Name - Nihalahmed Munir Barudwale

Project name - Prediction of Heart disease detection

Batch – Machine Learning With Python

Certificate Code – TCRIG02R28

CODE –

```
# -*- coding: utf-8 -*-
"""Interenship Project-2.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1Tt0h_wIEU1yCGHoTOFgfRNu8xQueBYH7

**Name** - Nihalahmed Munir Barudwale

**Project name** - Prediction of Heart disease detection
"""

#

"""**Import Required Libraries**"""

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline
import os
import warnings
warnings.filterwarnings('ignore')

"""**Import dataset**"""

dataset = pd.read_csv("/content/drive/MyDrive/TCR Internship
Project/heart.csv")

dataset

"""**Shape of dataset**"""

dataset.shape

"""**Some Operations on dataset**"""

dataset.head()

dataset.tail()

type(dataset)

dataset.info()

dataset.describe()
```

```
dataset.columns

"""**Checking total number of NA values** """

dataset.isna().sum()

"""**Checking total number of NULL values** """

dataset.isnull().sum()

#

"""**Exploratory Data Analysis (EDA)**

**Analysing the 'target' variable**
"""

dataset.target.describe()

dataset.target.unique()

#Checking correlation between columns
dataset.corr()["target"].abs().sort_values(ascending=False)

#This shows that most columns are moderately correlated with target, but
'fbs' is very weakly correlated.

dataset.target.value_counts()

"""Patient without heart problems - labeled as 0

Patient with heart problems - labeled as 1
"""

print("Percentage of patients without heart problems:
"+str(round(138*100/303,2)))
print("Percentage of patients with heart problems:
"+str(round(165*100/303,2)))

y = dataset["target"]
sns.countplot(y)

sns.distplot(dataset['target'])

"""**Analysing the 'sex' variable**"""

dataset.sex.value_counts()

sns.barplot(dataset["sex"],y)

"""We notice that the 'sex' feature has 2 unique features."""

sns.distplot(dataset['sex'])

"""**Analysing the 'cp' variable**"""

dataset.cp.value_counts()

sns.barplot(dataset["cp"],y)

""" The CP feature has values from 0 to 3.We notice, that chest pain of
```

```
'0', are much less likely to have heart problems"""

sns.distplot(dataset['cp'])

"""**Analysing the 'age' variable**"""

dataset.age.value_counts()

sns.barplot(dataset["age"],y)

"""Nothing special here."""

sns.distplot(dataset['age'])

"""**Analysing the 'trestbps' variable**"""

dataset.trestbps.value_counts()

sns.barplot(dataset["trestbps"],y)

"""Nothing special here."""

sns.distplot(dataset['trestbps'])

"""**Analysing the 'chol' variable**"""

dataset.chol.value_counts()

sns.barplot(dataset["chol"],y)

"""Nothing special here """

sns.distplot(dataset['chol'])

"""**Analysing the 'fbs' variable**"""

dataset.fbs.value_counts()

sns.barplot(dataset["fbs"],y)

"""Not much difference here."""

sns.distplot(dataset['fbs'])

"""**Analysing the 'restecg' variable**"""

dataset.restecg.value_counts()

sns.barplot(dataset["restecg"],y)

"""We realize that people with restecg '1' and '0' are much more likely to
have a heart disease than with restecg '2'"""

sns.distplot(dataset['restecg'])

"""**Analysing the 'exang' variable**"""

dataset.exang.value_counts()

sns.barplot(dataset["exang"],y)
```

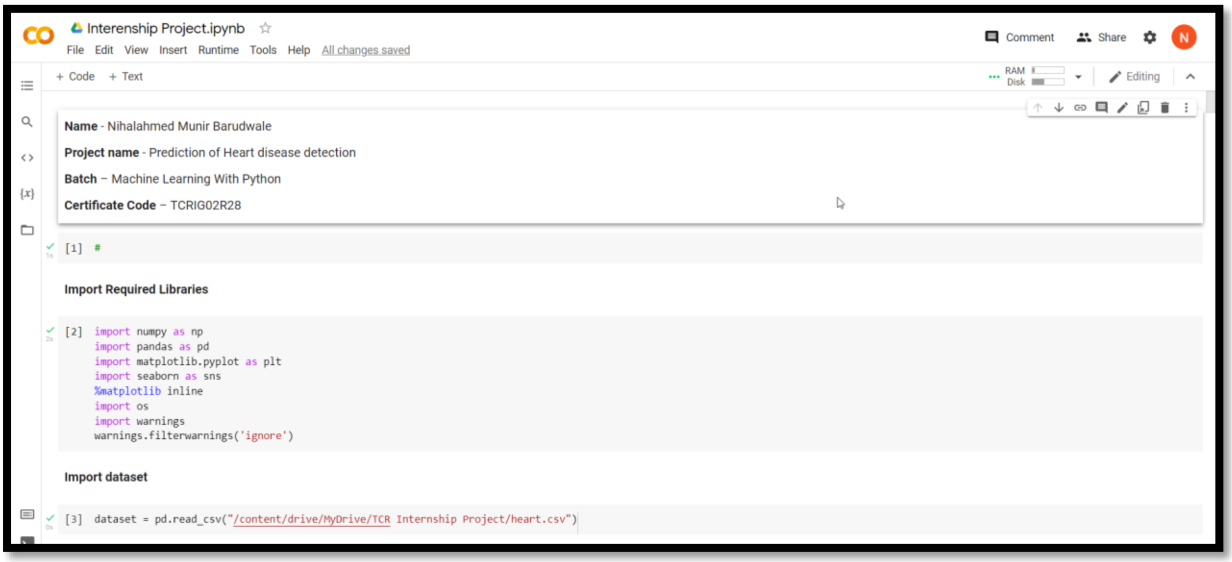
```
"""We notice here that people with exang=1, are much less likely to have
heart problems."""

sns.distplot(dataset['exang'])

"""**Analysing the 'slope' variable**"""

dataset.slope.value_counts()

sns.barplot(dataset["slope"],y)

"""We observe, that Slope '2' causes heart pain much more than Slope '0'
and '1'"""

sns.distplot(dataset['slope'])

"""**Analysing the 'ca' variable**"""

dataset.ca.value_counts()

sns.barplot(dataset["ca"],y)

"""We notice that ca=4 has large number of heart patients."""

sns.distplot(dataset['ca'])

"""**Analysing the 'thal' variable**"""

dataset.thal.value_counts()

sns.barplot(dataset["thal"],y)

"""thal=2 has large number of heart patients."""

sns.distplot(dataset['thal'])

"""**Get an overview distribution of each column**"""

dataset.hist(figsize=(16, 20), xlabelsize=8, ylabelsize=8)

sns.pairplot(dataset, hue='target')

"""**Correlation heatmap**"""

dataset.corr()

f, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(dataset.corr(),annot=True,cmap='PiYG',linewidths=.5)

"""**Splitting the data -
Train Test split**
"""

from sklearn.model_selection import train_test_split
x = dataset.drop("target",axis=1)
y= dataset["target"]

X_train,X_test,Y_train,Y_test =
train_test_split(x,y,test_size=0.20,random_state=0)

X_train.shape
```

```
X_test.shape

Y_train.shape

Y_test.shape

from sklearn.metrics import accuracy_score

"""**Logistic Regression**"""

from sklearn.linear_model import LogisticRegression
model_logistic_reg = LogisticRegression()
model_logistic_reg.fit(X_train,Y_train)
Y_pred_logistic_reg = model_logistic_reg.predict(X_test)

Y_pred_logistic_reg.shape

print("Predicted Values : ",Y_pred_logistic_reg)

Y_test[0:10] #You can check accuracy by observing predicted results and
test data.

accuracy_score_logistic_reg =
round(accuracy_score(Y_pred_logistic_reg,Y_test)*100,2)
print("The accuracy score achieved using Logistic Regression is:
"+str(accuracy_score_logistic_reg)+" %")

"""**SVM**"""

from sklearn import svm
model_svm = svm.SVC(kernel='linear')
model_svm.fit(X_train, Y_train)
Y_pred_svm = model_svm.predict(X_test)

Y_pred_svm.shape

print("Predicted Values : ",Y_pred_svm)

Y_test[0:10] #You can check accuracy by observing predicted results and
test data.

accuracy_score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)
print("The accuracy score achieved using Linear SVM is:
"+str(accuracy_score_svm)+" %")

"""**K Nearest Neighbors**"""

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
Y_pred_knn=knn.predict(X_test)

Y_pred_knn.shape

print("Predicted Values : ",Y_pred_knn)

Y_test[0:10] #You can check accuracy by observing predicted results and
test data.

accuracy_score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)
```

```
print("The accuracy score achieved using KNN is: 
"+str(accuracy_score_knn)+" %")

"""**Decision Tree**"""

from sklearn.tree import DecisionTreeClassifier
max_accuracy = 0
for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(X_train,Y_train)
    Y_pred_dt = dt.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

dt = DecisionTreeClassifier(random_state=best_x)
dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)

print(Y_pred_dt.shape)

print("Predicted Values : ",Y_pred_dt)

Y_test[0:10] #You can check accuracy by observing predicted results and 
test data.

accuracy_score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
print("The accuracy score achieved using Decision Tree is: 
"+str(accuracy_score_dt)+" %")

"""**Random Forest**"""

from sklearn.ensemble import RandomForestClassifier
max_accuracy = 0
for x in range(2000):
    rf = RandomForestClassifier(random_state=x)
    rf.fit(X_train,Y_train)
    Y_pred_rf = rf.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

rf = RandomForestClassifier(random_state=best_x)
rf.fit(X_train,Y_train)
Y_pred_rf = rf.predict(X_test)

Y_pred_rf.shape

print("Predicted Values : ",Y_pred_rf)

Y_test[0:10] #You can check accuracy by observing predicted results and 
test data.

accuracy_score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
print("The accuracy score achieved using Random Forest is: 
"+str(accuracy_score_rf)+" %")

"""**Summary of accuracy scores**"""
```

```python
all_accuracy_scores =
[accuracy_score_logistic_reg,accuracy_score_svm,accuracy_score_knn,accuracy
_score_dt,accuracy_score_rf]
algorithms_used = ["Logistic Regression","Support Vector Machine","K-
Nearest Neighbors","Decision Tree","Random Forest"]

for i in range(len(algorithms_used)):
    print("\nThe accuracy score achieved using "+algorithms_used[i]+" is:
"+str(all_accuracy_scores[i])+" %")

sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(algorithms_used,all_accuracy_scores)

"""**Here we can see that Random Forest is better than other
algorithms.**"""
```

SCREENSHOTS –



Interenship Project.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk Editing

Name - Nihalahmed Munir Barudwale

Project name - Prediction of Heart disease detection

Batch – Machine Learning With Python

Certificate Code – TCRIG02R28

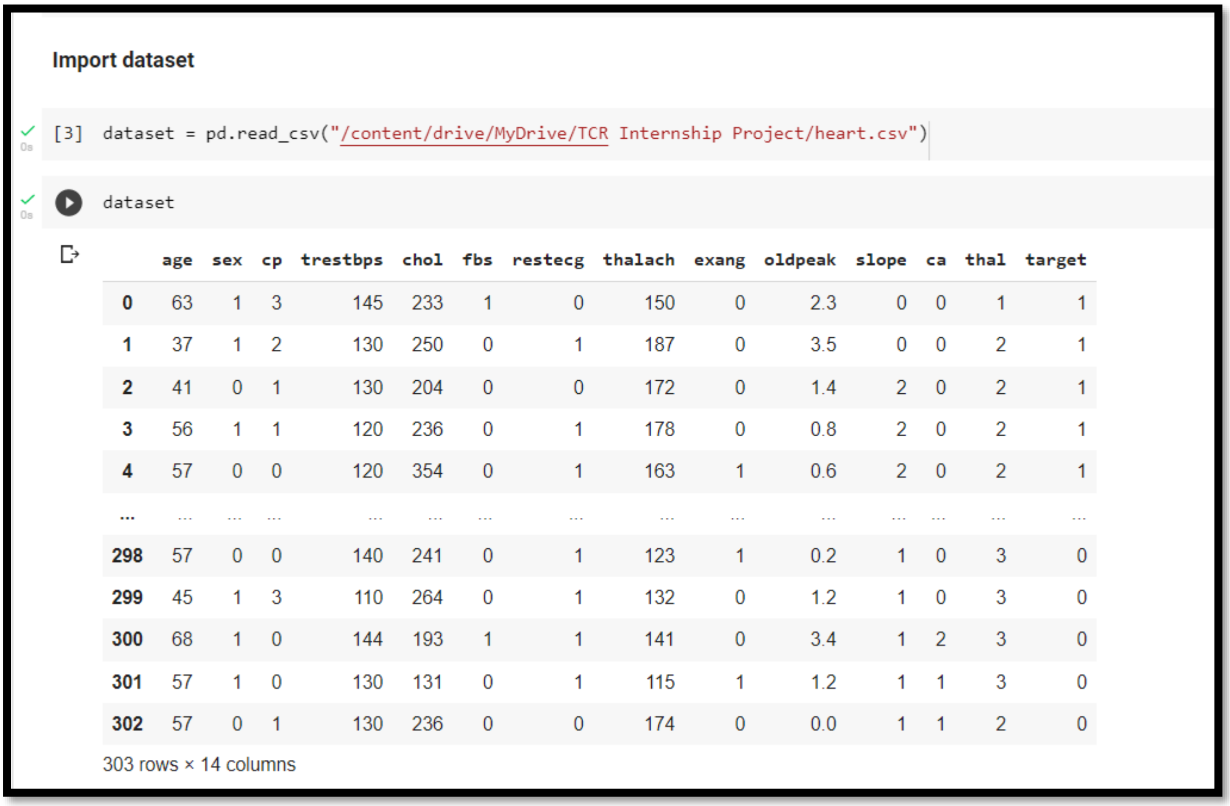
```
[1] #
```

Import Required Libraries

```
[2] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    %matplotlib inline
    import os
    import warnings
    warnings.filterwarnings('ignore')
```

Import dataset

```
[3] dataset = pd.read_csv("/content/drive/MyDrive/TCR Internship Project/heart.csv")
```



Import dataset

```
[3] dataset = pd.read_csv("/content/drive/MyDrive/TCR Internship Project/heart.csv")
```

```
dataset
```

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

303 rows × 14 columns

Shape of dataset

```
dataset.shape
```

```
(303, 14)
```

Some Operations on dataset

```
[6] dataset.head()
```

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

```
dataset.tail()
```

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

```
[8] type(dataset)
```

```
pandas.core.frame.DataFrame
```

```
dataset.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
```

```
[10] dataset.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
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%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

```
[11] dataset.columns
```

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

Checking total number of NA values

```
dataset.isna().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
```

Checking total number of NULL values

```
dataset.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
```

Analysing the 'target' variable

```
[15] dataset.target.describe()
```

```
count    303.000000
mean       0.544554
std        0.498835
min        0.000000
25%        0.000000
50%        1.000000
75%        1.000000
max        1.000000
Name: target, dtype: float64
```

```
dataset.target.unique()
```

```
array([1, 0])
```

```
#Checking correlation between columns
dataset.corr()["target"].abs().sort_values(ascending=False)
```

```
target      1.000000
exang       0.436757
cp          0.433798
oldpeak     0.430696
thalach     0.421741
ca          0.391724
slope       0.345877
thal        0.344029
sex         0.280937
age         0.225439
trestbps    0.144931
restecg     0.137230
chol        0.085239
fbs         0.028046
Name: target, dtype: float64
```

```
[18] #This shows that most columns are moderately correlated with target, but 'fbs' is very weakly correlated.
```

```
[19] dataset.target.value_counts()
```

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

Patient without heart problems - labeled as 0

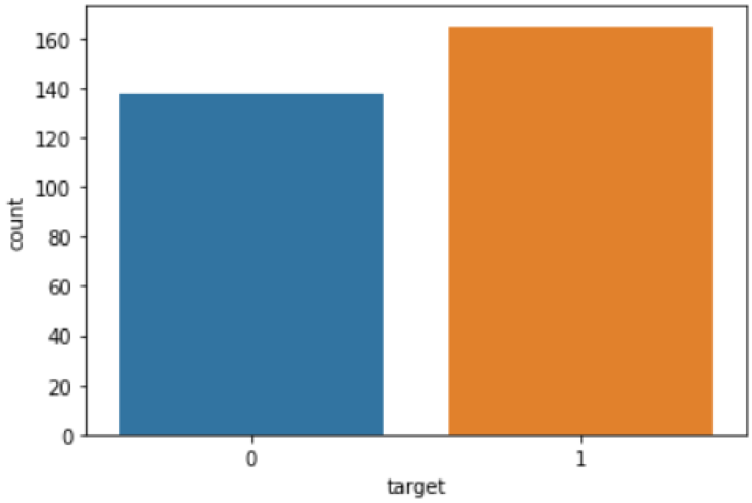
Patient with heart problems - labeled as 1

```
[20] print("Percentage of patients without heart problems: "+str(round(138*100/303,2)))
     print("Percentage of patients with heart problems: "+str(round(165*100/303,2)))
```

```
Percentage of patients without heart problems: 45.54
Percentage of patients with heart problems: 54.46
```

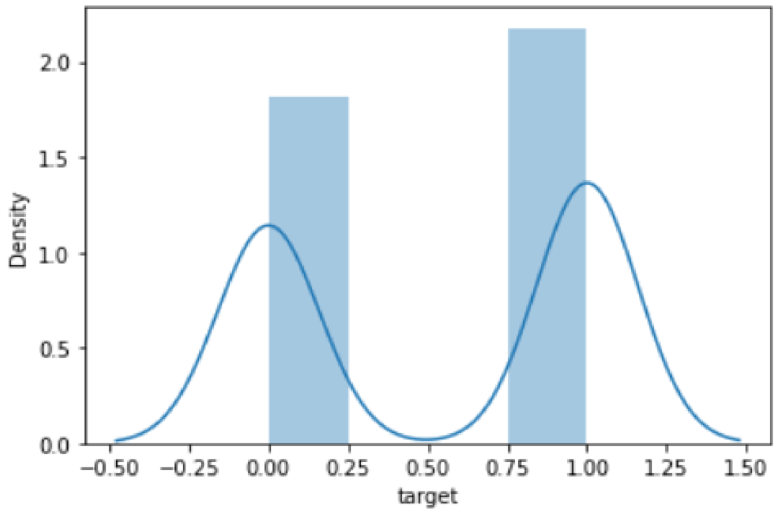
```
y = dataset["target"]
sns.countplot(y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c603f0d0>
```




```
sns.distplot(dataset['target'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c57f6a10>
```



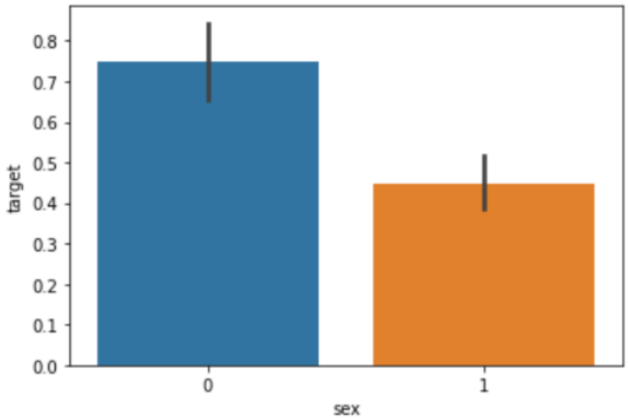
Analysing the 'sex' variable

```
[23] dataset.sex.value_counts()
```

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

```
sns.barplot(dataset["sex"],y)
```

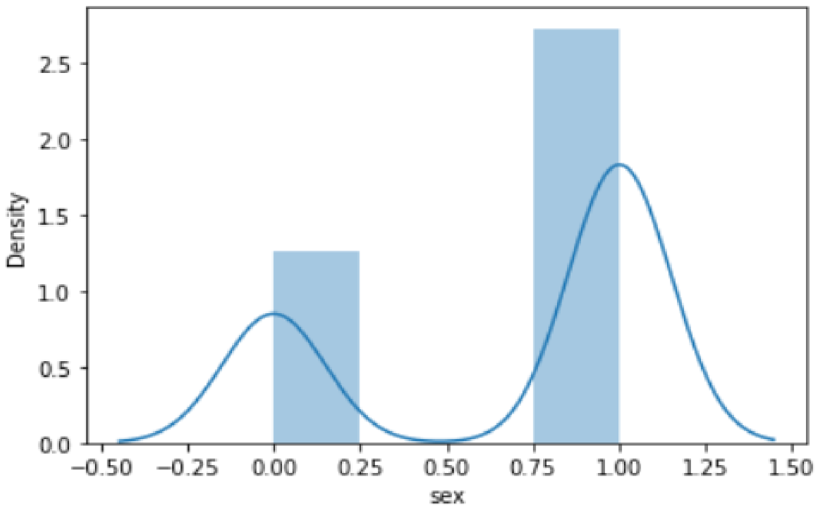
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c203efd0>
```



We notice that the 'sex' feature has 2 unique features.

```
[25] sns.distplot(dataset['sex'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c202f3d0>
```



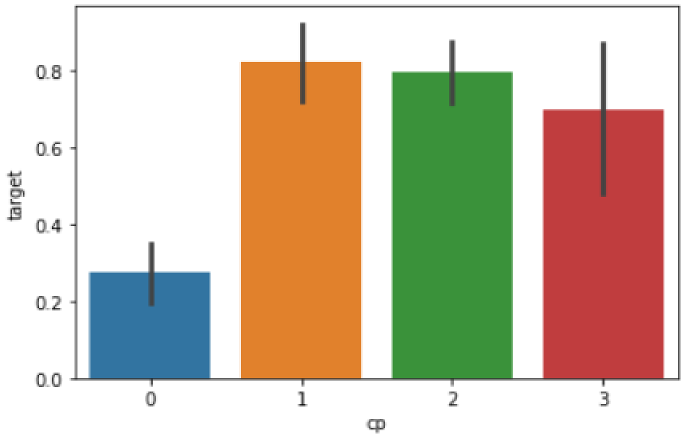
Analysing the 'cp' variable

```
[26] dataset.cp.value_counts()
```

```
0    143
2     87
1     50
3     23
Name: cp, dtype: int64
```

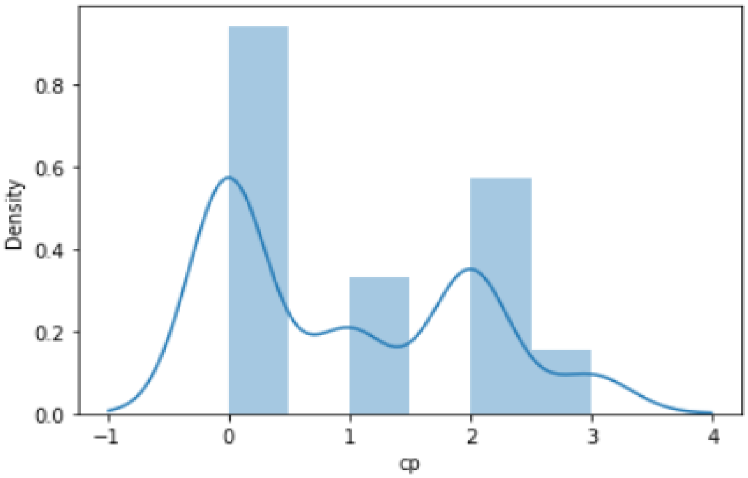
```
sns.barplot(dataset["cp"],y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c1f9f150>
```



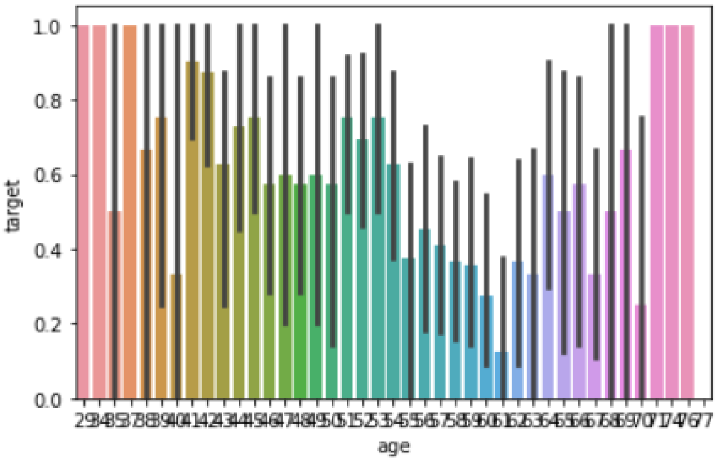

```
sns.distplot(dataset['cp'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c1fb9cd0>
```



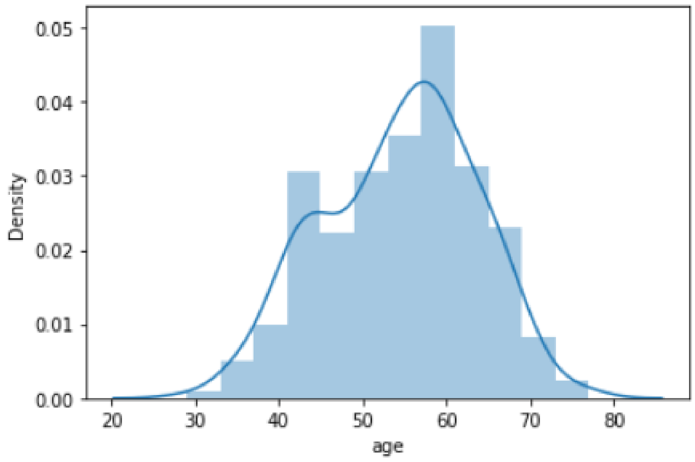
```
sns.barplot(dataset["age"],y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c2016810>
```



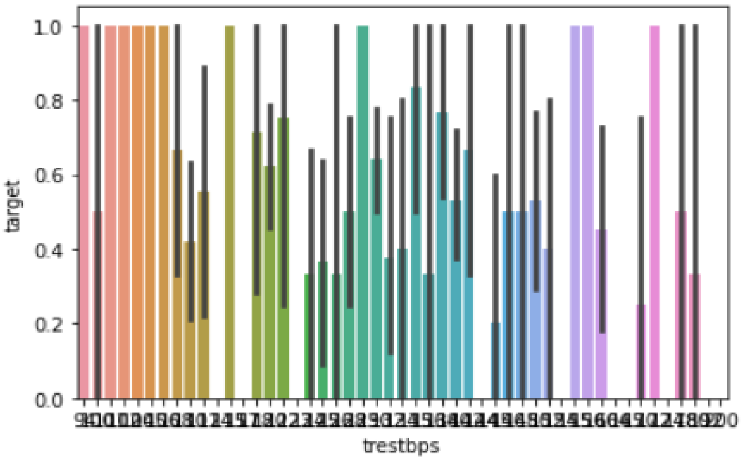

```
sns.distplot(dataset['age'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c1ed2950>
```



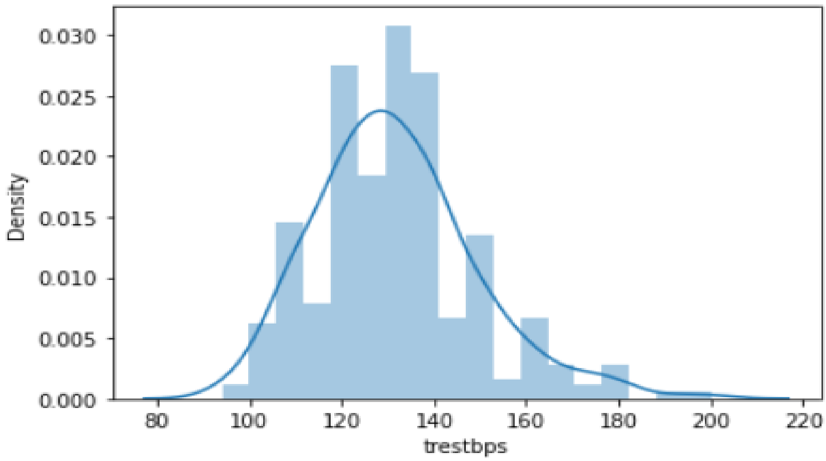
```
[33] sns.barplot(dataset["trestbps"],y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c1bffd90>
```




```
[34] sns.distplot(dataset['trestbps'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c1ceeb50>
```

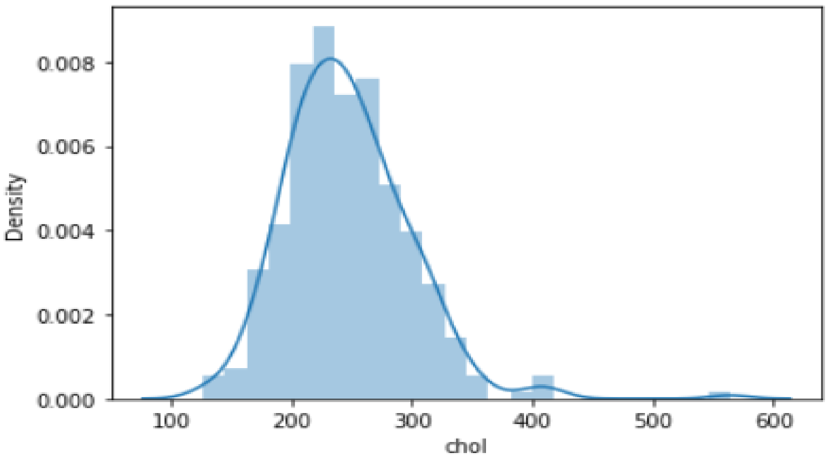


```
dataset.chol.value_counts()
```

```
234    6
204    6
197    6
269    5
212    5
      ..
278    1
281    1
284    1
290    1
564    1
Name: chol, Length: 152, dtype: int64
```

```
sns.distplot(dataset['chol'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c194b550>
```



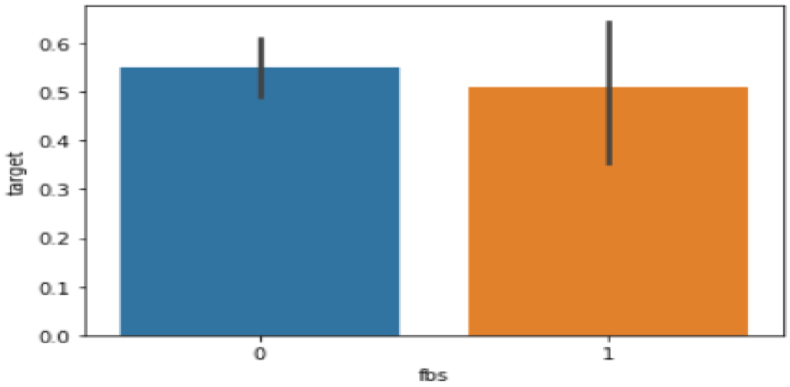
Analysing the 'fbs' variable

```
[38] dataset.fbs.value_counts()
```

```
0    258
1     45
Name: fbs, dtype: int64
```

```
sns.barplot(dataset["fbs"],y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c12f99d0>
```



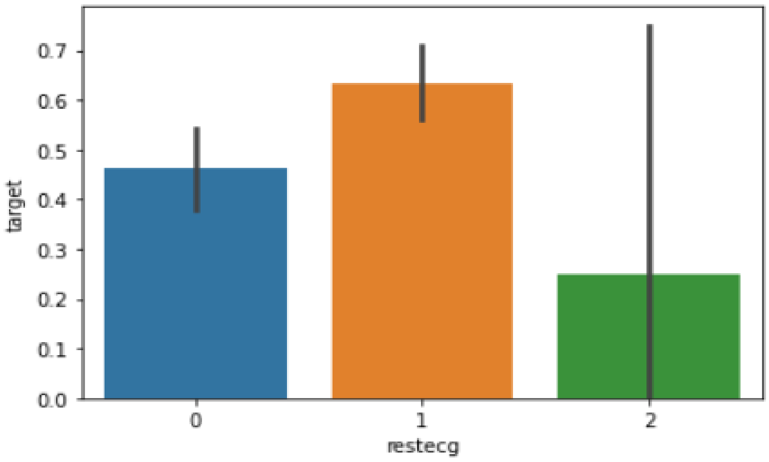
Analysing the 'restecg' variable

```
[41] dataset.restecg.value_counts()
```

```
1    152
0    147
2      4
Name: restecg, dtype: int64
```

```
sns.barplot(dataset["restecg"],y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c1267710>
```



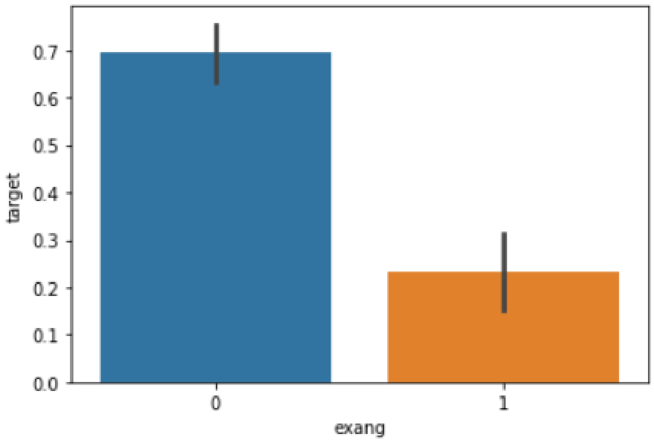
Analysing the 'exang' variable

```
[44] dataset.exang.value_counts()
```

```
0    204
1     99
Name: exang, dtype: int64
```

```
sns.barplot(dataset["exang"],y)
```

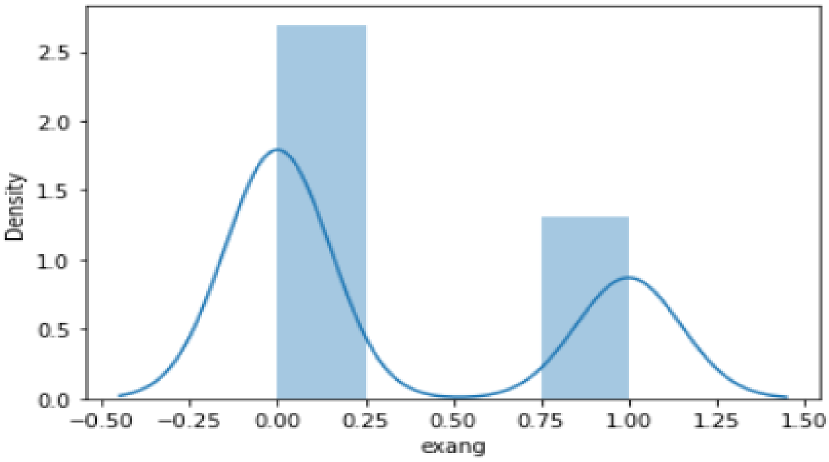
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c10c2ad0>
```



We notice here that people with exang=1, are much less likely to have heart problems.

```
sns.distplot(dataset['exang'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c108f250>
```



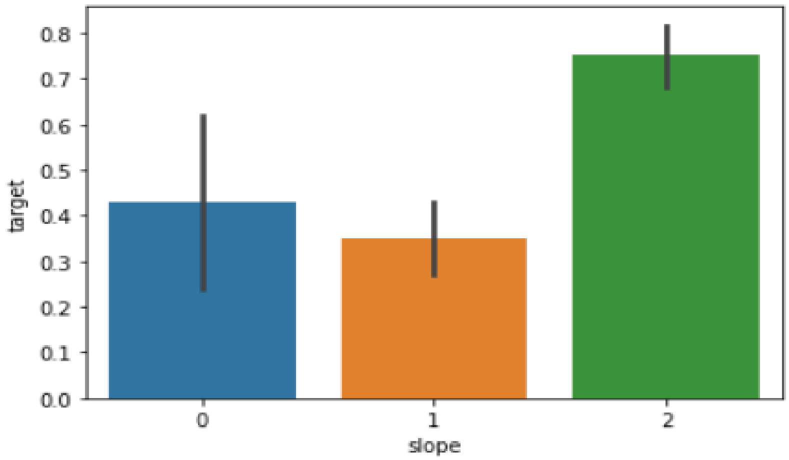
Analysing the 'slope' variable

```
[47] dataset.slope.value_counts()
```

```
2    142
1    140
0     21
Name: slope, dtype: int64
```

```
sns.barplot(dataset["slope"],y)
```

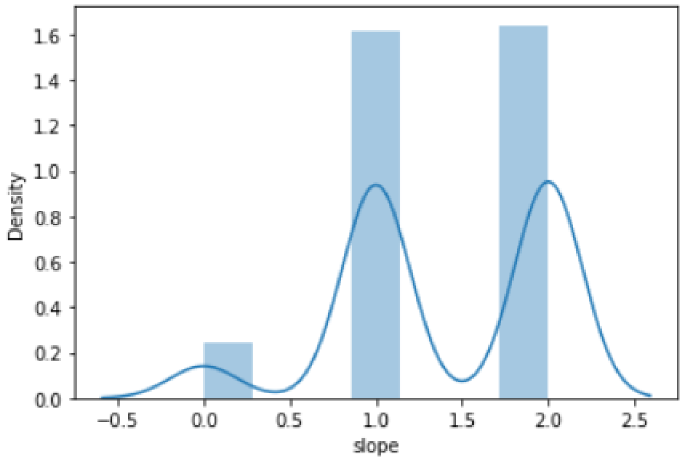
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c142a350>
```



We observe, that Slope '2' causes heart pain much more than Slope '0' and '1'

```
[49] sns.distplot(dataset['slope'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c141d190>
```



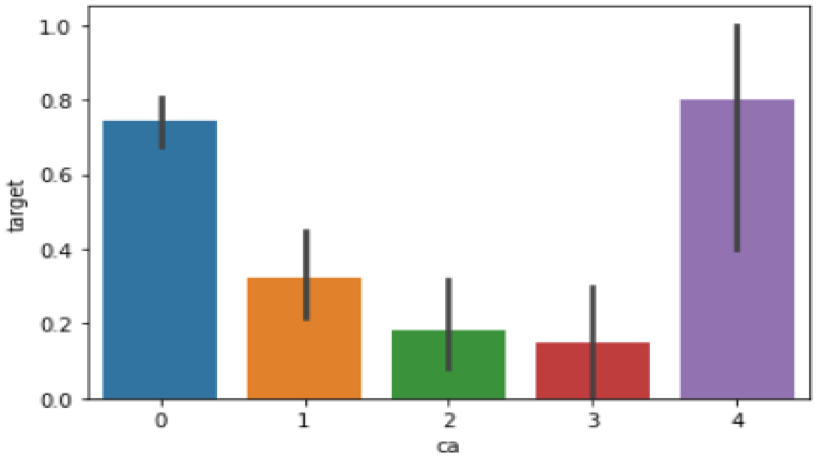
Analysing the 'ca' variable

```
dataset.ca.value_counts()
```

```
0    175
1     65
2     38
3     20
4      5
Name: ca, dtype: int64
```

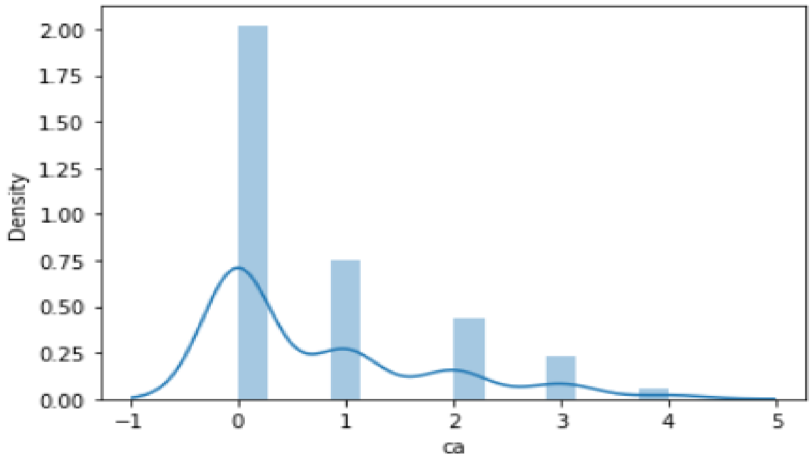
```
[51] sns.barplot(dataset["ca"],y)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f58c0f5e390>




```
sns.distplot(dataset['ca'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f58c0edc110>



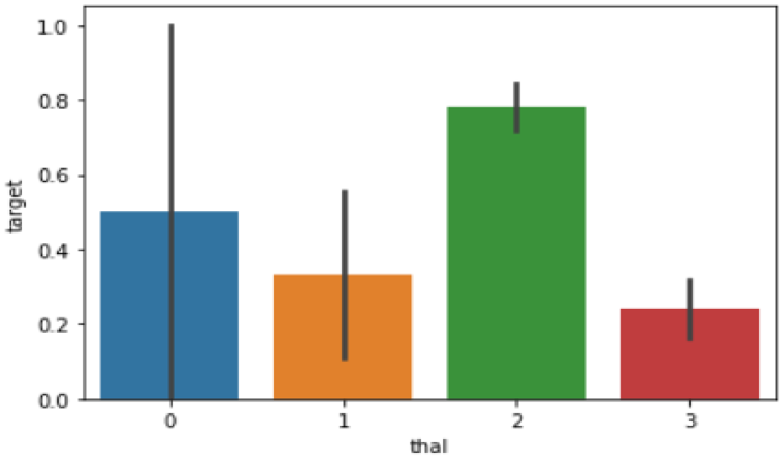
Analysing the 'thal' variable

```
[53] dataset.thal.value_counts()
```

```
2    166
3    117
1     18
0      2
Name: thal, dtype: int64
```

```
sns.barplot(dataset["thal"],y)
```

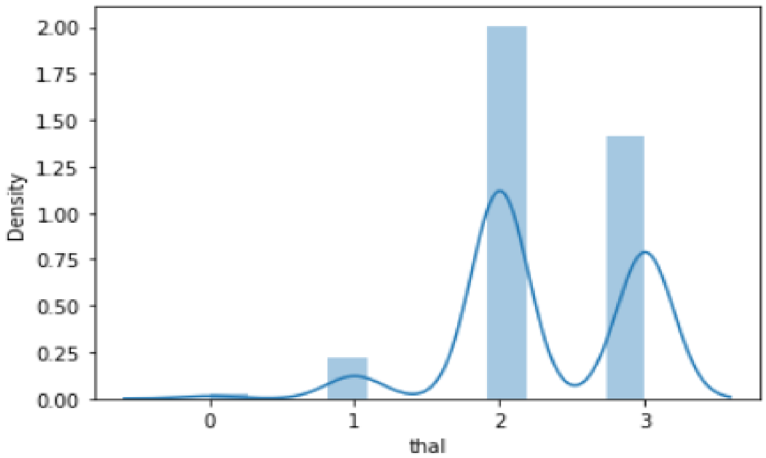
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58d9dffb90>
```



thal=2 has large number of heart patients.

```
sns.distplot(dataset['thal'])
```

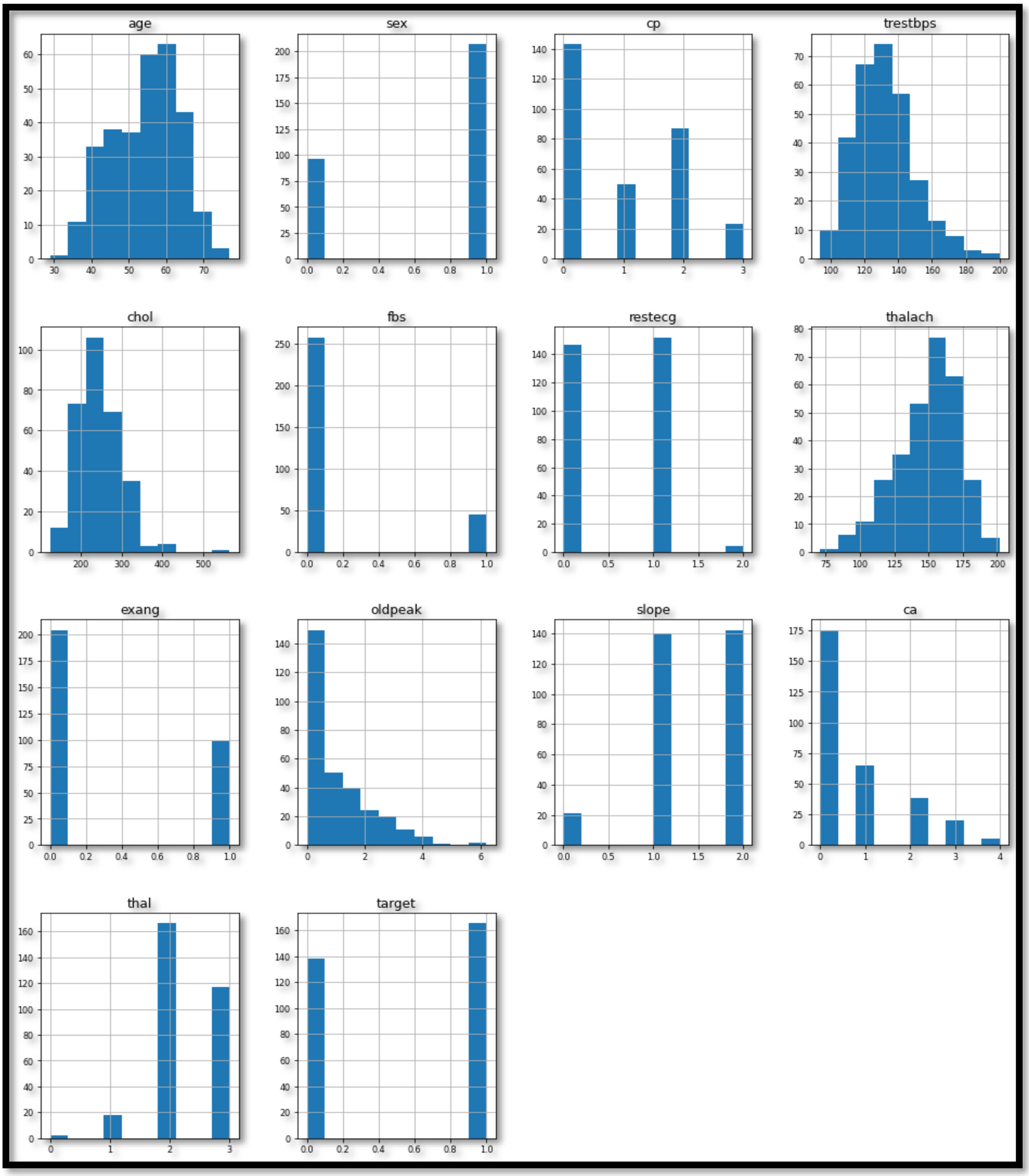
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c0deb690>
```

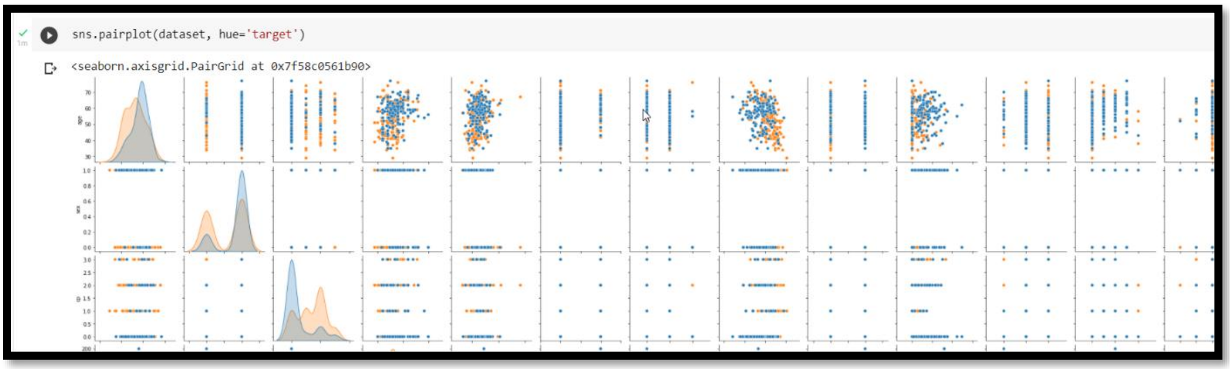


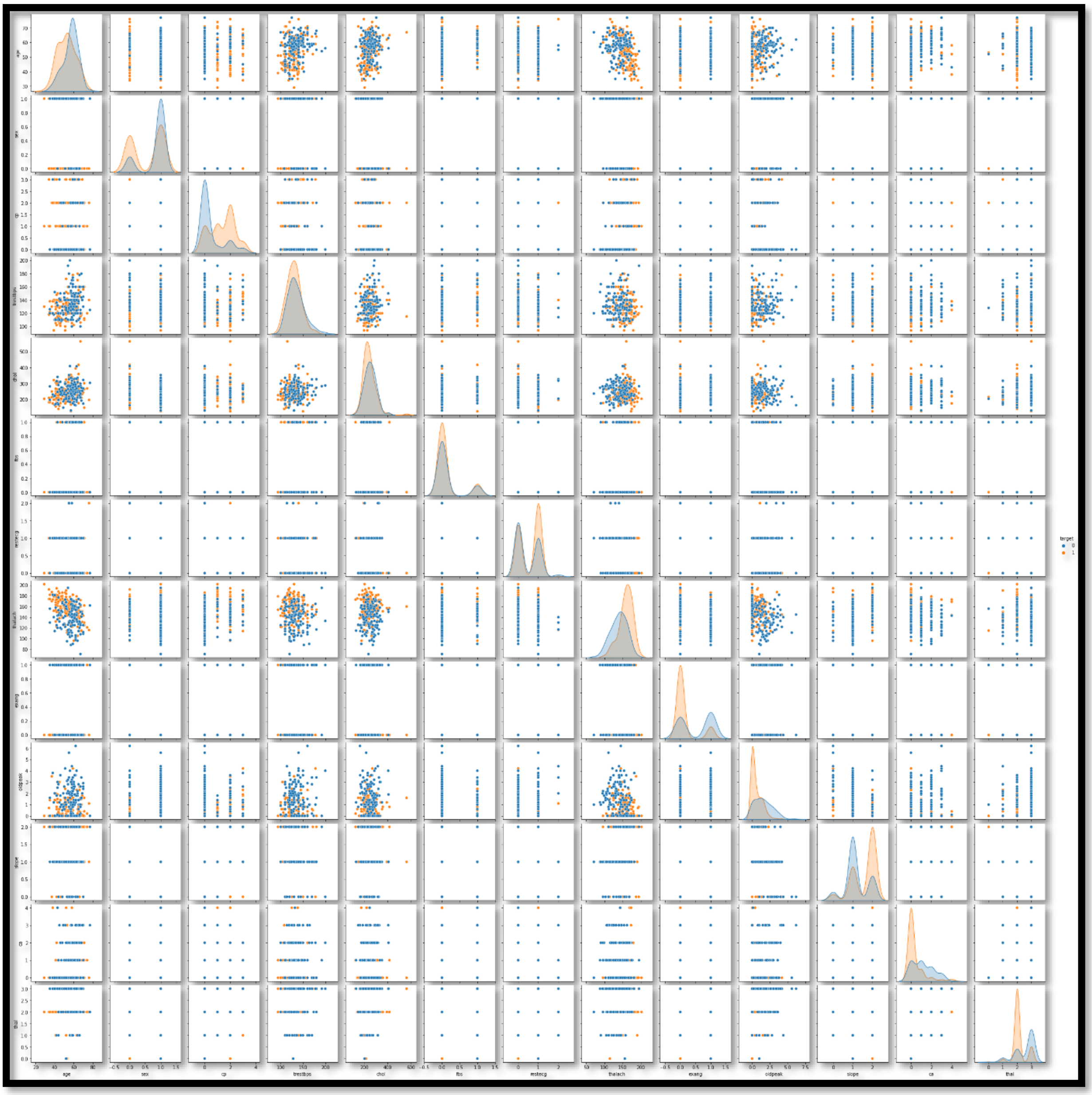
Get an overview distribution of each column

```
dataset.hist(figsize=(16, 20), xlabelsize=8, ylabelsize=8)
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0cc1a10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0ceffd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0cb0650>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0c64c50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0c28290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0bde890>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0b92f10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0b54490>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0b544d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0b0cbd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0a85710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c0a3dd10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f58c09fe590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c09b2a90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c09e9f90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58c09af590>]],
```


age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
target


sns.pairplot(dataset, hue='target')
<seaborn.axisgrid.PairGrid at 0x7f58c0561b90>



```
dataset.corr()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276326	0.068001	-0.225439
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118261	0.210041	-0.280937
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181053	-0.161736	0.433798
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101389	0.062210	-0.144931
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070511	0.098803	-0.085239
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137979	-0.032019	-0.028046
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072042	-0.011981	0.137230
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213177	-0.096439	0.421741
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115739	0.206754	-0.436757
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222682	0.210244	-0.430696
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080155	-0.104764	0.345877
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000000	0.151832	-0.391724
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151832	1.000000	-0.344029
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.391724	-0.344029	1.000000

```
f, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(dataset.corr(),annot=True,cmap='PiYG',linewidths=.5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58b406cb10>
```

Splitting the data - Train Test split

```
[60] from sklearn.model_selection import train_test_split
     x = dataset.drop("target",axis=1)
     y= dataset["target"]

     X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size=0.20,random_state=0)
```

```
[61] X_train.shape
```

```
(242, 13)
```

```
X_test.shape
```

```
(61, 13)
```

```
[63] Y_train.shape
```

```
(242,)
```

```
[64] Y_test.shape
```

```
(61,)
```

```
[65] from sklearn.metrics import accuracy_score
```

Logistic Regression

```
[66] from sklearn.linear_model import LogisticRegression
     model_logistic_reg = LogisticRegression()
     model_logistic_reg.fit(X_train,Y_train)
     Y_pred_logistic_reg = model_logistic_reg.predict(X_test)
```

```
[67] Y_pred_logistic_reg.shape

     (61,)
```

```
[68] print("Predicted Values : ",Y_pred_logistic_reg)

     Predicted Values :  [0 1 1 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 1 1 0 1 1 1 1 0
      1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1]
```

```
[69] Y_test[0:10] #You can check accuracy by observing predicted results and test data.

     225    0
     152    1
     228    0
     201    0
     52     1
     245    0
     175    0
     168    0
     223    0
     217    0
     Name: target, dtype: int64
```

```
[70] accuracy_score_logistic_reg = round(accuracy_score(Y_pred_logistic_reg,Y_test)*100,2)
     print("The accuracy score achieved using Logistic Regression is: "+str(accuracy_score_logistic_reg)+" %")

     The accuracy score achieved using Logistic Regression is: 85.25 %
```

SVM

```
[71] from sklearn import svm
     model_svm = svm.SVC(kernel='linear')
     model_svm.fit(X_train, Y_train)
     Y_pred_svm = model_svm.predict(X_test)
```

```
[72] Y_pred_svm.shape

     (61,)
```

```
print("Predicted Values : ",Y_pred_svm)

Predicted Values :  [0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1 1 0 0 1 1 1 0 1 1 1 1 0
 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1]
```

```
[74] Y_test[0:10] #You can check accuracy by observing predicted results and test data.

     225    0
     152    1
     228    0
     201    0
     52     1
     245    0
     175    0
     168    0
     223    0
     217    0
     Name: target, dtype: int64
```

```
accuracy_score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)
print("The accuracy score achieved using Linear SVM is: "+str(accuracy_score_svm)+" %")

The accuracy score achieved using Linear SVM is: 81.97 %
```

K Nearest Neighbors

```
[76] from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier(n_neighbors=7)
     knn.fit(X_train,Y_train)
     Y_pred_knn=knn.predict(X_test)
```

```
[77] Y_pred_knn.shape

     (61,)
```

```
[78] print("Predicted Values : ",Y_pred_knn)

     Predicted Values :  [0 0 1 0 1 1 0 0 0 0 1 1 0 1 1 1 0 1 0 1 1 1 0 0 0 0 1 0 1 1 0 0 1 0 1 1 0
      1 0 1 0 1 1 0 0 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0]
```

```
     Y_test[0:10] #You can check accuracy by observing predicted results and test data.

     225    0
     152    1
     228    0
     201    0
     52     1
     245    0
     175    0
     168    0
     223    0
     217    0
     Name: target, dtype: int64
```

```
[80] accuracy_score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)
     print("The accuracy score achieved using KNN is: "+str(accuracy_score_knn)+" %")

     The accuracy score achieved using KNN is: 67.21 %
```

Decision Tree

```
[81] from sklearn.tree import DecisionTreeClassifier
     max_accuracy = 0
     for x in range(200):
         dt = DecisionTreeClassifier(random_state=x)
         dt.fit(X_train,Y_train)
         Y_pred_dt = dt.predict(X_test)
         current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
         if(current_accuracy>max_accuracy):
             max_accuracy = current_accuracy
             best_x = x

     dt = DecisionTreeClassifier(random_state=best_x)
     dt.fit(X_train,Y_train)
     Y_pred_dt = dt.predict(X_test)
```

```
[82] print(Y_pred_dt.shape)

     (61,)
```

```
     print("Predicted Values : ",Y_pred_dt)

     Predicted Values :  [0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 1 1 0 0 0 1 1 0 0 1 1 1 0 1 1 1 1 0
      1 0 0 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1]
```

```
[84] Y_test[0:10] #You can check accuracy by observing predicted results and test data.

     225    0
     152    1
     228    0
     201    0
     52     1
     245    0
     175    0
     168    0
     223    0
     217    0
     Name: target, dtype: int64
```

```
[85] accuracy_score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
     print("The accuracy score achieved using Decision Tree is: "+str(accuracy_score_dt)+" %")

     The accuracy score achieved using Decision Tree is: 81.97 %
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
max_accuracy = 0
for x in range(2000):
    rf = RandomForestClassifier(random_state=x)
    rf.fit(X_train,Y_train)
    Y_pred_rf = rf.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

rf = RandomForestClassifier(random_state=best_x)
rf.fit(X_train,Y_train)
Y_pred_rf = rf.predict(X_test)
```

```
[87] Y_pred_rf.shape
```

```
(61,)
```

```
[88] print("Predicted Values : ",Y_pred_rf)
```

```
Predicted Values :  [0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 1 1 0 1 1 1 0 0
 1 0 0 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1]
```

```
[89] Y_test[0:10] #You can check accuracy by observing predicted results and test data.
```

```
225    0
152    1
228    0
201    0
52     1
245    0
175    0
168    0
223    0
217    0
Name: target, dtype: int64
```

```
[90] accuracy_score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
     print("The accuracy score achieved using Random Forest is: "+str(accuracy_score_rf)+" %")
```

```
The accuracy score achieved using Random Forest is: 90.16 %
```

Summary of accuracy scores

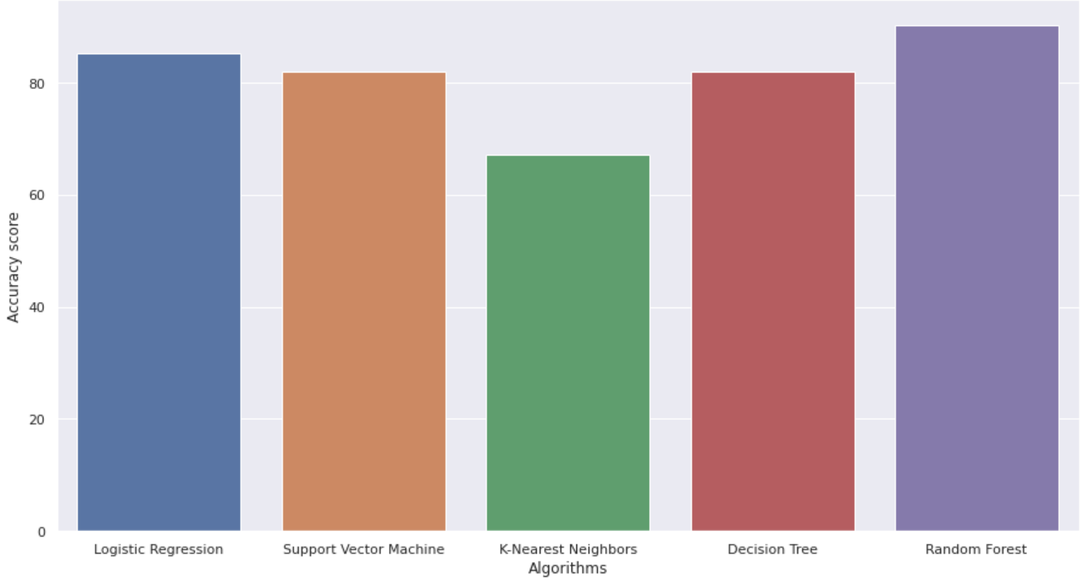
```
[91] all_accuracy_scores = [accuracy_score_logistic_reg,accuracy_score_svm,accuracy_score_knn,accuracy_score_dt,accuracy_score_rf]
     algorithms_used = ["Logistic Regression","Support Vector Machine","K-Nearest Neighbors","Decision Tree","Random Forest"]
```

```
[92] for i in range(len(algorithms_used)):
         print("\nThe accuracy score achieved using "+algorithms_used[i]+" is: "+str(all_accuracy_scores[i])+" %")
```

```
The accuracy score achieved using Logistic Regression is: 85.25 %

The accuracy score achieved using Support Vector Machine is: 81.97 %

The accuracy score achieved using K-Nearest Neighbors is: 67.21 %

The accuracy score achieved using Decision Tree is: 81.97 %

The accuracy score achieved using Random Forest is: 90.16 %
```

```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(algorithms_used,all_accuracy_scores)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58abab46d0>
```



Here we can see that Random Forest is better than other algorithms.