Task 1:

- Import the libraries

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split ,RandomizedSearchCV , GridSearchCV
from sklearn.metrics import classification_report  , f1_score
```

✓ 6.6s

Import the libraries that will be used later for executing certain functions.

- Load the data

```
# Load Data
data = pd.read_csv('../code/Training_set_heart.csv')
test_set = pd.read_csv('../code/Testing_set_heart.csv')

data.head()
```

✓ 0.0s

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

Load the dataset from the local directory for further processing.

- Perform basic EDA

```
data.info()

data.describe()
```
✓ 0.0s

```
<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
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000
mean	54.561321	0.688679	0.915094	132.127358	247.830189	0.165094	0.537736	148.995283	0.330189	1.008491	1.448113	0.783019	2.320755	0.542453
std	9.493376	0.464130	1.008193	17.440700	53.199877	0.372144	0.527437	23.332645	0.471394	1.141681	0.601850	1.066551	0.646495	0.499374
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%	47.000000	0.000000	0.000000	120.000000	211.750000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	57.000000	1.000000	1.000000	130.000000	240.500000	0.000000	1.000000	151.500000	0.000000	0.650000	2.000000	0.000000	2.000000	1.000000
75%	61.250000	1.000000	2.000000	140.000000	277.000000	0.000000	1.000000	165.250000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	192.000000	564.000000	1.000000	2.000000	202.000000	1.000000	5.600000	2.000000	4.000000	3.000000	1.000000

Gain insights from this dataset. From the above, we can see that the training dataset has 14 columns and 212 rows of data. All columns in the dataset are considered numeric and the descriptive statistics for this dataset are also provided.

```
# Checking whether any null value
data.isnull().sum()
```
✓ 0.0s

```
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
```

Check the dataset, whether it is any null value or NaN (Not a Number) in the dataset. It showed that no missing values found.

```
# Examine to the Training_set_heart.csv and Testing_set_heart.csv,
# there exists some columns having invalid value for some rows.
# Drop the invalid data

# Drop the rows of having invalid ca value
data = data[data['ca'] < 4]
test_set = test_set[test_set['ca'] < 4]

# Drop the rows of having invalid thal value
data = data[data['thal'] > 0]
test_set = test_set[test_set['thal'] > 0]

print('Shape of the data is ', data.shape)
```
✓ 0.0s

```
Shape of the data is  (206, 14)
```

```
print('Shape of the data is ', test_set.shape)
test_set.head()
```
✓ 0.0s

```
Shape of the data is  (90, 13)
```

When examining the provided training dataset and testing datasets manually, I found that there exists some invalid values in certain columns, so I specifically handle those invalid rows of data by dropping them from the pandas dataframe (data).

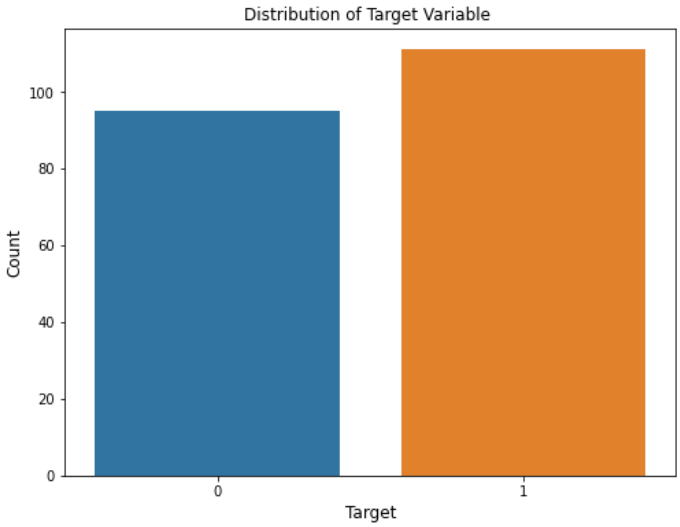
```
# Identify the categorical columns of independent variables (dependent variable is 'target')
# sex, chest pain type, fasting blood sugar > 120, resting electrocardiographic results, exercise-induced angina, thalassemia status
categorical_features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'thal']

# Replace the values in the categorical columns to their medical meaning for easier interpretation
data['sex'] = data.sex.replace({0:'male', 1:'female'})
data['cp'] = data.cp.replace({0:'typical angina', 1:'atypical angina', 2:'non-angina', 3:'asymptomatic'})
data['fbs'] = data.fbs.replace({0:'lower than 120 mg/dl', 1:'higher than 120 mg/dl'})
data['restecg'] = data.restecg.replace({0:'normal', 1:'stt abnormality', 2:'lv hypertrophy'})
data['exang'] = data.exang.replace({0:'True', 1:'False'})
data['thal'] = data.thal.replace({1:'normal', 2:'fixed defect', 3:'reversible defect'})

test_set['sex'] = test_set.sex.replace({0:'male', 1:'female'})
test_set['cp'] = test_set.cp.replace({0:'typical angina', 1:'atypical angina', 2:'non-angina', 3:'asymptomatic'})
test_set['fbs'] = test_set.fbs.replace({0:'lower than 120 mg/dl', 1:'higher than 120 mg/dl'})
test_set['restecg'] = test_set.restecg.replace({0:'normal', 1:'stt abnormality', 2:'lv hypertrophy'})
test_set['exang'] = test_set.exang.replace({0:'True', 1:'False'})
test_set['thal'] = test_set.thal.replace({1:'normal', 2:'fixed defect', 3:'reversible defect'})
```
✓ 0.0s

In order to make the interpretation of the XAI method results later easier, I choose to replace the values in the categorical columns with their medical meanings. It is more meaningful and understandable, instead of using only the numerical representations. Based on the questions, we can identify a total of 6 categorical input features for both datasets. Besides, an assumption has been made that the numbering of categorical features in the dataset follows the sequence they stated in the questions. The attributes that do not have specified as arrays will be considered as numerical features.

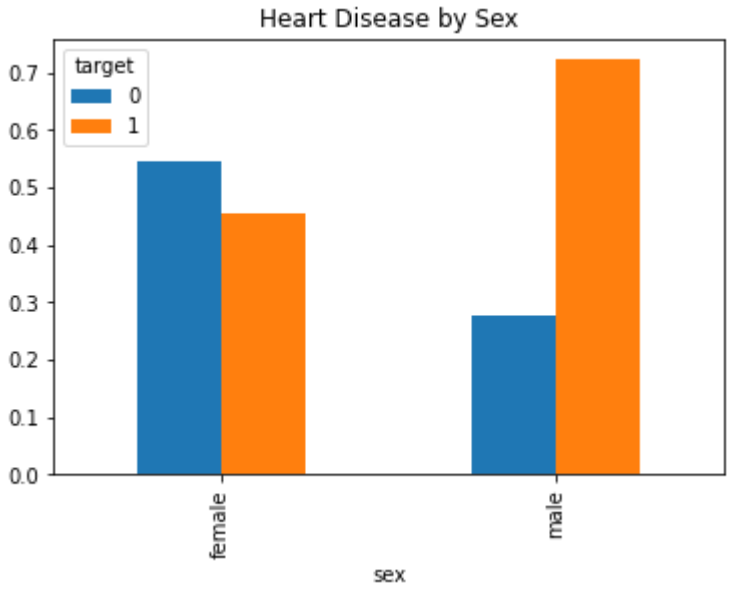
```
data.info()
```
✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 206 entries, 0 to 211
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       206 non-null    int64
 1   sex       206 non-null    object
 2   cp        206 non-null    object
 3   trestbps  206 non-null    int64
 4   chol      206 non-null    int64
 5   fbs       206 non-null    object
 6   restecg   206 non-null    object
 7   thalach   206 non-null    int64
 8   exang     206 non-null    object
 9   oldpeak   206 non-null    float64
 10  slope     206 non-null    int64
 11  ca        206 non-null    int64
 12  thal      206 non-null    object
 13  target    206 non-null    int64
dtypes: float64(1), int64(7), object(6)
memory usage: 24.1+ KB
```

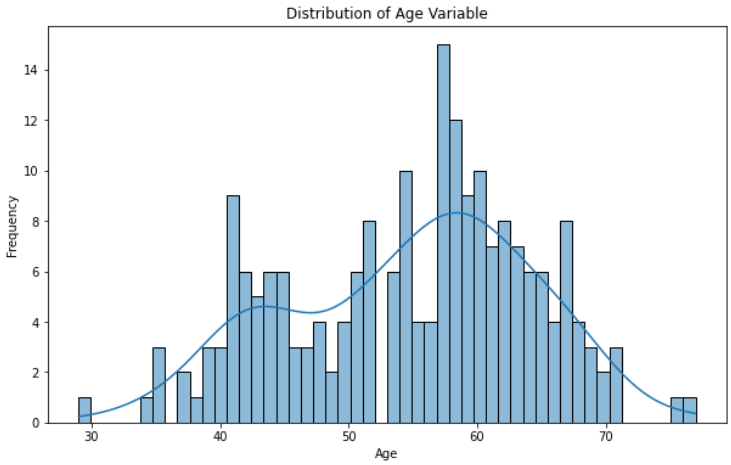
The characteristics of the training dataset after the replacements. We can see that there are 6 categorical features and a total of 8 numerical features in this training dataset.



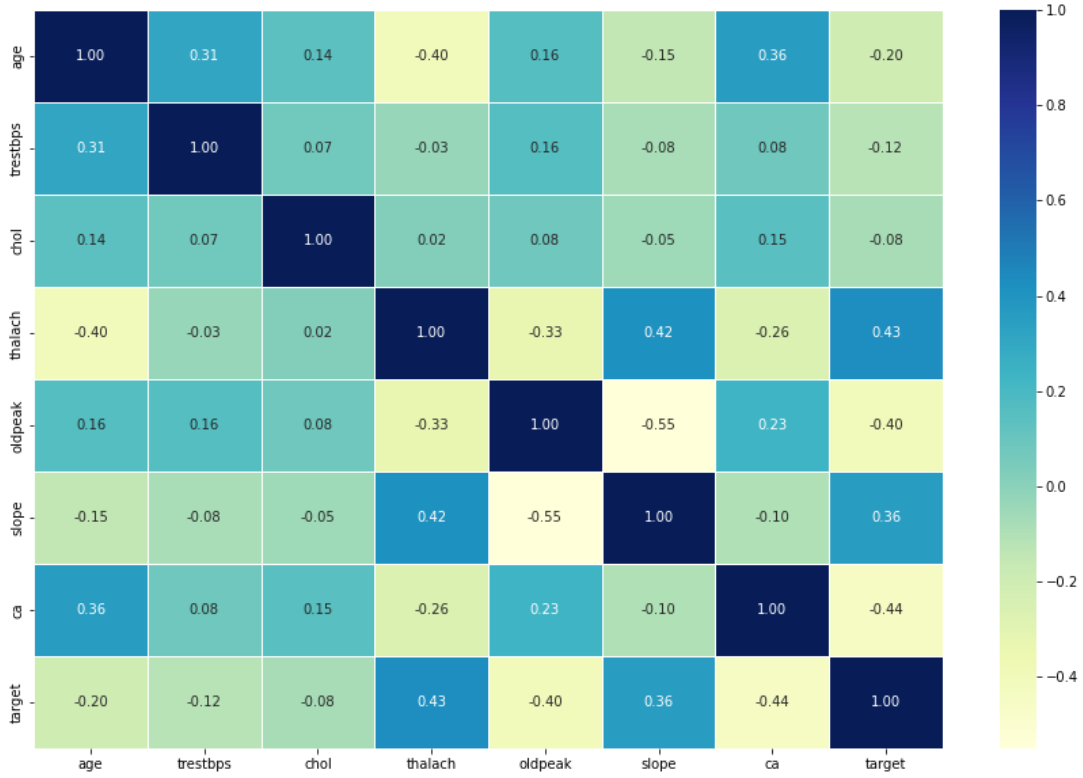
To explore the data, several graphs have been plotted. From the graph above, we can observe that this dataset is fairly balanced, as the number of samples between having heart disease and no heart disease does not significantly differ. Therefore, there is no need to address the issue of data imbalance in this case.



From the graph above, we can observe that the males have higher ratio of having heart disease compared to females, and the data showing samples of having heart disease is mostly from males.



Above is the distribution of the age variable in the training dataset, we can see that the majority of samples in the dataset fall within the age range of 50 to 70 years old.



We can get the correlation coefficients among the variables from the graph above.

- Separate input and target features of the data

```
# Seperate the input and target features of the data
y = data['target']
X = data.drop('target', axis=1)
```

✓ 0.0s

Since the objective of this project is to determine if heart disease is present or not, so the target variable will be 'target', which contains the ground truth for each row of data.

- Split the data into train set and validation set

```
#Split data into Train and Validation sets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20,random_state=42)
```

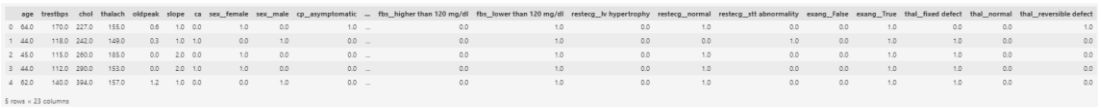
✓ 0.0s

The training dataset is divided into train set and validation set for models' evaluation later. The splitting ratio is set to 8:2, where 80% of the training dataset is assigned to train set and the remaining 20% is assigned to the evaluation set.

- Extra preprocessing

```
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

numerical_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'slope', 'ca' ]

# Encoding Categorical Features using Preprocessor
preprocessor = ColumnTransformer([("numerical", "passthrough", numerical_features),
                                  ("categorical", OneHotEncoder(sparse=False, handle_unknown="ignore"), categorical_features)])

# Train preprocessor
preprocessor.fit(X_train)

# Get the list of categories generated by the process
ohe_categories = preprocessor.named_transformers_["categorical"].categories_

# Create nice names for our one hot encoded features
new_ohe_features = [f"{col}__{val}" for col, vals in zip(categorical_features, ohe_categories) for val in vals]

# Create a new list with all names of features
all_features = numerical_features + new_ohe_features

# Save processed data
X_train = pd.DataFrame(preprocessor.transform(X_train), columns=all_features)
X_test = pd.DataFrame(preprocessor.transform(X_test), columns=all_features)
test_set = pd.DataFrame(preprocessor.transform(test_set), columns=all_features)

X_train.head()
```

✓ 0.0s



	age	trestbps	chol	thalach	oldpeak	slope	ca	sex_female	sex_male	cp_asymptomatic	...	fbs_higher than 120 mg/dl	fbs_lower than 120 mg/dl	restecg_lv hypertrophy	restecg_normal	restecg_stt abnormality	exang_False	exang_True	thal_fixed defect	thal_normal	thal_reversible defect
0	64.0	170.0	227.0	155.0	0.6	1.0	0.0	1.0	0.0	1.0	...	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
1	44.0	118.0	242.0	149.0	0.3	1.0	1.0	0.0	1.0	0.0	...	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0
2	45.0	115.0	260.0	185.0	0.0	2.0	0.0	1.0	0.0	0.0	...	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0
3	44.0	112.0	290.0	153.0	0.0	2.0	1.0	1.0	0.0	0.0	...	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0
4	62.0	140.0	394.0	157.0	1.2	1.0	0.0	0.0	1.0	0.0	...	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0

5 rows × 23 columns

The default dataset used label encoding to deal with categorical features. However, the model may misunderstand the data is in having some inherent order. Therefore, the better solution is to use one-hot encoding to encode the categorical features.

Task 2:

- Build ML model on train set

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score, accuracy_score, confusion_matrix

# Select and compare a sets of classification algorithms for finding algorithm that have best F1 score

# Model initialization
lr_model = LogisticRegression()
dt_model = DecisionTreeClassifier()
rf_model = RandomForestClassifier()
gb_model = GradientBoostingClassifier()
svc_model = SVC()

# Fitting models to train data
lr_model.fit(X_train, y_train)
dt_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)
gb_model.fit(X_train, y_train)
svc_model.fit(X_train, y_train)

# Make predictions on test data
lr_pred = lr_model.predict(X_test)
dt_pred = dt_model.predict(X_test)
rf_pred = rf_model.predict(X_test)
gb_pred = gb_model.predict(X_test)
svc_pred = svc_model.predict(X_test)
```

✓ 0.8s

Since this dataset is about classification, I have selected 5 commonly used models for handling classification problems, including Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Gradient Boosting Classifier and the Support Vector Classifier (SVC). The models are trained with their default parameters, and the predictions of the models on the validation data are collected.

- Evaluate the model using F1 Score

```
# Evaluate the model using F1 Score

print('Accuracy of Logistic Regression: ', accuracy_score(lr_pred, y_test))
print('Accuracy of Decision Tree Classifier: ', accuracy_score(dt_pred, y_test))
print('Accuracy of Random Forest Classifier: ', accuracy_score(rf_pred, y_test))
print('Accuracy of Gradient Boosting Classifier: ', accuracy_score(gb_pred, y_test))
print('Accuracy of Support Vector Classification: ', accuracy_score(svc_pred, y_test))

print('\nF1 Score of Logistic Regression: ', f1_score(lr_pred, y_test))
print('F1 Score of Decision Tree Classifier: ', f1_score(dt_pred, y_test))
print('F1 Score of Random Forest Classifier: ', f1_score(rf_pred, y_test))
print('F1 Score of Gradient Boosting Classifier: ', f1_score(gb_pred, y_test))
print('F1 Score of Support Vector Classification: ', f1_score(svc_pred, y_test))
```

✓ 0.0s

```
Accuracy of Logistic Regression:  0.7619047619047619
Accuracy of Decision Tree Classifier:  0.6904761904761905
Accuracy of Random Forest Classifier:  0.7380952380952381
Accuracy of Gradient Boosting Classifier:  0.7380952380952381
Accuracy of Support Vector Classification:  0.6428571428571429

F1 Score of Logistic Regression:  0.8000000000000002
F1 Score of Decision Tree Classifier:  0.7346938775510204
F1 Score of Random Forest Classifier:  0.7755102040816326
F1 Score of Gradient Boosting Classifier:  0.7659574468085107
F1 Score of Support Vector Classification:  0.7058823529411765
```

```
print(classification_report(y_test, lr_pred))
```

✓ 0.0s

	precision	recall	f1-score	support
0	0.71	0.71	0.71	17
1	0.80	0.80	0.80	25
accuracy			0.76	42
macro avg	0.75	0.75	0.75	42
weighted avg	0.76	0.76	0.76	42

Based on the evaluation results, the Logistic Regression model has the overall good performance in term of accuracy and F1 score. I have generated a classification report for the prediction result of the Logistic Regression model to understandots prediction performance. From the report, we can see that for the prediction of not having heart disease, both precision (percentage of correct positive relative to total positive predictions) and recall (percentage of correct positive predictions relative to total actual positives) are 71%, resulting in an F1 score of 0.71. While, for the prediction of having heart disease, both precision and recall are 80%, resulting in an F1 score of 0.8. The overall F1 score is 0.76. A higher F1 score, closer to 1, indicates a better model for predicting whether someone has heart disease. According to the general rule of thumb for F1 score, a score of 0.76 is considered as acceptable for model. The formula for calculating the F1 score is provided below.

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

TP = number of true positives
FP = number of false positives
FN = number of false negatives

Task 3:

- Use SHAP Explainer to derive SHAP Values for ML model

```
import shap
shap.initjs()

masker = shap.maskers.Independent(data=test_set)

# Use SHAP Explainer to derive SHAP Values for Logistic Regression Model
explainer = shap.Explainer(lr_model, masker=masker)

# Evaluate SHAP values
shap_values = explainer(test_set)

shap_values
```

✓ 3.2s

```
.values =
array([[-3.58785802e-02, -5.28115746e-02, -8.11440046e-05, ...,
         3.77982889e-01,  2.67902063e-03,  3.25506792e-01],
       [-4.04010903e-02, -5.28115746e-02,  3.32690419e-03, ...,
         3.77982889e-01,  2.67902063e-03,  3.25506792e-01],
       [ 2.29140512e-02, -2.05019606e-01, -1.46870648e-02, ...,
         3.77982889e-01,  2.67902063e-03,  3.25506792e-01],
       ...,
       [ 9.34652089e-03,  7.17222695e-02,  1.37944808e-03, ...,
         3.77982889e-01,  2.67902063e-03,  3.25506792e-01],
       [ 4.55266018e-02, -3.43390544e-01, -4.00039943e-02, ...,
         3.77982889e-01,  2.67902063e-03,  3.25506792e-01],
       [ 1.83915411e-02,  7.17222695e-02, -4.73069547e-02, ...,
         3.77982889e-01,  2.67902063e-03,  3.25506792e-01]])
```

The codes above showed how to create a SHAP Explainer to get the SHAP values for our prediction model - Logistic Regression model, when predicting on testing dataset (it is stated as test set here). The SHAP values have been widely acknowledged in machine learning (ML) model explanations. The SHAP values can be the approach to explain the outcomes of any ML model. The SHAP values for each feature indicate the contribution of that specific feature to the model's prediction. A positive SHAP value indicates the positive impact on the prediction while negative SHAP value indicates the negative impact prediction, for that particular instance. By analysis to the SHAP values, we can gain insights into our model decision making process.

Task 4:

- Plot SHAP force plot for first row of test data

```
test_set.head(1)
```

✓ 0.0s

Python

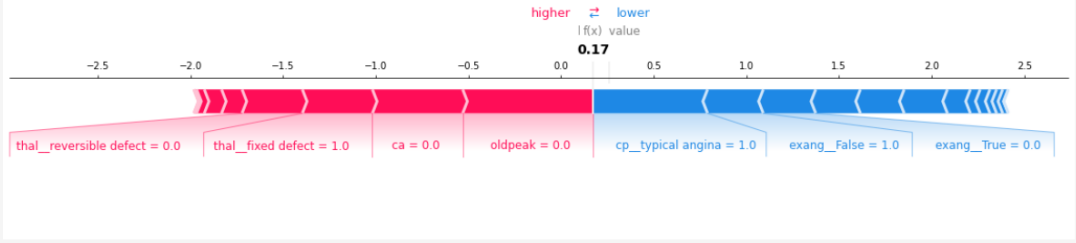
	age	trestbps	chol	thalach	oldpeak	slope	ca	sex_female	sex_male	cp_asymptomatic	...	fbs_higher than 120 mg/dl	fbs_lower than 120 mg/dl	restecg_lv hypertrophy	restecg_normal	restecg_stt abnormality	exang_False	exang_True	thal_fixed defect	thal_
0	46.0	138.0	243.0	152.0	0.0	1.0	0.0	0.0	1.0	0.0	...	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	

1 rows × 23 columns

```
# Plot the SHAP force plot for the first rows of test data

print(f"Predicted Value: {lr_model.predict(test_set)[0]}")
shap.force_plot(shap_values.base_values[0],
                shap_values.values[0],
                test_set.iloc[[0]],
                matplotlib=True)
```

✓ 0.4s

```
Predicted Value: 1
```



To interpret the local explanation of the SHAP, the first row of test data (instance '0' in the array of shap_values) is used. From the generated force plot, we can see that the model predicts a value of 1 for this particular row of test data, indicating 'having heart disease'. The features that positively impact the prediction result (shown in red) are displayed on the left side, while the features that negatively impact the prediction result (shown in blue) are displayed on the right side, resulting in an overall SHAP value of +0.17. The main reasons cause to the prediction of 'having heart disease' are related to the patient's ST depression value of 0 ('oldpeak' = 0), zero number of major vessels coloured by fluoroscopy ('ca' = 0) and having a fixed defect for his thalassemia ('thai_fixed defect' = 1). The patient's typical-angina-typed chest pain is identified as the most influential factor that causes him to be diagnosed as not having heart disease ('cp_typical angina' = 1).

Task 5:

- Plot SHAP force plot for all the rows of test data

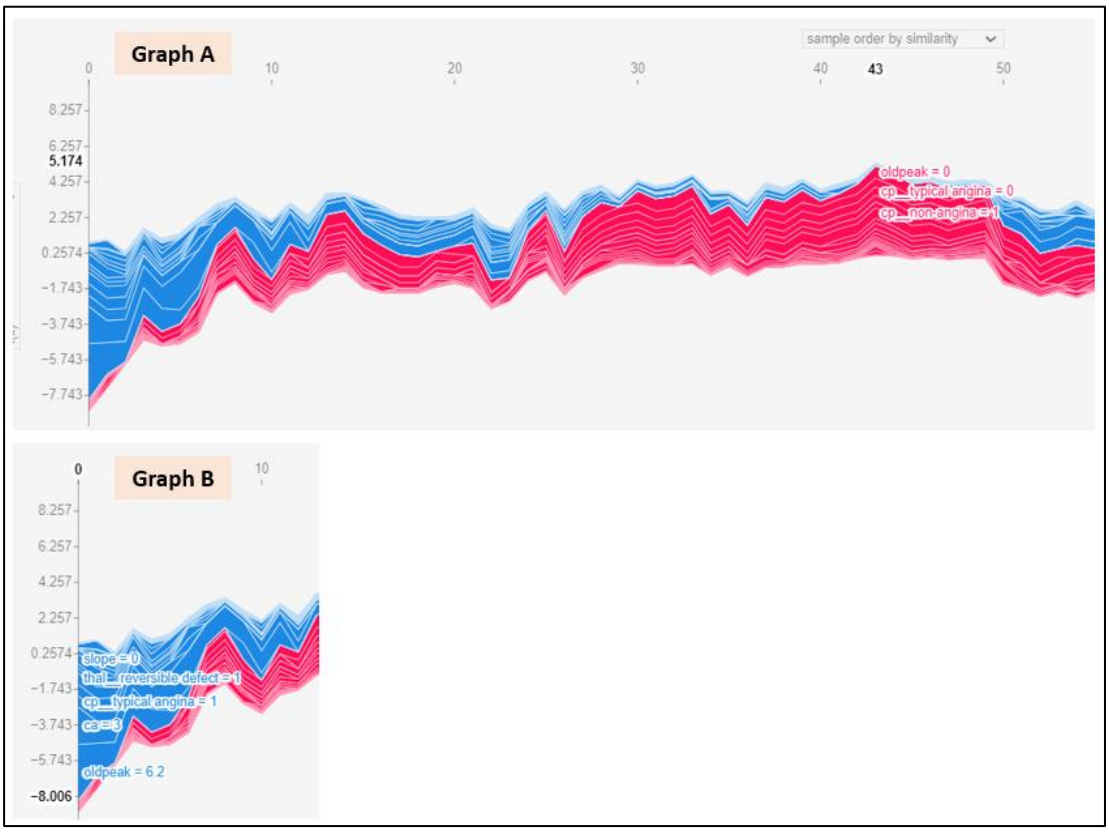
```
# Plot a SHAP force plot for the all rows of test data.
shap.initjs()
shap.force_plot(shap_values.base_values,
                shap_values.values,
                test_set)
```

✓ 0.0s Python

The above codes demonstrate the SHAP force plot for all data samples in test set (90 rows in total). It can give the overall distribution of SHAP values for every instance in the test set in a single graph. When we move our cursor to the upper peak of the largest region of red colour, we can see that the features 'oldpeak = 0', 'cp_typical angina = 0' and 'cp_non-angina = 1' are the most influential factors that cause to model to shift to predict 'having heart disease' in most of the sample's prediction. Which means that, the patients with an ST depression value of 0, no tyical-angina-typed chest pain, as well as non-angina-typed chest pain are the most likely to be diagnosed with having heart disease.

On the other hand, when the cursor is moved to the bottom peak of the largest region of blue colour, we can see that the most influential factors that cause to model to shift to have prediction of 'no heart disease' are 'oldpeak = 6.2', 'ca = 3', 'cp_typical angina = 1', 'thal_reversible defect = 1' and 'slope = 0'. This implies that the patients are likely to be diagnosed with 'no heart disease' is because of having an ST depression value of

6.2, 3 major vessels coloured by fluoroscopy, typical-angina-typed chest pain, reversible defect thalassemia, as well as a slope of zero in the peak exercise ST segment.

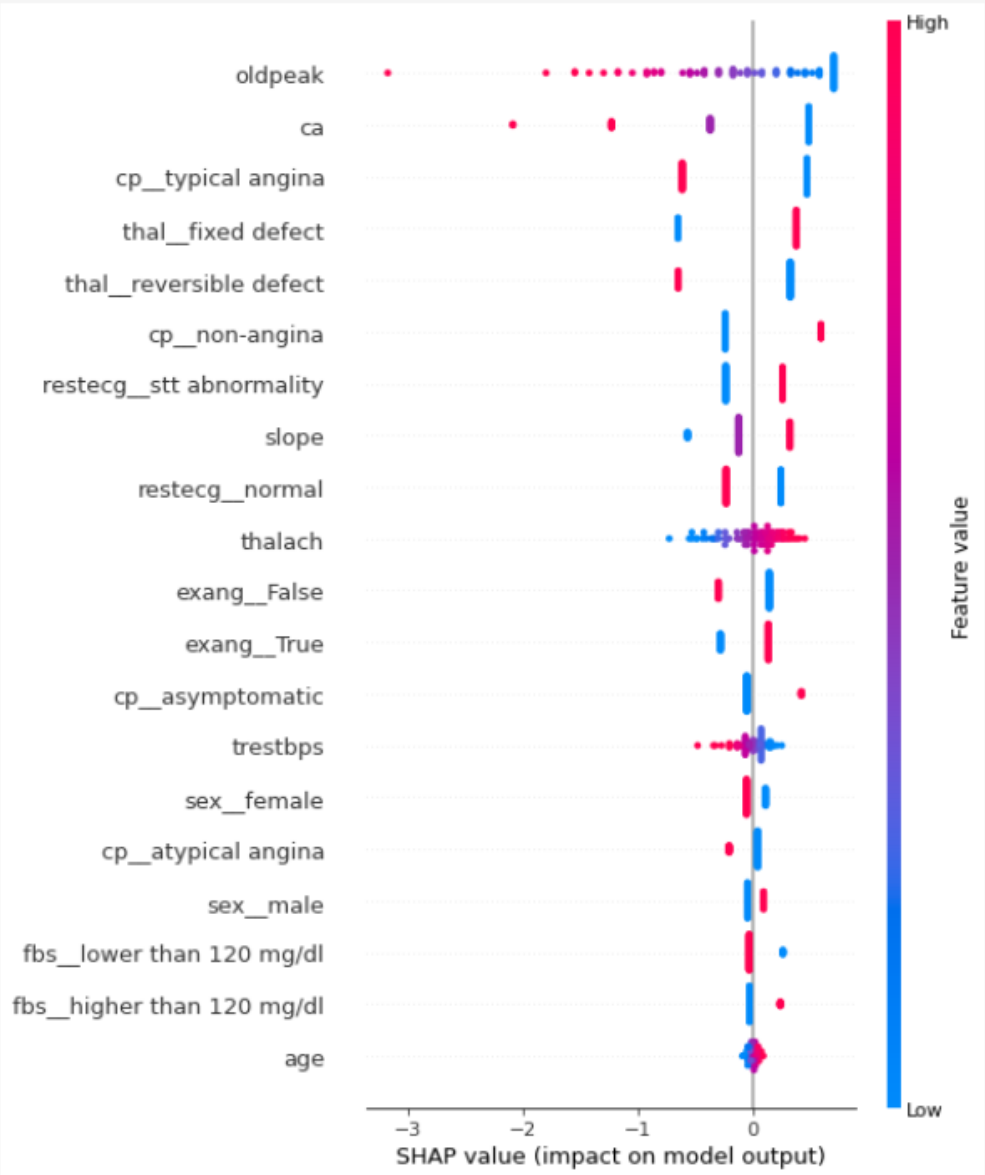


Task 6:

- Plot SHAP summary plot using all the features in the data

```
# Plot a SHAP summary plot using all the features in the data
shap.summary_plot(shap_values, feature_names=all_features)
```

✓ 0.4s



The codes above demonstrate the summary plot for all data samples in the test set with almost all the features. The maximum number of features that can be shown in summary plot is limited to 20, so the above plot displayed the top important features for the model prediction that indicated by SHAP. It shows the impacts of each feature on the model prediction using SHAP values as a metric. The colour represents that particular feature value, either high (red color), low (blue color) or medium (purple color), which is according to the color indicator on the right side.

Look at the example of ‘oldpeak’ feature, it has the widest range of SHAP values among other features. The graph shows that the high ‘oldpeak’ value has a negative impact on the model output and vice versa. The higher the value of ‘oldpeak’, the more impact it has on the prediction of the model. This implies that a higher ST depression value is beneficial in steering predictions away from the heart disease.

Another example is the ‘ca’, which has a total of 4 types of instance values (0 to 3) in the test set, so the plot has shows 4 dots. Each dot represents one instance value. Starting from the most left-hand side, the dots represent 3, 2, 1 and 0 accordingly. From there, we observe that the patients with the characteristic of 4 major vessels coloured by fluoroscopy are most likely to have no heart disease, as its impact on model prediction is considered as significantly negative. The patients with either 1, 2 or 3 major vessels coloured by fluoroscopy are predicted to not have heart disease overall, while having no major vessels coloured by fluoroscopy leads to a positive impact in diagnosis heart disease.

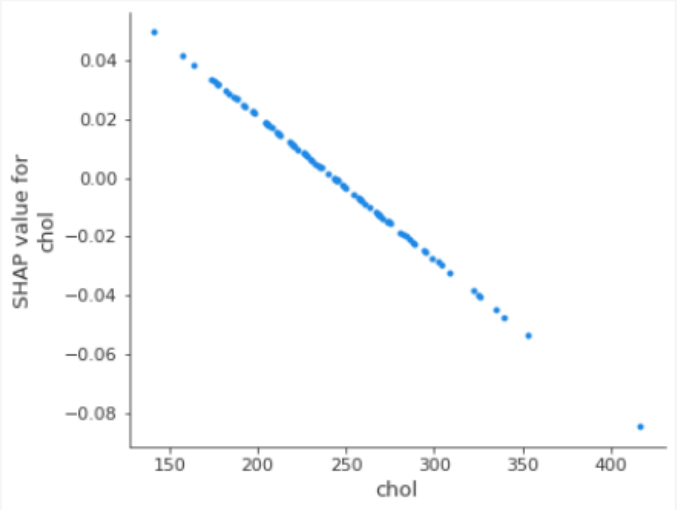
The next example is ‘age’. Age seems to have not much contribution to the model prediction. However, we still can observe that the ‘age’ can lead to a positive result in heart disease when it is getting higher, while it can lead to a negative impact on heart disease when it is getting lower. It seems that older people have a higher possibility of getting heart disease than younger persons, which matches with many people intuitive.

Task 7:

- Plot dependence plot to show the effect of ‘chol’ across the whole dataset

```
shap.dependence_plot("chol", shap_values.values, test_set, interaction_index=None)
```

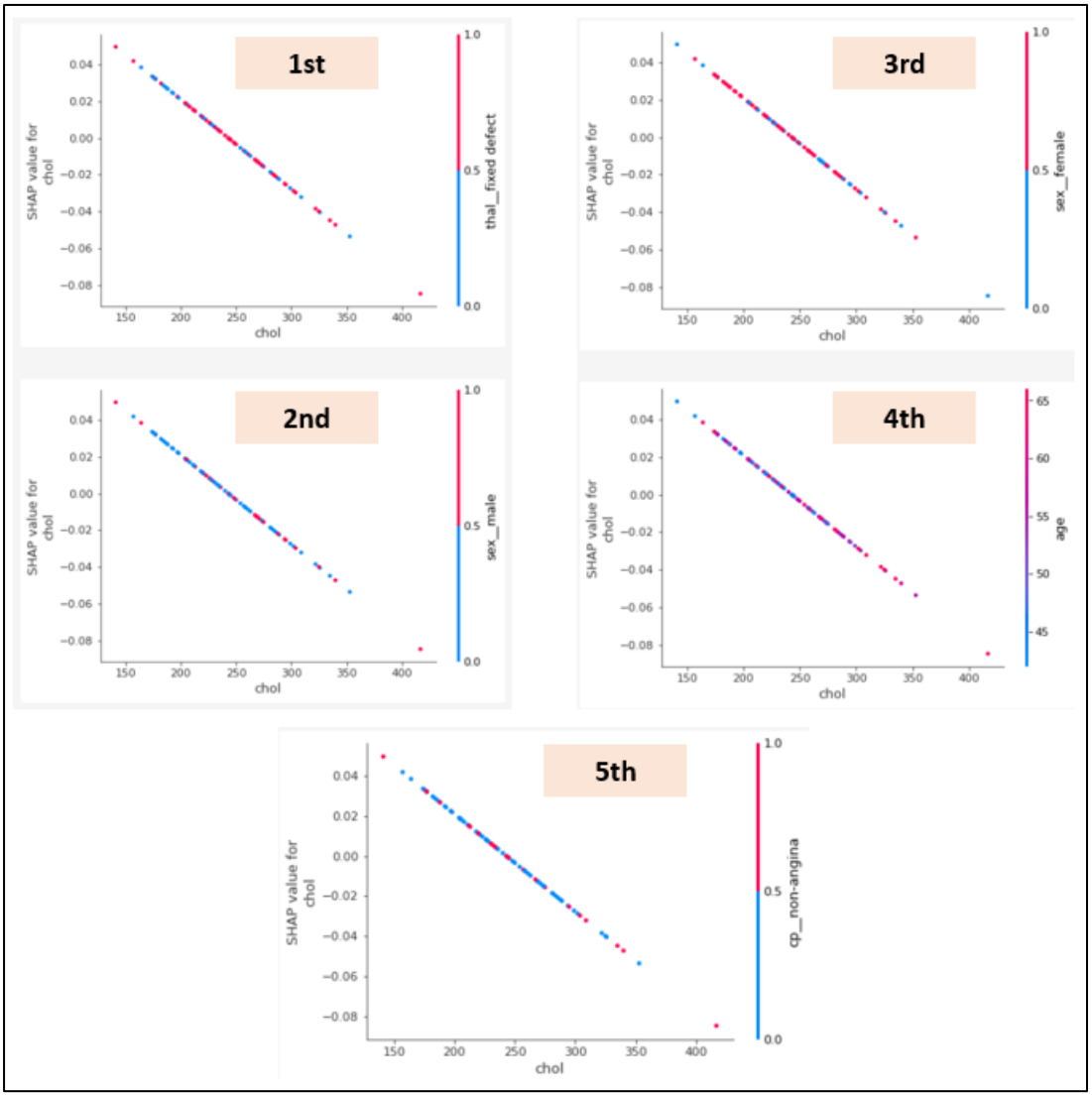
✓ 0.1s



The above code demonstrates the dependence plot of the feature ‘chol’ across the entire test set. It can show the effects that ‘chol’ has on the predictions of the model. The x-axis of the graph represents the value of the feature ‘chol’, while the y-axis represents the SHAP value for that ‘chol’. As shown in the graph above, the value of ‘chol’ has a perfect negative correlation with the SHAP value of itself. This means that when the ‘chol’ value is higher, the SHAP value for ‘chol’ will decrease. In other words, a higher serum cholesterol value leads to a lower SHAP value, and a more negative impact on the model prediction.

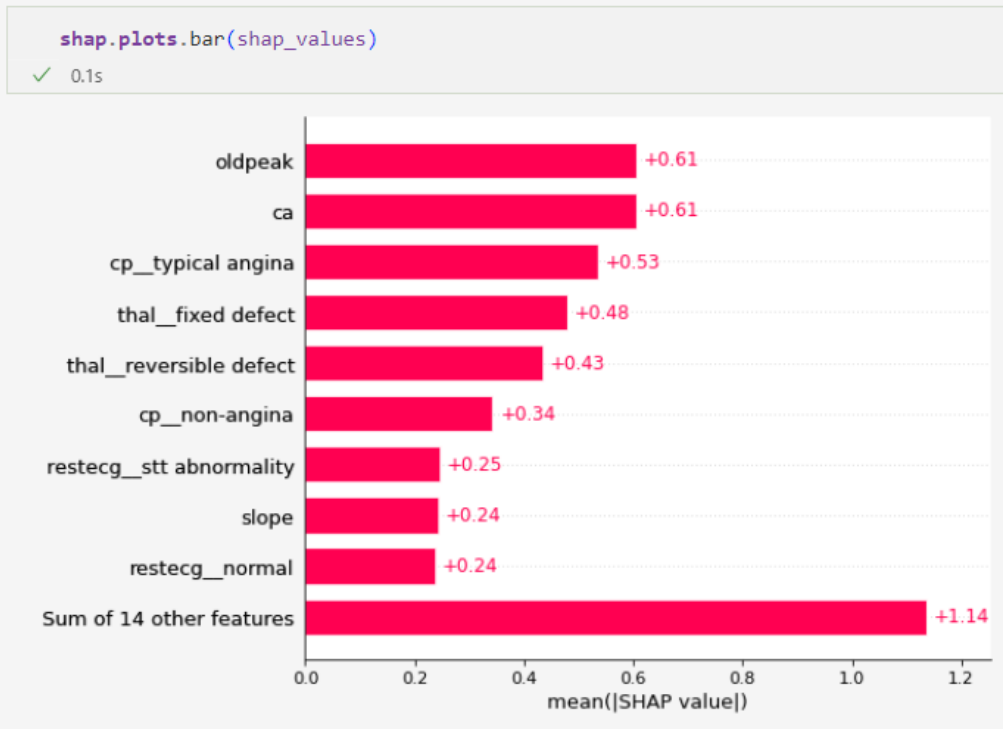
The serum cholesterol value of about 200 to 250 mg/dl has the least contribution to prediction outcome. According to the general indications, the normal rate of healthy serum cholesterol is less than 200mg/dl. The less impact ‘chol’ value falls at a borderline high level in this graph. However, the overall SHAP values of the ‘chol’ are significantly small. It does not even be ranked in the top 20 important features, as shown in the previous summary plot. Therefore, the serum cholesterol level appears not to be an important factor for this model. This is quite counterintuitive to most people. The picture below shows the graphs of the dependence plot of ‘chol’ when adding the interaction effect with other features. The top 5 possible interacting features with ‘chol’ approximated by SHAP are fixed defect thalassemia, sex in male, sex in female, age

and non-angina-typed chest pain. For instance, in the first graph, the sample with serum cholesterol value of less than 150 with fixed defect thalassemia is most likely to have heart disease.



Task 8:

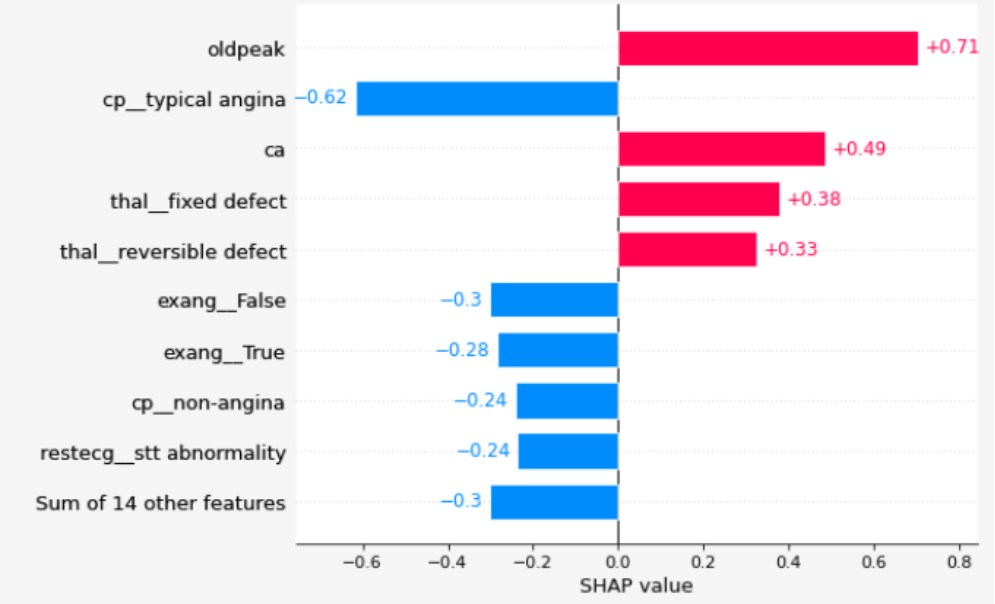
- Additional works
  - Plot SHAP bar plot using all the rows of the test data



To further explore the global interpretability of this heart disease model, the bar plot of the entire test set is generated. The bar plot shows the mean absolute SHAP value for each feature across the entire test set. A higher mean of feature indicates more influence on the prediction of model. The ranks of features in the bar plot are the same as their ranking in the previous summary plot. From the graph above, we can see that the 'oldpeak' and 'ca' have the equal mean absolute SHAP values and are ranked the highest in the graph. The sum of their mean values is already higher than the sum of other 14 features at the bottom of the ranking. (0.61+0.61 = 1.22 > 1.14) All features are in the positive direction, which means all have a positive magnitude in their contribution to the model's prediction.

- Plot SHAP bar plot for the first row of test data

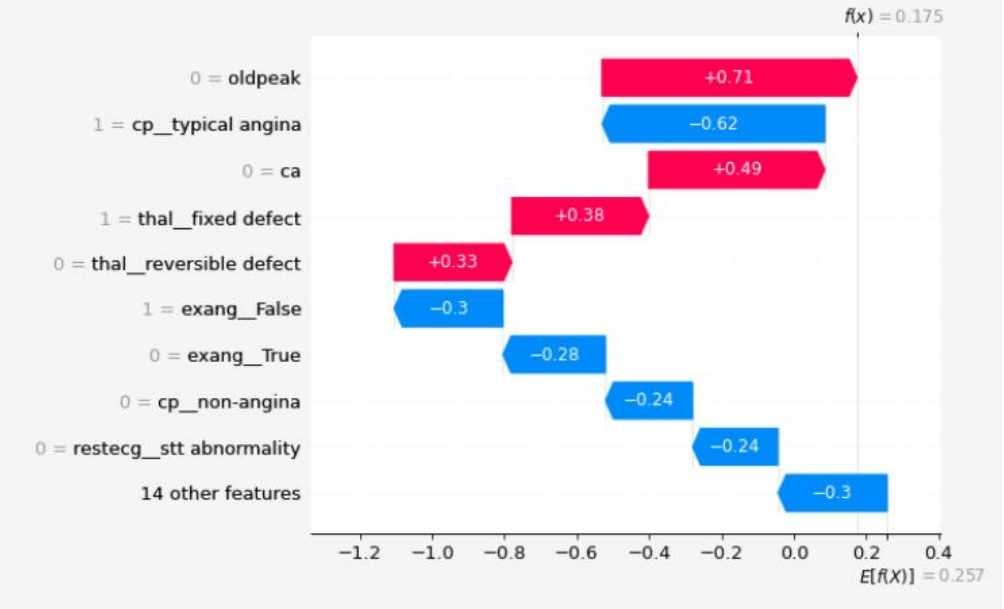
```
shap.plots.bar(shap_values[0])
```
✓ 0.1s



The bar plot can also be generated for a single row of data. The graph above is the bar plot for the first row of the test set. Similar to the previous bar plot, the features impacting the model outcome are depicted in either the positive impact (in red) or negative impact (in blue) for that feature on this prediction. Unlike the force plot in task 4, where we had to estimate the SHAP values of features by manually guessing the length of the bars, this bar plot clearly displays the actual value of each feature. This makes it more practical when we want to examine the SHAP values for each feature in a specific sample.

- Plot SHAP waterfall plot for the first row of test data

```
shap.plots.waterfall(shap_values[0])
```
✓ 0.5s

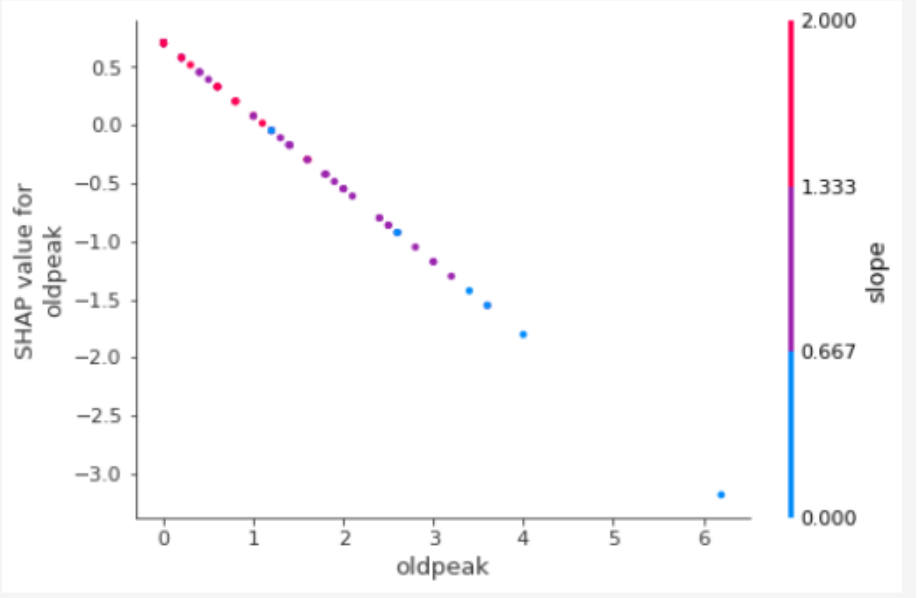


Same as the previous bar plot for a single row of data, the waterfall plot is also applicable for local interpretation. Additionally, same as the SHAP force plot, the arrows in red imply the features that force the model towards predicting a 'have heart disease' result, while the arrows in blue imply that the features that force the model towards predicting a 'no heart disease' result. The base value E[f(x)]) is 0.257, while the final prediction variable f(x) is 0.175 for the first row of test set. It shows the prediction of the model to this sample is 'having heart disease. The waterfall plot seems like the combination of bar plot and force plot. It emphasises the additive nature of the contribution (positive or negative), and how these contributions lead to the base value and finally generate the model's prediction.

- Plot dependency plot to show the effect of ‘oldpeak’ across the whole dataset

```
shap.dependence_plot("oldpeak", shap_values.values, test_set)
```

✓ 0.1s



Since the feature ‘oldpeak’ has been found to be the most influential factor in the model prediction. Let’s plot the dependence plot to gain insights into its impact on the entire test set. From the graph above, we can see that, the value of ‘oldpeak’ has a perfect negative correlation with the SHAP value of itself. In the region where ‘oldpeak’ values are larger than 3, it can be seen that the values of ‘slope’ are associated with a decrease in the SHAP values of ‘oldpeak’. The samples with higher ‘oldpeak’ value and lower ‘slope’ value are most likely to be predicted as ‘no heart disease’ as the SHAP values is significantly shift towards a negative impact on the model’s prediction. This aligns with real-world cases and current medical knowledge, making the model reasonable in its logic.