Chronic Kidney Disease Prediction

Rayyan Kazim

2024-04-04

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from sklearn import neighbors
from sklearn import metrics
from sklearn.preprocessing import scale
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA, TruncatedSVD, FactorAnalysis
```

```python
df = pd.read_csv("kidney_disease.csv")
df.head(10)
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800
6	6	68.0	70.0	1.010	0.0	0.0	NaN	normal	notpresent	notpresent	...	36	NaN
7	7	24.0	NaN	1.015	2.0	4.0	normal	abnormal	notpresent	notpresent	...	44	6900
8	8	52.0	100.0	1.015	3.0	0.0	normal	abnormal	present	notpresent	...	33	9600
9	9	53.0	90.0	1.020	2.0	0.0	abnormal	abnormal	present	notpresent	...	29	12100

Classification Problem

We would like to use supervised machine learning to predict chronic kidney disease, as well as understand which variables help with the diagnosis of it.

Variable Transformation

```
df.dtypes
```

```
id                  int64
age               float64
bp                float64
sg                float64
al                float64
su                float64
rbc                object
pc                 object
pcc                object
ba                 object
bgr               float64
bu                float64
sc                float64
sod               float64
pot               float64
hemo              float64
pcv                object
wc                 object
rc                 object
htn                object
dm                 object
cad                object
appet              object
```

```
pe                  object
ane                 object
classification      object
dtype: object
```

```
df['sg'] = pd.Categorical(df['sg'])
df['al'] = pd.Categorical(df['al'])
df['su'] = pd.Categorical(df['su'])
```

pcv, wc and rc should actually be labeled as float64 and int64 variables. We will not transform them just yet as they only appear as objects because of N/A values. Later, in this analysis, we will drop the N/A values. Some of the variables that are labeled as float64 (continuous) should actually be labeled as int64 (integers), however, this will not make a difference in our calculations (for example, 4 equivalent to 4.0).

```
df.classification.value_counts()
```

```
classification
ckd        248
notckd     150
ckd\t        2
Name: count, dtype: int64
```

We should make sure that our classification only has 2 classes as individuals will either have ckd, or they will not. We will convert the observation "ckd"̂ to"ckd".

```
df['classification'] = df['classification'].replace('ckd\t','ckd')
```

```
df.classification.value_counts()
```

```
classification
ckd        250
notckd     150
Name: count, dtype: int64
```

Dataset Overview

```
df.describe()
```

	id	age	bp	bgr	bu	sc	sod	pot
count	400.000000	391.000000	388.000000	356.000000	381.000000	383.000000	313.000000	312.000
mean	199.500000	51.483376	76.469072	148.036517	57.425722	3.072454	137.528754	4.62724
std	115.614301	17.169714	13.683637	79.281714	50.503006	5.741126	10.408752	3.19390
min	0.000000	2.000000	50.000000	22.000000	1.500000	0.400000	4.500000	2.50000
25%	99.750000	42.000000	70.000000	99.000000	27.000000	0.900000	135.000000	3.80000
50%	199.500000	55.000000	80.000000	121.000000	42.000000	1.300000	138.000000	4.40000
75%	299.250000	64.500000	80.000000	163.000000	66.000000	2.800000	142.000000	4.90000
max	399.000000	90.000000	180.000000	490.000000	391.000000	76.000000	163.000000	47.0000

This dataset consists of 400 observations (we may define this as 400 individuals). The youngest individual in this is 2 years old, and the oldest is 90 years old. The average age of the individuals we are looking at is 51 years old. As seen in part 2, there is a fairly even balance between binary (object) and numerical (integer and float variables) variables in this dataset. The average sodium of all individuals has been found as an exact value of 137.528753 milliequivalents per litre, with a standard deviation of 10.408752.

Association Between Variables

```
df_numeric = df.select_dtypes(include=['float64'])
df_category = df.select_dtypes(include=['category'])
df_numcat = pd.concat([df_numeric, df_category], axis=1).reindex(df_numeric.index)
df_numcat.corr()
```

	age	bp	bgr	bu	sc	sod	pot	hemo	sg
age	1.000000	0.159480	0.244992	0.196985	0.132531	-0.100046	0.058377	-0.192928	-0.191

	age	bp	bgr	bu	sc	sod	pot	hemo	sg
bp	0.159480	1.000000	0.160193	0.188517	0.146222	-0.116422	0.075151	-0.306540	-0.218
bgr	0.244992	0.160193	1.000000	0.143322	0.114875	-0.267848	0.066966	-0.306189	-0.374
bu	0.196985	0.188517	0.143322	1.000000	0.586368	-0.323054	0.357049	-0.610360	-0.314
sc	0.132531	0.146222	0.114875	0.586368	1.000000	-0.690158	0.326107	-0.401670	-0.361
sod	-0.100046	-0.116422	-0.267848	-0.323054	-0.690158	1.000000	0.097887	0.365183	0.4121
pot	0.058377	0.075151	0.066966	0.357049	0.326107	0.097887	1.000000	-0.133746	-0.072
hemo	-0.192928	-0.306540	-0.306189	-0.610360	-0.401670	0.365183	-0.133746	1.000000	0.6025
sg	-0.191096	-0.218836	-0.374710	-0.314295	-0.361473	0.412190	-0.072787	0.602582	1.0000
al	0.122091	0.160689	0.379464	0.453528	0.399198	-0.459896	0.129038	-0.634632	-0.469
su	0.220866	0.222576	0.717827	0.168583	0.223244	-0.131776	0.219450	-0.224775	-0.296

The strong associations exist between serum creatinene and sodium, as well as between hemoglobin and blood urea. We see an even stronger association between sugar and blood glucose random. Hemoglibin also has fairly strong associations with albumin and specific gravity. We see that age and bp have weak associations with other variables, however, it is still really important to consider these variables in this particular analysis because increasing age and high blood pressure can definitely correlate with kidney disorders.

Because of the strong associations, our feature selection and extraction must incorporate hemoglobin because of it's relationship with multiple variables. Sodium, serum creatinene, age, blood pressure and blood urea are other variables we can use (we mentioned that sugar and blood glucose random have a strong association, however, their association with other variables is weak.). These variables will help us predict whether or not one has chronic kidney disease.

```
x = df_numcat[['age','hemo', 'sod', 'sc', 'bu', 'bp']]
y = df['classification']
```

Missing Value Analysis

```
x.isnull().head(10)
```

	age	hemo	sod	sc	bu	bp
0	False	False	True	False	False	False
1	False	False	True	False	False	False
2	False	False	True	False	False	False
3	False	False	False	False	False	False
4	False	False	True	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False
7	False	False	True	False	False	True
8	False	False	True	False	False	False
9	False	False	False	False	False	False

"True" indicates missing values. Being able to identify these entries confirms to us that we have missing values.

We will not drop any missing values because that will reduce our row numbers for some columns. When it comes to splitting the data for the training and testing stage, we will need to have a consistent number of rows between our predictor and response variables. For our response variable y, we have 400 observations. Hence, for x, we need to keep 400 observations.

We can replace NA values in each column with the mean value for that column. NA values may give us problems in our classifications, so a good substitute for them would be average values.

```
x['hemo'].fillna(x['hemo'].mean(), inplace = True)
x['sod'].fillna(x['sod'].mean(), inplace = True)
x['sc'].fillna(x['sc'].mean(), inplace = True)
x['bu'].fillna(x['bu'].mean(), inplace = True)
x['bp'].fillna(x['bp'].mean(), inplace = True)
x['age'].fillna(x['age'].mean(), inplace = True)
```

Outlier Analysis

We can use interquartile range to identify outlier values. Values falling outside of this interval,(Q1-1.5xIQR, Q3+1.5xIQR), are cosidered outliers. However, it is not important to identify these values as we will not be removing them. We will not be removing them because outliers can sometimes have an influence. In some cases, the removal of outliers can lead to potential bias as it decreases the number of observations being analysed.

Data Splitting

```
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.3, random_state=1, stratify=y)
```

Classifier Choices

We will use the Decision Tree classifier, as well as the KNN classifier.

Decision Tree classifications are very useful for these types of scenerios. When predicting whether or not someone has chronic kidney disease, the final results might be "yes" (has ckd) or "no" (does not have ckd). Decision Tree's are extremely clear for this as they will set a path with conditions to decide whther or not someone has ckd. In general, having a decision tree is very useful for classification when the target variable is binary (in our case, the "classification" variable is a binary object.).

KNN classifications are something we will use mainly because of their simplicity and high accuracy. These are the main reasons as to why we would like to use KNN algorithms for our classication problem. From part 4, where we obtained all the correlations between the numeric variables in the dataset, we were able to extract relevent predictor variables for our classification problem. Using KNN modeling, these variables can help predict whether or not someone has chronic kidney disease with high accuracy.

Performance Metrics

We will use accuracy scores and confusion matrices to compare our models. In terms of accuracy, the model with the higher accuracy score will be better. When looking at a confusion matrices, we should be seeing that our predicted values are fairly close to the actual values.

Classifier Comparision

KNN Classification

```
knn = neighbors.KNeighborsClassifier()
```

```
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

```
prediction1 = knn.predict(X_test)
print("Accuracy Score:", round(metrics.accuracy_score(y_test, prediction1),5))
```

```
Accuracy Score: 0.85
```

```
y_test
```

```
198       ckd
129       ckd
25        ckd
320    notckd
253    notckd
        ...
86        ckd
219       ckd
383    notckd
```

```
133       ckd
71        ckd
Name: classification, Length: 120, dtype: object
```

```python
prediction1
```

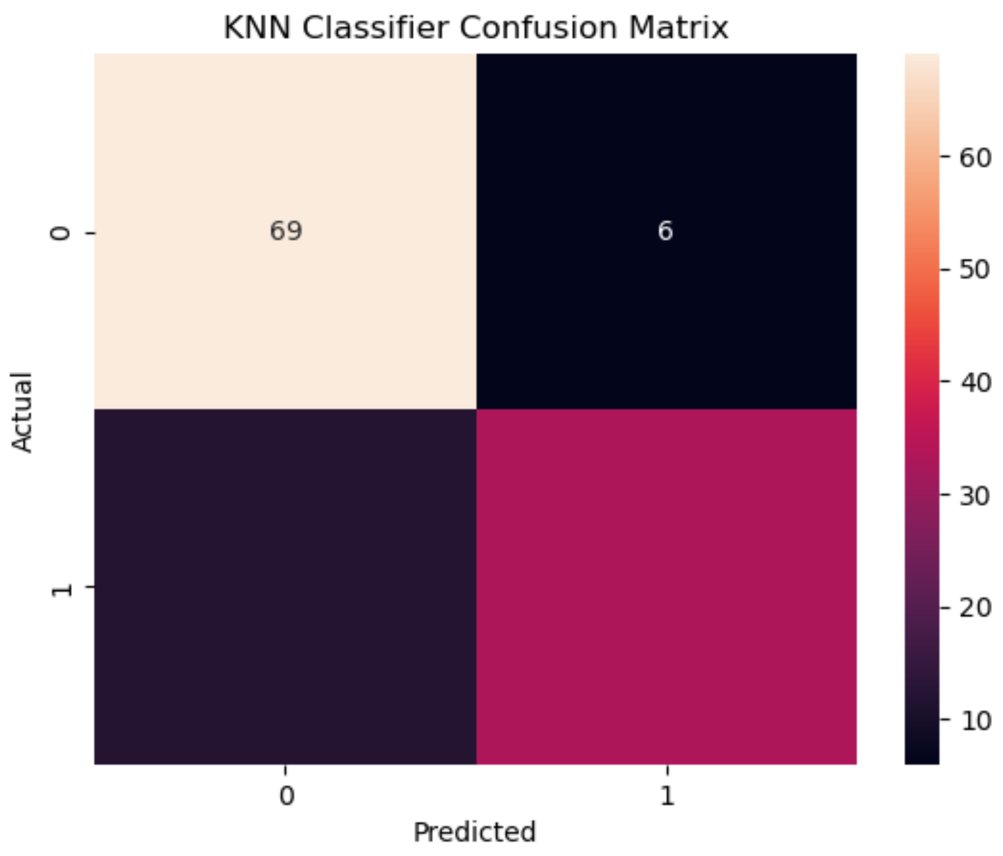
```
array(['ckd', 'ckd', 'ckd', 'notckd', 'notckd', 'ckd', 'ckd', 'ckd',
       'notckd', 'ckd', 'notckd', 'notckd', 'notckd', 'ckd', 'ckd', 'ckd',
       'notckd', 'ckd', 'notckd', 'ckd', 'ckd', 'notckd', 'ckd', 'ckd',
       'ckd', 'notckd', 'notckd', 'ckd', 'ckd', 'notckd', 'notckd', 'ckd',
       'notckd', 'ckd', 'notckd', 'notckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'notckd', 'ckd', 'notckd', 'ckd', 'ckd', 'ckd', 'notckd', 'ckd',
       'notckd', 'ckd', 'ckd', 'ckd', 'ckd', 'notckd', 'notckd', 'notckd',
       'ckd', 'notckd', 'ckd', 'ckd', 'ckd', 'ckd', 'notckd', 'ckd',
       'notckd', 'ckd', 'notckd', 'ckd', 'ckd', 'ckd', 'ckd', 'notckd',
       'ckd', 'ckd', 'ckd', 'notckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'notckd', 'ckd', 'notckd', 'ckd', 'ckd', 'notckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'notckd', 'ckd', 'ckd',
       'notckd', 'ckd', 'notckd', 'notckd', 'ckd', 'ckd', 'notckd',
       'notckd', 'notckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd'],
      dtype=object)
```

```python
confusion_matrix(y_test, prediction1)
```

```
array([[69,  6],
       [12, 33]])
```

```python
sns.heatmap(confusion_matrix(y_test, prediction1), annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("KNN Classifier Confusion Matrix")
```

```
Text(0.5, 1.0, 'KNN Classifier Confusion Matrix')
```



Decision Tree Classifier

```
dt = DecisionTreeClassifier()
```

```
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
prediction2 = dt.predict(X_test)
print("Accuracy Score:", round(metrics.accuracy_score(y_test, prediction2),5))
```

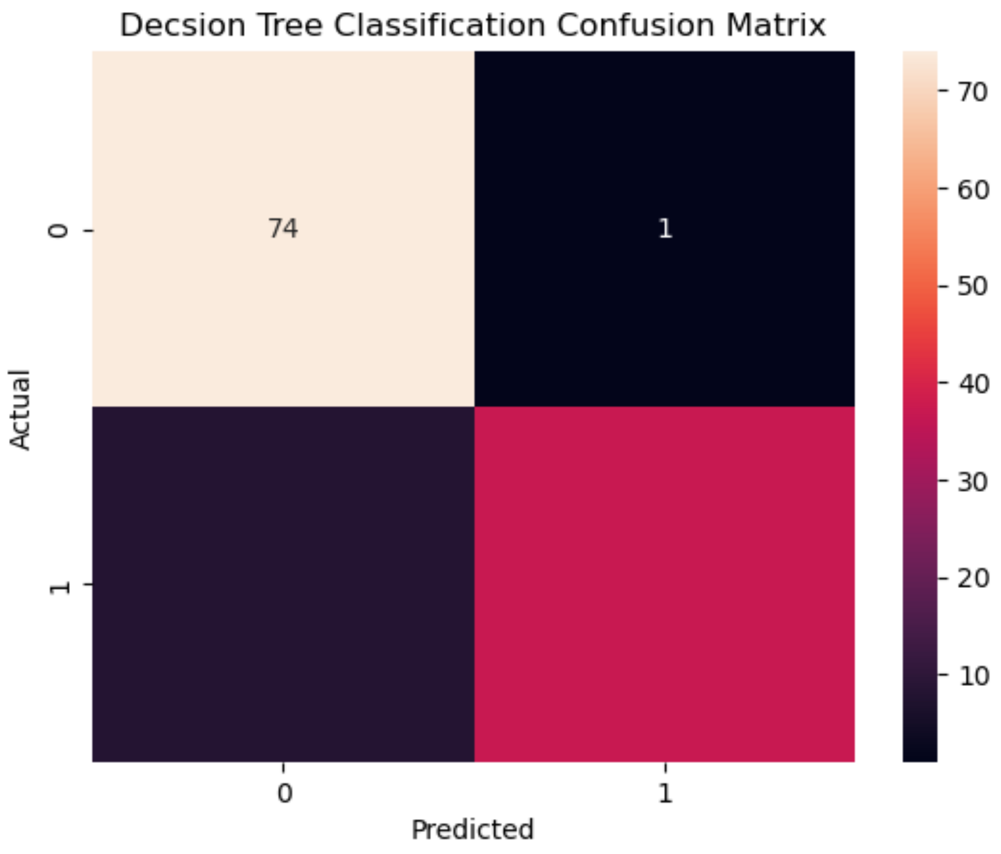
```
Accuracy Score: 0.925
```

```
confusion_matrix(y_test, prediction2)
```

```
array([[74,  1],
       [ 8, 37]])
```

```
sns.heatmap(confusion_matrix(y_test, prediction2), annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Decsion Tree Classification Confusion Matrix")
```

```
Text(0.5, 1.0, 'Decsion Tree Classification Confusion Matrix')
```



When comparing the classifiers based on the test set, we can see the the decision tree has the higher accuracy score, meaning that this classifier is better in terms of accuracy.

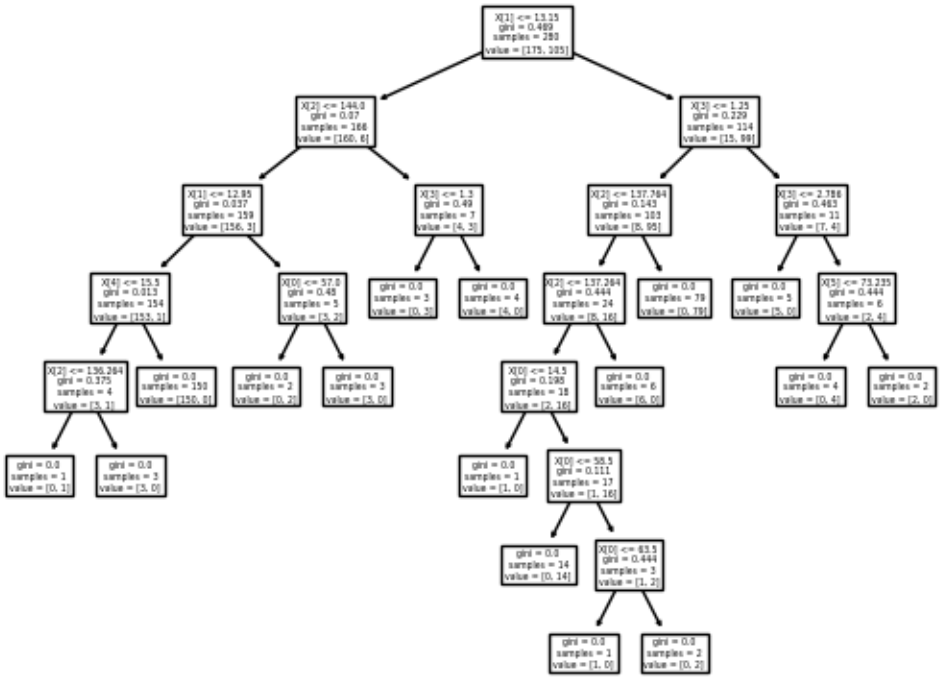
when comparing the two 2x2 confusion matrices, we will chose the one that has higher true positive and true negative values, and low false positive and false negative values. From these matrices, we can see that the decision tree classifier has the higher true positive and lower false positive value. This implies that the decision tree classifier is better, in terms of confusion matrix comparision.

Based on both of these performance metrics, we will conclude that Decision Tree is the better classifier, as it has a higher accuracy score.

Interpretable Classifier Insight

```
plot_tree(dt)
```

```
[Text(0.5595238095238095, 0.9375, 'X[1] <= 13.15\ngini = 0.469\nsamples = 280\nvalue = [175
 Text(0.35714285714285715, 0.8125, 'X[2] <= 144.0\ngini = 0.07\nsamples = 166\nvalue = [160
 Text(0.23809523809523808, 0.6875, 'X[1] <= 12.95\ngini = 0.037\nsamples = 159\nvalue = [15
 Text(0.14285714285714285, 0.5625, 'X[4] <= 15.5\ngini = 0.013\nsamples = 154\nvalue = [153
 Text(0.09523809523809523, 0.4375, 'X[2] <= 136.264\ngini = 0.375\nsamples = 4\nvalue = [3,
 Text(0.047619047619047616, 0.3125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.14285714285714285, 0.3125, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.19047619047619047, 0.4375, 'gini = 0.0\nsamples = 150\nvalue = [150, 0]'),
 Text(0.3333333333333333, 0.5625, 'X[0] <= 57.0\ngini = 0.48\nsamples = 5\nvalue = [3, 2]')
 Text(0.2857142857142857, 0.4375, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.38095238095238093, 0.4375, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.47619047619047616, 0.6875, 'X[3] <= 1.3\ngini = 0.49\nsamples = 7\nvalue = [4, 3]')
 Text(0.42857142857142855, 0.5625, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.5238095238095238, 0.5625, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.7619047619047619, 0.8125, 'X[3] <= 1.25\ngini = 0.229\nsamples = 114\nvalue = [15,
 Text(0.6666666666666666, 0.6875, 'X[2] <= 137.764\ngini = 0.143\nsamples = 103\nvalue = [8
 Text(0.6190476190476191, 0.5625, 'X[2] <= 137.264\ngini = 0.444\nsamples = 24\nvalue = [8,
 Text(0.5714285714285714, 0.4375, 'X[0] <= 14.5\ngini = 0.198\nsamples = 18\nvalue = [2, 16
 Text(0.5238095238095238, 0.3125, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.6190476190476191, 0.3125, 'X[0] <= 58.5\ngini = 0.111\nsamples = 17\nvalue = [1, 16
 Text(0.5714285714285714, 0.1875, 'gini = 0.0\nsamples = 14\nvalue = [0, 14]'),
 Text(0.6666666666666666, 0.1875, 'X[0] <= 63.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'
 Text(0.6190476190476191, 0.0625, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7142857142857143, 0.0625, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.6666666666666666, 0.4375, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.7142857142857143, 0.5625, 'gini = 0.0\nsamples = 79\nvalue = [0, 79]'),
 Text(0.8571428571428571, 0.6875, 'X[3] <= 2.786\ngini = 0.463\nsamples = 11\nvalue = [7, 4
 Text(0.8095238095238095, 0.5625, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.9047619047619048, 0.5625, 'X[5] <= 73.235\ngini = 0.444\nsamples = 6\nvalue = [2, 4
 Text(0.8571428571428571, 0.4375, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.9523809523809523, 0.4375, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]')]
```

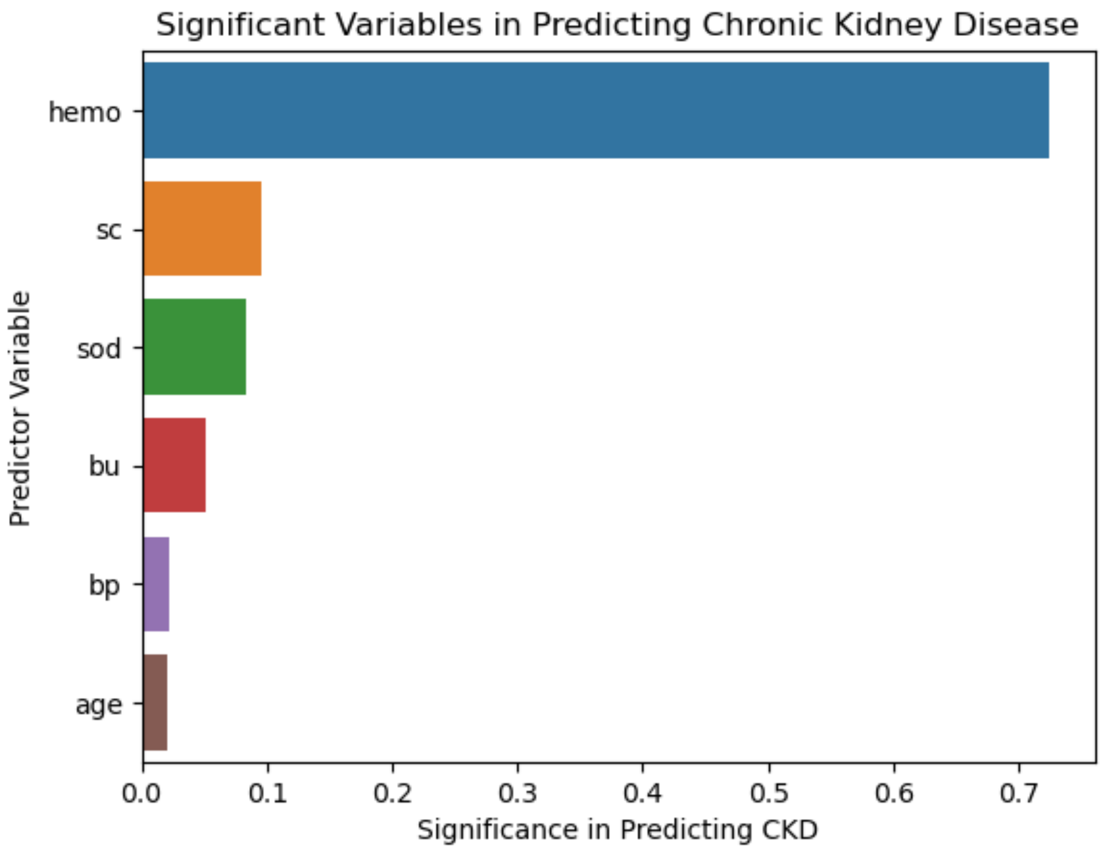


```python
dt_best = DecisionTreeClassifier(
    random_state=0
    )
dt_best.fit(x, y)
```

```
DecisionTreeClassifier(random_state=0)
```

```
feature_importances = dt_best.feature_importances_
sorted_indices = feature_importances.argsort()[::-1]
sorted_feature_names = X_train.columns[sorted_indices]
sorted_importances = feature_importances[sorted_indices]
```

```
sns.barplot(x = sorted_importances, y = sorted_feature_names)
plt.ylabel("Predictor Variable")
plt.xlabel("Significance in Predicting CKD")
plt.title("Significant Variables in Predicting Chronic Kidney Disease")
plt.show()
```



Out of all the predictor variables, the most important/significant one in predicting chronic kidney disease is hemoglobin. We also saw earlier, how strongly this variable is associated with the other variables of the original dataset.