ara218 / Ara218 Private

Code Issues Pull requests Projects Wiki Security Insights Sett

main

Ara218 / HomeWork / homework_1.ipynb

 ara218 finishing HW 1 History

1 contributor

1.07 MB

In [27]:
```
import pandas as pd
```

In [28]:
```
coulmns = ["Age", "Sex", "ChestPainType", "RestingBP", "Cholesterol",
    "FastingBloodSugar", "RestingECG", "MaxHeartRate", "ExerciseInducedAngina",
    "Oldpeak", "Slope", "MajorVessels", "Thal", "Target"]

df = pd.read_csv("C:\\Users\\alsae\\Desktop\\ML\\Ara218\\data\\heart.dat", sep=
```

In [29]:
```
df
```

Out[29]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBloodSugar	RestingECG
0	70.0	1.0	4.0	130.0	322.0	0.0	2.0
1	67.0	0.0	3.0	115.0	564.0	0.0	2.0
2	57.0	1.0	2.0	124.0	261.0	0.0	0.0
3	64.0	1.0	4.0	128.0	263.0	0.0	0.0
4	74.0	0.0	2.0	120.0	269.0	0.0	2.0
...	...	...	...	...	...	...	...
265	52.0	1.0	3.0	172.0	199.0	1.0	0.0
266	44.0	1.0	2.0	120.0	263.0	0.0	0.0
267	56.0	0.0	2.0	140.0	294.0	0.0	2.0
268	57.0	1.0	4.0	140.0	192.0	0.0	0.0
269	67.0	1.0	4.0	160.0	286.0	0.0	2.0

270 rows × 14 columns

In [30]:
```
df.head()
```

Out[30]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBloodSugar	RestingECG	M
0	70.0	1.0	4.0	130.0	322.0	0.0	2.0	
1	67.0	0.0	3.0	115.0	564.0	0.0	2.0	
2	57.0	1.0	2.0	124.0	261.0	0.0	0.0	
3	64.0	1.0	4.0	128.0	263.0	0.0	0.0	
4	74.0	0.0	2.0	120.0	269.0	0.0	2.0	

In [31]:
```
df.tail()
```

```
df.tail()
```

Out[31]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBloodSugar	RestingECG
265	52.0	1.0	3.0	172.0	199.0	1.0	0.0
266	44.0	1.0	2.0	120.0	263.0	0.0	0.0
267	56.0	0.0	2.0	140.0	294.0	0.0	2.0
268	57.0	1.0	4.0	140.0	192.0	0.0	0.0
269	67.0	1.0	4.0	160.0	286.0	0.0	2.0

In [32]:

```
print(df.size)
print(df.shape)
```

```
3780
(270, 14)
```

In [33]:

```
print('bellow are the amount of missing elemnts in the dataset')
df.isnull().sum()
```

```
bellow are the amount of missing elemnts in the dataset
```

Out[33]:

```
Age                    0
Sex                    0
ChestPainType          0
RestingBP              0
Cholesterol            0
FastingBloodSugar      0
RestingECG             0
MaxHeartRate           0
ExerciseInducedAngina  0
Oldpeak                0
Slope                  0
MajorVessels           0
Thal                   0
Target                 0
dtype: int64
```

In [34]:

```
cleaned_df = df.dropna()
cleaned_df
```

Out[34]:

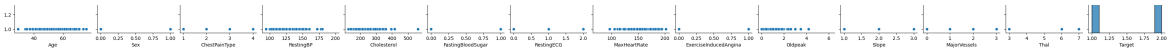
	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBloodSugar	RestingECG
0	70.0	1.0	4.0	130.0	322.0	0.0	2.0
1	67.0	0.0	3.0	115.0	564.0	0.0	2.0
2	57.0	1.0	2.0	124.0	261.0	0.0	0.0
3	64.0	1.0	4.0	128.0	263.0	0.0	0.0
4	74.0	0.0	2.0	120.0	269.0	0.0	2.0
...	...	...	...	...	...	...	...

265	52.0	1.0	3.0	172.0	199.0	1.0	0.0
266	44.0	1.0	2.0	120.0	263.0	0.0	0.0
267	56.0	0.0	2.0	140.0	294.0	0.0	2.0
268	57.0	1.0	4.0	140.0	192.0	0.0	0.0
269	67.0	1.0	4.0	160.0	286.0	0.0	2.0

270 rows × 14 columns

In [35]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [36]:
```
sns.pairplot(cleaned_df)
plt.show()

print("Statistics of the Dataframe:")
print(cleaned_df.describe())
```



```
Statistics of the Dataframe:
              Age         Sex  ChestPainType   RestingBP  Cholesterol  \
count  270.000000  270.000000     270.000000  270.000000   270.000000
mean    54.433333    0.677778       3.174074  131.344444   249.659259
std      9.109067    0.468195       0.950090   17.861608    51.686237
min     29.000000    0.000000       1.000000   94.000000   126.000000
25%     48.000000    0.000000       3.000000  120.000000   213.000000
50%     55.000000    1.000000       3.000000  130.000000   245.000000
75%     61.000000    1.000000       4.000000  140.000000   280.000000
max     77.000000    1.000000       4.000000  200.000000   564.000000

       FastingBloodSugar  RestingECG  MaxHeartRate  ExerciseInducedAngina  \
count         270.000000  270.000000    270.000000             270.000000
mean            0.148148    1.022222    149.677778               0.329630
std             0.355906    0.997891     23.165717               0.470952
min             0.000000    0.000000     71.000000               0.000000
25%             0.000000    0.000000    133.000000               0.000000
50%             0.000000    2.000000    153.500000               0.000000
75%             0.000000    2.000000    166.000000               1.000000
max             1.000000    2.000000    202.000000               1.000000

         Oldpeak       Slope  MajorVessels        Thal      Target
count  270.00000  270.000000    270.000000  270.000000  270.000000
mean     1.05000    1.585185      0.670370    4.696296    1.444444
std      1.14521    0.614390      0.943896    1.940659    0.497827
min      0.00000    1.000000      0.000000    3.000000    1.000000
25%      0.00000    1.000000      0.000000    3.000000    1.000000
50%      0.80000    2.000000      0.000000    3.000000    1.000000
75%      1.60000    2.000000      1.000000    7.000000    2.000000
max      6.20000    3.000000      3.000000    7.000000    2.000000
```

In [37]:

```
nominal_columns = ['Sex', 'ChestPainType', 'FastingBloodSugar', 'RestingECG', '
ordered_columns = ['MajorVessels']

categorical_columns = nominal_columns + ordered_columns

dummy_list = pd.get_dummies(cleaned_df, columns=categorical_columns, prefix=cat

dummy_list.head()
```

Out[37]:

	Age	RestingBP	Cholesterol	MaxHeartRate	Oldpeak	Target	Sex-0.0	Sex-1.0	ChestPain
0	70.0	130.0	322.0	109.0	2.4	2	0	1	
1	67.0	115.0	564.0	160.0	1.6	1	1	0	
2	57.0	124.0	261.0	141.0	0.3	2	0	1	
3	64.0	128.0	263.0	105.0	0.2	1	0	1	
4	74.0	120.0	269.0	121.0	0.2	1	1	0	

5 rows × 29 columns

the coorlation between age and heart rate is roughly a negtive linear corrlation

the correlation between age and heart rate is roughly a negative linear correlation

In [38]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

In [39]:
```
y = cleaned_df['Target'].values
cleaned_df.drop('Target', axis=1, inplace=True)
```

In [40]:
```
x = cleaned_df.values
```

In [41]:
```
print("Shape of x:", x.shape)
print("Shape of y:", y.shape)
```

```
Shape of x: (270, 13)
Shape of y: (270,)
```

In [42]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, stratify=y, test_size=0.2
```

In [43]:
```
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(xtrain,ytrain)
```

Out[43]:
```
KNeighborsClassifier(n_neighbors=4)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [44]:
```
#  Predict xtest and view first 25 predicitons
print(knn.predict(xtest)[:25])

# Compare prediction with real ytest 25 predictions
print(ytest[:25])

# Print the score with test data
print(knn.score(xtest,ytest))

#rescale only real value columns
realcols = cleaned_df.select_dtypes(include=np.number).columns

# For each column normalize ```df[col] as (x - mean) / standard_deviation```
for col in realcols:
  mean = cleaned_df[col].mean()
  std = cleaned_df[col].std()
  cleaned_df[col] = (cleaned_df[col]-mean) / std
```

```
[1 1 1 1 2 2 2 1 1 1 2 1 2 2 2 1 1 1 1 1 1 2 1 1 1]
[1 2 2 2 2 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 2 2 2 1 1]
0.6617647058823529
```

In [45]:
```
x = cleaned_df.values
```

In [46]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, stratify=y, test_size=0.2
```

In [47]:
```
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(xtrain,ytrain)
```

Out[47]:
```
KNeighborsClassifier(n_neighbors=4)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [48]:
```
knn.score(xtest,ytest)
```

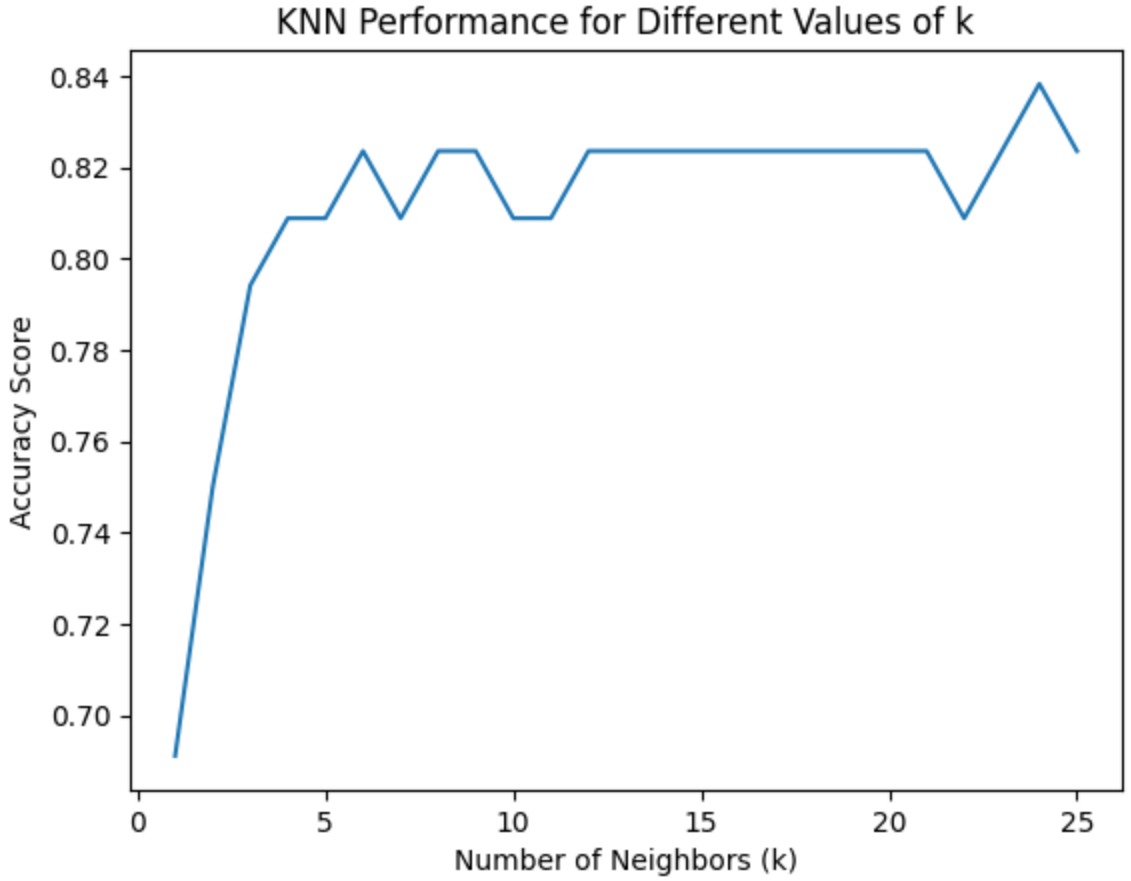
Out[48]:
```
0.8088235294117647
```

Lets analyze the difference between two modeling strategies (data normalization)
Compare score with and without data normalization process and explain

without the data normalization was .661 with the data normalization it is .808 normalization helps ensure fair and meaningful comparisons between features. It contributes to the stability, efficiency, and overall performance of the model.

In [53]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split


def returnScore(k, xtrain, xtest, ytrain, ytest):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(xtrain, ytrain)
    return knn.score(xtest, ytest)

k_values = range(1, 26)

result = [returnScore(k, xtrain, xtest, ytrain, ytest) for k in k_values]

plt.plot(k_values, result)
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy Score')
plt.title('KNN Performance for Different Values of k')
plt.show()

best_k = np.argmax(result) + 1
print('Best Value of k:', best_k)

best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(xtrain, ytrain)

best_knn_score = best_knn.score(xtest, ytest)
```

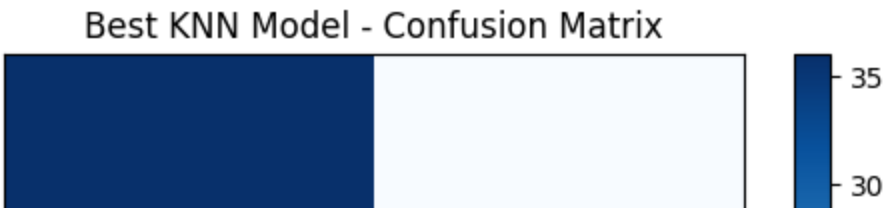
```
best_knn_score = best_knn.score(xtest, ytest)
print('Accuracy on Test Data (using best k):', best_knn_score)

ypred = best_knn.predict(xtest)
matrix = confusion_matrix(ytest, ypred)
print('Confusion Matrix:')
print(matrix)
```

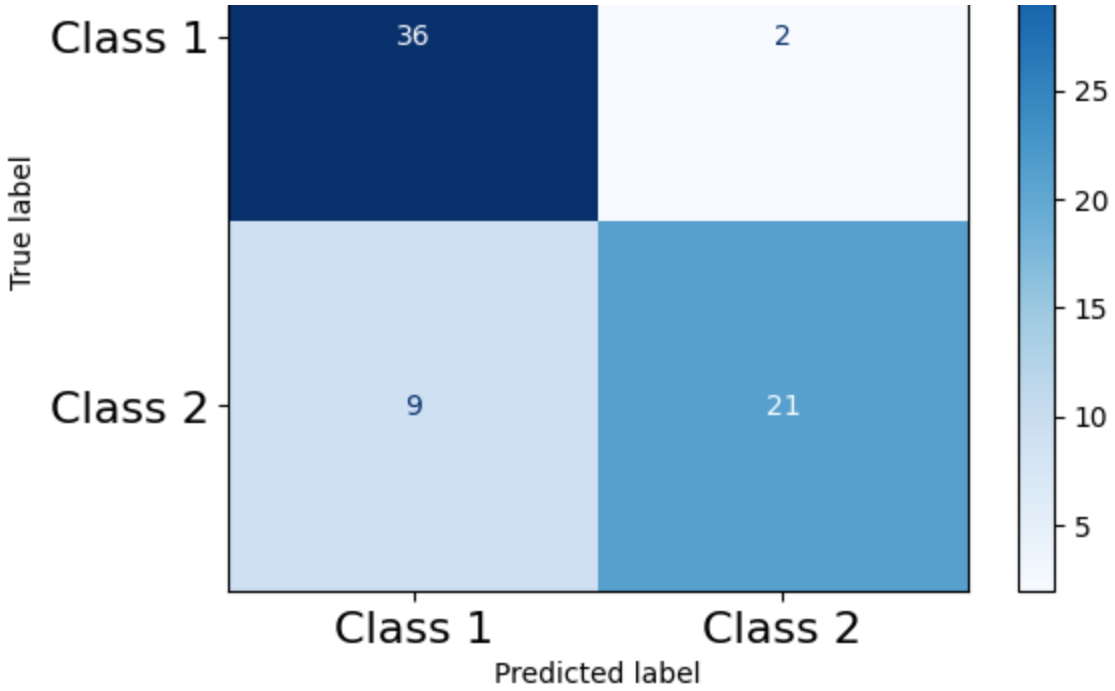



```
Best Value of k: 24
Accuracy on Test Data (using best k): 0.8382352941176471
Confusion Matrix:
[[36  2]
 [ 9 21]]
```

In [54]:

```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm_display = ConfusionMatrixDisplay(confusion_matrix=matrix, display_labels=["C
cm_display.plot(cmap='Blues', values_format='d')

plt.title("Best KNN Model - Confusion Matrix")
plt.xticks(range(2), ["Class 1","Class 2"], fontsize=16)
plt.yticks(range(2), ["Class 1","Class 2"], fontsize=16)
plt.show()
```

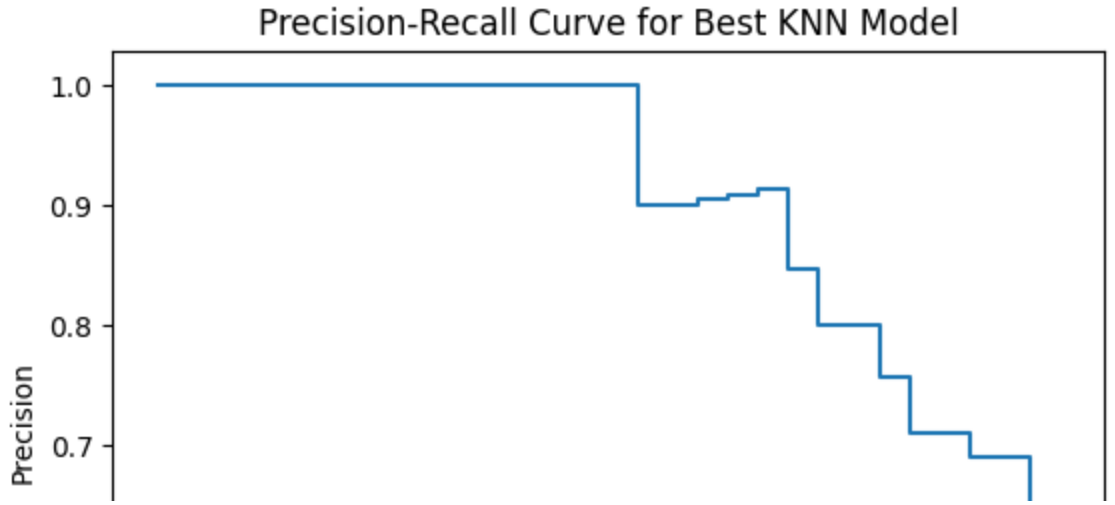


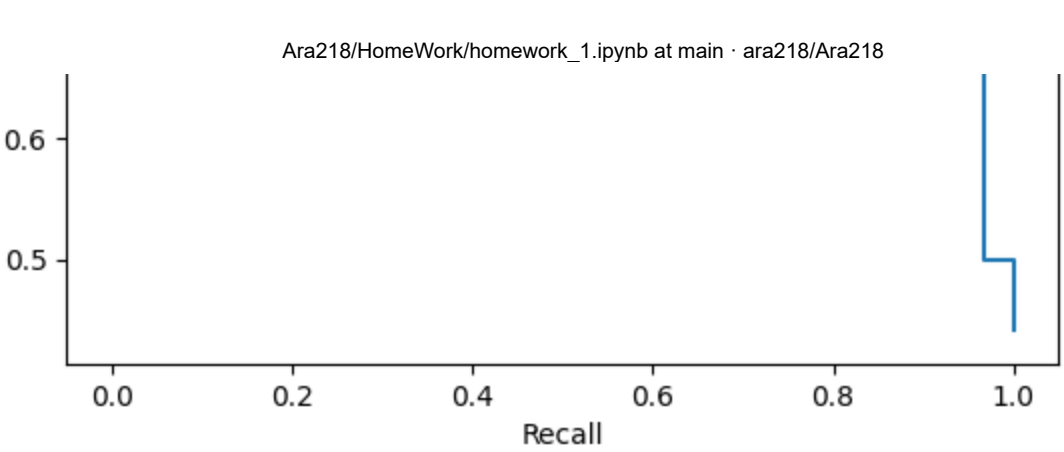


In [57]:

```
from sklearn.metrics import mean_squared_error, precision_recall_curve, Precisi
import matplotlib.pyplot as plt

mse = mean_squared_error(ytest, ypred)  # Calculate the test MSE
print("Test mean squared error (MSE): {:.2f}".format(mse))

accuracy = best_knn.score(xtest, ytest)
print("Test Accuracy: {:.2f}".format(accuracy))

ytest_binary = (ytest == 2).astype(int)
ypred_binary = (ypred == 2).astype(int)

precision, recall, _ = precision_recall_curve(ytest_binary, best_knn.predict_pr
pr_display = PrecisionRecallDisplay(precision=precision, recall=recall)
pr_display.plot()

plt.title("Precision-Recall Curve for Best KNN Model")
plt.show()
```

```
Test mean squared error (MSE): 0.16
Test Accuracy: 0.84
```




0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0
Recall