```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import datasets
```

Data Collection and Processing

```
## loading the data
heart_data = pd.read_csv("CARDIO_ML_PROJ (1).csv")
```

```
heart_data.head()
```

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

```
heart_data.tail()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
298	57	0	0	140	241	0	1	123	1	0.2	1	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2
301	57	1	0	130	131	0	1	115	1	1.2	1	1
302	57	0	1	130	236	0	0	174	0	0.0	1	1

```
heart_data.shape
```

```
(303, 14)
```

```
heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
heart_data.isnull().sum() ## we dont hv any missing values
```

```
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
```

```
heart_data.dropna(how ='any', inplace = True)
```

```
heart_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
```

```
heart_data.describe() ## statistical calculation
```

	age	sex	cp	trestbps	chol	fbs	res
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.52
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.52
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.00
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.00
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.00
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.00
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.00

```
## checking the distribution of target variable
heart_data['target'].value_counts() ## 0 shows healthy heart, 1 shows unhealthy heart
```

```
1    165
0    138
Name: target, dtype: int64
```

```
## splitting the features and target
```

```
x = heart_data.drop(columns ='target', axis =1)
y = heart_data['target']
print(x)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   3       145   233    1        0      150      0      2.3
1     37    1   2       130   250    0        1      187      0      3.5
2     41    0   1       130   204    0        0      172      0      1.4
3     56    1   1       120   236    0        1      178      0      0.8
4     57    0   0       120   354    0        1      163      1      0.6
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0
```

```
         slope  ca  thal
    0        0   0     1
    1        0   0     2
    2        2   0     2
    3        2   0     2
    4        2   0     2
    ..     ...  ..   ...
    298      1   0     3
    299      1   0     3
    300      1   2     3
    301      1   1     3
    302      1   1     2

    [303 rows x 13 columns]
```

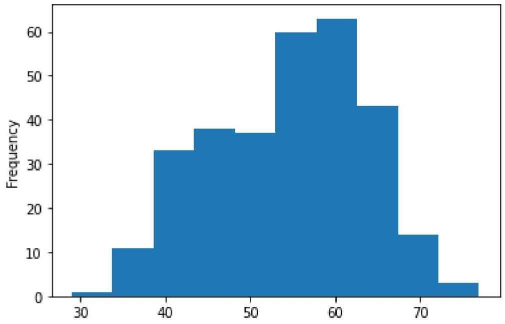
```
print(y)
```

```
    0      1
    1      1
    2      1
    3      1
    4      1
          ..
    298    0
    299    0
    300    0
    301    0
    302    0
    Name: target, Length: 303, dtype: int64
```

```
heart_data.age.value_counts() ### ques:2 [c]
```

```
    58    19
    57    17
    54    16
    59    14
    52    13
    51    12
    62    11
    60    11
    44    11
    56    11
    64    10
    41    10
    63     9
    67     9
    65     8
    43     8
    45     8
    55     8
    42     8
    61     8
    53     8
    46     7
    48     7
    66     7
    50     7
    49     5
    47     5
    70     4
    39     4
    35     4
    68     4
    38     3
    71     3
    40     3
    69     3
    34     2
    37     2
    29     1
    74     1
    76     1
    77     1
    Name: age, dtype: int64
```

```
#Distribution of age according to the heart disease
heart_data.age.plot.hist()
#Show plot
plt.show()
```



```
heart_data.sex.value_counts() ### [d]
```

```
1    207
0     96
Name: sex, dtype: int64
```

```
heart_data.sex.describe()
```

```
count    303.000000
mean       0.683168
std        0.466011
min        0.000000
25%        0.000000
50%        1.000000
75%        1.000000
max        1.000000
Name: sex, dtype: float64
```

```
heart_data.trestbps.value_counts()
```

```
120    37
130    36
140    32
110    19
150    17
138    13
128    12
160    11
125    11
112     9
132     8
118     7
124     6
135     6
108     6
152     5
134     5
145     5
122     4
170     4
100     4
105     3
126     3
115     3
180     3
136     3
142     3
102     2
148     2
178     2
94      2
144     2
146     2
200     1
114     1
154     1
123     1
192     1
174     1
165     1
104     1
117     1
101     1
156     1
106     1
155     1
```

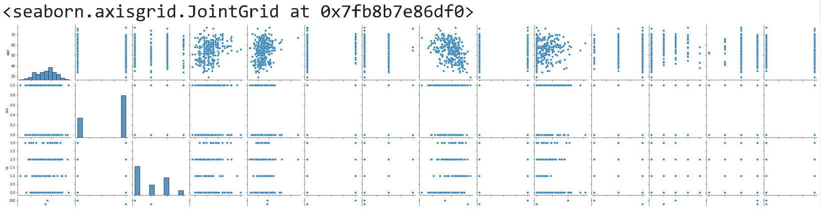
```
    129     1
    172     1
    164     1
    Name: trestbps, dtype: int64
```

```
len(heart_data[heart_data.trestbps ==1]) ## [e]
```

```
    0
```

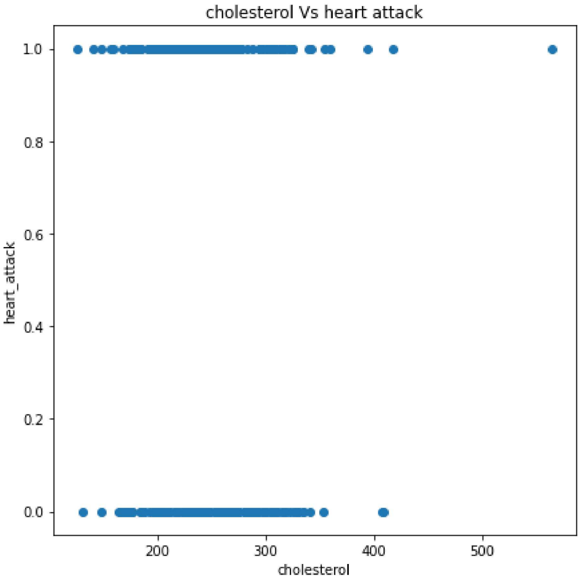
```
sns.pairplot(data=heart_data) ## [f]
heart_data.corr()
sns.jointplot(heart_data.chol   , heart_data.target)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning:
  warnings.warn(
<seaborn.axisgrid.JointGrid at 0x7fb8b7e86df0>
```



```
plt.figure(figsize=(7,7))                 ### [f]
plt.scatter(x='chol',y='target',data=heart_data)
plt.xlabel('cholesterol')
plt.ylabel('heart_attack')
plt.title('cholesterol Vs heart attack')
```

```
Text(0.5, 1.0, 'cholesterol Vs heart attack')
```



```
heart_data.thalach.describe() ###[h]
```

```
count    303.000000
mean     149.646865
std       22.905161
min       71.000000
25%      133.500000
50%      153.000000
75%      166.000000
max      202.000000
Name: thalach, dtype: float64
```

```
heart_data.info()###[i]
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 43.6 KB
```

```
heart_data.all
```

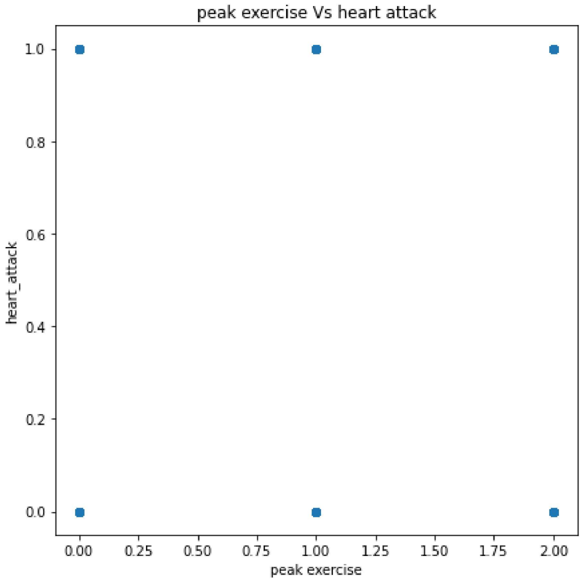
```
<bound method NDFrame._add_numeric_operations.<locals>.all of      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   3       145   233    1        0      150      0      2.3   
1     37    1   2       130   250    0        1      187      0      3.5   
2     41    0   1       130   204    0        0      172      0      1.4   
3     56    1   1       120   236    0        1      178      0      0.8   
4     57    0   0       120   354    0        1      163      1      0.6   
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...   
298   57    0   0       140   241    0        1      123      1      0.2   
299   45    1   3       110   264    0        1      132      0      1.2   
300   68    1   0       144   193    1        1      141      0      3.4   
301   57    1   0       130   131    0        1      115      1      1.2   
302   57    0   1       130   236    0        0      174      0      0.0   

     slope  ca  thal  target  
0        0   0     1       1  
1        0   0     2       1  
2        2   0     2       1  
3        2   0     2       1  
4        2   0     2       1  
..     ...  ..   ...     ...  
298      1   0     3       0  
299      1   0     3       0  
300      1   2     3       0  
301      1   1     3       0  
302      1   1     2       0  

[303 rows x 14 columns]>
```

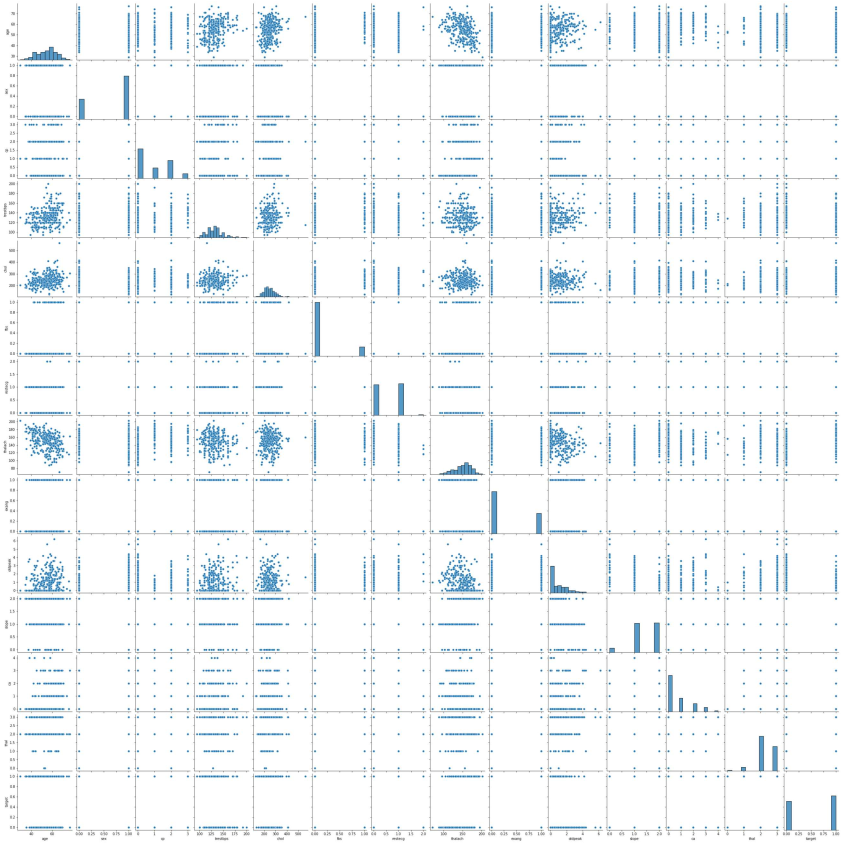
```
plt.figure(figsize=(7,7))                 ### [g]
plt.scatter(x='slope',y='target',data=heart_data)
plt.xlabel('peak exercise')
plt.ylabel('heart_attack')
plt.title('peak exercise Vs heart attack')
```

```
Text(0.5, 1.0, 'peak exercise Vs heart attack')
```



```
sns.pairplot(data=heart_data) ###[j]
```

```
<seaborn.axisgrid.PairGrid at 0x7fb8b8b2f880>
```



ANS:3

```
## splitting into trin and test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y, random_state=2)
```

```
print(x.shape, x_train.shape, x_test.shape)
```

```
    (303, 13) (242, 13) (61, 13)
```

```
## model training
```

```
model = LogisticRegression()
model.fit(x_train, y_train)
```

```
    /usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
    STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

    Increase the number of iterations (max_iter) or scale the data as shown in:
        https://scikit-learn.org/stable/modules/preprocessing.html
    Please also refer to the documentation for alternative solver options:
        https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
      n_iter_i = _check_optimize_result(
    LogisticRegression()
```

```
## model evaluation
```

```
## ACCURACY OF TRAINING DATA
x_train_prediction = model.predict(x_train)
training_data_accuracy = accuracy_score(x_train_prediction, y_train)
print('Accuracy on Training Data: ', training_data_accuracy)
```

```
    Accuracy on Training Data:  0.8512396694214877
```

```
## ACCURACY SCORE ON TEST DATA
x_test_prediction = model.predict(x_test)
```

```
test_data_accuracy = accuracy_score(x_test_prediction, y_test)
print('Accuracy on Test Data: ', test_data_accuracy)
     Accuracy on Test Data:  0.819672131147541
```

```
## building the predictive system
input_data = (41,0,1,130,204,0,0,172,0,1.4,2,0,2)

## change the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)
# reshape the numpy array as we are predicting for only one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = model.predict(input_data_reshaped)
print(prediction)

if(prediction[0]==0):
  print("The person having healthy heart")
else:
  print("The person having heart Disease")

     [1]
     The person having heart Disease
     /usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but LogisticRegression was
       warnings.warn(
```

```
# Decision tree model
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(x_train,y_train)

     DecisionTreeClassifier()
```

```
# Classification report
predictions = dtree.predict(x_test)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,predictions))

                   precision    recall  f1-score   support

                0       0.80      0.71      0.75        28
                1       0.78      0.85      0.81        33

         accuracy                           0.79        61
        macro avg       0.79      0.78      0.78        61
     weighted avg       0.79      0.79      0.79        61
```

```
# Creating a confusion matrix
print(confusion_matrix(y_test,predictions))

     [[20  8]
      [ 5 28]]
```

```
# Random forest model
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100,class_weight='balanced_subsample')
rfc.fit(x_train,y_train)

     RandomForestClassifier(class_weight='balanced_subsample')
```

```
predictions = rfc.predict(x_test)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,predictions))

                   precision    recall  f1-score   support

                0       0.76      0.79      0.77        28
                1       0.81      0.79      0.80        33

         accuracy                           0.79        61
        macro avg       0.79      0.79      0.79        61
     weighted avg       0.79      0.79      0.79        61
```

```
print(confusion_matrix(y_test,predictions))
```

```
[[22  6]
 [ 7 26]]
```