Heart Diseases Classification using SVM

May 16, 2020

Data Exploration

[1]:
```
# importing require libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
```

[2]:
```
# reading dataset
train_df=pd.read_csv('data/Heart_train.csv')
train_df.head()
```

[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	67	1	2	152	212	0	0	150	0	0.8	1	0	3	0
1	53	1	2	130	246	1	0	173	0	0.0	2	3	2	1
2	61	1	3	134	234	0	1	145	0	2.6	1	2	2	0
3	45	1	1	128	308	0	0	170	0	0.0	2	0	2	1
4	50	1	0	144	200	0	0	126	1	0.9	1	0	3	0

[3]:
```
test_df = pd.read_csv('data/Heart_test.csv')
test_df.head()
```

[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
0	58	1	1	120	284	0	0	160	0	1.8	1
1	52	1	0	112	230	0	1	160	0	0.0	2
2	42	0	2	120	209	0	1	173	0	0.0	1
3	55	1	1	130	262	0	1	155	0	0.0	2
4	53	0	0	130	264	0	0	143	0	0.4	1

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

[4]:
```
# checking the shape of dataset
print('train_df shape: ', train_df.shape)
print('test_df shape: ', test_df.shape)
```

```
train_df shape:  (242, 14)
test_df shape:  (61, 14)
```

[5]:
```
# checking the null values
train_df.isnull().sum()
```

[5]:
```
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
```

[6]:
```
test_df.isnull().sum()
```

[6]:
```
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
```

[7]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 242 entries, 0 to 241
Data columns (total 14 columns):
age         242 non-null int64
sex         242 non-null int64
cp          242 non-null int64
trestbps    242 non-null int64
chol        242 non-null int64
fbs         242 non-null int64
restecg     242 non-null int64
thalach     242 non-null int64
exang       242 non-null int64
oldpeak     242 non-null float64
slope       242 non-null int64
ca          242 non-null int64
thal        242 non-null int64
target      242 non-null int64
dtypes: float64(1), int64(13)
memory usage: 26.6 KB
```

[8]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61 entries, 0 to 60
Data columns (total 14 columns):
age         61 non-null int64
sex         61 non-null int64
cp          61 non-null int64
trestbps    61 non-null int64
chol        61 non-null int64
fbs         61 non-null int64
restecg     61 non-null int64
thalach     61 non-null int64
exang       61 non-null int64
oldpeak     61 non-null float64
slope       61 non-null int64
ca          61 non-null int64
thal        61 non-null int64
target      61 non-null int64
dtypes: float64(1), int64(13)
memory usage: 6.8 KB
```

[9]:
```
# columns
print(train_df.columns)
```

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

[10]:
```
train_df.describe()
```

[10]:

	age	sex	cp	trestbps	chol	fbs
count	242.000000	242.000000	242.000000	242.000000	242.000000	242.000000
mean	54.603306	0.690083	1.024793	132.243802	244.760331	0.157025
std	9.200829	0.463418	1.034370	17.524212	52.774558	0.364578
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	210.250000	0.000000
50%	56.000000	1.000000	1.000000	130.000000	240.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	272.500000	0.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000

	restecg	thalach	exang	oldpeak	slope	ca
count	242.000000	242.000000	242.000000	242.00000	242.000000	242.000000
mean	0.520661	149.809917	0.326446	1.02686	1.396694	0.760331
std	0.508829	23.129193	0.469885	1.15571	0.617576	1.034868
min	0.000000	71.000000	0.000000	0.00000	0.000000	0.000000
25%	0.000000	133.000000	0.000000	0.00000	1.000000	0.000000
50%	1.000000	152.000000	0.000000	0.80000	1.000000	0.000000
75%	1.000000	166.750000	1.000000	1.60000	2.000000	1.000000
max	2.000000	202.000000	1.000000	6.20000	2.000000	4.000000

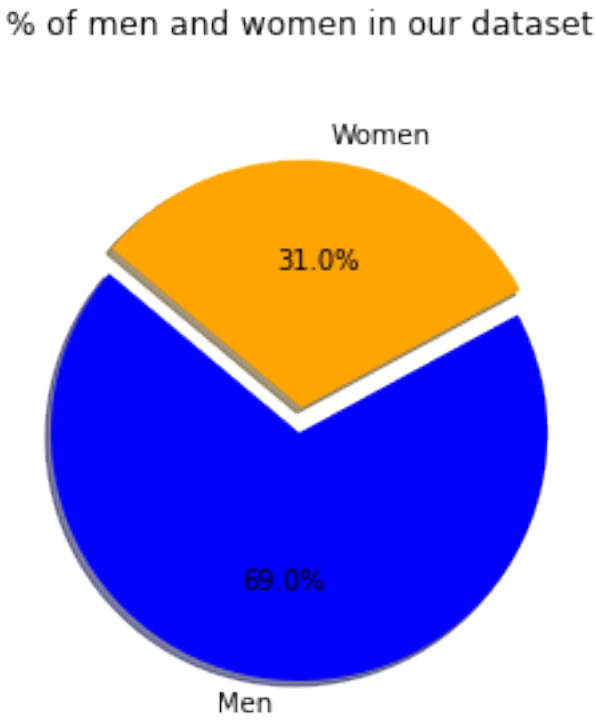
	thal	target
count	242.000000	242.000000
mean	2.309917	0.545455
std	0.617034	0.498962
min	0.000000	0.000000
25%	2.000000	0.000000
50%	2.000000	1.000000
75%	3.000000	1.000000
max	3.000000	1.000000

Visualization

[11]:
```
# checking the number of male and female
train_df.sex.value_counts()
```

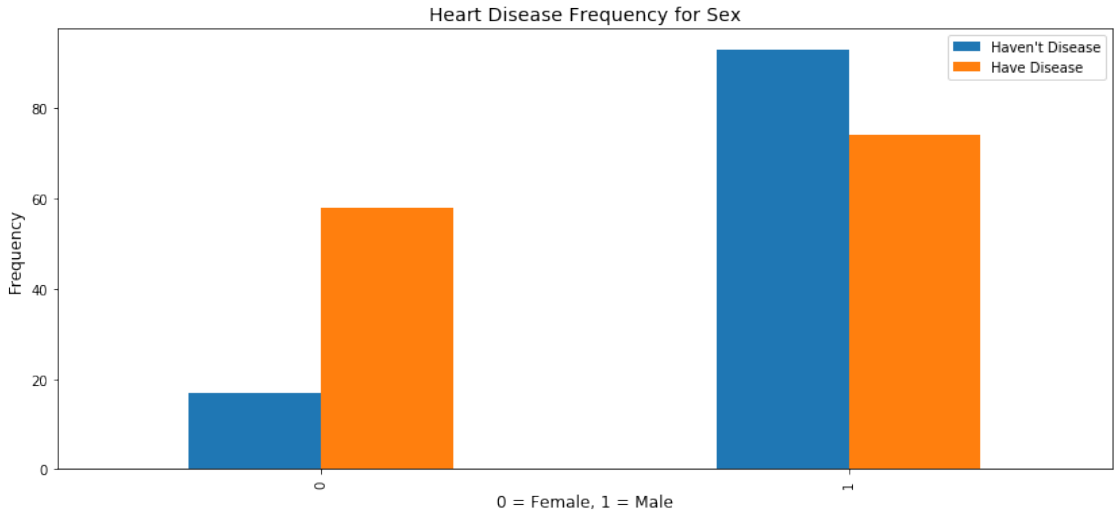
[11]:
```
1    167
0     75
Name: sex, dtype: int64
```

[12]:
```
label = ['Men', 'Women']
sizes = [167, 75]
colors = ['blue','orange']
explode = (0.1, 0)

# Plot
plt.pie(sizes, explode=explode, labels=label, colors=colors,autopct='%1.1f%%',␣
 ↪shadow=True, startangle=140)

plt.title('% of men and women in our dataset\n\n')
plt.axis('equal')
plt.show()
```



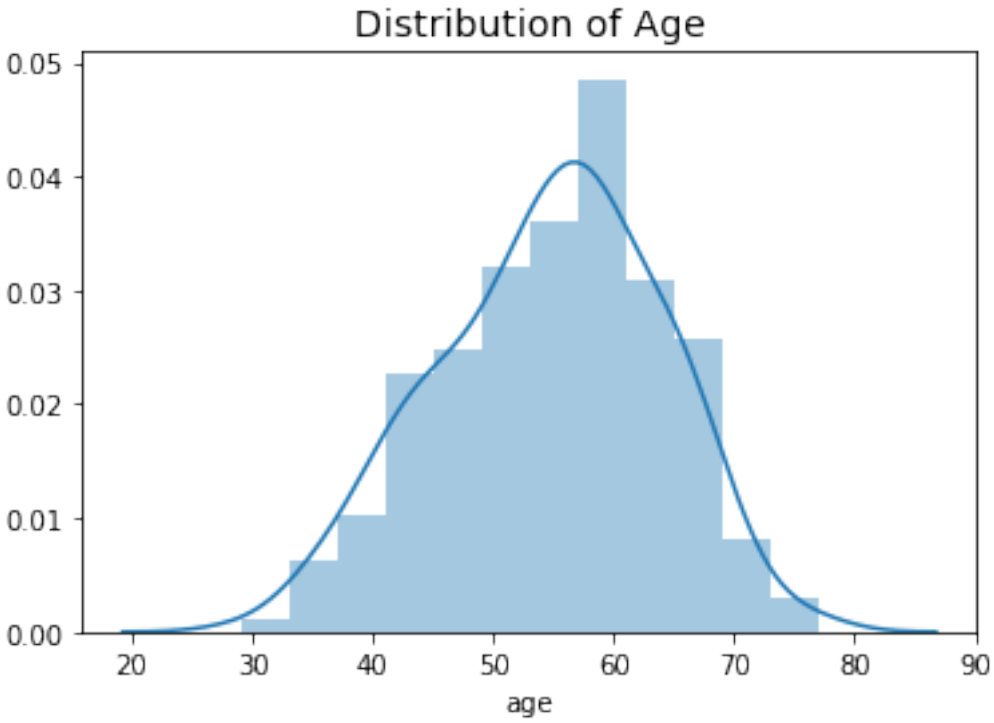
[13]:
```
# heart diseases frequency with male or female
crossTab = pd.crosstab(train_df.sex,train_df.target)
crossTab.plot(kind="bar",figsize=(14,6))

plt.title('Heart Disease Frequency for Sex', fontsize=14)

plt.xlabel('0 = Female, 1 = Male', fontsize = 12)
plt.ylabel('Frequency', fontsize = 12)

plt.legend(["Haven't Disease", "Have Disease"])
```

```
plt.show()
```



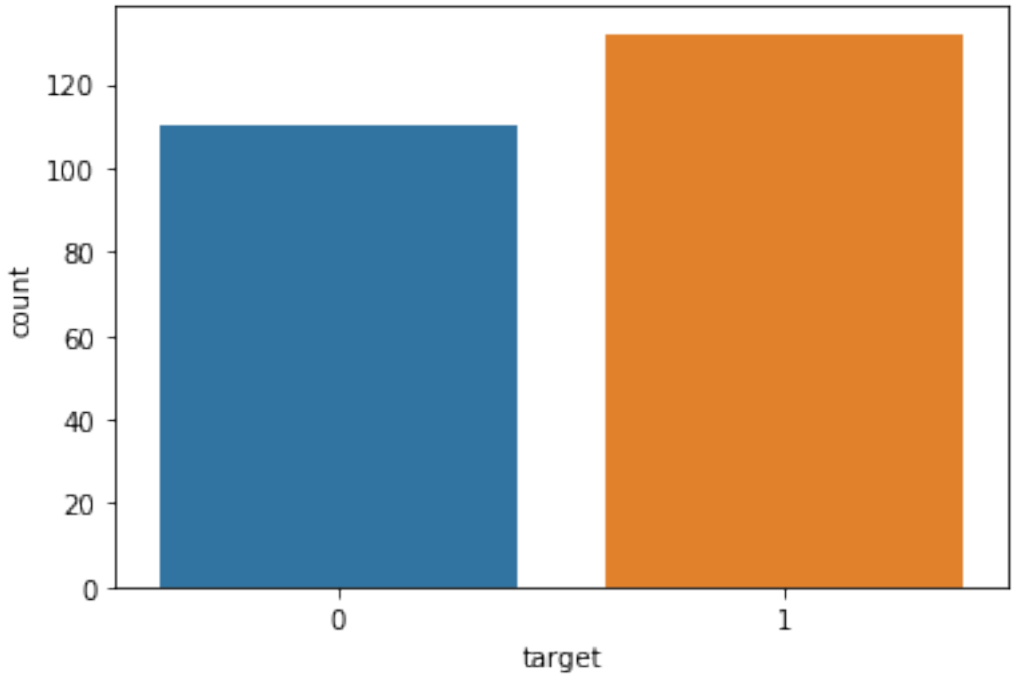
[14]:
```
# distribution of age in the patient

sns.distplot(train_df['age'])
plt.title('Distribution of Age', fontsize=14)
```

[14]: Text(0.5, 1.0, 'Distribution of Age')



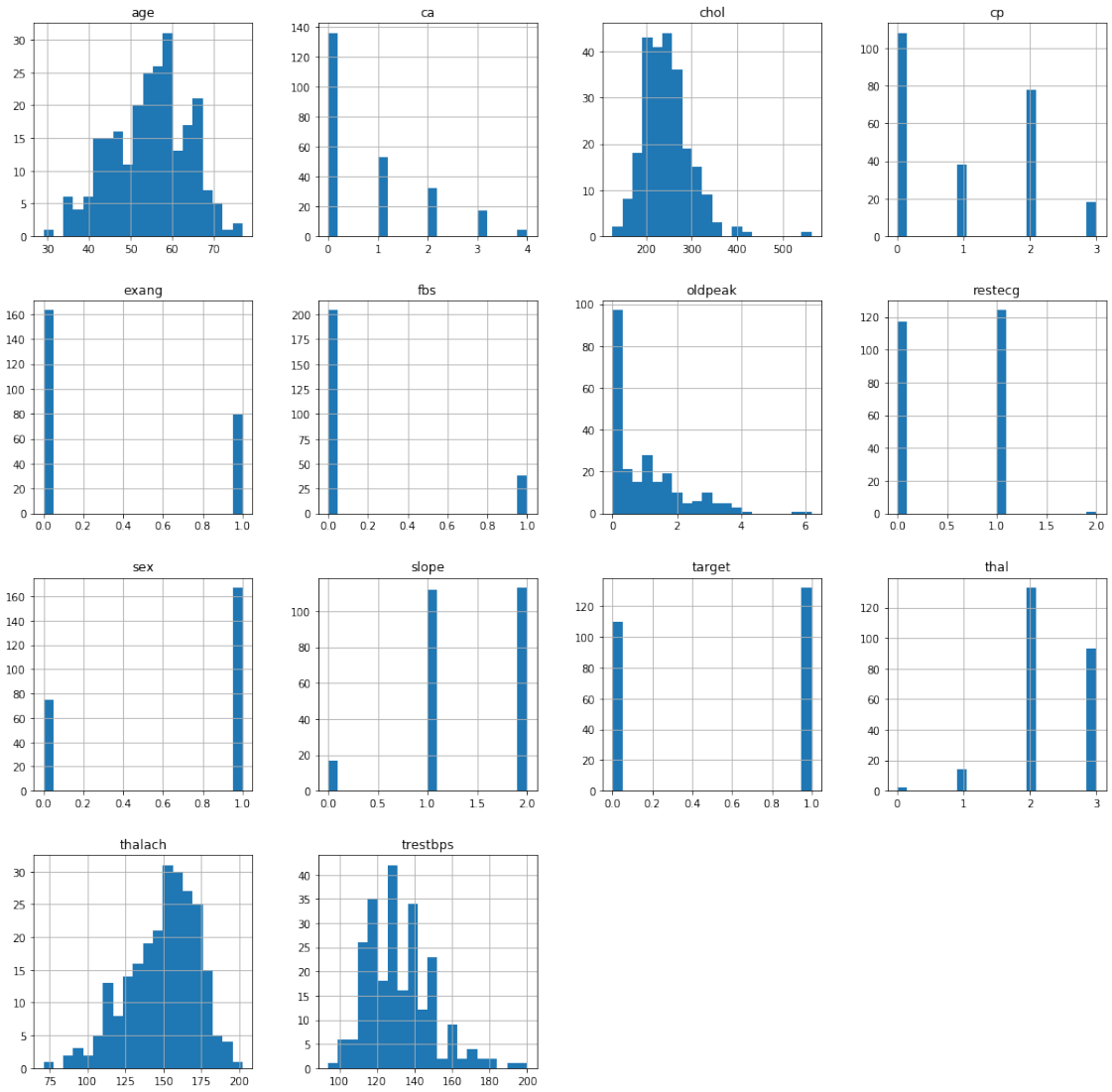
[15]:
```
# checking the target number
sns.countplot(train_df['target'])
```

[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb01f143090>
```

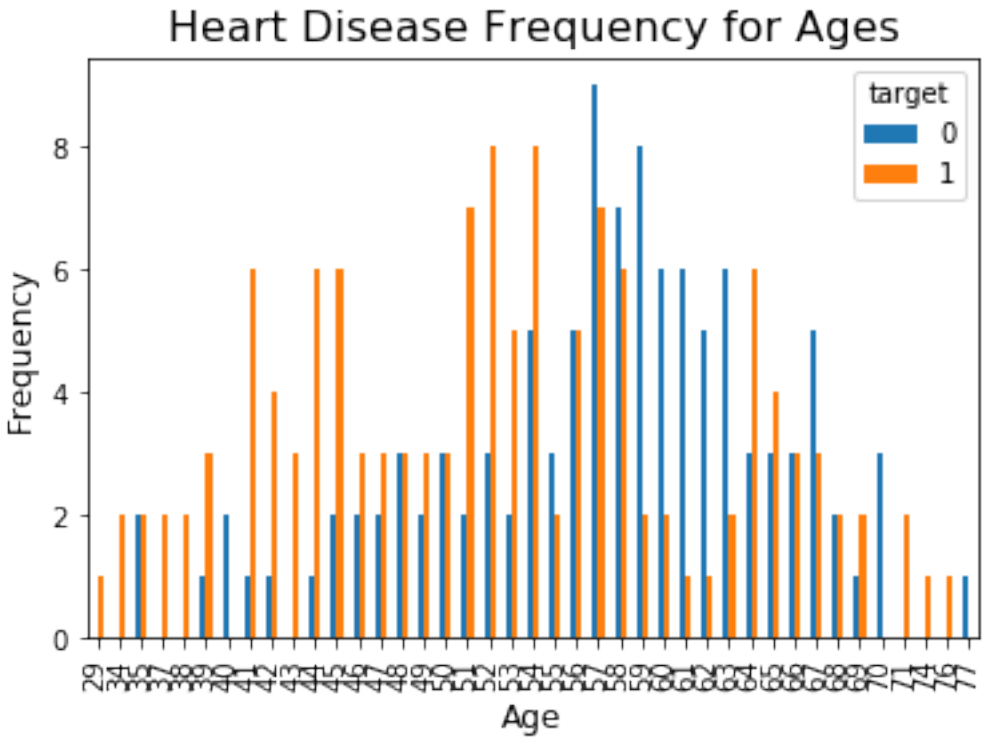


[16]:
```
# histogram of every datapoint
fig = plt.figure(figsize=(18, 18))
ax = fig.gca()

train_df.hist(ax=ax, bins=20)

plt.show()
```

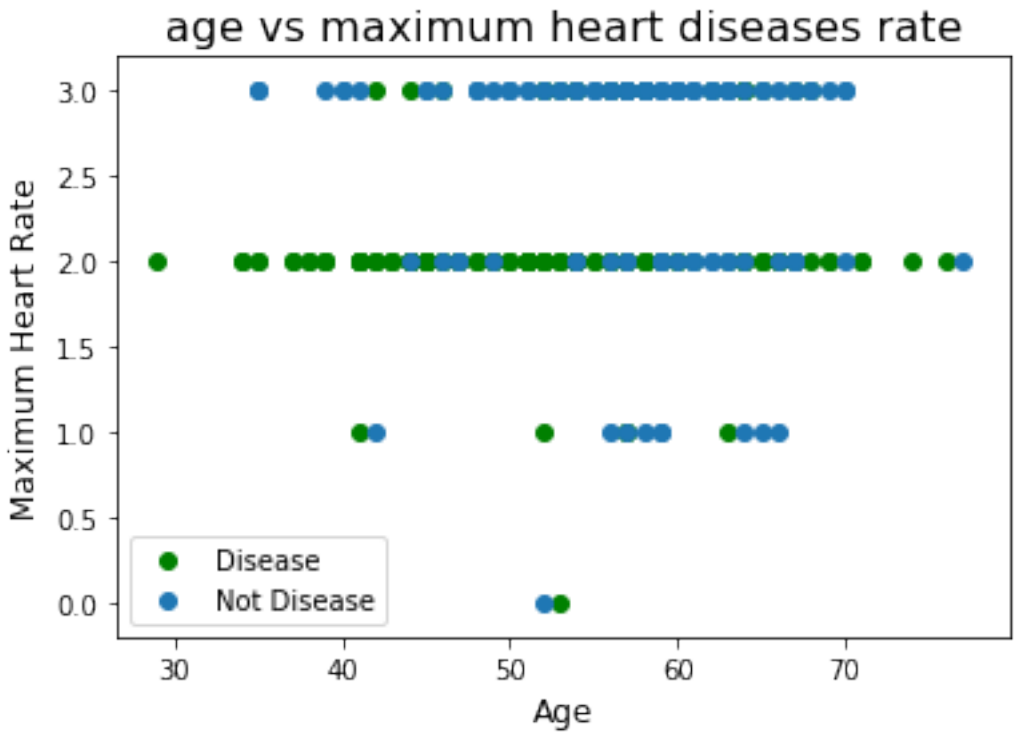
```
/home/code_monkey/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
UserWarning: To output multiple subplots, the figure containing the passed axes
is being cleared
  """
```



[17]:
```
# heart disease frequecy by all age
ctab = pd.crosstab(train_df.age, train_df.target)
ctab.plot(kind="bar")

plt.title('Heart Disease Frequency for Ages', fontsize=16)
plt.xlabel('Age', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

plt.show()
```



[18]:
```
# age vs maximum heart diseases rate
plt.scatter(x=train_df.age[train_df.target==1], y=train_df.thal[(train_df.
 ↪target==1)], c="green")
plt.scatter(x=train_df.age[train_df.target==0], y=train_df.thal[(train_df.
 ↪target==0)])

plt.title('age vs maximum heart diseases rate', fontsize=16)
plt.xlabel("Age", fontsize=12)
plt.ylabel("Maximum Heart Rate", fontsize=12)

plt.legend(["Disease", "Not Disease"])
plt.show()
```



Feature Extraction and Preprocessing

[19]:
```
# checking the nay row is duplicate or not
dup_row = train_df[train_df.duplicated(keep=False)]
dup_row
```

[19]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
98	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1
144	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1

[20]:
```
# removing duplicate row
train_df.drop(144, inplace=True)
print('train_df shape:', train_df.shape)
```

```
train_df shape: (241, 14)
```

[21]:
```
train_df.columns
```

[21]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[22]:
```
# seperating train set and test set
X_train = train_df.drop(columns='target')
y_train = train_df['target']

X_test = test_df.drop(columns='target')
y_test = test_df['target']

print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)

print('\nX_test shape:', X_test.shape)
print('y_test shape:', y_test.shape)
```

```
X_train shape: (241, 13)
y_train shape: (241,)

X_test shape: (61, 13)
y_test shape: (61,)
```

[23]:
```
# seperating categorical column and numberical columns
categorical_columns = ['cp', 'exang', 'slope', 'thal']
non_categorical_columns = [c for c in X_train.columns if c not in␣
↪categorical_columns]
```

Using One-Hot Encoding.

[24]:
```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(handle_unknown='ignore')
encoder.fit(X_train[categorical_columns])
```

[24]:
```
OneHotEncoder(handle_unknown='ignore')
```

[25]:
```
column_names = encoder.get_feature_names(input_features=categorical_columns)
column_names
```

[25]:
```
array(['cp_0', 'cp_1', 'cp_2', 'cp_3', 'exang_0', 'exang_1', 'slope_0',
       'slope_1', 'slope_2', 'thal_0', 'thal_1', 'thal_2', 'thal_3'],
      dtype=object)
```

[26]:
```
def encode_dataframe(df, column_names, index):
    category_encoded = encoder.transform(df[categorical_columns])
    category_encoded_df = pd.DataFrame(category_encoded.todense(),␣
↪columns=column_names, index=index)
```

```
    return pd.concat([df[non_categorical_columns], category_encoded_df], axis=1)
```

[27]:
```
X_train_encoded = encode_dataframe(X_train, column_names, X_train.index)
X_train_encoded.drop(non_categorical_columns, inplace=True, axis=1)

X_train_encoded
```

[27]:

	cp_0	cp_1	cp_2	cp_3	exang_0	exang_1	slope_0	slope_1	slope_2	\
0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	
1	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	
2	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	
4	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	
..	…	…	…	…	…	…	…	…	…	
237	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	
238	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	
239	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	
240	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	
241	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	

	thal_0	thal_1	thal_2	thal_3
0	0.0	0.0	0.0	1.0
1	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0
3	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0
..	…	…	…	…
237	0.0	0.0	1.0	0.0
238	0.0	0.0	0.0	1.0
239	0.0	0.0	0.0	1.0
240	0.0	0.0	0.0	1.0
241	0.0	0.0	1.0	0.0

[241 rows x 13 columns]

[28]:
```
X_test_encoded = encode_dataframe(X_test, column_names, X_test.index)
X_test_encoded.drop(non_categorical_columns, inplace=True, axis=1)

X_test_encoded.head()
```

[28]:

	cp_0	cp_1	cp_2	cp_3	exang_0	exang_1	slope_0	slope_1	slope_2	\
0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	
1	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	
2	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	
4	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	

```
   thal_0  thal_1  thal_2  thal_3
0     0.0     0.0     1.0     0.0
1     0.0     0.0     1.0     0.0
2     0.0     0.0     1.0     0.0
3     0.0     0.0     1.0     0.0
4     0.0     0.0     1.0     0.0
```

[29]:
```
print('X_train_encoded shape:', X_train_encoded.shape)
print('X_test_encoded shape:', X_test_encoded.shape)
```

```
X_train_encoded shape: (241, 13)
X_test_encoded shape: (61, 13)
```

[30]:
```
# make the balance sampling
from collections import Counter
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
x_smote, y_smote = smote.fit_resample(X_train_encoded, y_train)
Counter(y_smote)
```

```
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.neighbors.base module is  deprecated in version 0.22 and will be removed
in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.neighbors. Anything that cannot be imported from
sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.ensemble.bagging module is  deprecated in version 0.22 and will be
removed in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.ensemble. Anything that cannot be imported from
sklearn.ensemble is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.ensemble.base module is  deprecated in version 0.22 and will be removed
in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.ensemble. Anything that cannot be imported from
sklearn.ensemble is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.ensemble.forest module is  deprecated in version 0.22 and will be
removed in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.ensemble. Anything that cannot be imported from
```

```
sklearn.ensemble is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.utils.testing module is  deprecated in version 0.22 and will be removed
in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.utils. Anything that cannot be imported from sklearn.utils
is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.metrics.classification module is  deprecated in version 0.22 and will be
removed in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.metrics. Anything that cannot be imported from
sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
/home/code_monkey/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:86: FutureWarning: Function safe_indexing
is deprecated; safe_indexing is deprecated in version 0.22 and will be removed
in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

[30]:
```
Counter({0: 131, 1: 131})
```

Grid Search

[31]:
```
grid_params={
    'kernel':('linear','rbf'),
    'C':(1,20)
}
```

[32]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer,f1_score

scorer = make_scorer(f1_score, average='micro')
clf = GridSearchCV(SVC(), grid_params, scoring=scorer)

clf.fit(X_train_encoded, y_train)
```

[32]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': (1, 20), 'kernel': ('linear', 'rbf')},
             scoring=make_scorer(f1_score, average=micro))
```

[33]:
```
best_score = clf.best_score_

print(best_score)
```

```
0.8216836734693876
```

[34]:
```
best_params = clf.best_params_

print(best_params)
```

```
{'C': 20, 'kernel': 'rbf'}
```

[35]:
```
# printing the best parameter
print("The best score is: {} with params: {}".format(clf.best_score_,clf.
 ↪best_params_))
```

```
The best score is: 0.8216836734693876 with params: {'C': 20, 'kernel': 'rbf'}
```

Model Evaluation

[36]:
```
from sklearn import metrics
model = SVC(random_state=1,kernel='linear',C=20)
model.fit(X_train,y_train)

y_pred = model.predict(X_test_encoded)
print(metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.11      0.04      0.05        28
           1       0.48      0.76      0.59        33

    accuracy                           0.43        61
   macro avg       0.30      0.40      0.32        61
weighted avg       0.31      0.43      0.34        61
```

[ ]:
```

```