AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

408/1, Kuratoli, Khilkhet, Dhaka 1229, Bangladesh

Assignment Title:	Final Term Assignment (PiP-B)		
Assignment No:	1	Date of Submission:	10-12-2021
Course Title:	PROGRAMMING IN PYTHON		
Course Code:	01462	Section:	B
Semester:	FALL 20 21- 22	Course Teacher:	Akinul Islam Jony

Declaration and Statement of Authorship:

1. I/we hold a copy of this Assignment/Case-Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case-Study is my/our original work and no part of it has been copied from any other student's work or from any other source except where due acknowledgement is made.
3. No part of this Assignment/Case-Study has been written for me/us by any other person except where such collaborationhas been authorized by the concerned teacher and is clearly acknowledged in the assignment.
4. I/we have not previously submitted or currently submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared and archived for the purpose of detecting plagiarism.
6. I/we give permission for a copy of my/our marked work to be retained by the Faculty for review and comparison, including review by external examiners.
7. I/we understand thatPlagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a formofcheatingandisaveryseriousacademicoffencethatmayleadtoexpulsionfromtheUniversity. Plagiarized material can be drawn from, and presented in, written, graphic and visual form, including electronic data, and oral presentations. Plagiarism occurs when the origin of them arterial used is not appropriately cited.
8. I/we also understand that enabling plagiarism is the act of assisting or allowing another person to plagiarize or to copy my/our work.

* *Student(s) must complete all details except the faculty use part.*
** Please submit all assignments to your course teacher or the office of the concerned teacher.

Group Name/No.:

No	Name	ID	Program	Signature
1	RIDOY,RAYHAN KHAN	17-34755-2	CSE	Ridoy
2				
3				
4				
5				
6				
7				
8				
9				
10				

Faculty use only		
FACULTYCOMMENTS	**Marks Obtained**	
	Total Marks	

In [151]:
```
import numpy as np
import pandas as pd
# Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets # for using built-in datasets
from sklearn import metrics # for checking the model accuracy
#To plot the graph embedded in the notebook
%matplotlib inline
```

I downloaded the dataset from kaggle based on HeartDisease's data

This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease.

Content Attribute Information:

age sex chest pain type (4 values) resting blood pressure serum cholestoral in mg/dl fasting blood sugar > 120 mg/dl resting electrocardiographic results (values 0,1,2) maximum heart rate achieved exercise induced angina oldpeak = ST depression induced by exercise relative to rest the slope of the peak exercise ST segment number of major vessels (0-3) colored by flourosopy thal: 0 = normal; 1 = fixed defect; 2 = reversable defect The names and social security numbers of the patients were recently removed from the database, replaced with dummy values. """

In [152]:
```
heart=pd.read_csv('heart.csv') #read the dataset from local drive

print(heart.head())  #top 5 rows from dataset
```

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

In [153]:
```
heart.keys()   #column names of dataset
```

Out[153]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

In [154]:
```
print(heart.describe())  # Return numerical summary of each attribute of iris
```

```
              Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
count  918.000000  918.000000   918.000000  918.000000  918.000000
mean    53.510893  132.396514   198.799564    0.233115  136.809368
std      9.432617   18.514154   109.384145    0.423046   25.460334
min     28.000000    0.000000     0.000000    0.000000   60.000000
25%     47.000000  120.000000   173.250000    0.000000  120.000000
50%     54.000000  130.000000   223.000000    0.000000  138.000000
75%     60.000000  140.000000   267.000000    0.000000  156.000000
max     77.000000  200.000000   603.000000    1.000000  202.000000

          Oldpeak  HeartDisease
count  918.000000    918.000000
mean     0.887364      0.553377
std      1.066570      0.497414
min     -2.600000      0.000000
25%      0.000000      0.000000
50%      0.600000      1.000000
75%      1.500000      1.000000
max      6.200000      1.000000
```

In [155]:
```
heartDF = pd.DataFrame(heart) #loading the data in a dataframe named as heartDF

heartDF
```

Out[155]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1

4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	0

918 rows × 12 columns

In [156]:
```
heartDF.info()    #there are 918 entries, I found
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

In [157]:
```
heartDF.isnull().sum()    #checking the null value
```

Out[157]:
```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

In [158]:
```
heartDF.info() #so the are 918 samples available
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

In [159]:
```
heartDF  #lets see our final dataframe again
```

Out[159]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	0

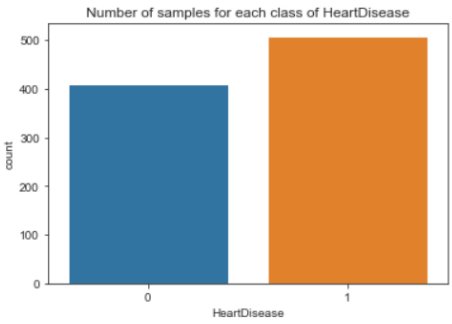
918 rows × 12 columns

In [160]:
```
heartDF.groupby('HeartDisease').size()  # let's check number of samples for each class of heartDF
```

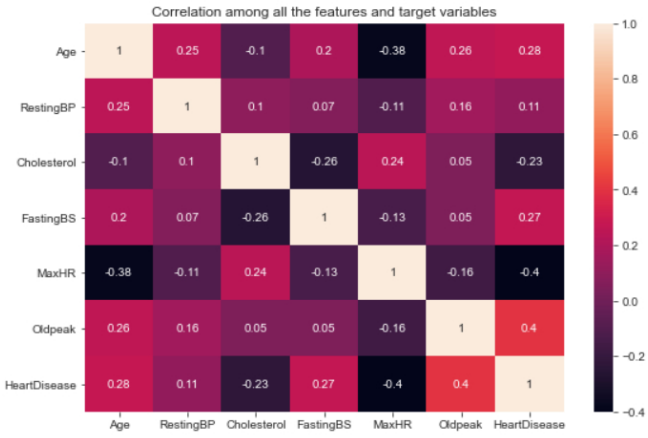
Out[160]:
```
HeartDisease
0    410
1    508
dtype: int64
```

"""upto this i found there are no disease for 410 samples and there are heart diseases for 508 samples.So I can easily find the difference(508-410=98) between '0'&'1' from whole dataset.Where dataset has total 918 samples. Next I will visualize this """

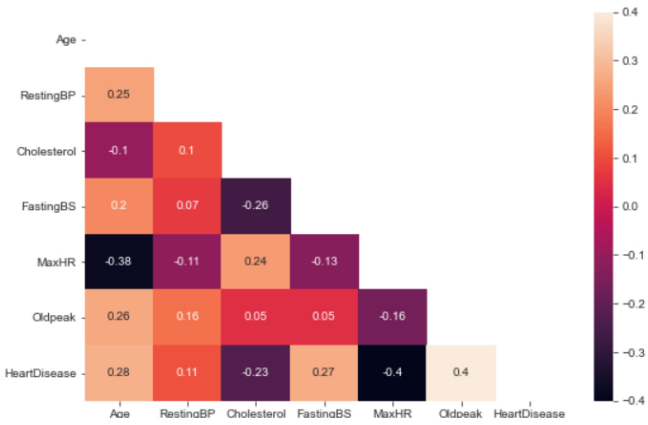
In [161]:
```
sns.countplot(x='HeartDisease', data=heartDF)   # let's visualise the number of samples for each class with count plot
plt.title("Number of samples for each class of HeartDisease");
```



In [162]:
```
#Lets find the correlation between features for higly positive/negative intercorrelation ship

correlation_matrix = heartDF.corr().round(2) ## corr() to calculate the correlation between feature and target variables

plt.figure(figsize = (9, 6)) # changing the figure size

sns.heatmap(data=correlation_matrix, annot=True);   # "annot = True" to print the values inside the square
plt.title("Correlation among all the features and target variables");
```



In [163]:
```
# Steps to remove redundant values
# Return a array filled with zeros
mask = np.zeros_like(correlation_matrix)
mask[np.triu_indices_from(mask)] = True   # Return the indices for the upper-triangle of array
plt.figure(figsize = (9, 6))     # changing the figure size
sns.heatmap(data=correlation_matrix, annot=True, mask=mask);           # "annot = True" to print the values inside the square
```

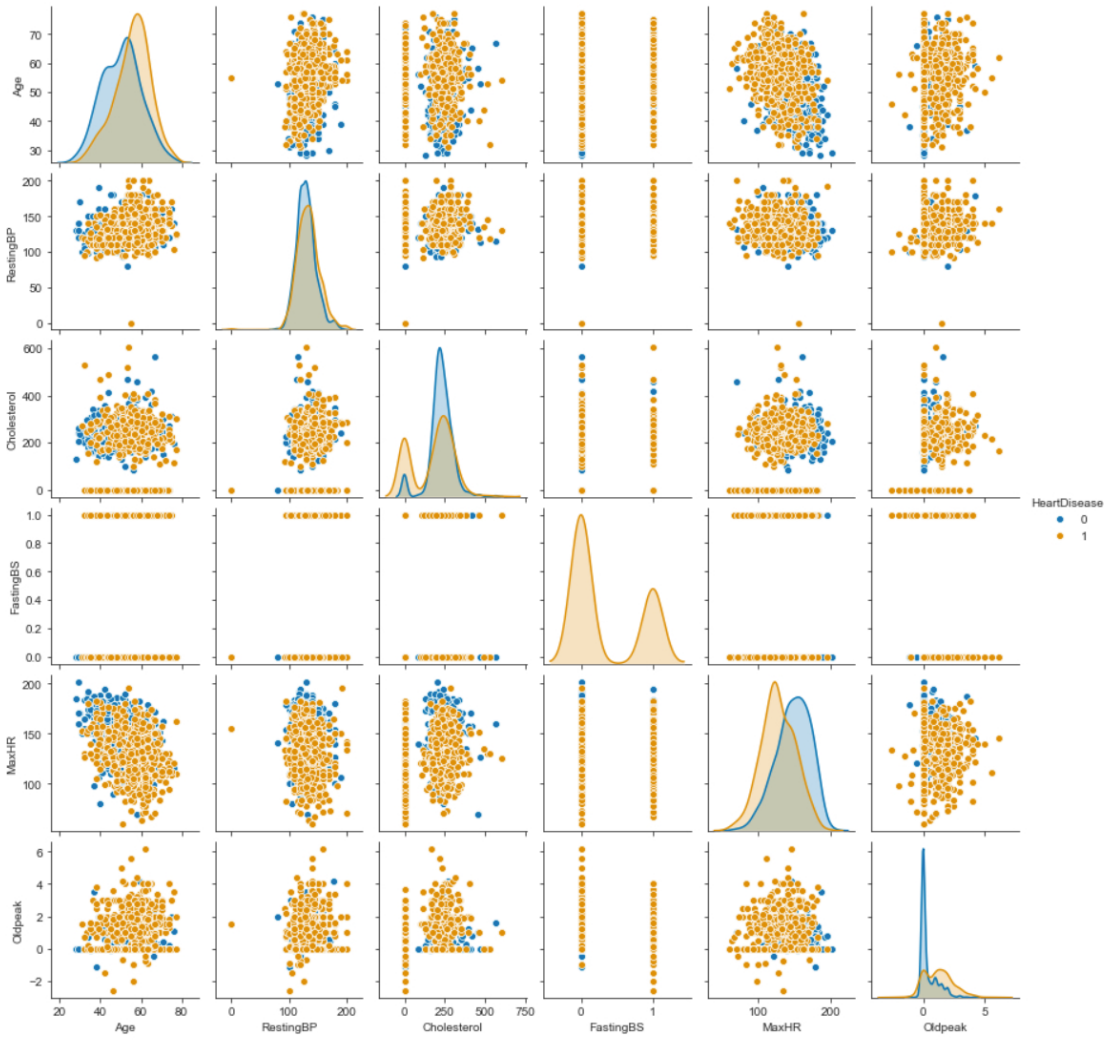


"""from heatmap I saw there are no values near about 1 or -1. Then, from the existing values I took top there features values which have higher correlation with target.

these are: 'Age'--> -0.38,'MaxHR'--> -0.4,'Oldpeak'--> 0.4"""

In [164]:
```
, hue="HeartDisease", height = 2, palette = 'colorblind'); #pairplot---> All the points are overlapping.Its hard to seperate 0,1.
```

```
C:\Users\GHJ\anaconda3\lib\site-packages\seaborn\distributions.py:369: UserWarning: Default bandwidth for data is 0; skipping d
ensity estimation.
  warnings.warn(msg, UserWarning)
```



In [165]:
```
#created a new dataframe as named new_heartDF with top correlated features from the existing
new_heartDF = pd.DataFrame(heartDF, columns=['Age','MaxHR','Oldpeak','HeartDisease'])
# Print the DataFrame
new_heartDF
```

Out[165]:

	Age	MaxHR	Oldpeak	HeartDisease
0	40	172	0.0	0
1	49	156	1.0	1
2	37	98	0.0	0
3	48	108	1.5	1
4	54	122	0.0	0
...	...	...	...	...
913	45	132	1.2	1
914	68	141	3.4	1
915	57	115	1.2	1
916	57	174	0.0	1
917	38	173	0.0	0

918 rows × 4 columns

In [166]:
```
X1=new_heartDF[['Age','MaxHR','Oldpeak']]   # Feature matrix
```

```
X1
```

Out[166]:

	Age	MaxHR	Oldpeak
0	40	172	0.0
1	49	156	1.0
2	37	98	0.0
3	48	108	1.5
4	54	122	0.0
...	...	...	...
913	45	132	1.2
914	68	141	3.4
915	57	115	1.2
916	57	174	0.0
917	38	173	0.0

918 rows × 3 columns

In [167]:
```
y1=new_heartDF[['HeartDisease']]    # Target variable
y1
```

Out[167]:

	HeartDisease
0	0
1	1
2	0
3	1
4	0
...	...
913	1
914	1
915	1
916	1
917	0

918 rows × 1 columns

Split the dataset

In [168]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size = 0.3, random_state = 2)
print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
print("y_train shape: ", y_train.shape)
print("y_test shape: ", y_test.shape)
```

```
X_train shape:  (642, 3)
X_test shape:  (276, 3)
y_train shape:  (642, 1)
y_test shape:  (276, 1)
```

"""for checking linear relationship I will call linear regression model"""

In [169]:
```
from sklearn.linear_model import LinearRegression
# Create LinearRegression Instance
lrm = LinearRegression()

# Fit data on to the model
lrm.fit(X_train, y_train)

# Predict
y_predicted = lrm.predict(X_test)
```

Evaluate Model

Now, let's evaluate how well our model did using metrics r-squared and root mean squared error (rmse). The r-squared value shows how strong our features determined the target value. The rmse defines the difference between predicted and the test values. The higher the value of the rmse, the less accurate the model.

In [170]:
```
from sklearn.metrics import mean_squared_error
# calculate r-squared values
r2 = lrm.score(X_test, y_test)
# calculate the difference between predicted and the test values.
rmse = (np.sqrt(mean_squared_error(y_test, y_predicted)))
# print the both values
print('-----------------------------------------')
print('r-squared: {}'.format(r2))
print('-----------------------------------------')
print('root mean squared error: {}'.format(rmse))
```

```
print('-------------------------------------------')
```

```
-------------------------------------------
r-squared: 0.2693969214142561
-------------------------------------------
root mean squared error: 0.4270960020347691
-------------------------------------------
```

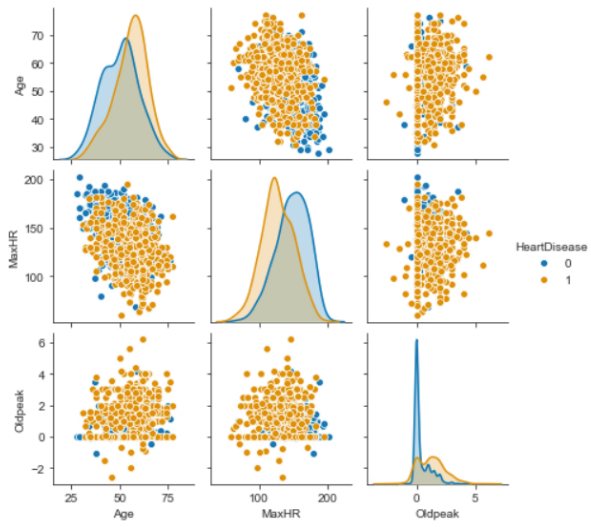
""" I found a high rmse value which means features are not very much strong"""

Let's plot predictions vs actual

In [171]:
```
plt.figure(figsize=(10,8))
sns.regplot(x=y_predicted, y=y_test)
plt.xlabel('Predicted HeartDisease', fontsize=15, color='blue')
plt.ylabel('Actual  HeartDisease', fontsize=15, color='green')
plt.title("Predicted Boston Housing Prices vs. Actual Price (in $1000)",
 fontsize=18, color='magenta');
```

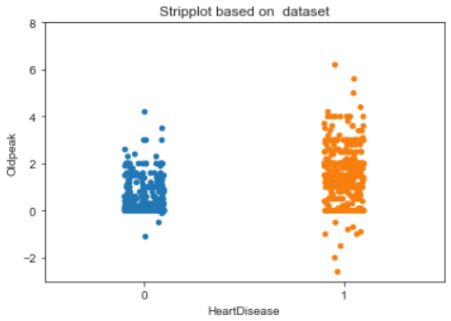
"""I found from the figure predicted and actual disease arenot in a linear relationship"""

In [172]:
```
#this is our pairplot from sorted  top correlated features

sns.pairplot(new_heartDF, hue="HeartDisease", height = 2, palette = 'colorblind');
```



In [173]:
```
#stripplot hearddisease vc oldpeak

sns.stripplot(x='HeartDisease', y='Oldpeak', data=new_heartDF,
```

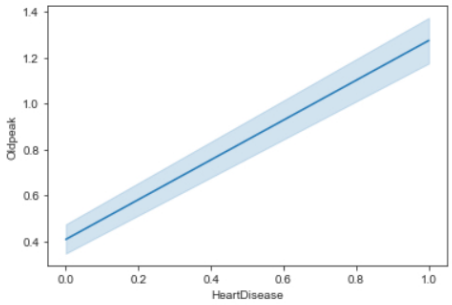
```
                  jitter=True,
                  dodge=True)
plt.ylim(-3, 8) # Setting ylim
plt.title("Stripplot based on  dataset");
```



""" from stripplot I observed:

no HeartDisease---> has maximum range(0 to 2) of Oldpeak and belongs -1 to 4 range HeartDisease---> has maximum range(0 to 4) of Oldpeak and belongs -2 to 6 range """

In [174]:
```
sns.lineplot(x='HeartDisease', y='Oldpeak', data=new_heartDF)
sns.set_style("ticks")
```

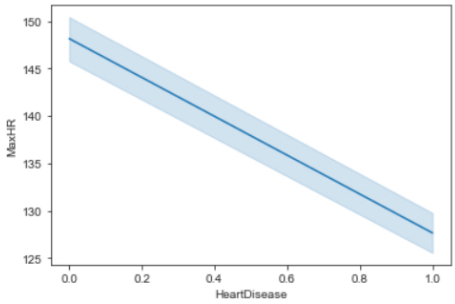


from 'HeartDisease' & 'Oldpeak' lineplot, I observed:

'HeartDisease' & 'Oldpeak' have linear relationship. IF Oldpeak is increased then heart disease also increased and reversed. """

In [175]:
```
sns.lineplot(x='HeartDisease', y='MaxHR', data=new_heartDF)
```

Out[175]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24760517340>
```

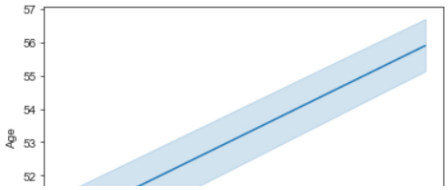


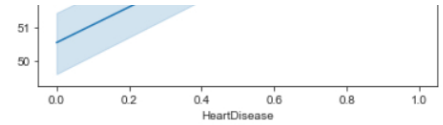
from 'HeartDisease' vs 'MaxHR' lineplot ,I observed:

'HeartDisease' & 'MaxHR' have linear relationship. IF MaxHR is increased then heart disease is decreased and reversed. """

In [176]:
```
sns.lineplot(x='HeartDisease', y='Age', data=new_heartDF)
```

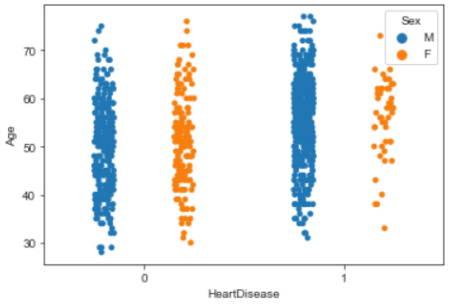
Out[176]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24760c099d0>
```





In [177]:
```
sns.stripplot(x='HeartDisease', y='Age', data=heartDF,
              jitter=True,
              hue='Sex',
              dodge=True)
```

Out[177]: <matplotlib.axes._subplots.AxesSubplot at 0x247608ba9a0>

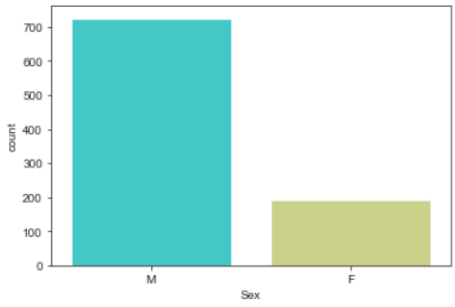


from 'HeartDisease' vs 'Age' stripplot based on gender ,I observed:

in case of no HeartDisease: both male and female belongs same result for 30 to 70+ age in case of HeartDisease: male--> max range of age is 30 to 70 female--> max range of age is 45 to 65

In [178]:
```
sns.countplot(x="Sex", data=heartDF,
 palette="rainbow")
```

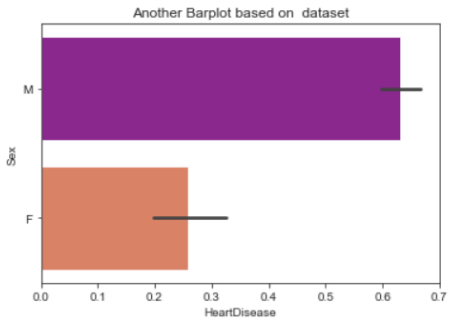
Out[178]: <matplotlib.axes._subplots.AxesSubplot at 0x247608422b0>



from countplot based on gender ,I observed:

male samples are greater than female samples in my dataset

In [179]:
```
sns.barplot(x="HeartDisease", y="Sex", data=heartDF,
 palette="plasma")
plt.title("Another Barplot based on  dataset");
```



from barplot based on HeartDisease vs sex ,I observed:

in range of 0 to 0.5,female has HeartDisease in range of 0 to 0.6,male has HeartDisease

In [180]:
```
_heartDF=heartDF.drop(columns=['ChestPainType', 'RestingECG'])   #here I dropped two colums and stored in _heartDF

_heartDF
```

Out[180]:

	Age	Sex	RestingBP	Cholesterol	FastingBS	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	140	289	0	172	N	0.0	Up	0
1	49	F	160	180	0	156	N	1.0	Flat	1
2	37	M	130	283	0	98	N	0.0	Up	0
3	48	F	138	214	0	108	Y	1.5	Flat	1
4	54	M	150	195	0	122	N	0.0	Up	0
...	...	...	...	...	...	...	...	...	...	...
913	45	M	110	264	0	132	N	1.2	Flat	1
914	68	M	144	193	1	141	N	3.4	Flat	1
915	57	M	130	131	0	115	Y	1.2	Flat	1
916	57	F	130	236	0	174	N	0.0	Flat	1
917	38	M	138	175	0	173	N	0.0	Up	0

918 rows × 10 columns

In [181]:
```
#here I replace string values with numerical value for easy analysis

_heartDF['Sex'] = heartDF['Sex'].replace(['F','M'], [0,1])
_heartDF['ExerciseAngina'] = heartDF['ExerciseAngina'].replace(['N','Y'], [0,1])
_heartDF['ST_Slope'] = heartDF['ST_Slope'].replace(['Flat','Up','Down'], [0,1,2])
# Print the DataFrame
_heartDF
```

Out[181]:

	Age	Sex	RestingBP	Cholesterol	FastingBS	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	1	140	289	0	172	0	0.0	1	0
1	49	0	160	180	0	156	0	1.0	0	1
2	37	1	130	283	0	98	0	0.0	1	0
3	48	0	138	214	0	108	1	1.5	0	1
4	54	1	150	195	0	122	0	0.0	1	0
...	...	...	...	...	...	...	...	...	...	...
913	45	1	110	264	0	132	0	1.2	0	1
914	68	1	144	193	1	141	0	3.4	0	1
915	57	1	130	131	0	115	1	1.2	0	1
916	57	0	130	236	0	174	0	0.0	0	1
917	38	1	138	175	0	173	0	0.0	1	0

918 rows × 10 columns

lets create feature and target matrix

In [182]:
```
X=_heartDF.iloc[:,:-1].values
y=_heartDF.iloc[:,-1].values
```

splitting dataset in 70% , 30% for train & test

In [183]:
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test=train_test_split(X,y,random_state=101,test_size=0.3)
```

importing the classifiers

In [184]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC
import pandas as pd
import xgboost as xgb
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

In [185]:
```
#calling classifiers

GNB_model = GaussianNB()
SVC_model=svm.SVC()
```

```
LogisticR_model=LogisticRegression(random_state=101)
KNN_model=KNeighborsClassifier(n_neighbors=5)
Dtree_model=DecisionTreeClassifier()
Random_model=RandomForestClassifier(n_estimators=10)
XGBOOST_model=xgb.XGBClassifier(objective='binary:logistic',seed=101)
GRADIENT_model=GradientBoostingClassifier(n_estimators=10, learning_rate=1.0,max_depth=1, random_state=101)
ADABOOST_model=AdaBoostClassifier(n_estimators=10,learning_rate=1)
BAGGINGBOOST_model = BaggingClassifier(base_estimator=SVC(),n_estimators=10, random_state=101)
```

fit the model

In [186]:
```
GNB_model.fit(X_train,y_train)
SVC_model.fit(X_train,y_train)
LogisticR_model.fit(X_train,y_train)
KNN_model.fit(X_train,y_train)
Dtree_model.fit(X_train,y_train)
Random_model.fit(X_train,y_train)
XGBOOST_model.fit(X_train,y_train)
GRADIENT_model.fit(X_train,y_train)
ADABOOST_model.fit(X_train,y_train)
BAGGINGBOOST_model.fit(X_train,y_train)
```

```
[18:52:52] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
```

```
C:\Users\GHJ\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (s
tatus=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\GHJ\anaconda3\lib\site-packages\xgboost\sklearn.py:1146: UserWarning: The use of label encoder in XGBClassifier is dep
recated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=Fal
se when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_c
lass - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

Out[186]:
```
BaggingClassifier(base_estimator=SVC(), random_state=101)
```

lets predict

In [187]:
```
#GNB_prediction=GNB_model.predict(X_test)
SVC_prediction=SVC_model.predict(X_test)
LogisticR_prediction=LogisticR_model.predict(X_test)
KNN_prediction=KNN_model.predict(X_test)
Dtree_prediction=Dtree_model.predict(X_test)
Random_prediction=Random_model.predict(X_test)
XGBOOST_prediction=XGBOOST_model.predict(X_test)
GRADIENT_prediction=GRADIENT_model.predict(X_test)
ADABOOST_prediction=ADABOOST_model.predict(X_test)
BAGGINGBOOST_prediction=BAGGINGBOOST_model.predict(X_test)
```

accuracy of prediction

In [188]:
```
#print(accuracy_score(GNB_prediction,y_test))
print(accuracy_score(SVC_prediction,y_test))
print(accuracy_score(LogisticR_prediction,y_test))
print(accuracy_score(KNN_prediction,y_test))
print(accuracy_score(Dtree_prediction,y_test))
print(accuracy_score(Random_prediction,y_test))
print(accuracy_score(XGBOOST_prediction,y_test))
print(accuracy_score(GRADIENT_prediction,y_test))
print(accuracy_score(ADABOOST_prediction,y_test))
print(accuracy_score(BAGGINGBOOST_prediction,y_test))
```

```
0.7028985507246377
0.8297101449275363
0.6920289855072463
0.8188405797101449
0.822463768115942
0.822463768115942
0.8405797101449275
0.7934782608695652
0.6884057971014492
```

from the result of accuracy ,I observed:

The gradient boost has the best accuracy 0.84057 among all.

In [ ]:
```

```

THANK YOU