[22]:

	age	sex	cp	trestbps	chol	fbs
count	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000
mean	54.311321	0.688679	0.957547	131.784610	244.133256	0.132075
std	9.145339	0.464130	1.022537	17.755169	45.330324	0.339374
min	29.000000	0.000000	0.000000	93.944184	126.085811	0.000000
25%	47.000000	0.000000	0.000000	119.987220	212.793680	0.000000
50%	55.000000	1.000000	1.000000	130.021392	243.475116	0.000000
75%	61.000000	1.000000	2.000000	139.959811	269.275502	0.000000
max	77.000000	1.000000	3.000000	192.020200	406.932689	1.000000

	restecg	thalach	exang	oldpeak	slope	ca
count	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000
mean	0.570755	149.863490	0.344340	1.010168	1.419811	0.731132
std	0.532982	21.648149	0.476277	1.071093	0.622016	1.038762
min	0.000000	88.032613	0.000000	-0.185668	0.000000	0.000000
25%	0.000000	137.712696	0.000000	0.083715	1.000000	0.000000
50%	1.000000	150.955534	0.000000	0.889500	1.000000	0.000000
75%	1.000000	164.991594	1.000000	1.569735	2.000000	1.000000
max	2.000000	202.138041	1.000000	4.404773	2.000000	4.000000

	thal	target
count	212.000000	212.000000
mean	2.353774	0.542453
std	0.586042	0.499374
min	1.000000	0.000000
25%	2.000000	0.000000
50%	2.000000	1.000000
75%	3.000000	1.000000
max	3.000000	1.000000

Data Cleaning :

- the NaN values (missing values) were replaced with feature mean for numeric and median for other type of features.
- the noise in 'thal' (non categorical values) were handled by rounding to integer.

If we attempt to drop the missing values, the performance of the classifier was observed to be low. Moreover, dropping the values reduces the size of the dataset affecting performance.

Question 2: KNN Classification

[CM6]

Basic Model

[23]:
```
# Basic Model

# importing libraries
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
```

[24]:
```
#  one hot encoding the categorical features
df_heart_ohe=df_heart[choosen_features]
for i in choosen_features_cats:
    df_heart_ohe[i] = df_heart_ohe[i].astype(int)

df_heart_ohe=pd.get_dummies(df_heart_ohe, columns = choosen_features_cats)

df_heart_ohe.head()
```

```
<ipython-input-24-a5b7c39d0f66>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_heart_ohe[i] = df_heart_ohe[i].astype(int)
```

[24]:

	oldpeak	thalach	cp_0	cp_1	cp_2	cp_3	exang_0	exang_1	slope_0	slope_1	slope_2
0	1.284822	115.952071	0	0	1	0	1	0	0	1	0
1	3.110483	135.970028	1	0	0	0	0	1	0	1	0
2	-0.023723	152.210039	0	0	1	0	1	0	0	0	1
3	1.195082	143.049207	0	0	1	0	0	1	0	1	0
4	3.082052	143.099327	1	0	0	0	0	1	0	1	0

[25]:
```
# dividing data and target
y = df_heart['target']
X = df_heart_ohe
```

[26]:
```
# dividing the data into train, validation, and test sets (60%, 20%, 20%) with␣
 ↪random_state=275
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=275)

X_test, X_val , y_test, y_val = train_test_split(X_test, y_test, test_size=0.5,␣
 ↪random_state=275)
```

```
print(X_train.shape)
print(X_test.shape)
print(X_val.shape)
```

```
(127, 11)
(42, 11)
(43, 11)
```

[27]:
```
# train the model with the classifier's default parameters
knn = KNeighborsClassifier()
knn.fit(X_train, y_train.values.ravel())
y_pred = knn.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print('The accuracy of the basic KNN model with default parameters on the test set␣
 ↪is', accuracy * 100, '%')
```

```
The accuracy of the basic KNN model with default parameters on the test set is
73.80952380952381 %
```

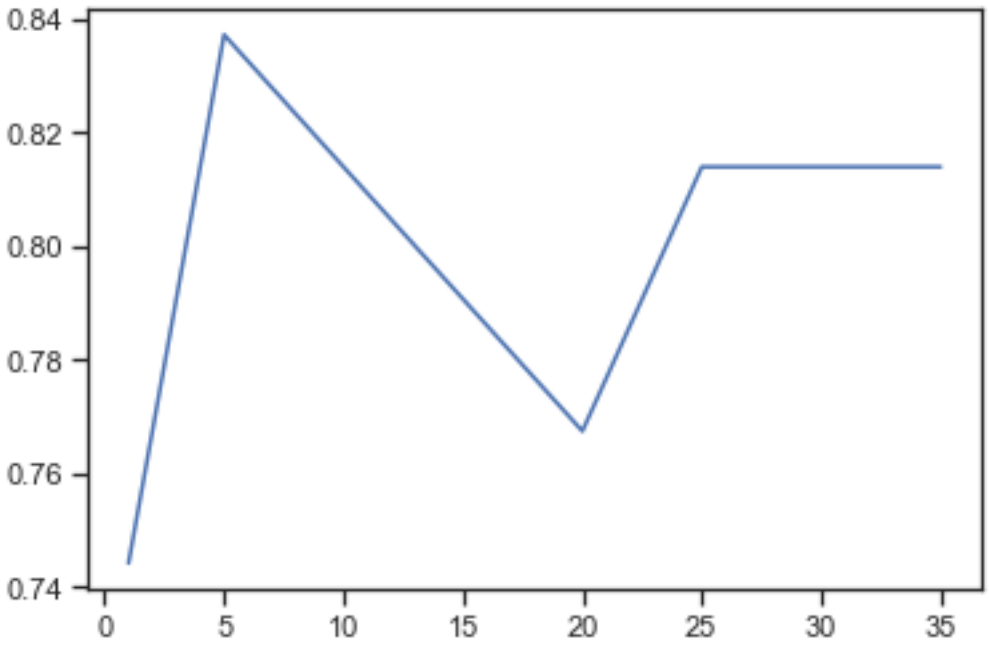
[28]:
```
# finding best parameter for the classifier
k_range = [1,5,10,15,20,25,30,35]
Scores = {}
Scores_list = []
best_k = 0
accuracy_max = 0
for k in k_range:
    knn =  KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train.values.ravel())
    y_pred = knn.predict(X_val)
    Scores[k] = metrics.accuracy_score(y_val, y_pred)
    Scores_list.append(metrics.accuracy_score(y_val, y_pred))

# plotting the graph showing relationship between the accuracy and parameters
plt.plot(k_range, Scores_list)

# finding best k value
accuracy = max(Scores_list)
key_list = list(Scores.keys())
val_list = list(Scores.values())
position = val_list.index(accuracy)
best_k = key_list[position]

print('The best value of k is', best_k)
print(val_list)
print('The accuracy of the basic KNN model on the validation set is', accuracy * 100,␣
 ↪'%')
```

```
The best value of k is 5
[0.7441860465116279, 0.8372093023255814, 0.813953488372093, 0.7906976744186046,
0.7674418604651163, 0.813953488372093, 0.813953488372093, 0.813953488372093]
The accuracy of the basic KNN model on the validation set is 83.72093023255815 %
```



We find that the model has the highest accuracy score for k=5

[CM7]

[29]:
```
# fitting the model on training set
knn =  KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train.values.ravel())

# predicting the target on test set
y_pred = knn.predict(X_test)

#calculating accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print('The accuracy of the basic KNN model with best k on the test set is', accuracy *␣
 ↪100, '%')

# calculating AUC
pred_prob = knn.predict_proba(X_test)
auc = roc_auc_score(y_test, pred_prob[:, 1], average = 'macro', multi_class = 'ovr')
print('AUC:', auc)

# calculating f-score
f_score = f1_score(y_test, y_pred, average = 'weighted')
print('f-score:', f_score)
```

```
The accuracy of the basic KNN model with best k on the test set is 73.80952380952381 %
AUC: 0.8901234567901234
f-score: 0.7433664667707222
```