

[File](#) [+](#) [X](#) [Run](#) [Kernel](#) [Settings](#) [Help](#)[Code](#) [▼](#)[JupyterLab](#) [Python 3 \(ipykernel\)](#) [☰](#)[1]:

```
# Step 1 : Data Collection and Understanding
# Import Libraries used in data preparation
import pandas as pd
import numpy as np
```

[2]:

```
# Load the heart dataset .. in same folder
dataset = pd.read_csv("heart.csv") #Load dataset into a pandas DataFrame.
dataset #to ensure read it successfully
```

[2]:

	age	sex	cp	trestbps	chol	fb	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

[3]:

```
# get the dataset Size
size = dataset.shape
size
```

[3]: (1025, 14)

[4]:

```
# get the dataset dimension
dimension = dataset.ndim
dimension
```

[4]: 2

[5]:

```
# get columns' titles
titles = dataset.columns
titles
```

[5]: Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fb', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'], dtype='object')

[6]:

```
#describing a certain column with some basic statistics
desc = dataset['target'].describe()
desc
```

[6]:

count	1025.000000
mean	0.513171
std	0.500070
min	0.000000
25%	0.000000
50%	1.000000
75%	1.000000
max	1.000000
Name:	target, dtype: float64

[7]:

```
# Step 2 : Data Exploration
# Display the first 5 rows of the dataset
head = dataset.head()
head
```

[7]:

	age	sex	cp	trestbps	chol	fb	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0

```
4 62 0 0 138 294 1 1 106 0 1.9 1 3 2 0
```

```
[8]: # Display information about the dataset
info = dataset.info()
info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   age         1025 non-null   int64  
 1   sex          1025 non-null   int64  
 2   cp           1025 non-null   int64  
 3   trestbps    1025 non-null   int64  
 4   chol         1025 non-null   int64  
 5   fbs          1025 non-null   int64  
 6   restecg     1025 non-null   int64  
 7   thalach      1025 non-null   int64  
 8   exang        1025 non-null   int64  
 9   oldpeak      1025 non-null   float64 
 10  slope        1025 non-null   int64  
 11  ca           1025 non-null   int64  
 12  thal         1025 non-null   int64  
 13  target       1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

```
[9]: # Display descriptive statistics of the dataset
desc = dataset.describe()
desc
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.000000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366	
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	
25%	48.000000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	
50%	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	
75%	61.000000	1.000000	2.000000	140.000000	275.000000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	

```
[10]: # Step 3 : Data Cleaning and preparation
data = pd.read_csv("heart_after_edit.csv") #Load edit dataset into a pandas DataFrame.
data #to ensure read it successfully
```

	index	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	1	52.0	1	0	125	212.0	0.0	1	168	0.0	1	2	2	3	0.0
1	2	53.0	5	0	120	203.0	1.0	0	155	1.0	3.1	0	0	3	0.0
2	3	NaN	1	0	145	174.0	0.0	1	125	1.0	2.6	0	0	3	0.0
3	4	61.0	1	0	148	203.0	0.0	1	161	0.0	0	2	1	3	NaN
4	5	62.0	0	0	138	294.0	1.0	1	106	0.0	1.9	1	3	2	0.0
...
1020	1021	59.0	1	1	140	221.0	0.0	1	164	1.0	0	2	0	2	1.0
1021	1022	60.0	1	0	125	258.0	0.0	0	141	1.0	2.8	1	1	3	0.0
1022	1023	47.0	1	0	110	275.0	0.0	0	118	1.0	1	1	1	2	0.0
1023	1024	50.0	0	0	110	254.0	0.0	0	159	0.0	0	2	0	2	1.0
1024	1025	54.0	1	0	120	188.0	0.0	1	113	0.0	1.4	1	1	3	0.0

1025 rows × 15 columns

```
[11]: data.rename(columns={'sex': 'gender'}, inplace=True) #rename column
data.columns
```

```
[11]: Index(['index', 'age', 'gender', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
dtype='object')
```

```
[12]: #rename column - more clear
data.columns = ['index', 'age', 'gender', 'chest_pain_type', 'resting_blood_pressure', 'cholesterol', 'fasting_blood_sugar', 'rest_electro'
'exercise_induced_angina', 'st_depression', 'st_slope', 'num_major_vessels', 'thal', 'target']

data.columns
```

```
[12]: Index(['index', 'age', 'gender', 'chest_pain_type', 'resting_blood_pressure',
   'cholesterol', 'fasting_blood_sugar', 'rest_electro', 'max_heart_rate',
   'exercise_induced_angina', 'st_depression', 'st_slope',
   'num_major_vessels', 'thal', 'target'],
  dtype='object')

[13]: desc = data.describe()
desc # Display descriptive statistics of the edited dataset
```

	index	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	ex
count	1025.000000	1024.000000	1025.000000	1025.000000	1025.000000	1024.000000	1024.000000	1025.000000	1025.000000	
mean	512.649756	54.418945	0.699512	0.942439	184.712195	246.045898	0.149414	0.534634	149.114146	
std	296.416605	9.063654	0.479512	1.029641	1699.772051	51.596779	0.356670	0.545776	23.005724	
min	1.000000	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	
25%	256.000000	48.000000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	132.000000	
50%	513.000000	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	152.000000	
75%	769.000000	61.000000	1.000000	2.000000	140.000000	275.000000	0.000000	1.000000	166.000000	
max	1025.000000	77.000000	5.000000	3.000000	54548.000000	564.000000	1.000000	5.000000	202.000000	

```
[14]: #count missing values in each attribute
missing = pd.DataFrame({'missing': data.isnull().sum()})

missing
```

	missing
index	0
age	1
gender	0
chest_pain_type	0
resting_blood_pressure	0
cholesterol	1
fasting_blood_sugar	1
rest_electro	0
max_heart_rate	0
exercise_induced_angina	1
st_depression	0
st_slope	0
num_major_vessels	0
thal	0
target	2

```
[15]: # print target coulmn to say data
data['target']
```

```
[15]: 0      0.0
1      0.0
2      0.0
3    NaN
4      0.0
...
1020    1.0
1021    0.0
1022    0.0
1023    1.0
1024    0.0
Name: target, Length: 1025, dtype: float64
```

```
[16]: # target column is numeric
data['target'] = pd.to_numeric(data['target'], errors='coerce')

# Drop rows with missing target values empty or null
# data = data.dropna(subset=['target']) or
data = data[pd.notnull(data['target'])]

# convert to int type
data['target'] = data['target'].astype(int)

# drop rows where target values are not 0 or 1
data = data[data['target'].isin([0, 1])].reset_index(drop=True)
```

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1023 entries, 0 to 1022
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   index            1023 non-null   int64  
 1   age              1022 non-null   float64 
 2   gender           1023 non-null   int64  
 3   chest_pain_type 1023 non-null   int64  
 4   resting_blood_pressure 1023 non-null   int64  
 5   cholesterol      1022 non-null   float64 
 6   fasting_blood_sugar 1022 non-null   float64 
 7   rest_electro     1023 non-null   int64  
 8   max_heart_rate   1023 non-null   int64  
 9   exercise_induced_angina 1022 non-null   float64 
 10  st_depression    1023 non-null   object  
 11  st_slope          1023 non-null   int64  
 12  num_major_vessels 1023 non-null   int64  
 13  thal              1023 non-null   int64  
 14  target            1023 non-null   int32  
dtypes: float64(4), int32(1), int64(9), object(1)
memory usage: 116.0+ KB
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_19128\3058665240.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
data['target'] = data['target'].astype(int)
```

```
[17]: data['target']
```

```
[17]: 0      0
 1      0
 2      0
 3      0
 4      1
 ...
1018    1
1019    0
1020    0
1021    1
1022    0
Name: target, Length: 1023, dtype: int32
```

```
[18]: data = data[pd.notnull(data['cholesterol'])]      remove rows with null value in cholesterol Att
```

```
[19]: #count missing values in each attribute
missing = pd.DataFrame({'missing': data.isnull().sum()})
```

```
[19]:      missing
       index      0
       age        1
      gender      0
  chest_pain_type  0
resting_blood_pressure  0
      cholesterol  0
fasting_blood_sugar    1
      rest_electro  0
max_heart_rate         0
exercise_induced_angina  1
      st_depression  0
      st_slope      0
num_major_vessels      0
      thal         0
      target       0
```

```
[20]: # show age before processing
data['age']
```

```
[20]: 0      52.0
 1      53.0
 2      NaN
 3      62.0
 4      58.0
 ...
1018    59.0
```

```
1019    60.0
1020    47.0
1021    50.0
1022    54.0
Name: age, Length: 1022, dtype: float64
```

```
[21]: # Ensure age column is numeric
data['age'] = pd.to_numeric(data['age'], errors='coerce')

# Drop rows with missing age values and reset index
data = data.dropna(subset=['age']).reset_index(drop=True)
# or do this data = data['age'].fillna(data['age'].mean())
data['age']
```

```
[21]: 0      52.0
1      53.0
2      62.0
3      58.0
4      58.0
...
1016    59.0
1017    60.0
1018    47.0
1019    50.0
1020    54.0
Name: age, Length: 1021, dtype: float64
```

```
[22]: data.describe()
```

```
[22]:   index      age     gender chest_pain_type resting_blood_pressure cholesterol fasting_blood_sugar rest_electro max_heart_rate ex
count 1021.000000 1021.000000 1021.000000 1021.000000 1021.000000 1021.000000 1020.000000 1021.000000 1021.000000
mean  514.476983  54.421156  0.698335  0.941234  184.874633  246.218413  0.150000  0.533790  149.092067
std   295.524507  9.061711  0.480081  1.028264  1703.099676  51.570039  0.357247  0.546005  23.016902
min   1.000000  29.000000  0.000000  0.000000  94.000000  126.000000  0.000000  0.000000  71.000000
25%   259.000000  48.000000  0.000000  0.000000  120.000000  211.000000  0.000000  0.000000  132.000000
50%   515.000000  56.000000  1.000000  1.000000  130.000000  240.000000  0.000000  1.000000  152.000000
75%   770.000000  61.000000  1.000000  2.000000  140.000000  275.000000  0.000000  1.000000  166.000000
max   1025.000000 77.000000  5.000000  3.000000  54548.000000  564.000000  1.000000  5.000000  202.000000
```

```
[23]: # removing rows where gender is either 0 or 1 and reset index
data = data[data['gender'].isin([0, 1])]

data.reset_index(drop=True, inplace=True)

data.describe()
```

```
[23]:   index      age     gender chest_pain_type resting_blood_pressure cholesterol fasting_blood_sugar rest_electro max_heart_rate ex
count 1020.000000 1020.000000 1020.000000 1020.000000 1020.000000 1020.000000 1019.000000 1020.000000 1020.000000
mean  514.979412  54.422549  0.694118  0.942157  184.938235  246.260784  0.149166  0.534314  149.086275
std   295.232878  9.066047  0.461006  1.028346  1703.933930  51.577553  0.356427  0.546017  23.027449
min   1.000000  29.000000  0.000000  0.000000  94.000000  126.000000  0.000000  0.000000  71.000000
25%   259.750000  48.000000  0.000000  0.000000  120.000000  211.000000  0.000000  0.000000  132.000000
50%   515.500000  56.000000  1.000000  1.000000  130.000000  240.000000  0.000000  1.000000  152.000000
75%   770.250000  61.000000  1.000000  2.000000  140.000000  275.250000  0.000000  1.000000  166.000000
max   1025.000000 77.000000  1.000000  3.000000  54548.000000  564.000000  1.000000  5.000000  202.000000
```

```
[24]: # rows where the fasting_blood_sugar column has empty values
before = data[data['fasting_blood_sugar'].isna()]
before
```

```
[24]:   index      age     gender chest_pain_type resting_blood_pressure cholesterol fasting_blood_sugar rest_electro max_heart_rate exercise_induced_angina
978    984    64.0      1          0            128        263.0           NaN             1            105                 1.0
```

```
[25]: # Replace NaN values in fasting_blood_sugar column with the mean
data['fasting_blood_sugar'].fillna(data['fasting_blood_sugar'].min(), inplace=True)

#row after replacement
after_row = data.loc[978]
print("Row with index 978 after replacement:")
print(after_row)
```

```
Row with index 978 after replacement:
index          984
```

```

age           64.0
gender        1
chest_pain_type    0
resting_blood_pressure 128
cholesterol     263.0
fasting_blood_sugar 0.0
rest_electro      1
max_heart_rate     105
exercise_induced_angina 1.0
st_depression      0.2
st_slope          1
num_major_vessels   1
thal              3
target            1
Name: 978, dtype: object

```

C:\Users\HP\AppData\Local\Temp\ipykernel_19128\410803621.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
data['fasting_blood_sugar'].fillna(data['fasting_blood_sugar'].min(), inplace=True)
```

[26]: `data.describe()`

	index	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	ex
count	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	
mean	514.979412	54.422549	0.694118	0.942157	184.938235	246.260784	0.149020	0.534314	149.086275	
std	295.232878	9.066047	0.461006	1.028346	1703.933930	51.577553	0.356282	0.546017	23.027449	
min	1.000000	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	
25%	259.750000	48.000000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	132.000000	
50%	515.500000	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	152.000000	
75%	770.250000	61.000000	1.000000	2.000000	140.000000	275.250000	0.000000	1.000000	166.000000	
max	1025.000000	77.000000	1.000000	3.000000	54548.000000	564.000000	1.000000	5.000000	202.000000	

[27]: `# rows where the exercise_induced_angina column has empty values
before = data[data['exercise_induced_angina'].isna()]
before`

	index	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina	ex
4	8	55.0	1	0	160	289.0	0.0	0	145	Nan	

[28]: `# Replace Nan values in exercise_induced_angina column with the median
data['exercise_induced_angina'].fillna(data['exercise_induced_angina'].median(), inplace=True)

#row after replacement
after_row = data.loc[4]
print("Row with index 4 after replacement:")
print(after_row)`

```

Row with index 4 after replacement:
index          8
age         55.0
gender        1
chest_pain_type    0
resting_blood_pressure 160
cholesterol     289.0
fasting_blood_sugar 0.0
rest_electro      0
max_heart_rate     145
exercise_induced_angina 0.0
st_depression      0.8
st_slope          1
num_major_vessels   1
thal              3
target            0
Name: 4, dtype: object

```

C:\Users\HP\AppData\Local\Temp\ipykernel_19128\2203889533.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
data['exercise_induced_angina'].fillna(data['exercise_induced_angina'].median(), inplace=True)
```

	index	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	ex
count	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000
mean	514.979412	54.422549	0.694118	0.942157	184.938235	246.260784	0.149020	0.534314	149.086275	
std	295.232878	9.066047	0.461006	1.028346	1703.933930	51.577553	0.356282	0.546017	23.027449	
min	1.000000	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	
25%	259.750000	48.000000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	132.000000	
50%	515.500000	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	152.000000	
75%	770.250000	61.000000	1.000000	2.000000	140.000000	275.250000	0.000000	1.000000	166.000000	
max	1025.000000	77.000000	1.000000	3.000000	54548.000000	564.000000	1.000000	5.000000	202.000000	

```
[30]: # Find max value in the resting_blood_pressure column
index = data['resting_blood_pressure'].idxmax()
#or data[data.resting_blood_pressure == 54548]

print(f"index : {index}, \nrow data : \n{data.loc[index]} ")

index : 2,
row data :
index              6
age            58.0
gender           0
chest_pain_type  0
resting_blood_pressure 54548
cholesterol      248.0
fasting_blood_sugar  0.0
rest_electro      0
max_heart_rate    122
exercise_induced_angina 0.0
st_depression     1
st_slope          1
num_major_vessels 0
thal             2
target            1
Name: 2, dtype: object
```

```
[31]: # Manually update the value of resting_blood_pressure
data.at[index, 'resting_blood_pressure'] = 120
# or data [120 , 4] = 120

data.loc[index]
```

```
[31]: index              6
age            58.0
gender           0
chest_pain_type  0
resting_blood_pressure 120
cholesterol      248.0
fasting_blood_sugar  0.0
rest_electro      0
max_heart_rate    122
exercise_induced_angina 0.0
st_depression     1
st_slope          1
num_major_vessels 0
thal             2
target            1
Name: 2, dtype: object
```

	index	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	ex
count	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000
mean	514.979412	54.422549	0.694118	0.942157	131.577451	246.260784	0.149020	0.534314	149.086275	
std	295.232878	9.066047	0.461006	1.028346	17.516987	51.577553	0.356282	0.546017	23.027449	
min	1.000000	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	
25%	259.750000	48.000000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	132.000000	
50%	515.500000	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	152.000000	
75%	770.250000	61.000000	1.000000	2.000000	140.000000	275.250000	0.000000	1.000000	166.000000	
max	1025.000000	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	5.000000	202.000000	

```
[33]: # Define the List of columns to check for duplicates
subset_list = ['index', 'age', 'gender', 'chest_pain_type', 'resting_blood_pressure', 'cholesterol',
               'fasting_blood_sugar', 'rest_electro', 'max_heart_rate', 'exercise_induced_angina',
               'st_depression', 'st_slope', 'num_major_vessels', 'thal', 'target']
```

```
#duplicate rows
duplicates = data[data.duplicated(subset=subset_list, keep='first')]

duplicates
```

[33]:

	index	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina
365	12	43.0	0	0	132	341.0	1.0	0	136	1.0

[34]: # Remove duplicate rows, keeping the first occurrence
data.drop_duplicates(subset=subset_list, keep='first', inplace=True)
data.head()

[34]:

	index	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina
0	1	52.0	1	0	125	212.0	0.0	1	168	0.0
1	5	62.0	0	0	138	294.0	1.0	1	106	0.0
2	6	58.0	0	0	120	248.0	0.0	0	122	0.0
3	7	58.0	1	0	114	318.0	0.0	2	140	0.0
4	8	55.0	1	0	160	289.0	0.0	0	145	0.0

[35]: data.drop(columns=['index'], inplace=True)
after delete column
data.head()

[35]:

	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina	st_depression
0	52.0	1	0	125	212.0	0.0	1	168	0.0	
1	62.0	0	0	138	294.0	1.0	1	106	0.0	
2	58.0	0	0	120	248.0	0.0	0	122	0.0	
3	58.0	1	0	114	318.0	0.0	2	140	0.0	
4	55.0	1	0	160	289.0	0.0	0	145	0.0	

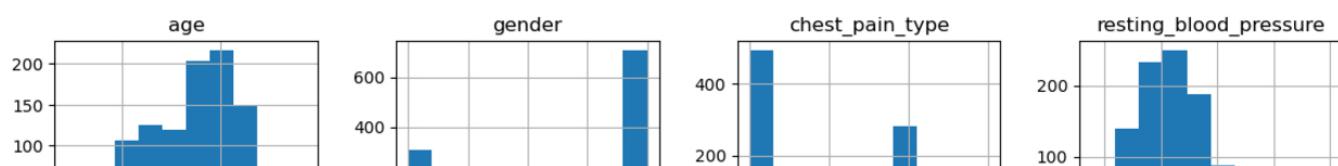
[36]: #count missing values in each attribute
missing = pd.DataFrame({'missing': data.isnull().sum()})
missing

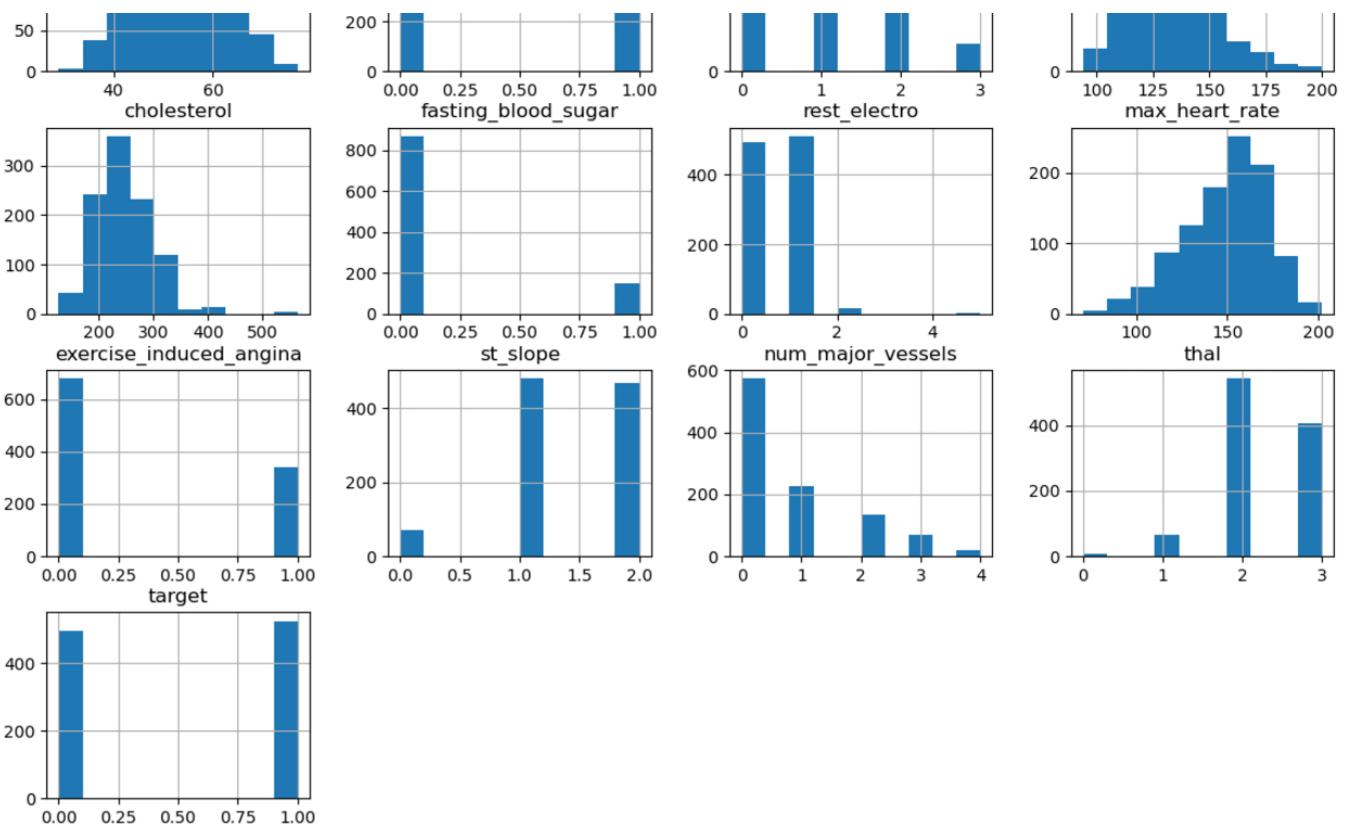
[36]:

	missing
age	0
gender	0
chest_pain_type	0
resting_blood_pressure	0
cholesterol	0
fasting_blood_sugar	0
rest_electro	0
max_heart_rate	0
exercise_induced_angina	0
st_depression	0
st_slope	0
num_major_vessels	0
thal	0
target	0

[37]: import matplotlib.pyplot as plt

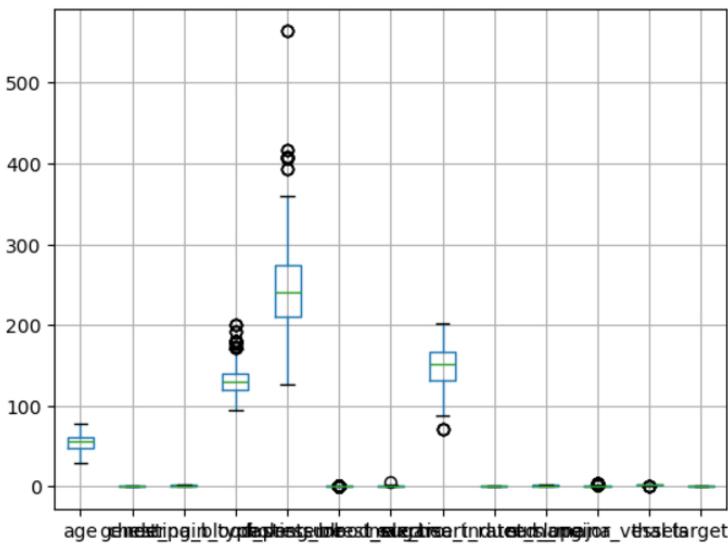
Generate histograms for each column
data.hist(figsize=(14, 10))
plt.show()





```
[38]: data.boxplot()
```

```
[38]: <Axes: >
```



```
[39]: data.corr() correlation between inputs
```

	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina	st_depression	st_slope	num_major_vessels	thal
age	1.000000	-0.106591	-0.069117	0.271897	0.226052	0.126386	-0.141009	-0.389869					
gender	-0.106591	1.000000	-0.042919	-0.083037	-0.193476	0.030504	-0.049456	-0.051202					
chest_pain_type	-0.069117	-0.042919	1.000000	0.037958	0.037958	-0.080453	0.084900	0.033273	0.305136				
resting_blood_pressure	0.271897	-0.083037	0.037958	1.000000	0.132058	0.182737	-0.133394	-0.041252					
cholesterol	0.226052	-0.193476	-0.080453	0.132058	1.000000	0.023048	-0.149652	-0.020255					
fasting_blood_sugar	0.126386	0.030504	0.084900	0.182737	0.023048	1.000000	-0.100011	-0.007794					
rest_electro	-0.141009	-0.049456	0.033273	-0.133394	-0.149652	-0.100011	1.000000	0.057245					
max_heart_rate	-0.389869	-0.051202	0.305136	-0.041252	-0.020255	-0.007794	0.057245	1.000000					
exercise_induced_angina	0.091559	0.139062	-0.402207	0.056057	0.066689	0.044622	-0.068052	-0.381805					
st_depression	0.210532	0.085781	-0.175246	0.186931	0.066780	0.003614	-0.053729	-0.350413					
st_slope	-0.166943	-0.025784	0.127964	-0.120529	-0.017228	-0.056255	0.089820	0.394794					
num_major_vessels	0.272324	0.112788	-0.176758	0.105512	0.073088	0.140798	-0.074744	-0.208526					
thal	0.073728	0.198584	-0.162205	0.056386	0.102687	-0.046563	-0.025777	-0.098842					

```
target -0.227754 -0.280452 0.433408 -0.136193 -0.102280 -0.037566 0.119488 0.423103
```

```
[40]: # Step 4: Feature Engineering  
# Create a new feature based on age  
bin_labels = ['Child', 'Teenager', 'Young Adult', 'Middle Aged', 'Senior']  
data['age_group'] = pd.cut(data['age'], bins=[0, 12, 19, 35, 55, 100], labels=bin_labels)  
  
# new feature  
data[['age', 'age_group']]
```

```
[40]: age age_group
```

0	52.0	Middle Aged
1	62.0	Senior
2	58.0	Senior
3	58.0	Senior
4	55.0	Middle Aged
...
1015	59.0	Senior
1016	60.0	Senior
1017	47.0	Middle Aged
1018	50.0	Middle Aged
1019	54.0	Middle Aged

1019 rows × 2 columns

```
[41]: # Create a new feature based on cholesterol 3 bins 0-200 , 200-240 , 240-1000  
bin_labels = ['Good', 'Borderline', 'High']  
data['chol_level'] = pd.cut(data['cholesterol'], bins=[0, 200, 240, 1000], labels=bin_labels)  
  
data[['cholesterol', 'chol_level']]
```

```
[41]: cholesterol chol_level
```

0	212.0	Borderline
1	294.0	High
2	248.0	High
3	318.0	High
4	289.0	High
...
1015	221.0	Borderline
1016	258.0	High
1017	275.0	High
1018	254.0	High
1019	188.0	Good

1019 rows × 2 columns

```
[42]: # step 5 : Data Transformation  
from sklearn.preprocessing import MinMaxScaler  
  
# 1. scaling/normalization numerical Features  
minMaxScaler = MinMaxScaler()  
  
# numeric features to normaliz  
numeric_features = ['resting_blood_pressure', 'cholesterol', 'max_heart_rate']  
  
# fit scaler to numeric features  
minMaxScaler.fit(data[numeric_features])  
  
# transform the numeric features  
data[numeric_features] = minMaxScaler.fit_transform(data[numeric_features])  
  
data.head()
```

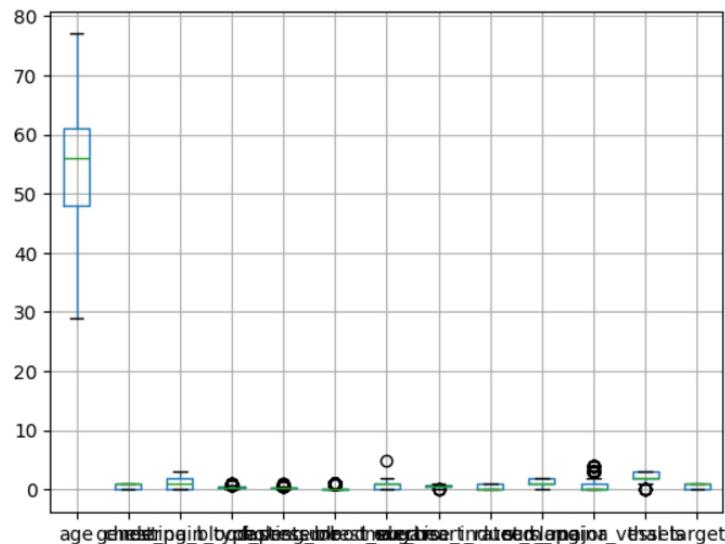
```
[42]: age gender chest_pain_type resting_blood_pressure cholesterol fasting_blood_sugar rest_electro max_heart_rate exercise_induced_angina st_depre
```

0	52.0	1	0	0.292453	0.196347	0.0	1	0.740458	0.0
1	62.0	0	0	0.415094	0.383562	1.0	1	0.267176	0.0

2	58.0	0	0	0.245283	0.278539	0.0	0	0.389313	0.0
3	58.0	1	0	0.188679	0.438356	0.0	2	0.526718	0.0
4	55.0	1	0	0.622642	0.372146	0.0	0	0.564885	0.0

```
[43]: data.boxplot()
```

```
[43]: <Axes: >
```

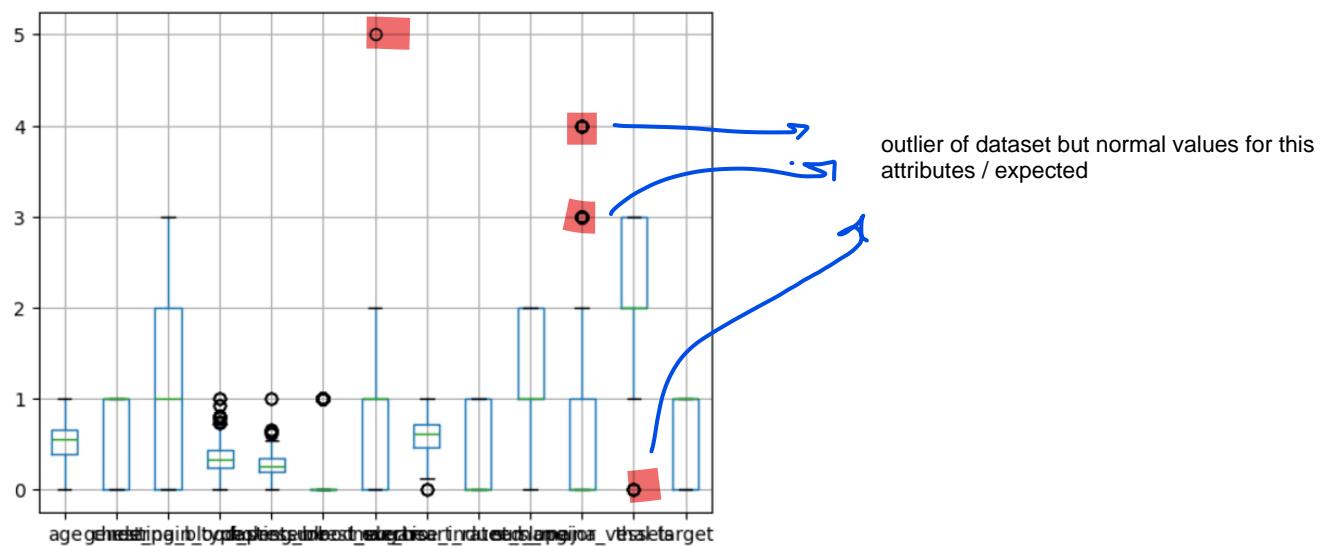


```
[44]: # normalize age
data['age'] = minMaxScaler.fit_transform(data['age'].values.reshape(-1, 1))
data.head()
```

	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_ecro	max_heart_rate	exercise_induced_angina	st_c
0	0.479167	1	0	0.292453	0.196347	0.0	1	0.740458	0.0	
1	0.687500	0	0	0.415094	0.383562	1.0	1	0.267176	0.0	
2	0.604167	0	0	0.245283	0.278539	0.0	0	0.389313	0.0	
3	0.604167	1	0	0.188679	0.438356	0.0	2	0.526718	0.0	
4	0.541667	1	0	0.622642	0.372146	0.0	0	0.564885	0.0	

```
[45]: data.boxplot()
```

```
[45]: <Axes: >
```



```
[46]: # Calculate the IQR for rest_ecro - interquartile range measure of spread of the data
```

```
Q1 = data['rest_ecro'].quantile(0.25) # Q1 represents the value below which 25% of the data points fall.
Q3 = data['rest_ecro'].quantile(0.75) # Q3 represents the value below which 75% of the data points fall.
```

```
IQR = Q3 - Q1
```

```
# Lower and upper bounds for outliers [-1 min +1 max]
```

again Data Cleaning

```

    lower_bound = Q1 - 1 * IQR
    upper_bound = Q3 + 1 * IQR

    # outlier rows
    outliers = data[(data['rest_electro'] < lower_bound) | (data['rest_electro'] > upper_bound)] #outside the range

    outliers

```

[46]:

	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina	st_slope
973	0.3125	1	0	0.150943	0.1621	0.0	5	0.80916	0.0	0.0

[47]:

```

# Drop the row with index 973
data.drop([973], inplace=True, axis=0)
data.reset_index(drop=True, inplace=True)

after_row = data.loc[973]
print("Row with index 973 after delete previous:")
print(after_row)

```

Row with index 973 after delete previous:

age	0.583333
gender	1
chest_pain_type	0
resting_blood_pressure	0.433962
cholesterol	0.150685
fasting_blood_sugar	0.0
rest_electro	1
max_heart_rate	0.587786
exercise_induced_angina	0.0
st_depression	0.4
st_slope	1
num_major_vessels	0
thal	1
target	1
age_group	Senior
chol_level	Good

Name: 973, dtype: object

[48]:

```

# set of acceptable values
acceptable_values = {0, 1, 2, 3}

# outliers
outliers = data[data['num_major_vessels'].isin(acceptable_values)]

outliers

```

here num-major_vessels:

can be 4 in special case .. used to flag special or severe cases that are outside the typical range.
So this values normal after research on ca attribute

[48]:

	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina	st_slope
48	0.187500	1	2	0.415094	0.111872	0.0	1	0.778626	0.0	0.0
79	0.187500	1	2	0.415094	0.111872	0.0	1	0.778626	0.0	0.0
124	0.479167	1	2	0.415094	0.221461	0.0	1	0.748092	0.0	0.0
203	0.187500	1	2	0.415094	0.111872	0.0	1	0.778626	0.0	0.0
237	0.187500	1	2	0.415094	0.111872	0.0	1	0.778626	0.0	0.0
285	0.479167	1	2	0.415094	0.221461	0.0	1	0.748092	0.0	0.0
335	0.187500	1	2	0.415094	0.111872	0.0	1	0.778626	0.0	0.0
343	0.291667	1	0	0.358491	0.276256	1.0	0	0.549618	1.0	0.0
411	0.479167	1	2	0.415094	0.221461	0.0	1	0.748092	0.0	0.0
422	0.291667	1	0	0.358491	0.276256	1.0	0	0.549618	1.0	0.0
459	0.187500	1	2	0.415094	0.111872	0.0	1	0.778626	0.0	0.0
515	0.604167	1	1	0.292453	0.214612	0.0	1	0.557252	0.0	0.0
591	0.187500	1	2	0.415094	0.111872	0.0	1	0.778626	0.0	0.0
737	0.604167	1	1	0.292453	0.214612	0.0	1	0.557252	0.0	0.0
743	0.604167	1	1	0.292453	0.214612	0.0	1	0.557252	0.0	0.0
825	0.604167	1	1	0.292453	0.214612	0.0	1	0.557252	0.0	0.0

823	0.004107	1	1	0.292453	0.214012	0.0	1	0.331252	0.0
964	0.187500	1	2	0.415094	0.111872	0.0	1	0.778626	0.0
986	0.291667	1	0	0.358491	0.276256	1.0	0	0.549618	1.0

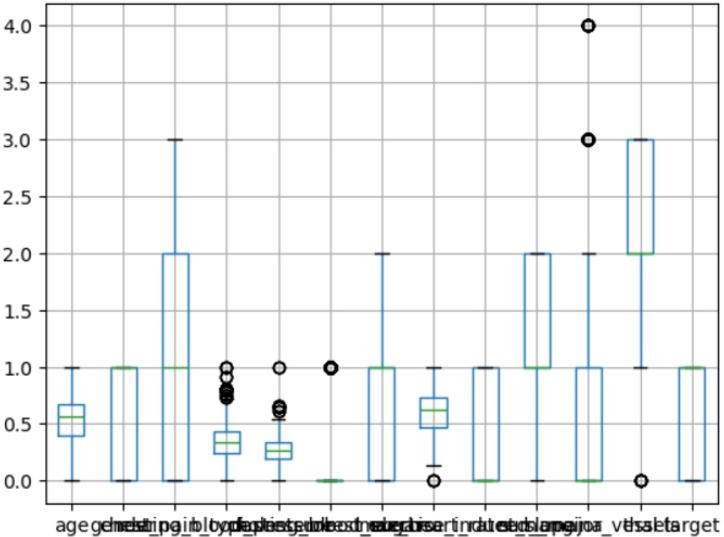
[49]: data

	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina
0	0.479167	1	0	0.292453	0.196347	0.0	1	0.740458	0.0
1	0.687500	0	0	0.415094	0.383562	1.0	1	0.267176	0.0
2	0.604167	0	0	0.245283	0.278539	0.0	0	0.389313	0.0
3	0.604167	1	0	0.188679	0.438356	0.0	2	0.526718	0.0
4	0.541667	1	0	0.622642	0.372146	0.0	0	0.564885	0.0
...
1013	0.625000	1	1	0.433962	0.216895	0.0	1	0.709924	1.0
1014	0.645833	1	0	0.292453	0.301370	0.0	0	0.534351	1.0
1015	0.375000	1	0	0.150943	0.340183	0.0	0	0.358779	1.0
1016	0.437500	0	0	0.150943	0.292237	0.0	0	0.671756	0.0
1017	0.520833	1	0	0.245283	0.141553	0.0	1	0.320611	0.0

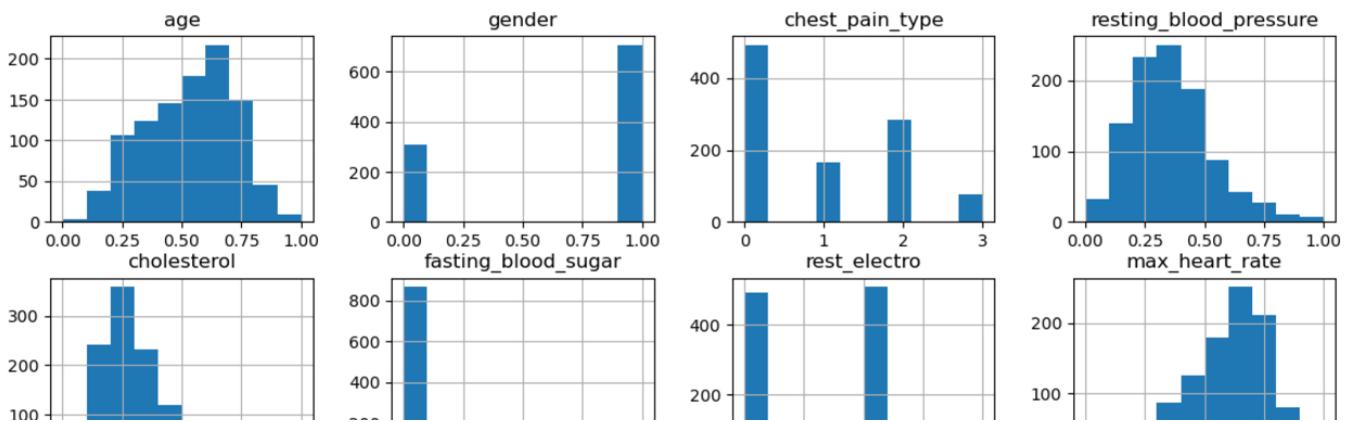
1018 rows × 16 columns

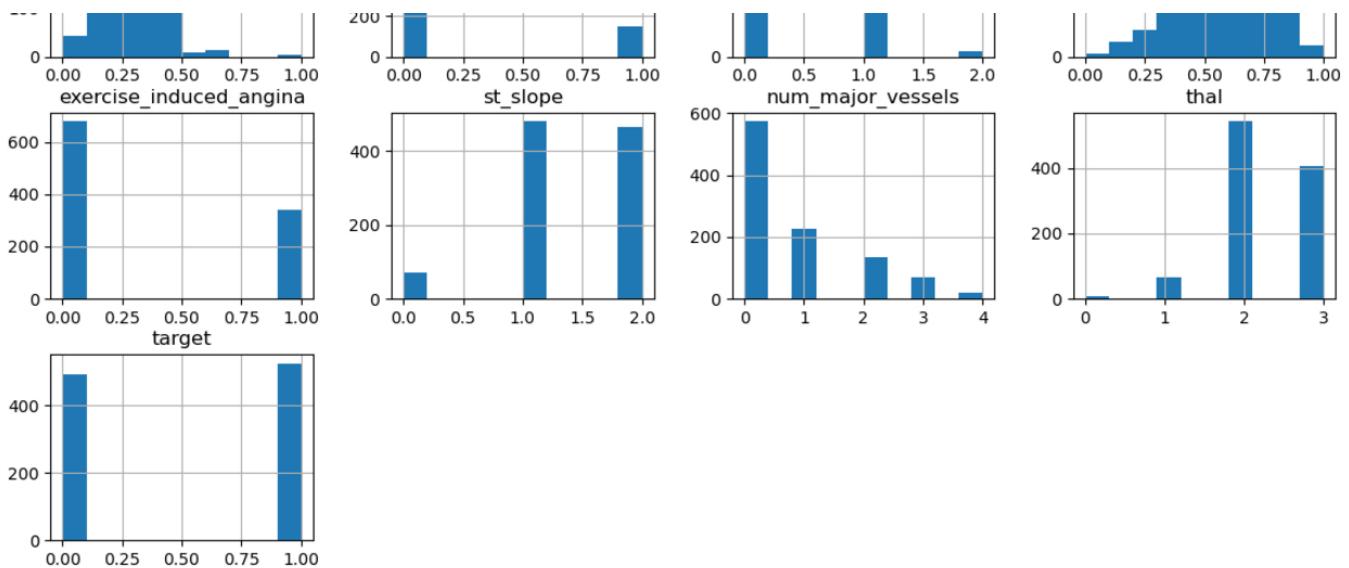
[50]: data.boxplot()

[50]: <Axes: >



```
[51]: # Generate histograms for each column
data.hist(figsize=(14, 10))
plt.show()
```





```
[52]: #2 Categorical Encoding - Ordinal/Label Encoder
```

```
from sklearn.preprocessing import LabelEncoder

# Encode non-numeric columns
encoder = LabelEncoder()
data['st_depression'] = encoder.fit_transform(data['st_depression'])
data['age_group'] = encoder.fit_transform(data['age_group'])
data['chol_level'] = encoder.fit_transform(data['chol_level'])
```

```
[53]: from sklearn.model_selection import train_test_split as tts
```

Split data , train model and test

```
# Split the data into training and testing sets
trainSet, testSet = tts(data, test_size=0.3)
```

```
trainSet
```

	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina	s
795	0.708333	1	3	0.481132	0.244292	1.0	0	0.603053	0.0	
290	0.791667	1	0	0.056604	0.394977	0.0	0	0.412214	1.0	
906	0.125000	1	0	0.245283	0.164384	0.0	1	0.450382	1.0	
533	0.729167	1	3	0.716981	0.230594	0.0	0	0.641221	0.0	
703	0.333333	1	3	0.150943	0.315068	0.0	1	0.465649	0.0	
...
150	0.708333	1	0	0.339623	0.465753	1.0	0	0.465649	1.0	
454	0.833333	1	2	0.433962	0.292237	0.0	0	0.572519	0.0	
461	0.375000	1	0	0.150943	0.340183	0.0	0	0.358779	1.0	
714	0.729167	1	2	0.433962	0.477169	0.0	1	0.664122	0.0	
496	0.354167	0	2	0.452830	0.116438	0.0	0	0.679389	1.0	

712 rows × 16 columns

```
[54]: testSet
```

	age	gender	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_electro	max_heart_rate	exercise_induced_angina	s
716	0.791667	0	2	0.547170	0.344749	0.0	1	0.770992	0.0	
422	0.291667	1	0	0.358491	0.276256	1.0	0	0.549618	1.0	
210	0.416667	1	1	0.339623	0.319635	0.0	1	0.763359	0.0	
385	0.333333	1	3	0.150943	0.315068	0.0	1	0.465649	0.0	
847	0.791667	1	0	0.245283	0.235160	0.0	0	0.442748	1.0	
...
121	0.645833	0	3	0.528302	0.260274	0.0	1	0.763359	0.0	
178	0.270833	1	2	0.339623	0.123288	0.0	1	0.603053	0.0	
215	0.583333	0	1	0.339623	0.251142	0.0	0	0.786260	0.0	
755	0.416667	0	0	0.339623	0.326484	0.0	1	0.702290	0.0	
167	0.562500	1	1	0.245283	0.260274	0.0	1	0.748092	0.0	

306 rows × 16 columns

```
[55]: testSet.to_csv('testSet.csv')
trainSet.to_csv('trainSet.csv')
```

```
[56]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1018 entries, 0 to 1017
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   age              1018 non-null    float64
 1   gender            1018 non-null    int64  
 2   chest_pain_type  1018 non-null    int64  
 3   resting_blood_pressure  1018 non-null    float64
 4   cholesterol       1018 non-null    float64
 5   fasting_blood_sugar  1018 non-null    float64
 6   rest_electro       1018 non-null    int64  
 7   max_heart_rate    1018 non-null    float64
 8   exercise_induced_angina  1018 non-null    float64
 9   st_depression      1018 non-null    int32  
 10  st_slope            1018 non-null    int64  
 11  num_major_vessels  1018 non-null    int64  
 12  thal               1018 non-null    int64  
 13  target              1018 non-null    int32  
 14  age_group           1018 non-null    int32  
 15  chol_level          1018 non-null    int32  
dtypes: float64(6), int32(4), int64(6)
memory usage: 111.5 KB
```

```
[57]: from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

addition: calc accuracy, Recall & Precision

```
# we should encode non-numeric columns previously LogisticRegression deal with numerics
# Separate features and target
train_features = trainSet.drop('target', axis=1)
train_target = trainSet['target']
test_features = testSet.drop('target', axis=1)
test_target = testSet['target']

# Train the model
model = LogisticRegression(max_iter=1000)
model.fit(train_features, train_target)

# Predict on the test set
predictions = model.predict(test_features)

# Calculate scores as percentages
accuracy = accuracy_score(test_target, predictions) * 100
precision = precision_score(test_target, predictions) * 100
recall = recall_score(test_target, predictions) * 100
```

```
# Print scores
print(f"Accuracy: {accuracy:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"Recall: {recall:.2f}%")
```

```
Accuracy: 83.33%
Precision: 79.77%
Recall: 89.61%
```



