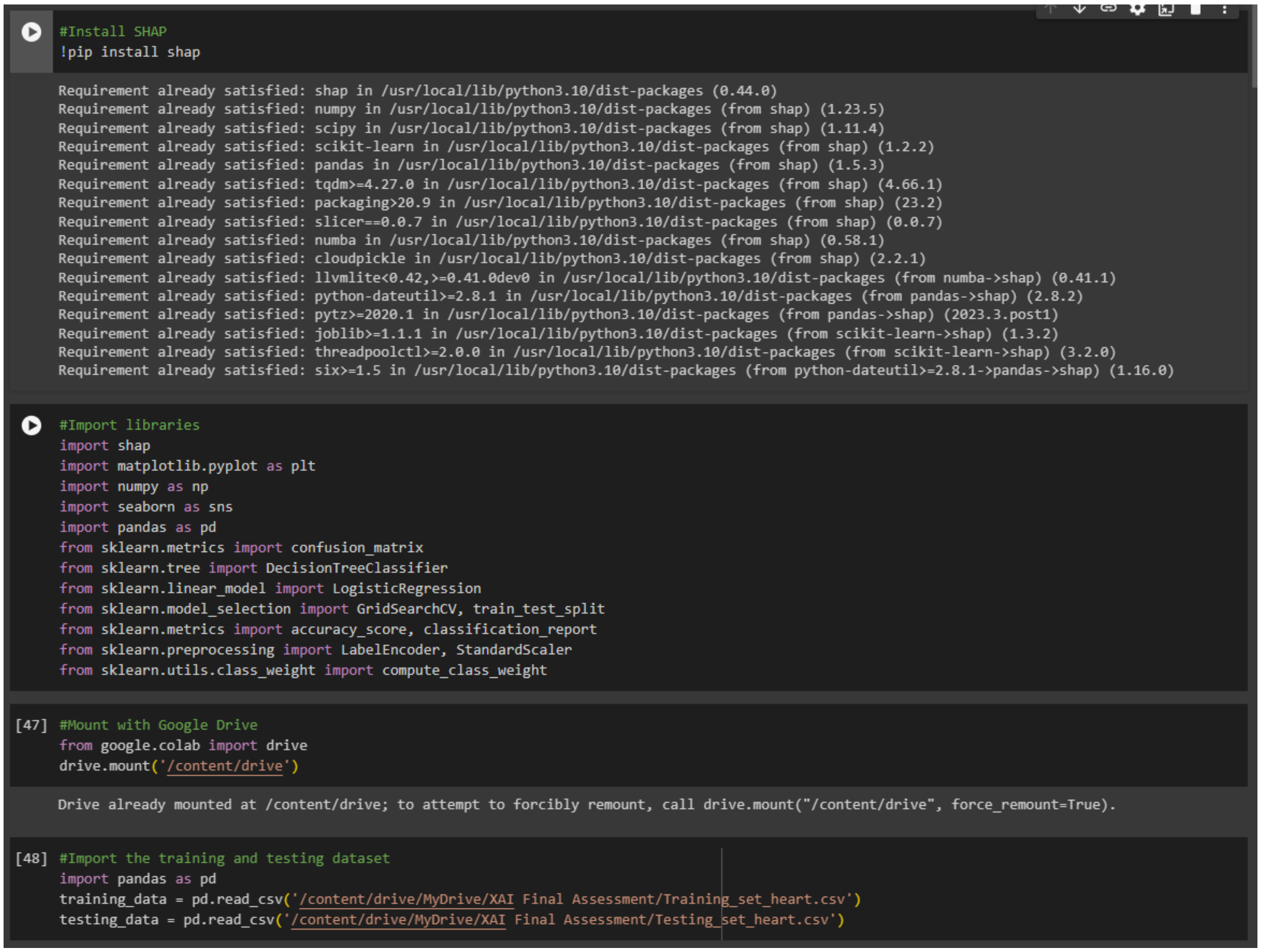
Problem Statement

The healthcare team at XYZ's Doctor of Health is keen on integrating Data Science into their operations. Given the alarming surge in heart disease cases, their primary focus lies in leveraging existing data to predict the likelihood of an individual having heart disease.

<u>Dataset description:</u>

- id: Unique id for each patient
- age: Age of the patient in years
- origin: place of study
- sex: Gender (Male of Female)
- cp chest pain type: (typical angina, atypical angina, non-anginal, asymptomatic)
- trestbps resting blood pressure: resting blood pressure (in mm Hg on admission to the hospital)
- chol: serum cholesterol in mg/dl
- fbs: if fasting blood sugar > 120 mg/dl
- restecg: resting electrocardiographic results (normal, stt abnormality, lv hypertrophy)
- thalach: maximum heart rate achieved
- exang: exercise-induced angina (True or False)
- oldpeak: ST depression induced by exercise relative to rest
- slope: the slope of the peak exercise ST segment
- ca: number of major vessels (0-3) colored by fluoroscopy
- thal: (normal, fixed defect, reversible defect)
- num: predicted attribute

Import Libraries and Load the data



```
#Install SHAP
!pip install shap
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.10/dist-packages (0.44.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.66.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (23.2)
Requirement already satisfied: slicer==0.0.7 in /usr/local/lib/python3.10/dist-packages (from shap) (0.0.7)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.58.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap) (0.41.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2023.3.post1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->shap) (1.16.0)
```

```
#Import libraries
import shap
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.utils.class_weight import compute_class_weight
```

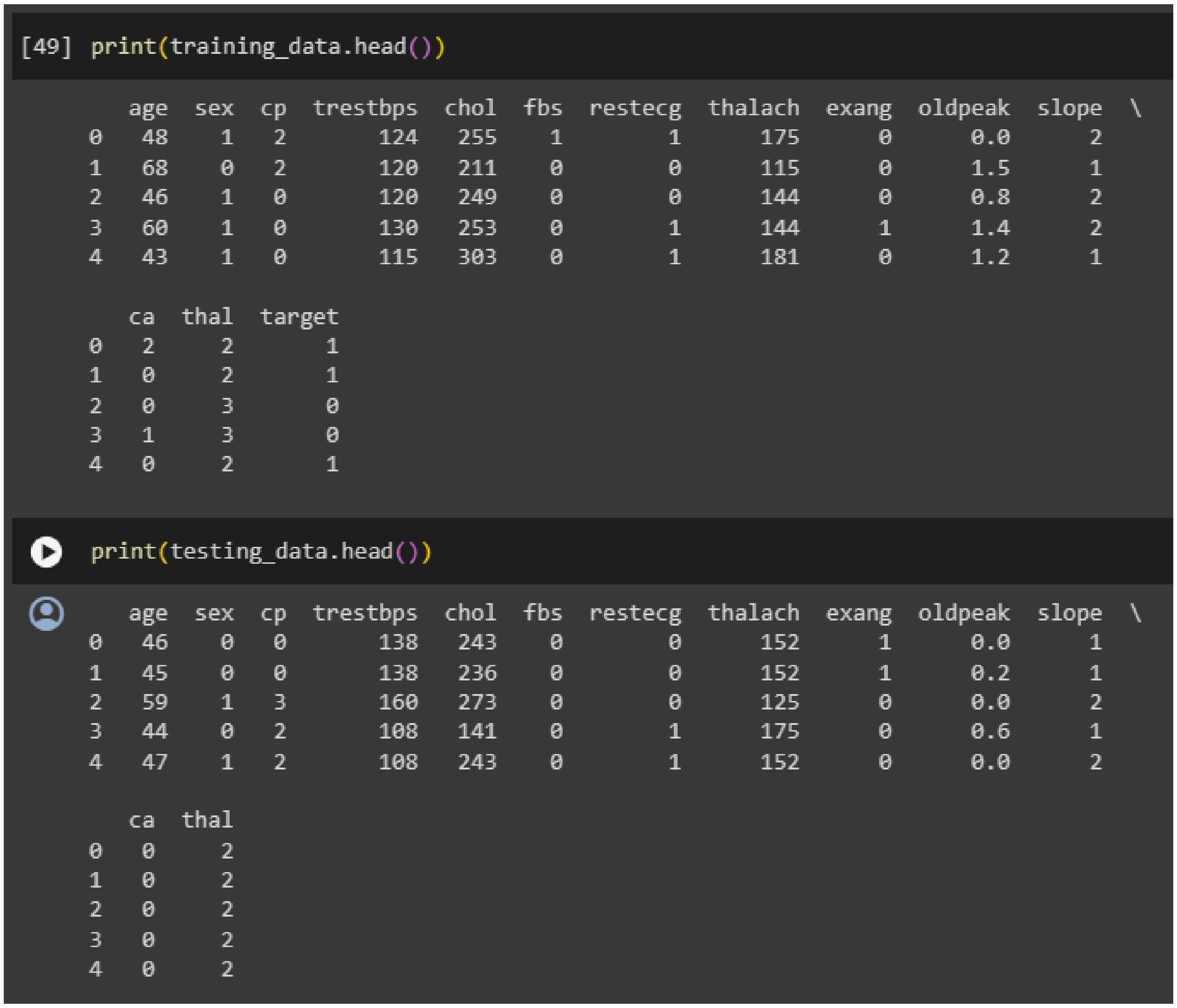
```
[47] #Mount with Google Drive
     from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[48] #Import the training and testing dataset
     import pandas as pd
     training_data = pd.read_csv('/content/drive/MyDrive/XAI Final Assessment/Training_set_heart.csv')
     testing_data = pd.read_csv('/content/drive/MyDrive/XAI Final Assessment/Testing_set_heart.csv')
```

Figure 1: import libraries and data loading

Figure 1 above shows the procedure of importing libraries and loading the dataset. As this project is using SHAP as XAI model, hence it needs to be installed. Upon installing SHAP, import all related libraries such as NumPy, Pandas and sklearn. For data loading, the datasets are loaded from Google Drive. training_data is for importing training dataset and testing_data for importing testing dataset. The provided training and testing are in two separate CSV files.

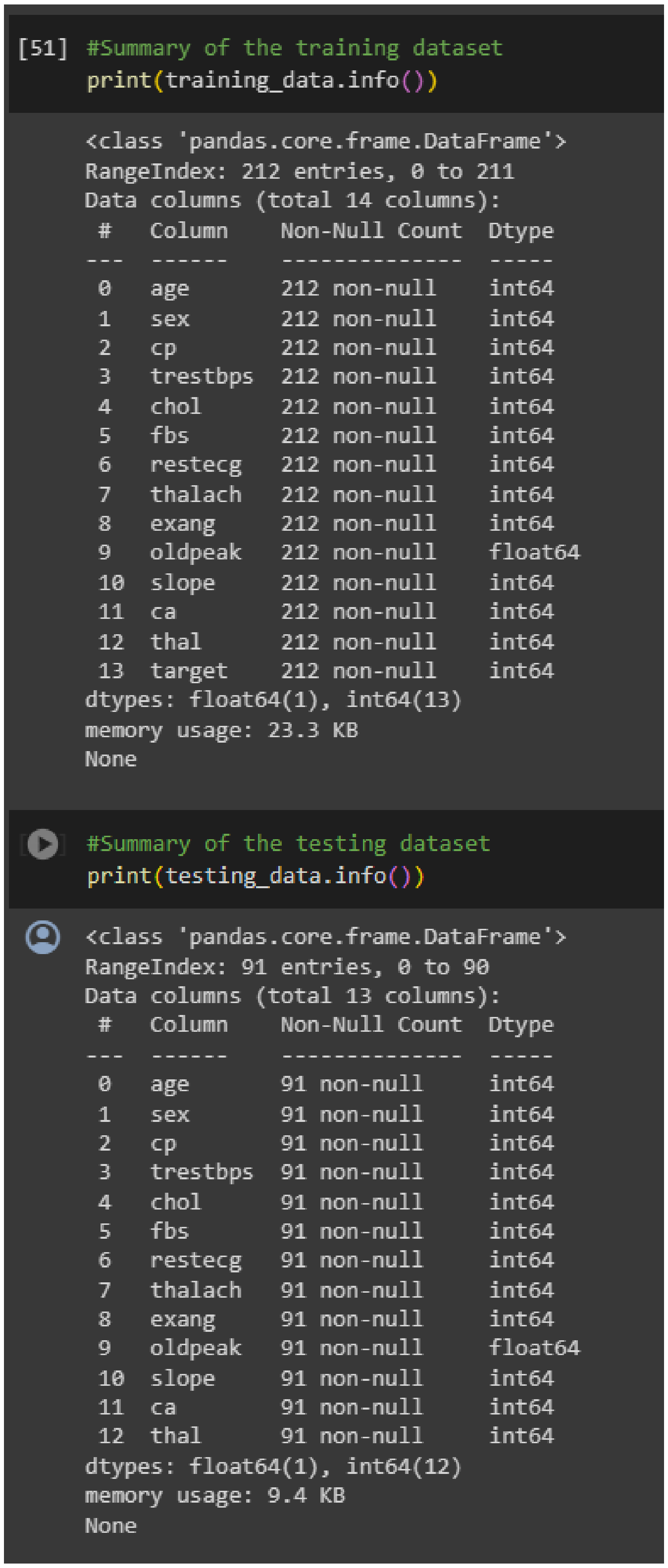


```
[49] print(training_data.head())

   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   48    1   2       124   255    1        1      175      0      0.0      2
1   68    0   2       120   211    0        0      115      0      1.5      1
2   46    1   0       120   249    0        0      144      0      0.8      2
3   60    1   0       130   253    0        1      144      1      1.4      2
4   43    1   0       115   303    0        1      181      0      1.2      1

   ca  thal  target
0   2     2       1
1   0     2       1
2   0     3       0
3   1     3       0
4   0     2       1
```

```
print(testing_data.head())

   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   46    0   0       138   243    0        0      152      1      0.0      1
1   45    0   0       138   236    0        0      152      1      0.2      1
2   59    1   3       160   273    0        0      125      0      0.0      2
3   44    0   2       108   141    0        1      175      0      0.6      1
4   47    1   2       108   243    0        1      152      0      0.0      2

   ca  thal
0   0     2
1   0     2
2   0     2
3   0     2
4   0     2
```

Figure 2: displaying first 5 rows

Upon loading the dataset, display first 5 rows of training and testing dataset using data.head. This provide the structure of the dataset such as the columns and data type.

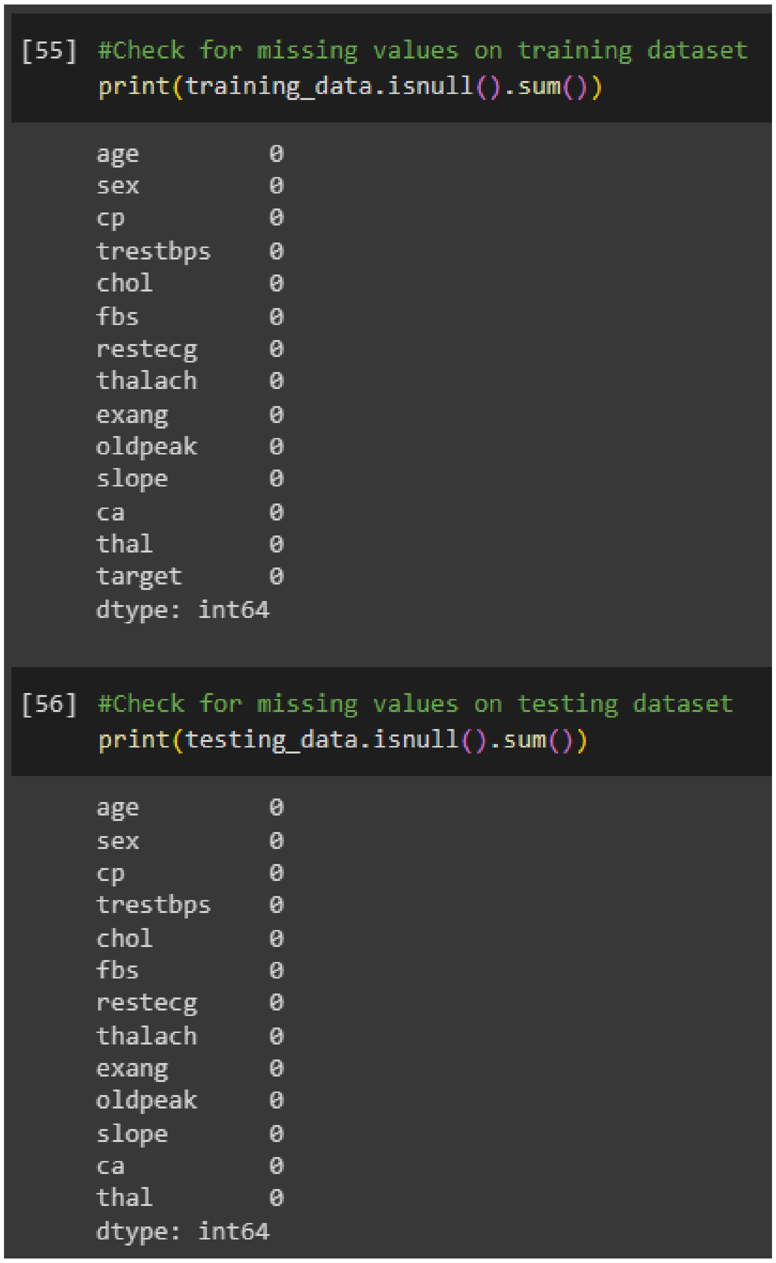


```
[51] #Summary of the training dataset
     print(training_data.info())

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 212 entries, 0 to 211
     Data columns (total 14 columns):
      #   Column    Non-Null Count  Dtype
     ---  ------    --------------  -----
      0   age       212 non-null    int64
      1   sex       212 non-null    int64
      2   cp        212 non-null    int64
      3   trestbps  212 non-null    int64
      4   chol      212 non-null    int64
      5   fbs       212 non-null    int64
      6   restecg   212 non-null    int64
      7   thalach   212 non-null    int64
      8   exang     212 non-null    int64
      9   oldpeak   212 non-null    float64
      10  slope     212 non-null    int64
      11  ca        212 non-null    int64
      12  thal      212 non-null    int64
      13  target    212 non-null    int64
     dtypes: float64(1), int64(13)
     memory usage: 23.3 KB
     None
```

```
     #Summary of the testing dataset
     print(testing_data.info())

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 91 entries, 0 to 90
     Data columns (total 13 columns):
      #   Column    Non-Null Count  Dtype
     ---  ------    --------------  -----
      0   age       91 non-null     int64
      1   sex       91 non-null     int64
      2   cp        91 non-null     int64
      3   trestbps  91 non-null     int64
      4   chol      91 non-null     int64
      5   fbs       91 non-null     int64
      6   restecg   91 non-null     int64
      7   thalach   91 non-null     int64
      8   exang     91 non-null     int64
      9   oldpeak   91 non-null     float64
      10  slope     91 non-null     int64
      11  ca        91 non-null     int64
      12  thal      91 non-null     int64
     dtypes: float64(1), int64(12)
     memory usage: 9.4 KB
     None
```

Figure 3: summary of the dataset

Exploratory Data Analysis (EDA)



```
[55] #Check for missing values on training dataset
     print(training_data.isnull().sum())

     age         0
     sex         0
     cp          0
     trestbps    0
     chol        0
     fbs         0
     restecg     0
     thalach     0
     exang       0
     oldpeak     0
     slope       0
     ca          0
     thal        0
     target      0
     dtype: int64
```

```
[56] #Check for missing values on testing dataset
     print(testing_data.isnull().sum())

     age         0
     sex         0
     cp          0
     trestbps    0
     chol        0
     fbs         0
     restecg     0
     thalach     0
     exang       0
     oldpeak     0
     slope       0
     ca          0
     thal        0
     dtype: int64
```

Figure 4: check for missing values

Figure 4 is the procedure of checking for any missing values on both datasets. Based on the outcome shown, there are no missing values.

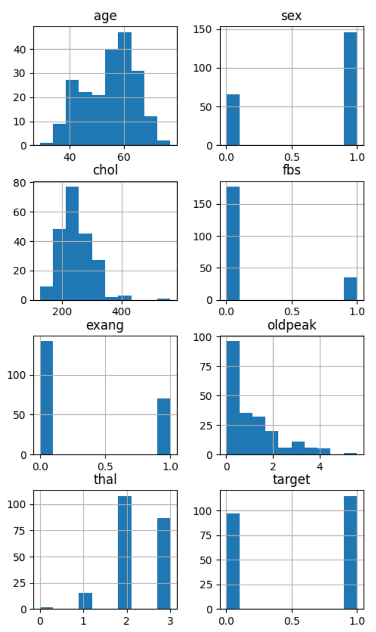
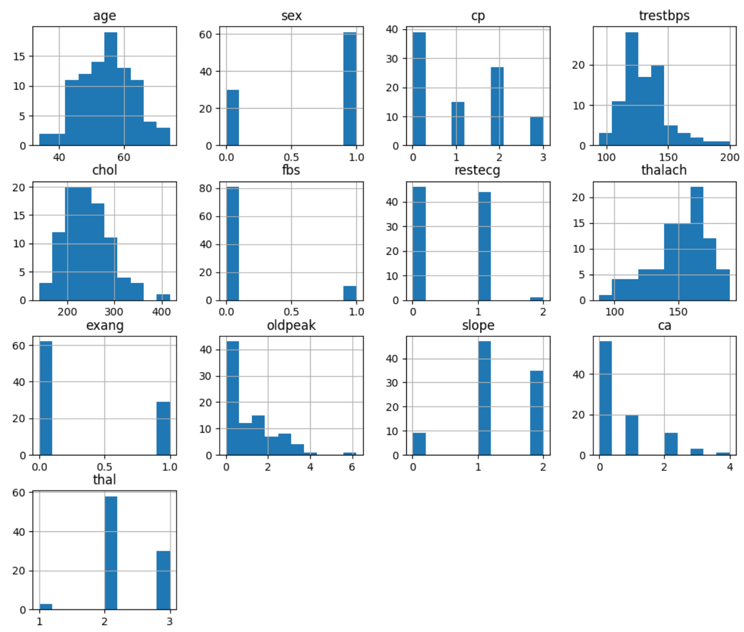



Figure 5 and 6: distribution for each feature in training and testing dataset

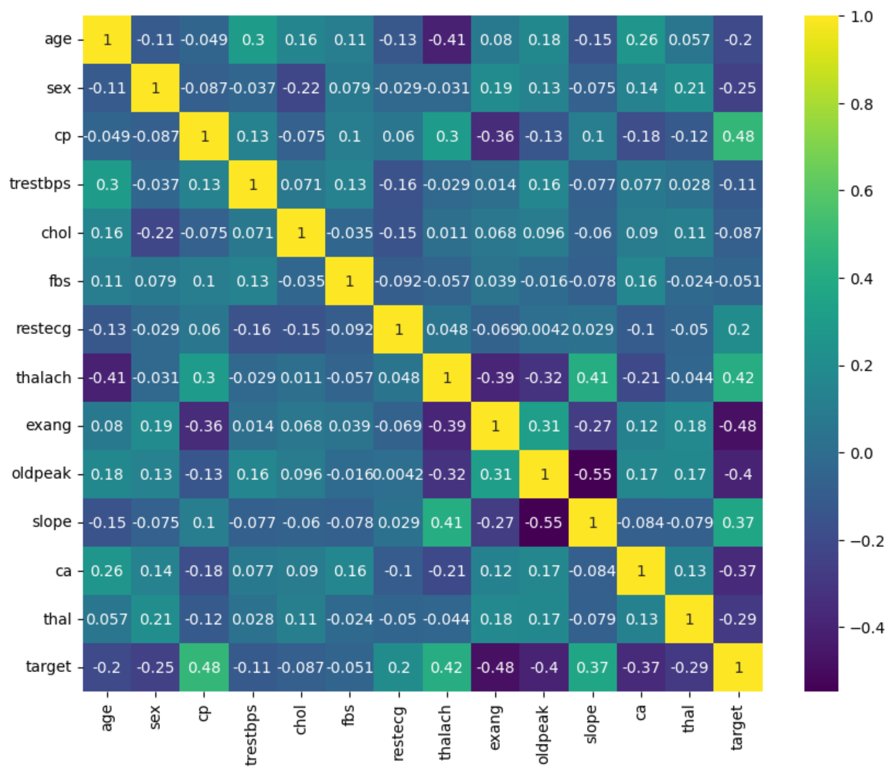


Figure 7: correlation matrix on training dataset

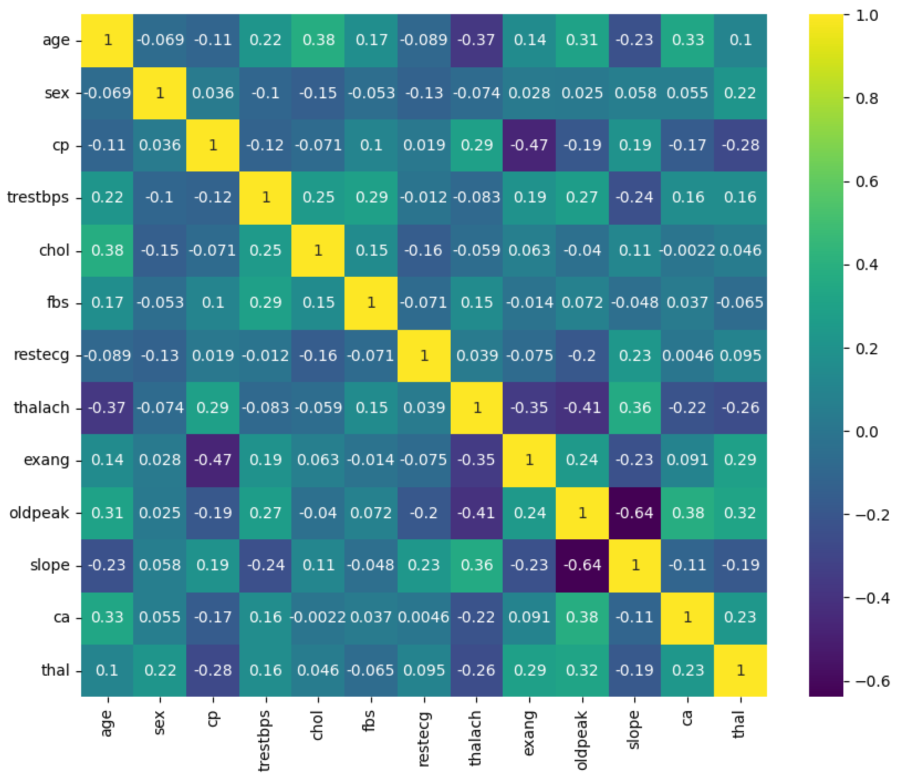
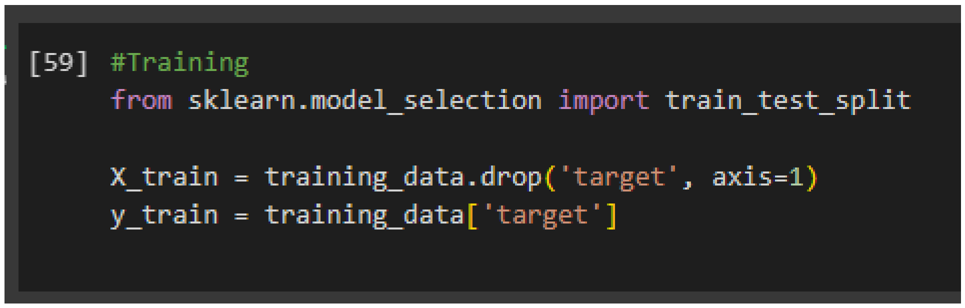


Figure 8: correlation matrix on testing dataset

Figure 5 to Figure 8 is the visualization for both datasets. Figure 5 and Figure 6 are the histograms that shows the distribution of each feature within training (Figure 5) and testing dataset (Figure 6). Figure 7 and Figure 8 is the heatmap that shows the correlation between each feature in training (Figure 7) and testing dataset (Figure 8).

Separate the Input and Target Features of the Data

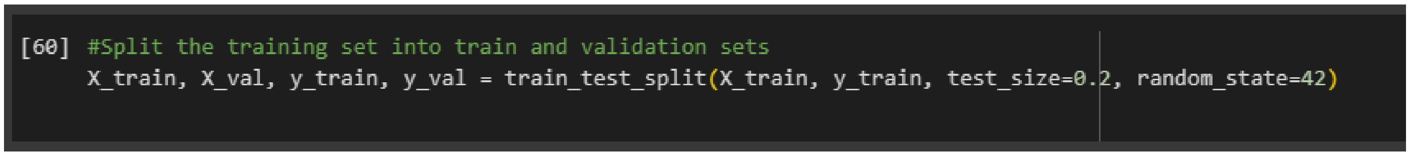


```
[59] #Training
     from sklearn.model_selection import train_test_split

     X_train = training_data.drop('target', axis=1)
     y_train = training_data['target']
```

Figure 9: separate target feature

Figure 9 shows the separation of the target feature of the data from the training dataset. This feature labeled as 'target' shows the actual case whether the patient has heart attack or not. Also, this feature does not exist in the testing dataset.

Split the data into Train and Test Sets

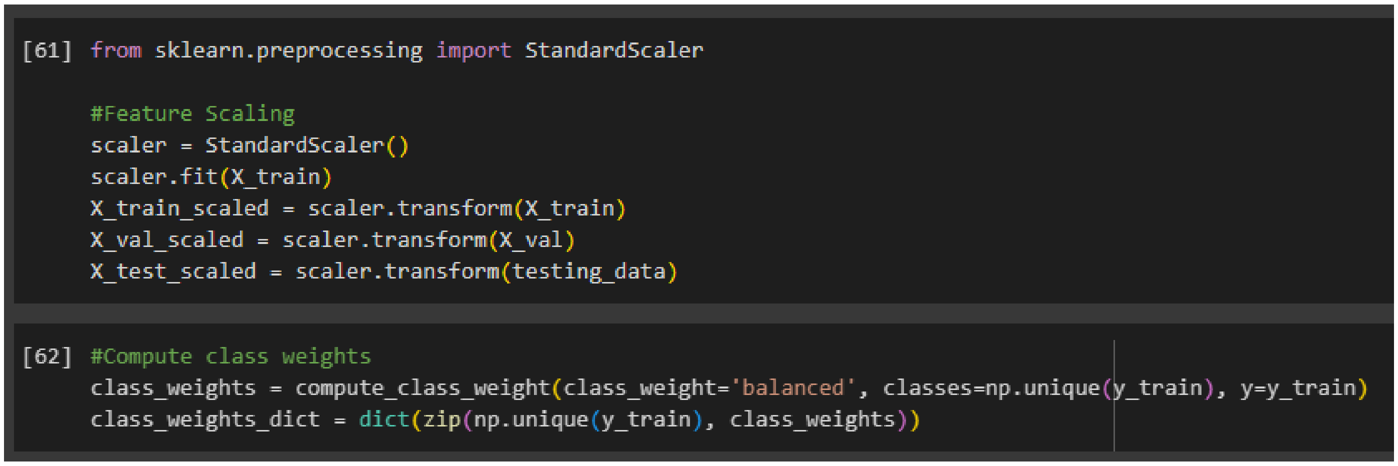


```
[60] #Split the training set into train and validation sets
     X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

Figure 10: splitting dataset

Figure 10 shows the splitting of the datasets. For this project, the training data is split on a ratio of 80:20. 80% will be training data and 20% will be validation data. The testing dataset will not be split.

Scaling data

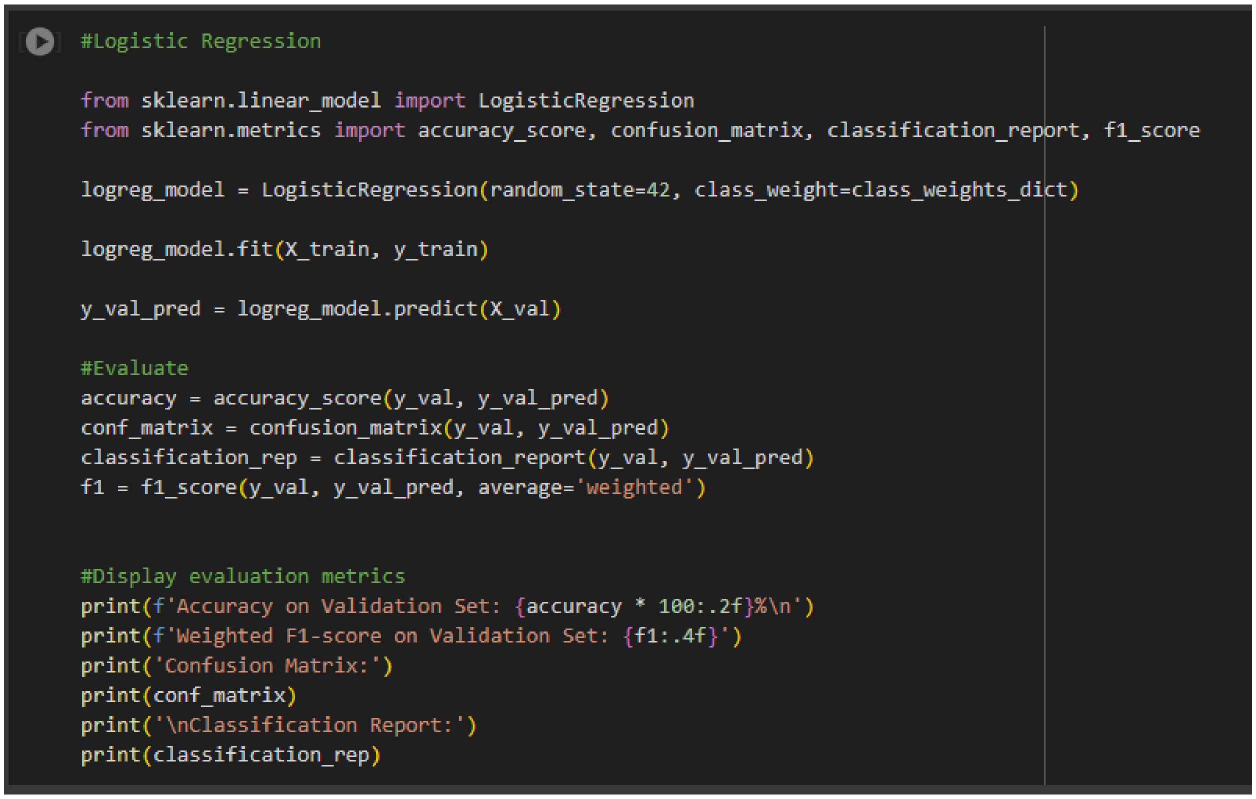


```
[61] from sklearn.preprocessing import StandardScaler

     #Feature Scaling
     scaler = StandardScaler()
     scaler.fit(X_train)
     X_train_scaled = scaler.transform(X_train)
     X_val_scaled = scaler.transform(X_val)
     X_test_scaled = scaler.transform(testing_data)

[62] #Compute class weights
     class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y_train), y=y_train)
     class_weights_dict = dict(zip(np.unique(y_train), class_weights))
```

Figure 11: Feature Scaling and compute class weights

Figure 11: shows the employment of feature scaling and the computation of class weights, which is to balance the classes by giving more weightage on the minor class to avoid biases.

Applying machine learning model on train set

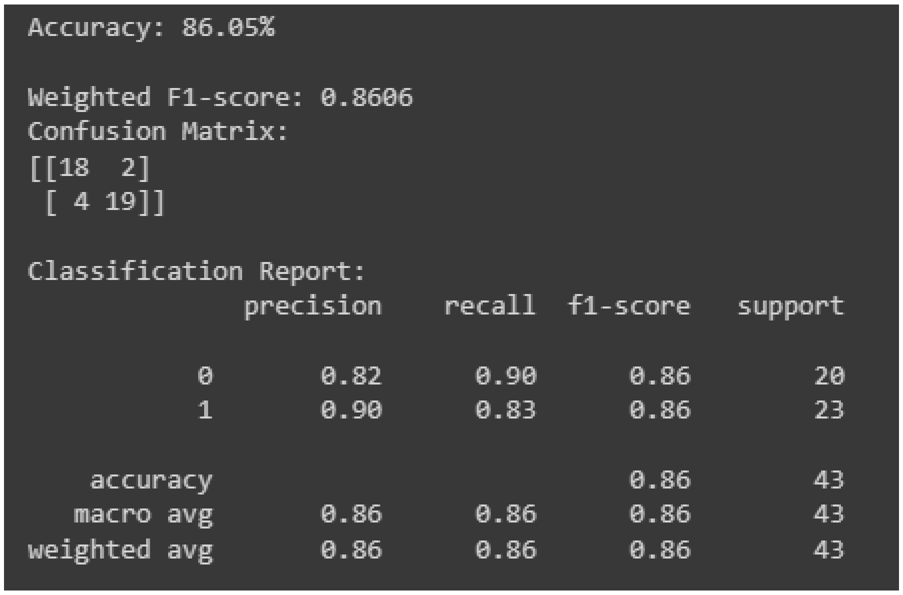


```
#Logistic Regression

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score

logreg_model = LogisticRegression(random_state=42, class_weight=class_weights_dict)

logreg_model.fit(X_train, y_train)

y_val_pred = logreg_model.predict(X_val)

#Evaluate
accuracy = accuracy_score(y_val, y_val_pred)
conf_matrix = confusion_matrix(y_val, y_val_pred)
classification_rep = classification_report(y_val, y_val_pred)
f1 = f1_score(y_val, y_val_pred, average='weighted')


#Display evaluation metrics
print(f'Accuracy on Validation Set: {accuracy * 100:.2f}%\n')
print(f'Weighted F1-score on Validation Set: {f1:.4f}')
print('Confusion Matrix:')
print(conf_matrix)
print('\nClassification Report:')
print(classification_rep)
```

Figure 12: apply logistic regression

Figure 12 shows the application of logistic regression as chosen classification model. As this project is to determine if heart disease is present and risk assessment (binary predictions, 0 or 1), logistic regression would be a suitable model.

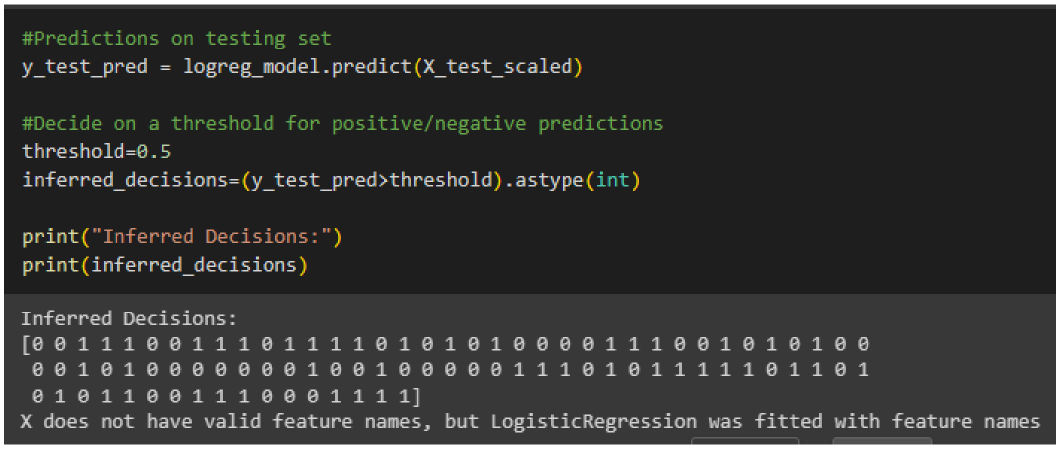
Evaluation using F1-score



```
Accuracy: 86.05%

Weighted F1-score: 0.8606
Confusion Matrix:
[[18  2]
 [ 4 19]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.90      0.86        20
           1       0.90      0.83      0.86        23

    accuracy                           0.86        43
   macro avg       0.86      0.86      0.86        43
weighted avg       0.86      0.86      0.86        43
```

Figure 13: classification report

Figure 13 above indicates the classification report after training the logistic regression model. This classification report shows the details of evaluation metrics such as accuracy, precision, recall, F1-score, and support. This model achieved an accuracy of 86.05% and the F1-score is 0.8606.

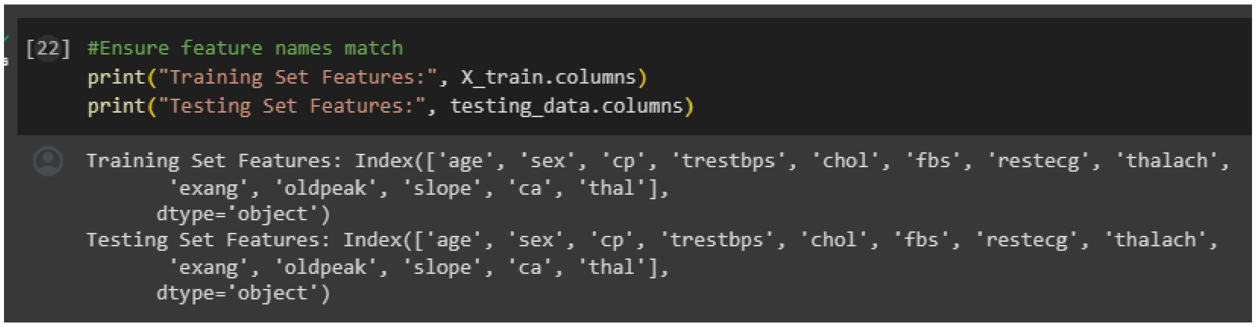


```
#Predictions on testing set
y_test_pred = logreg_model.predict(X_test_scaled)

#Decide on a threshold for positive/negative predictions
threshold=0.5
inferred_decisions=(y_test_pred>threshold).astype(int)

print("Inferred Decisions:")
print(inferred_decisions)
```

```
Inferred Decisions:
[0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 0 0
 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1
 0 1 0 1 1 0 0 1 1 1 0 0 0 1 1 1 1]
X does not have valid feature names, but LogisticRegression was fitted with feature names
```

Figure 14: predictions on testing set

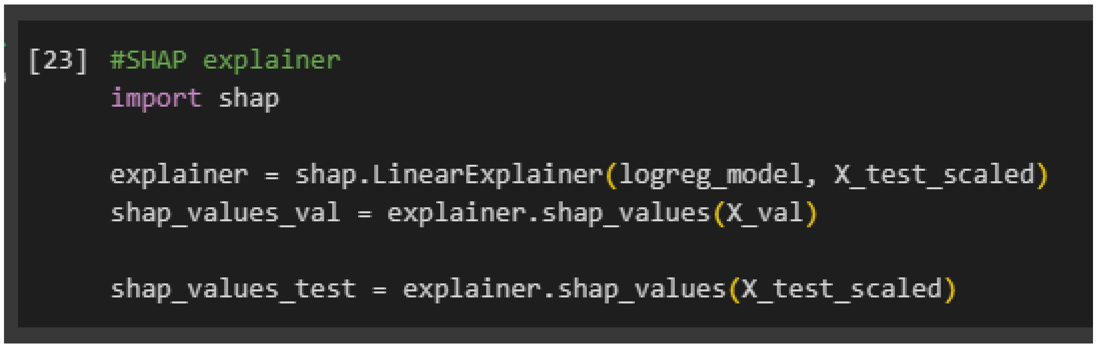
Figure 14 shows the predictions made on the testing set using the trained model from the training and validation dataset. The results shows the results of each patients. There was a warning message which says “X does not have valid feature names, but LogisticRegression was fitted with feature names.” This warning message is about possible feature names mismatch. Hence, Figure 15 shows that I displayed the feature names from both training and testing dataset. And the outcome did not show any feature names mismatch.



```
[22] #Ensure feature names match
     print("Training Set Features:", X_train.columns)
     print("Testing Set Features:", testing_data.columns)

     Training Set Features: Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
            'exang', 'oldpeak', 'slope', 'ca', 'thal'],
           dtype='object')
     Testing Set Features: Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
            'exang', 'oldpeak', 'slope', 'ca', 'thal'],
           dtype='object')
```

Figure 15: checking feature names

Using SHAP explainer to derive SHAP Values for the Machine Learning model.



```
[23] #SHAP explainer
     import shap

     explainer = shap.LinearExplainer(logreg_model, X_test_scaled)
     shap_values_val = explainer.shap_values(X_val)

     shap_values_test = explainer.shap_values(X_test_scaled)
```

Figure 16: applying SHAP model

Figure 16 shows the application of SHAP model as XAI model. With these calculated SHAP values, it will be used to interpret the predictions through global and local explanations.

SHAP force plot for the first row of test data.

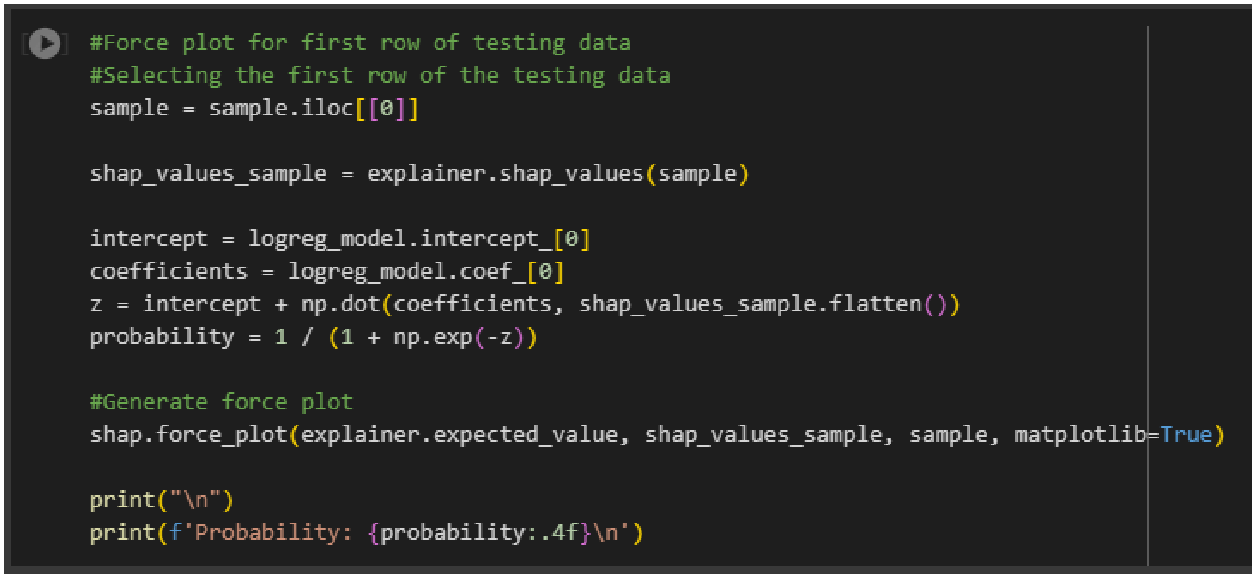

```
#Force plot for first row of testing data
#Selecting the first row of the testing data
sample = sample.iloc[[0]]

shap_values_sample = explainer.shap_values(sample)

intercept = logreg_model.intercept_[0]
coefficients = logreg_model.coef_[0]
z = intercept + np.dot(coefficients, shap_values_sample.flatten())
probability = 1 / (1 + np.exp(-z))

#Generate force plot
shap.force_plot(explainer.expected_value, shap_values_sample, sample, matplotlib=True)

print("\n")
print(f'Probability: {probability:.4f}\n')
```

Figure 17: Force plot for first row of test data, with probability

Figure 17 shows the coding where I develop the force plot with SHAP. I also included a feature to display the probability of the patient to develop heart disease. Refer Figure 18 and Figure 19.

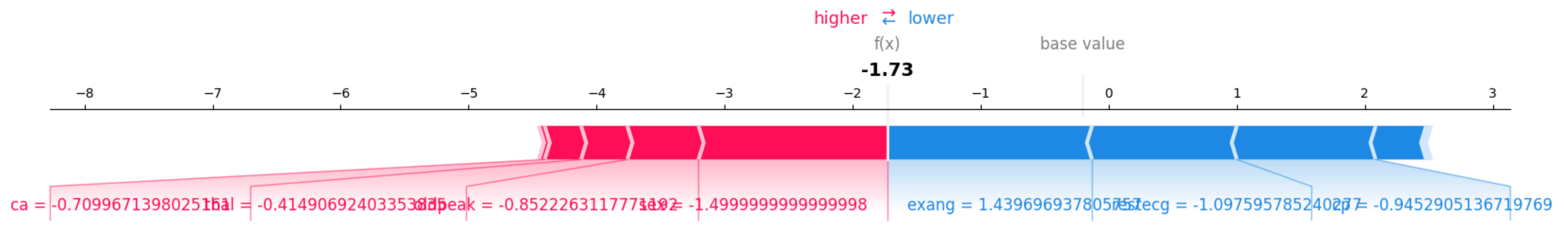
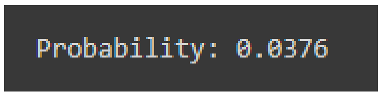


Figure 18: force plot

Figure 18 indicates the force plot for the first row of testing data. This interpretation from this plot is local explanation since I only take the first row of the testing data, which means only one patient. The bold black number in the middle shows the SHAP value. In this case, the lower the SHAP value, the lower the risk of having heart disease. The blue lines shows what are the features that contribute to lower the risk of heart disease, while the pink line are features that raises the risk of heart disease. For this patient, the feature that contributes the most towards lowering the risk of heart disease is the ‘exang’ which stands for exercise-induced angina. Exercise-induced angina is a type of chest pain during physical activity. Having less angina is better for our body’s health. On the other hand, the feature that contributes most towards increasing the risk of heart disease is ‘sex’ which stands for gender. Referring to Figure 19 shows the probability of this patient develop heart disease which is 0.0376 or 3.76%.

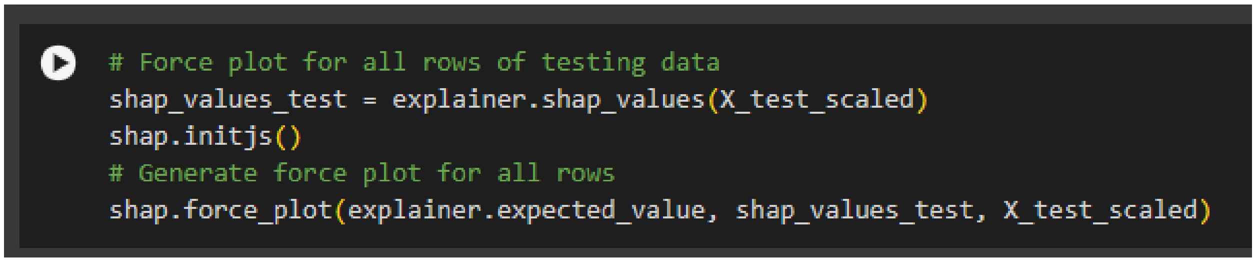


```
Probability: 0.0376
```

Figure 19: probability

Having a very low probability shows that this patient might be a very healthy man. In order to decrease the risk, doctors can advise this patient to maintain a healthy body by keeping fit such as taking sports and have enough rest to avoid the increasing the exercise-induced angina.

SHAP force plot for all the rows in the data.



```
# Force plot for all rows of testing data
shap_values_test = explainer.shap_values(X_test_scaled)
shap.initjs()
# Generate force plot for all rows
shap.force_plot(explainer.expected_value, shap_values_test, X_test_scaled)
```

Figure 20: force plot for all the rows

Figure 20 is the coding to display force plot for all the rows in the testing data. To display all the rows, the model loops until the end of the rows.

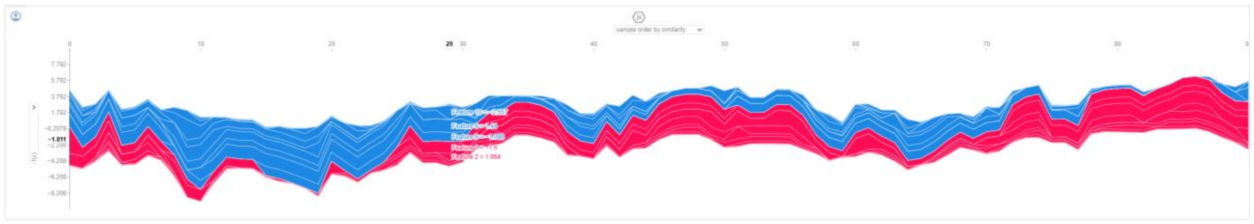
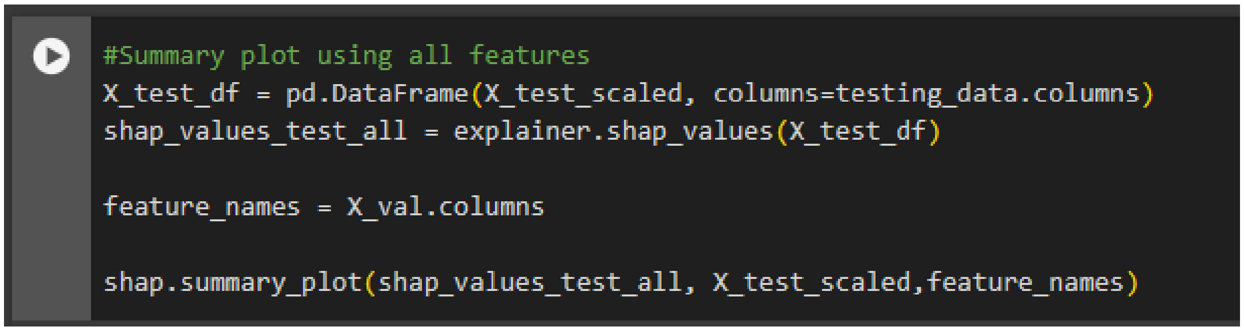


Figure 21: force plots

Figure 21 above shows the force plots from all rows. Every single waves shows every single factors in the dataset on what contributes to the risk of heart disease. *The higher the waves means the higher predicted risk of getting heart disease (in average). The red area represents what or which of the factors increases the overall risk and the blue area represents the factors that decreases the overall risk of heart disease. y-axis is the SHAP value and x-axis is the order by rows. As we hover through the plot, the force plot will show the features with SHAP values on a particular row, based on location of the mouse cursor align with x-axis.

<u>SHAP summary plot using all features in the data.</u>



```
#Summary plot using all features
X_test_df = pd.DataFrame(X_test_scaled, columns=testing_data.columns)
shap_values_test_all = explainer.shap_values(X_test_df)

feature_names = X_val.columns

shap.summary_plot(shap_values_test_all, X_test_scaled,feature_names)
```

Figure 22: develop summary plot for all features

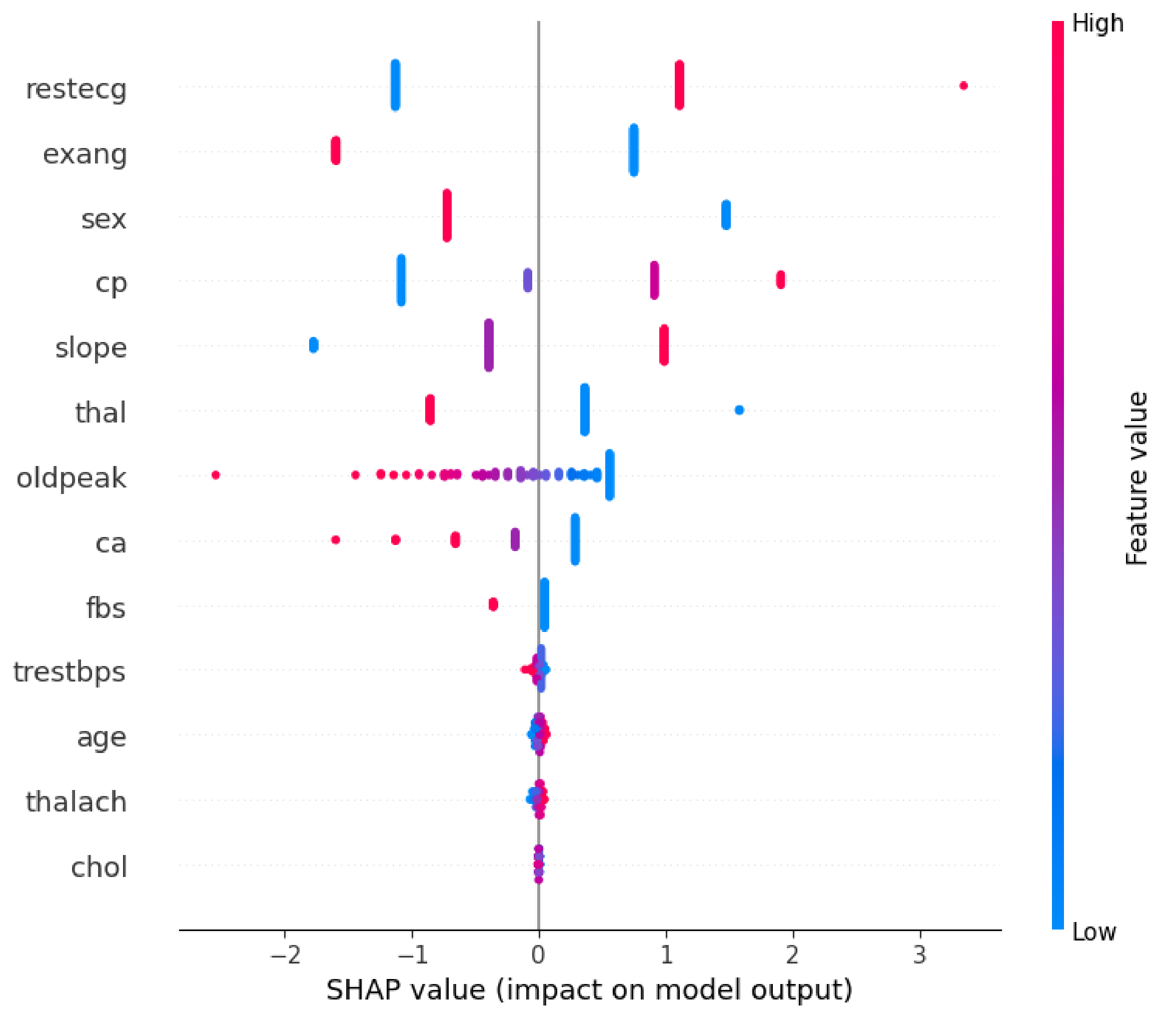


Figure 23: summary plot

Figure 22 shows the coding for develop and display the summary plot using SHAP for all the features, as shown in Figure 23 is the summary plot. This summary plot consists of all features from all the rows, which suggests global explanation.

The components in this summary plot includes:

- y-axis: List of features, ranked by importance in descending order
- x-axis: SHAP values, indicates the impact on the prediction
- Dots: each of the dots represents feature contributions
- Feature value: color that indicates feature value

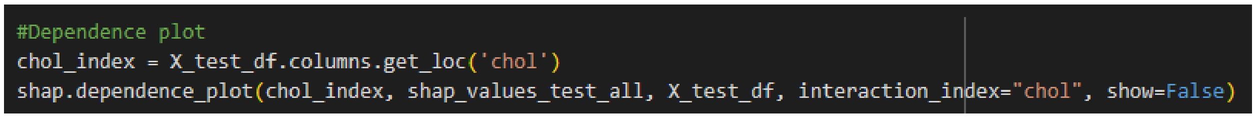
According to the summary plot, top three features with the highest impact on the predictions are:

- restecg
- exang
- sex

Some details we can see from these three features from the plot are:

- They are ranked the highest in the summary plot, which arranged in descending order.
- The dots on the features have significant SHAP values as they are located further from zero, either less than or greater than zero.
    - Zero (0) SHAP values means less or no contribution towards the model's predictions.
    - Positive SHAP values (to the right) contributes to lower risk of heart disease.
    - Negative SHAP values (to the left) contributes to higher risk of heart disease.
- The dots with wider span means it has more impacts on multiple rows.
- The dots on the three features also have wider span and red in colour, which means they pose a high value in model's prediction.

SHAP dependence plot to show the effect of 'chol' across the whole dataset.



```
#Dependence plot
chol_index = X_test_df.columns.get_loc('chol')
shap.dependence_plot(chol_index, shap_values_test_all, X_test_df, interaction_index="chol", show=False)
```

Figure 24: develop dependence plot to show the effect of 'chol' feature across the dataset.

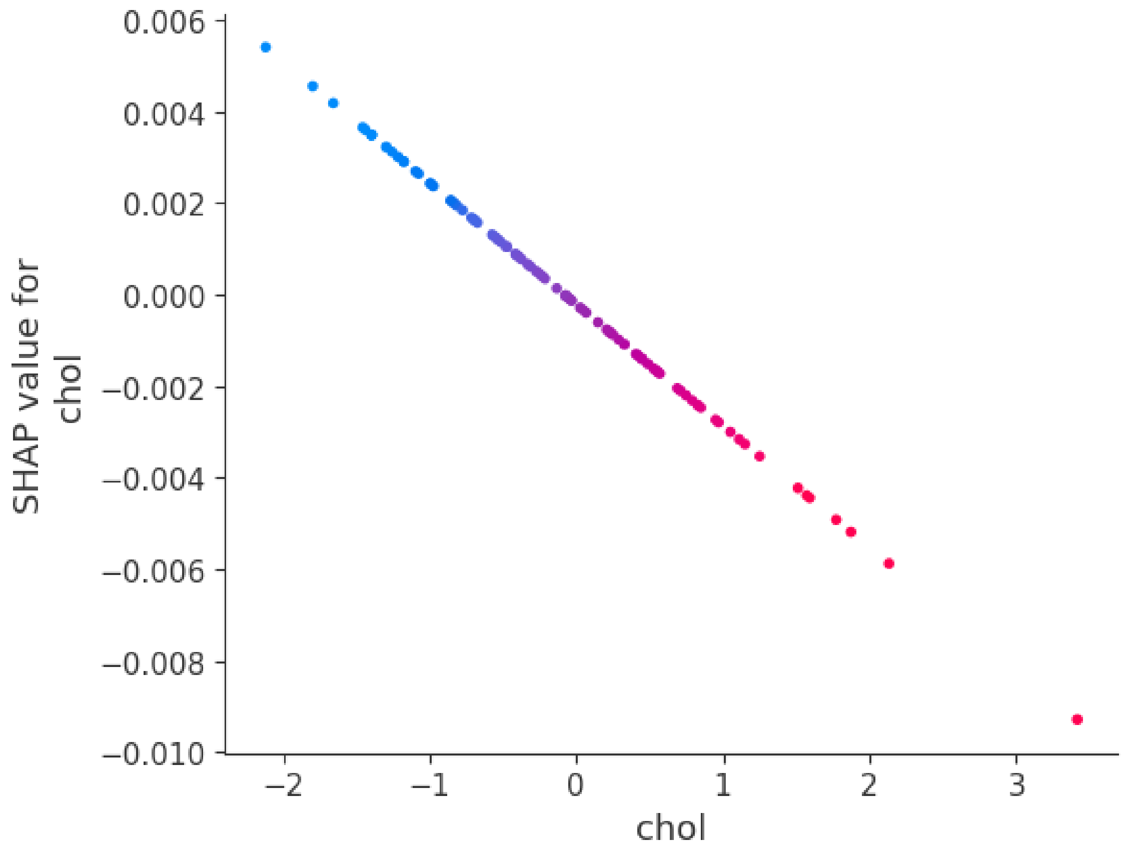


Figure 25: dependence plot

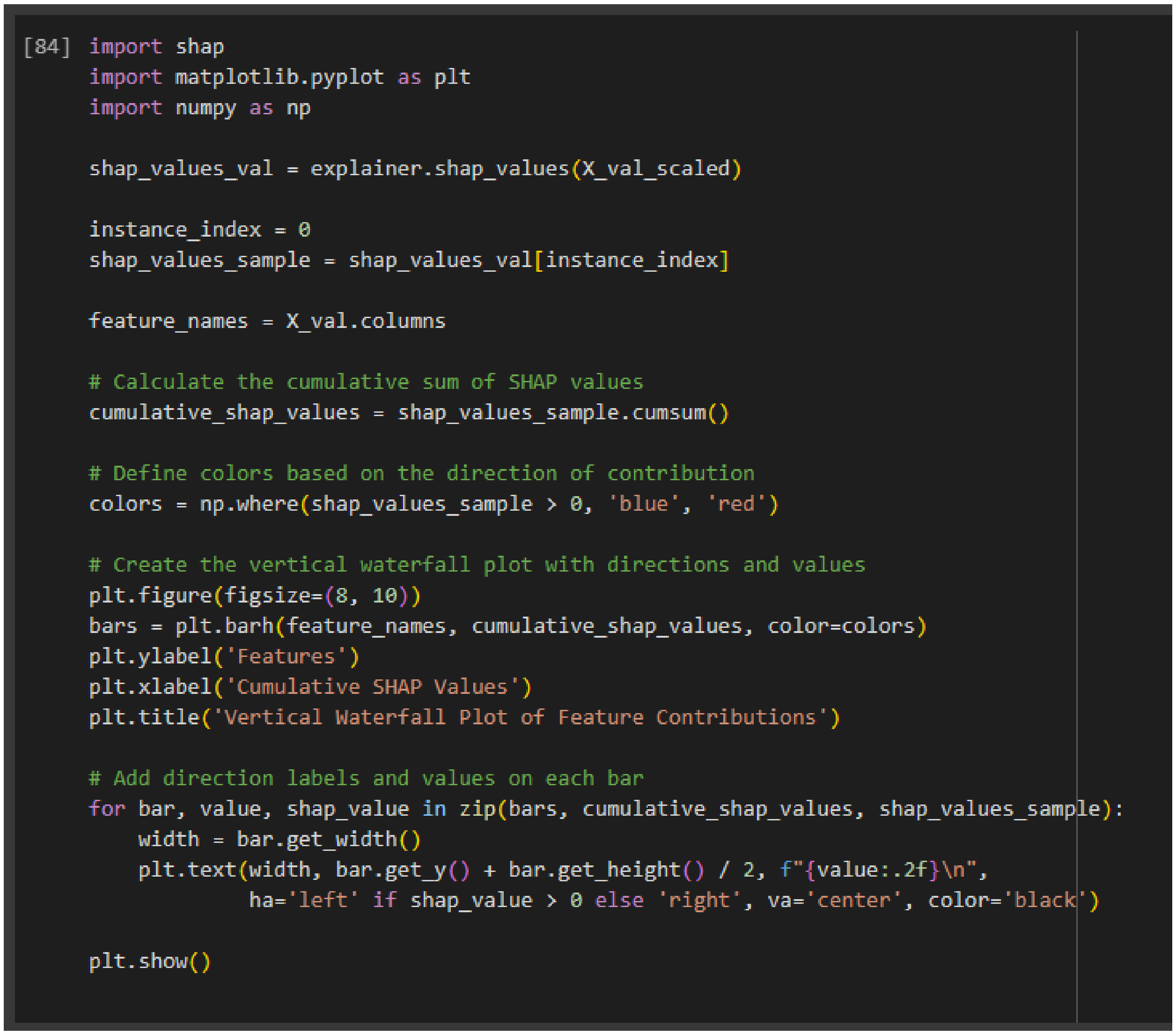
Figure 24 is the code for developing the dependence plot, which shows the effect of ‘chol’ feature across the dataset as shown in Figure 25. In Figure 25, we can see that:

- The higher chol value leads to lower SHAP value for chol. Which means it leads to higher risk of heart disease.
- The dots that incline towards negative SHAP values are colored pink, which means it poses a higher feature value in model’s prediction.
- However, referring to summary plot (Figure 23), chol ranked last in the plot, which means it has the least overall contribution towards model’s prediction.

This dependence plot still shows that chol feature has minimal impact towards heart disease risk. Hence to mitigate heart disease risk based on this feature, doctors can advise patients to decrease cholesterol intake. Some suggested steps include consuming more fiber-rich foods, limit processed foods, regular exercise and decrease or quit smoking.

Explorations and more visualizations with SHAP

- Waterfall plot



```
[84] import shap
     import matplotlib.pyplot as plt
     import numpy as np

     shap_values_val = explainer.shap_values(X_val_scaled)

     instance_index = 0
     shap_values_sample = shap_values_val[instance_index]

     feature_names = X_val.columns

     # Calculate the cumulative sum of SHAP values
     cumulative_shap_values = shap_values_sample.cumsum()

     # Define colors based on the direction of contribution
     colors = np.where(shap_values_sample > 0, 'blue', 'red')

     # Create the vertical waterfall plot with directions and values
     plt.figure(figsize=(8, 10))
     bars = plt.barh(feature_names, cumulative_shap_values, color=colors)
     plt.ylabel('Features')
     plt.xlabel('Cumulative SHAP Values')
     plt.title('Vertical Waterfall Plot of Feature Contributions')

     # Add direction labels and values on each bar
     for bar, value, shap_value in zip(bars, cumulative_shap_values, shap_values_sample):
         width = bar.get_width()
         plt.text(width, bar.get_y() + bar.get_height() / 2, f"{value:.2f}\n",
                  ha='left' if shap_value > 0 else 'right', va='center', color='black')

     plt.show()
```

Figure 26: develop waterfall plot

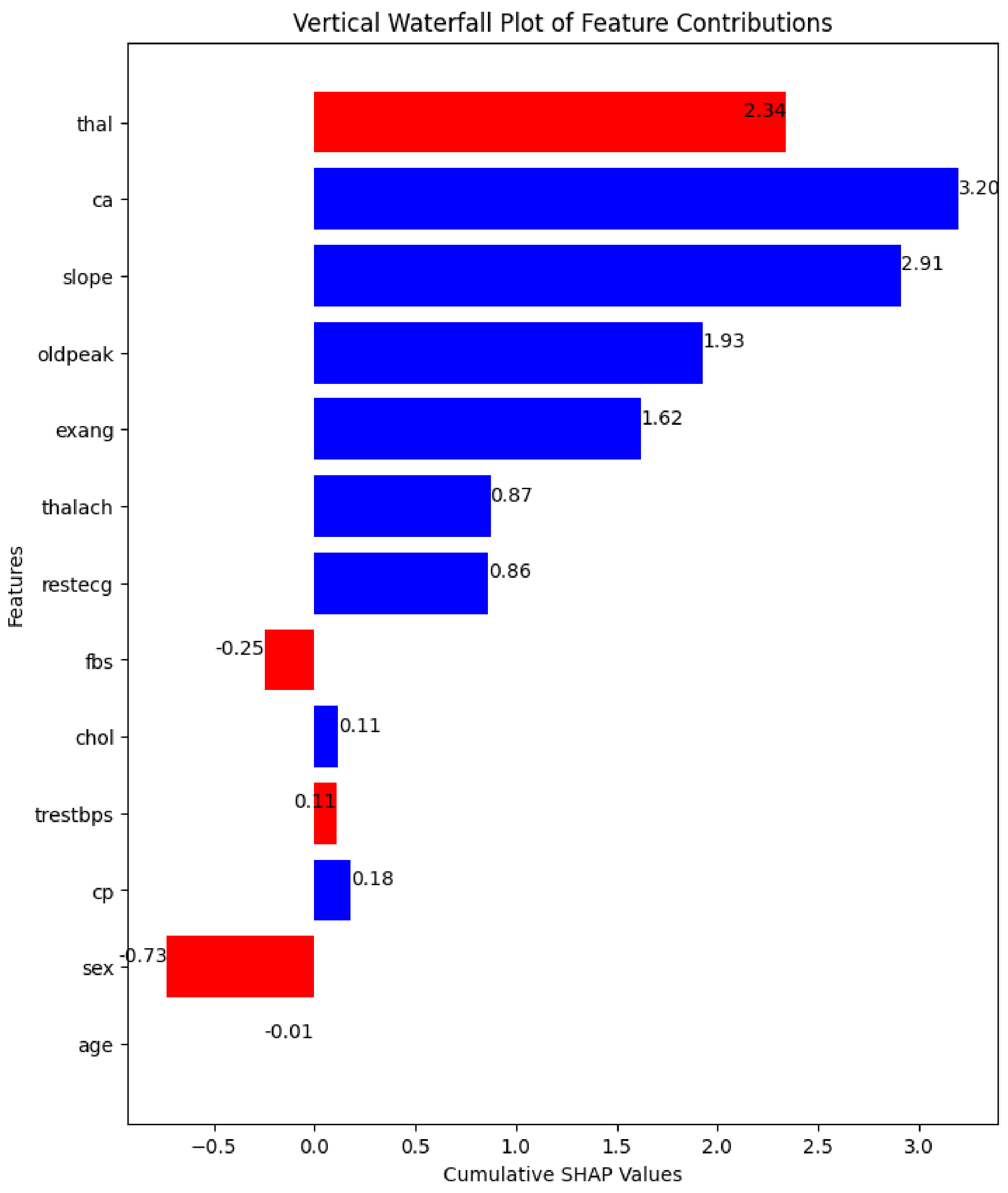


Figure 27: Waterfall plot

Figure 26 is the code to develop the waterfall plot as shown in Figure 27. Based on the code, we considered instance_index=0 which means that this waterfall plot are generated for the first row in the testing dataset, which suggests local explanation.

The characteristics of this waterfall plot are as below:

- x-axis = Cumulative SHAP values calculated on each features
- y-axis = features in the testing dataset
- Colors = features that contribute to increase the heart disease risk indicates in red, and features that contribute to decrease the heart disease risk indicates in blue.
- Bar lengths = corresponds to the cumulative SHAP values for each features
- The values shown on each bar is the cumulated SHAP values for the corresponding feature.

Analysis:

- There are 8 features that contribute towards decreasing heart disease risk, which are:
  - ca
  - slope
  - oldpeak
  - exang
  - thalach
  - restecg
  - chol
  - cp
- There are 4 features that contribute towards increasing heart disease risk, include:
  - thal
  - fbs
  - trestbps
  - sex
- Since 'age' features bar cannot be seen, which means that this feature contributes least to none to the model's predictions.

As data scientists, we can show the healthcare experts that these red coded features are the ones they can plan for their patients' personalized treatment/mitigation plan.

As healthcare experts, considering some of the features that can be mitigated, we can plan on any treatment plan for the patients to manage heart disease risk such as personalized prescriptions on medications.