projectwork

July 10, 2023

[1]:
```
#importing lybraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
#Importing the dataset
df = pd.read_csv('heart_dataset.csv')
```

[3]:
```
df.shape
```

[3]:
```
(303, 14)
```

[4]:
```
type(df)
```

[4]:
```
pandas.core.frame.DataFrame
```

[5]:
```
#checking top hive rows of the data
df.head()
```

[5]:

	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

[6]:
```
#checking last five rows of data
df.tail()
```

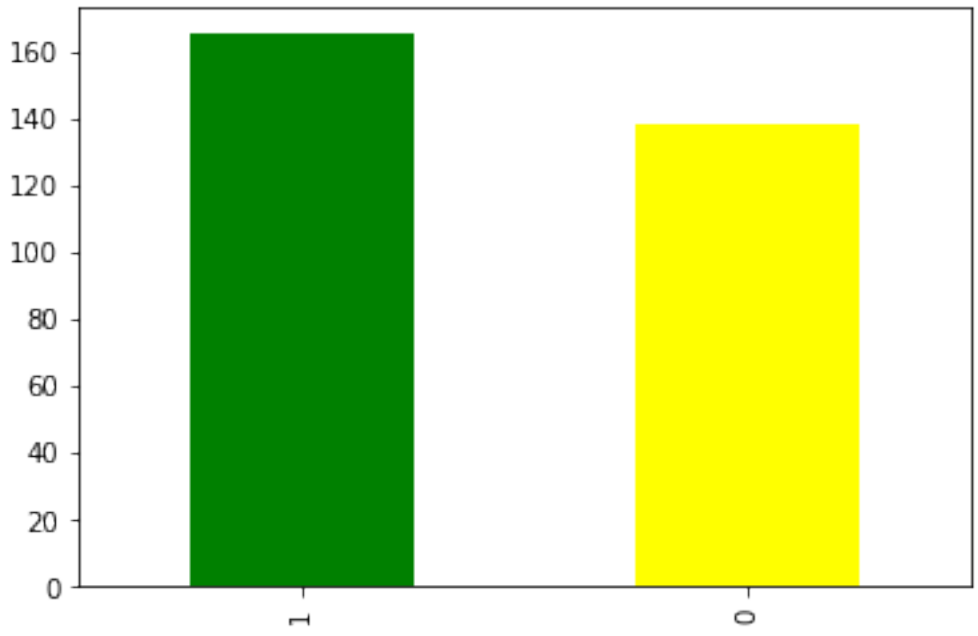
[6]:

	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

[7]:
```
#how many class of one feature or target
df["target"].value_counts()   #balanced data
```

[7]:
```
1    165
0    138
Name: target, dtype: int64
```

[9]:
```
#plotting the same
#bar chart
df["target"].value_counts().plot(kind='bar', color=["green","yellow"])
plt.show()
```



[14]:
```
#We have 165 person with heart disease and 138 person without heart disease, so␣
↪our dataset is balanced.
```

[11]:
```
#info
df.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
```

[12]:
```
#checking for missing values
df.isna().sum()
```

[12]:
```
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
```

[16]:
```
#it will give the all d statistics of numerical
df.describe()
```

[16]:

	age	sex	cp	trestbps	chol	fbs	\
count	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	
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	

	restecg	thalach	exang	oldpeak	slope	ca	\
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	
mean	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	
std	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	
min	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	
50%	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	
75%	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	
max	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	

	thal	target
count	303.000000	303.000000
mean	2.313531	0.544554
std	0.612277	0.498835
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

[17]:
```
#Checking correlation between columns
print(df.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]:
```
#Heart Disease Frequency according to Sex
df.sex.value_counts()
```

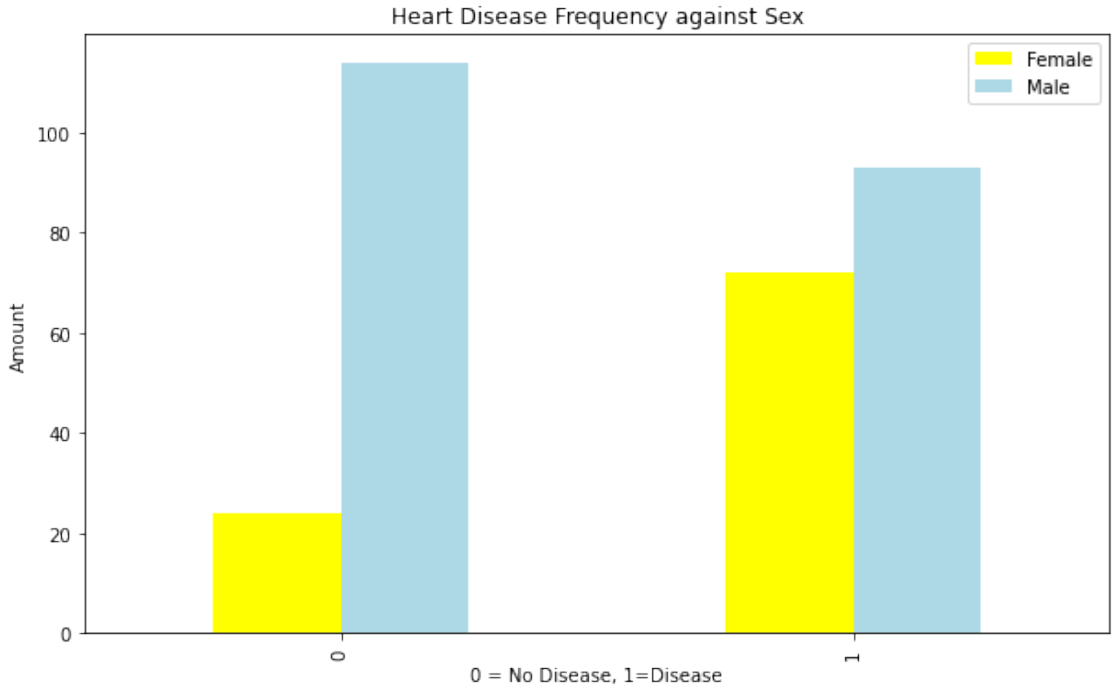
[18]:
```
1    207
0     96
Name: sex, dtype: int64
```

[19]:
```
#Creating contingency table to compare sex with target
pd.crosstab(df.target, df.sex)
```

[19]:
```
sex      0    1
target
0       24  114
1       72   93
```

[20]:
```
#Create plot of heart disease against sex
pd.crosstab(df.target, df.sex).
 ↪plot(kind="bar",figsize=(10,6),color=["yellow","lightblue"])
plt.title("Heart Disease Frequency against Sex")
plt.xlabel("0 = No Disease, 1=Disease")
plt.ylabel("Amount")
plt.legend(["Female","Male"])
```

[20]:
```
<matplotlib.legend.Legend at 0x7f3380d3ce90>
```



[21]:
```
#Heart Disease Frequency vs Chest Pain
#chest pain type
#0: Typical angina: chest pain related decrease blood supply to the heart
#1: Atypical angina: chest pain not related to heart
#2: Non-anginal pain: typically esophageal spasms (non heart related)
#3: Asymptomatic: chest pain not showing signs of disease
```

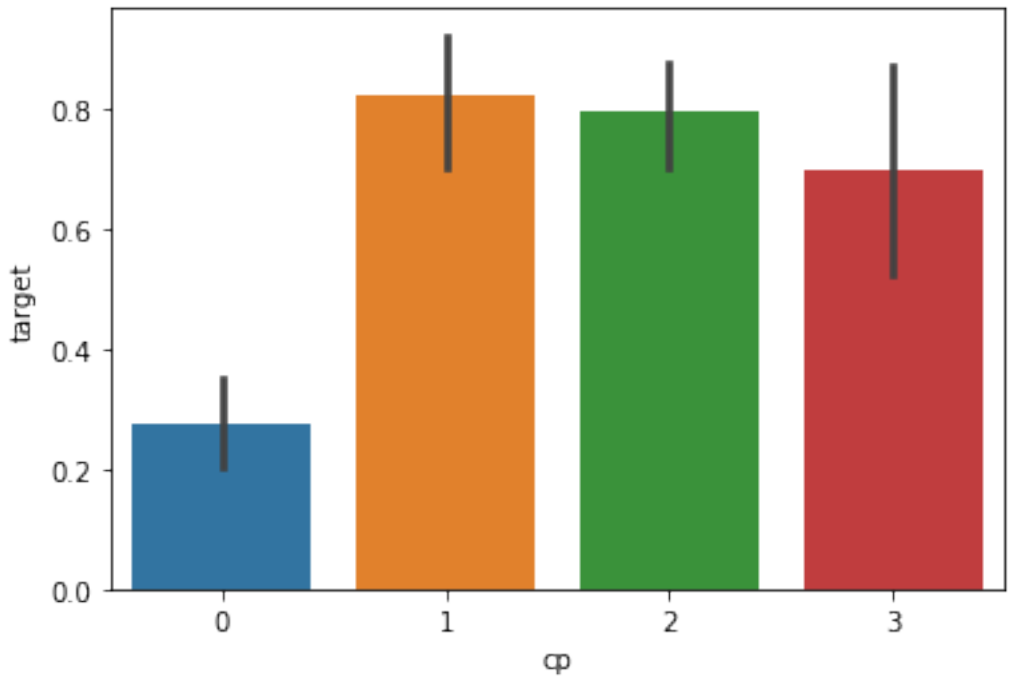
[22]:
```
#creating crosstab
pd.crosstab(df.cp,df.target)
```

[22]:
```
target    0   1
cp
0       104  39
1         9  41
2        18  69
3         7  16
```

[23]:
```
#Analysing the 'Chest Pain Type' feature
df["cp"].unique()
```

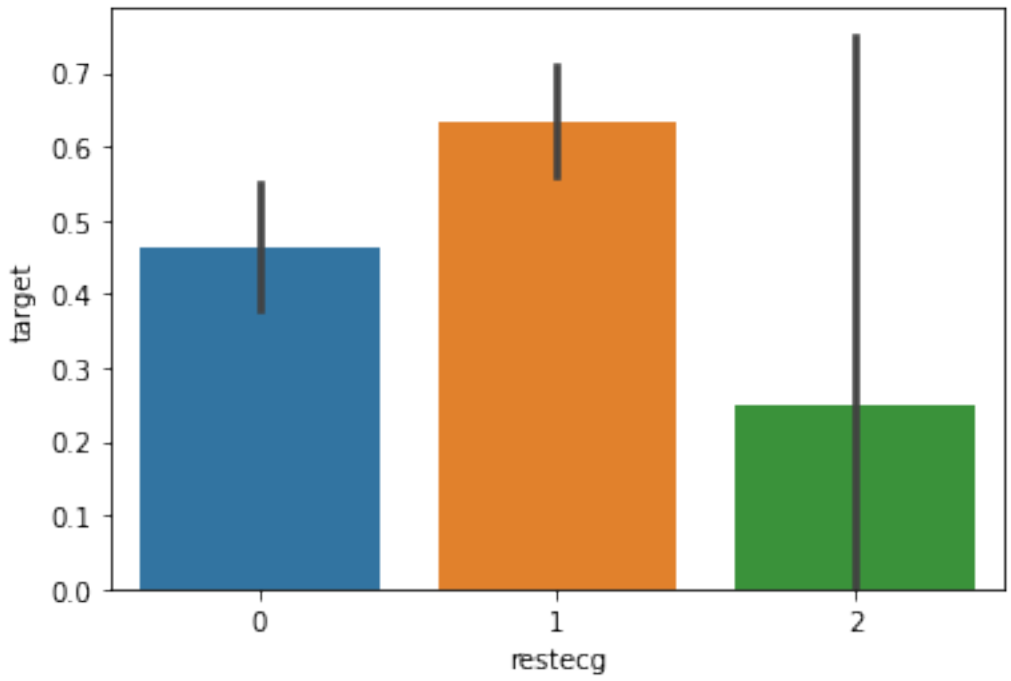
[23]:
```
array([3, 2, 1, 0])
```

[24]:
```
sns.barplot(df["cp"],df['target'])
plt.show()
```



[25]:
```
#We notice, that chest pain of '0', i.e. the ones with typical angina are much␣
↪less likely to have heart problems
```

[26]:
```
#Analysing the restecg feature
df["restecg"].unique()
sns.barplot(df["restecg"],df["target"])
plt.show()
```



[27]:
```
#### We will continue exploring other pairwise relationships
```

[28]:
```
## Data Processing
#### We need to convert some categorical variables into dummy variables and␣
↪scale all the values before training the Machine Learning models. We will␣
↪use the get_dummies method to create dummy columns for categorical variables.
↪"
```

[29]:
```
pd.set_option("display.float", "{:.2f}".format)
```

[30]:
```
df.describe()
```

[30]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang
count	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00
mean	54.37	0.68	0.97	131.62	246.26	0.15	0.53	149.65	0.33
std	9.08	0.47	1.03	17.54	51.83	0.36	0.53	22.91	0.47
min	29.00	0.00	0.00	94.00	126.00	0.00	0.00	71.00	0.00
25%	47.50	0.00	0.00	120.00	211.00	0.00	0.00	133.50	0.00
50%	55.00	1.00	1.00	130.00	240.00	0.00	1.00	153.00	0.00
75%	61.00	1.00	2.00	140.00	274.50	0.00	1.00	166.00	1.00
max	77.00	1.00	3.00	200.00	564.00	1.00	2.00	202.00	1.00

	oldpeak	slope	ca	thal	target
count	303.00	303.00	303.00	303.00	303.00
mean	1.04	1.40	0.73	2.31	0.54

```
std       1.16   0.62   1.02   0.61    0.50
min       0.00   0.00   0.00   0.00    0.00
25%       0.00   1.00   0.00   2.00    0.00
50%       0.80   1.00   0.00   2.00    1.00
75%       1.60   2.00   1.00   3.00    1.00
max       6.20   2.00   4.00   3.00    1.00
```

[34]:
```
#segregarting the categorical variables and continuous ones.
categorical_val = []
continous_val = []
for column in df.columns:
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

[35]:
```
categorical_val
```

[35]:
```
['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', 'target']
```

[36]:
```
categorical_val.remove('target')
df = pd.get_dummies(df, columns = categorical_val)
```

[37]:
```
df.head()
```

[37]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	\
0	63	145	233	150	2.30	1	0	1	0	0	
1	37	130	250	187	3.50	1	0	1	0	0	
2	41	130	204	172	1.40	1	1	0	0	1	
3	56	120	236	178	0.80	1	0	1	0	1	
4	57	120	354	163	0.60	1	1	0	1	0	

	…	slope_2	ca_0	ca_1	ca_2	ca_3	ca_4	thal_0	thal_1	thal_2	thal_3
0	…	0	1	0	0	0	0	0	1	0	0
1	…	0	1	0	0	0	0	0	0	1	0
2	…	1	1	0	0	0	0	0	0	1	0
3	…	1	1	0	0	0	0	0	0	1	0
4	…	1	1	0	0	0	0	0	0	1	0

[5 rows x 31 columns]

[38]:
```
#standardizing the data
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
scaled_columns = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
df[scaled_columns] = scale.fit_transform(df[scaled_columns])
```

[39]:
```
df.head()
```

[39]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	\
0	0.95	0.76	-0.26	0.02	1.09	1	0	1	0	0	
1	-1.92	-0.09	0.07	1.63	2.12	1	0	1	0	0	
2	-1.47	-0.09	-0.82	0.98	0.31	1	1	0	0	1	
3	0.18	-0.66	-0.20	1.24	-0.21	1	0	1	0	1	
4	0.29	-0.66	2.08	0.58	-0.38	1	1	0	1	0	

	…	slope_2	ca_0	ca_1	ca_2	ca_3	ca_4	thal_0	thal_1	thal_2	thal_3
0	…	0	1	0	0	0	0	0	1	0	0
1	…	0	1	0	0	0	0	0	0	1	0
2	…	1	1	0	0	0	0	0	0	1	0
3	…	1	1	0	0	0	0	0	0	1	0
4	…	1	1	0	0	0	0	0	0	1	0

[5 rows x 31 columns]

[41]:
```
#Train Test split
from sklearn.model_selection import train_test_split
predictors = df.drop("target",axis=1)
target = df["target"]
X_train,X_test,Y_train,Y_test = train_test_split(predictors,target,test_size=0.
↪20,random_state=0)
```

[42]:
```
X_train.shape
```

[42]: (242, 30)

[43]:
```
X_test.shape
```

[43]: (61, 30)

[44]:
```
Y_train.shape
```

[44]: (242,)

[45]:
```
Y_test.shape
```

[45]: (61,)

[46]:
```
from sklearn.metrics import accuracy_score
```

[47]:
```
import xgboost as xgb
xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(X_train, Y_train)
Y_pred_xgb = xgb_model.predict(X_test)
```

[48]:
```
score_xgb = round(accuracy_score(Y_pred_xgb,Y_test)*100,2)
print("The accuracy score achieved using XGBoost is: "+str(score_xgb)+" %")
```

```
The accuracy score achieved using XGBoost is: 83.61 %
```

[49]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,Y_train)
Y_pred_lr = lr.predict(X_test)
```

[50]:
```
score_lr = round(accuracy_score(Y_pred_lr,Y_test)*100,2)
print("The accuracy score achieved using Logistic Regression is:␣
↪"+str(score_lr)+" %")
```

```
The accuracy score achieved using Logistic Regression is: 88.52 %
```

[51]:
```
#SVM
from sklearn import svm
sv = svm.SVC(kernel='linear')
sv.fit(X_train, Y_train)
Y_pred_svm = sv.predict(X_test)
```

[52]:
```
score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)
print("The accuracy score achieved using Linear SVM is: "+str(score_svm)+" %")
```

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

[53]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, Y_train)
Y_pred_clf = knn_clf.predict(X_test)
score_knn = round(accuracy_score(Y_pred_clf,Y_test)*100,2)
print("The accuracy score achieved using KNN Classifier is: "+str(score_knn)+"␣
↪%")
```

```
The accuracy score achieved using KNN Classifier is: 78.69 %
```

[ ]:
```
```