STATS 3DA3

Homework Assignment 6

Lily Seebach (400331131)

2024-04-18

1. A classification problem based on the dataset by Rubini, Soundarapandian, and Eswaran (2015) is to classify an individual into has or doesn't have chronic kidney disease based on the following predictor variables: age, blood pressure (mm/Hg), specific gravity, albumin, sugar, red blood cells, pus cell, pus cell clumps, bacteria, blood glucose random (mgs/dl), blood urea (mgs/dl), serum creatinine (mgs/dl), sodium (mEq/L), potassium (mEq/L), hemoglobin (gms), packed cell volume, white blood cell count (cells/cmm), red blood cell count (millions/cmm), hypertension, diabetes mellitus, coronary artery disease, appetite, pedal edema, and anemia.

2. We import the data and transform it as necessary.

```python
import pandas as pd
import numpy as np
from ucimlrepo import fetch_ucirepo
import certifi
chronic_kidney_disease = fetch_ucirepo(id=336)
X = chronic_kidney_disease.data.features
y = chronic_kidney_disease.data.targets
data = pd.concat([X, y], axis = 1)
```

To check if any variable transformations are necessary, we check that the data types of the variables in the dataframe match the description in the data dictionary.

```python
print(data.dtypes)
```

```
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        float64
wbcc       float64
rbcc       float64
htn         object
dm          object
cad         object
appet       object
pe          object
ane         object
class       object
dtype: object
```

We can see that age, blood pressure (bp), blood glucose random (bgr), blood urea (bu), serum creatinine (sc), sodium (sod), potassium (pot), hemoglobin (hemo), packed cell volume (pcv), white blood cell count (wbcc), and red blood cell count (rbcc) are all numerical values as they should be according to the data dictionary (note the data dictionary differentiates between integer and continuous values but changing numeric data types will not affect results). However, specific gravity (sg), albumin (al), and sugar (su) should be nominal but are float values. Red blood cells (rbc), pus cell (pc), pus cell clumps (pcc), bacteria (ba), hypertension (htn), diabetes mellitus (dm), coronary artery disease (cad), appette (appet), pedal edema (pe), anemia (ane), and class should all be nominal instead of object as they appear now. We correct the data type errors by converting the noted variables to categorical/nominal.

```
columns_to_convert = ['sg','al','su','rbc','pc','pcc','ba','htn','dm','cad','appet','pe','ane'
for col in columns_to_convert:
    data[col] = pd.Categorical(data[col])
```

In order to make the variables match the abbreviations found in the data dictionary, we rename wbcc and rbcc to wc and rc, respectively.

```
data = data.rename(columns={'wbcc':'wc'})
data = data.rename(columns={'rbcc':'rc'})
```

3. We explore the data and give a detailed description of the dataset.

```
print(data.shape)
print(data.iloc[0:2,0:12])
print(data.iloc[0:2,12:26])
```

```
(400, 25)
    age    bp    sg   al   su  rbc      pc         pcc          ba    bgr  \
0  48.0  80.0  1.02  1.0  0.0  NaN  normal  notpresent  notpresent  121.0
1   7.0  50.0  1.02  4.0  0.0  NaN  normal  notpresent  notpresent    NaN

     bu   sc
0  36.0  1.2
1  18.0  0.8
   sod  pot  hemo   pcv      wc   rc  htn   dm cad appet  pe ane class
0  NaN  NaN  15.4  44.0  7800.0  5.2  yes  yes  no  good  no  no   ckd
1  NaN  NaN  11.3  38.0  6000.0  NaN   no   no  no  good  no  no   ckd
```

There are 25 variables and 400 observations. The variables include an individual's age in years, their blood pressue in mm/Hg, their specific gravity, their albumin level, their sugar level, the normality of their red blood cells, the normality of their pus cells, their presence of pus cell clumps, their presence of bacteria, their blood glucose random level in mgs/dl, their blood urea level in mgs/dl, their serum creatinine level in mgs/dl, their sodium level in mEq/L, their potassium level in mEq/L, their hemoglobinl level in gms, their packed cell volume, their white blood cell count in cells/cumm, their red blood cell count in millions/cmm, if they have hypertension, if they have diabetes mellitus, if they have coronary artery disease, if they have a good or poor appetite, if they have pedal edema, if they have anemia, and if they have chronic kidney diease.

Next we create data summaries, including observation proportions.

```python
for col in columns_to_convert:
    print(data[col].value_counts(normalize=True))
data.describe(include = 'all')
```

```
sg
1.020    0.300283
1.010    0.237960
1.025    0.229462
1.015    0.212465
1.005    0.019830
Name: proportion, dtype: float64
al
0.0    0.562147
1.0    0.124294
2.0    0.121469
3.0    0.121469
4.0    0.067797
5.0    0.002825
Name: proportion, dtype: float64
su
0.0    0.826211
2.0    0.051282
3.0    0.039886
1.0    0.037037
4.0    0.037037
5.0    0.008547
Name: proportion, dtype: float64
rbc
```

```
normal      0.810484
abnormal    0.189516
Name: proportion, dtype: float64
pc
normal      0.773134
abnormal    0.226866
Name: proportion, dtype: float64
pcc
notpresent    0.893939
present       0.106061
Name: proportion, dtype: float64
ba
notpresent    0.944444
present       0.055556
Name: proportion, dtype: float64
htn
no     0.630653
yes    0.369347
Name: proportion, dtype: float64
dm
no      0.653266
yes     0.344221
\tno    0.002513
Name: proportion, dtype: float64
cad
no     0.914573
yes    0.085427
Name: proportion, dtype: float64
appet
good    0.794486
poor    0.205514
Name: proportion, dtype: float64
```

```
pe
no     0.809524
yes    0.190476
Name: proportion, dtype: float64
ane
no     0.849624
yes    0.150376
Name: proportion, dtype: float64
class
ckd       0.620
notckd    0.375
ckd\t     0.005
Name: proportion, dtype: float64
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	cla
count	391.000000	388.000000	353.00	354.0	351.0	248	335	396	396	356.0000											
unique	NaN	NaN	5.00	6.0	6.0	2	2	2	2	NaN											
top	NaN	NaN	1.02	0.0	0.0	normal	normal	notpresent	notpresent	NaN											
freq	NaN	NaN	106.00	199.0	290.0	201	259	354	374	NaN											
mean	51.483376	76.469072	NaN	NaN	NaN	NaN	NaN	NaN	NaN	148.0365											
std	17.169714	13.683637	NaN	NaN	NaN	NaN	NaN	NaN	NaN	79.2817											
min	2.000000	50.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.00000											
25%	42.000000	70.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	99.00000											
50%	55.000000	80.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	121.0000											
75%	64.500000	80.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	163.0000											
max	90.000000	180.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	490.0000											

We can see that all variables, except chronic kidney disease, have missing values since no counts equal 400. We note that dm and class have 3 levels instead of their expected two. From the proportion counts of categorical variables, we can see that all categories are represented however some are severely under-represented. After correcting dm and ckd, we explore these distributions more through histograms and bar plots.

```
data = data.replace('\tno', 'no')
data = data.replace('ckd\t', 'ckd')
```

```
C:\Users\seeba\AppData\Local\Temp\ipykernel_4472\2882268402.py:1: FutureWarning: The behavior
  data = data.replace('\tno', 'no')
C:\Users\seeba\AppData\Local\Temp\ipykernel_4472\2882268402.py:2: FutureWarning: The behavior
  data = data.replace('ckd\t', 'ckd')
```

Next we re-examine the data types.

```
data.dtypes
```

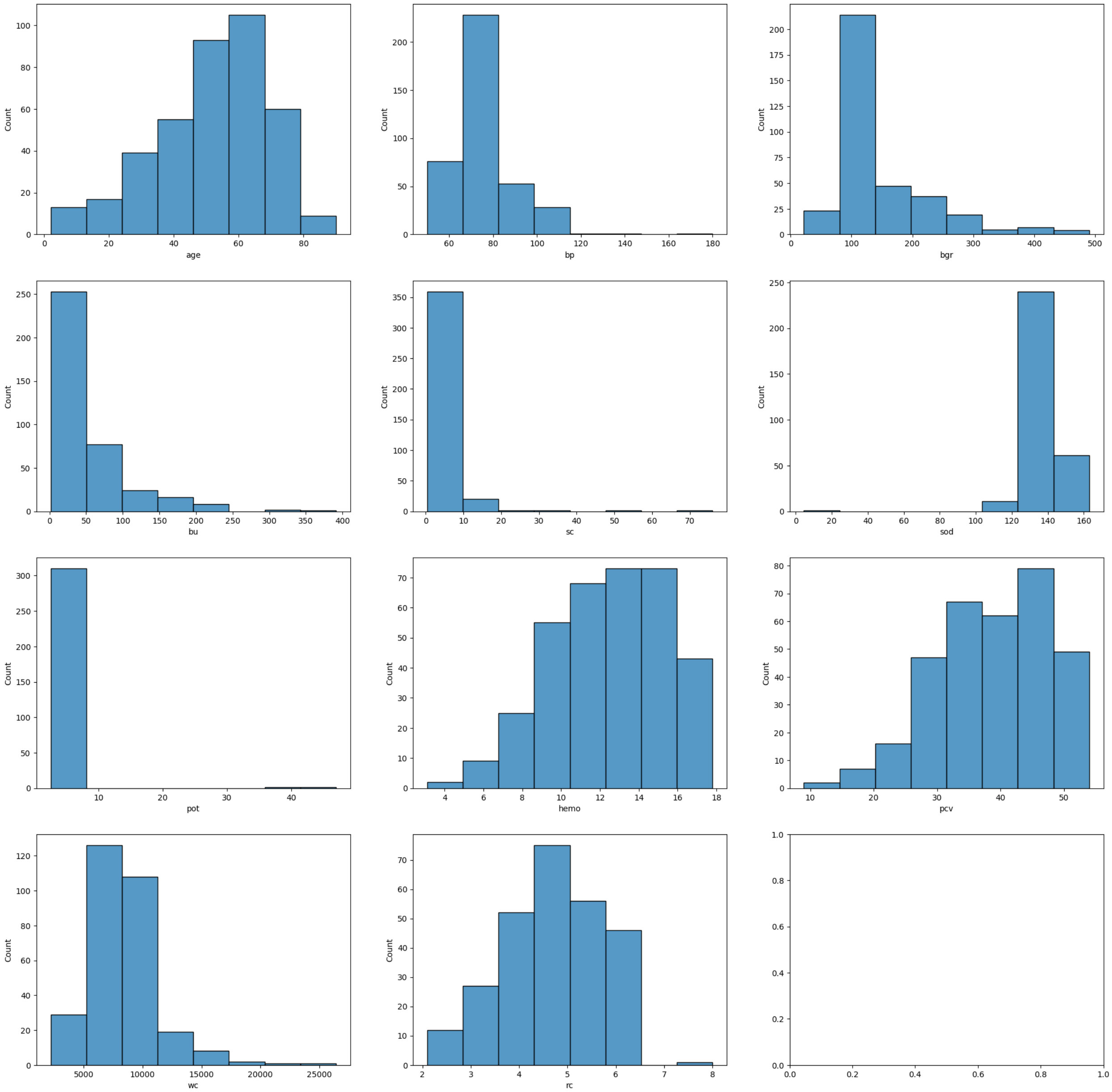
```
age        float64
bp         float64
sg        category
al        category
su        category
rbc       category
pc        category
pcc       category
ba        category
bgr        float64
bu         float64
sc         float64
sod        float64
pot        float64
hemo       float64
pcv        float64
wc         float64
rc         float64
htn       category
dm        category
```

```
cad      category
appet    category
pe       category
ane      category
class    category
dtype: object
```

The data types are only category (nominal) or numerical, as we previously transformed them to be such.

Finally, we look at variables' distributions, starting with numeric variables.

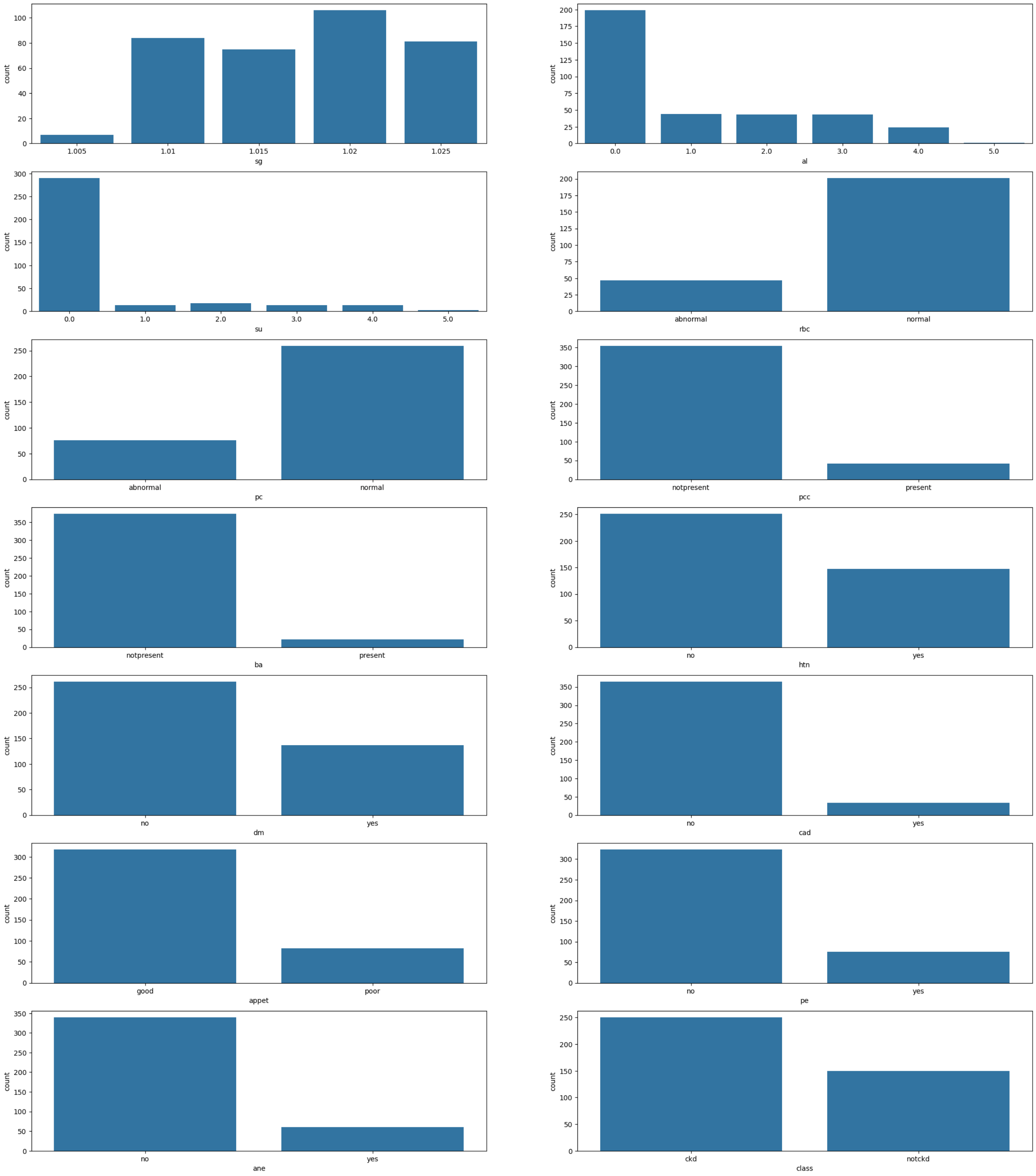
```
import seaborn as sns
import matplotlib.pyplot as plt
figure, axes = plt.subplots(4,3, sharex= False, figsize=(24,24))
k = 0
numeric_vars =  data.select_dtypes(include='float64').columns
for i in range(3):
    for j in range(3):
        sns.histplot(ax = axes[i,j], data=data, x = str(numeric_vars[k]), binwidth=(data[str(nu
        k = k+1
sns.histplot(ax = axes[3,0], data=data, x = str(numeric_vars[9]), binwidth=(data[str(numeric_va
sns.histplot(ax = axes[3,1], data=data, x = str(numeric_vars[10]), binwidth=(data[str(numeric_v
plt.show()
```



Some variables are heavily skewed such as white blood cell count, potassium, serum creatinine, blood urea, blood pressure, and blood glucose random. Conversely, red blood cell count appears relatively normally distribtued.

```
import seaborn as sns
import matplotlib.pyplot as plt
figure, axes = plt.subplots(7,2, sharex= False, figsize=(26,30))
k = 0
cat_vars =  data.select_dtypes(include='category').columns
```

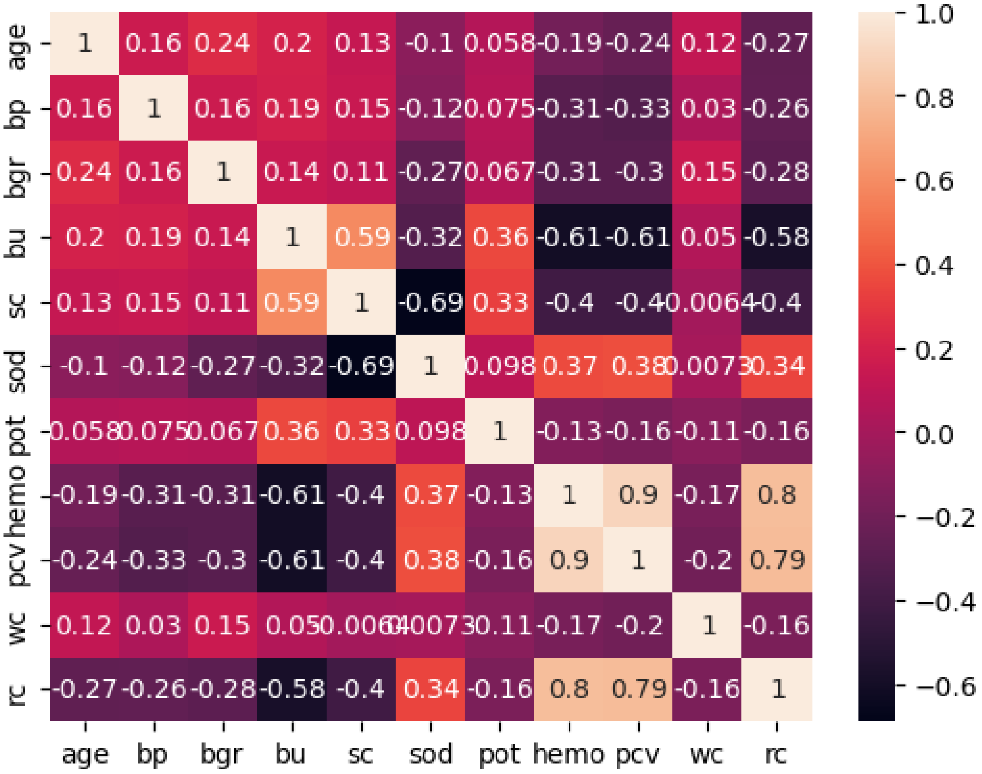
```
for i in range(7):
    for j in range(2):
        sns.countplot(ax = axes[i,j], data=data, x = str(cat_vars[k]))
        k = k+1
plt.show()
```



We notice most categorical variables are heavily skewed. For example, hypertension, diabetes mellitus, coronary artery disease, appetite, pedal edema, and anemia all have many more instances of “no” than “yes”. Red blood cells and pus cell display similar trends where by “normal” is more frequent than “abnormal”. Pus cell clumps and bacteria also display this trend where “present” is far less frequent than “not present”. It is important to note that class (ckd, notckd) does not display this heavy skew. Although “ckd” is more frequent than “notckd”, the two are relatively even.

4. We now analyze variable relationships. We use a heatmap to check for correlation between numerical variables.

```python
num_col = data.select_dtypes(include = 'float64').columns
sns.heatmap(data[num_col].corr(), annot=True)
```



From this heatmap we can see hemoglobin and packed cell volume, hemoglobin and red blood cell count, and packed cell volume and red blood cell count have a high positive correlation (>0.75). Conversely, sodium and serum creatinine, hemoglobin and blood urea, and packed cell volume and blood urea have a high negative correlation (<-0.6). The least correlated variables are white blood cell count and serum creatinine (0.0064). These correlations will likely cause feature selection to drop one of the variables in highly correlated variables.

5. We remove observations with missing values in order to conduct subgroup analysis and for future use.

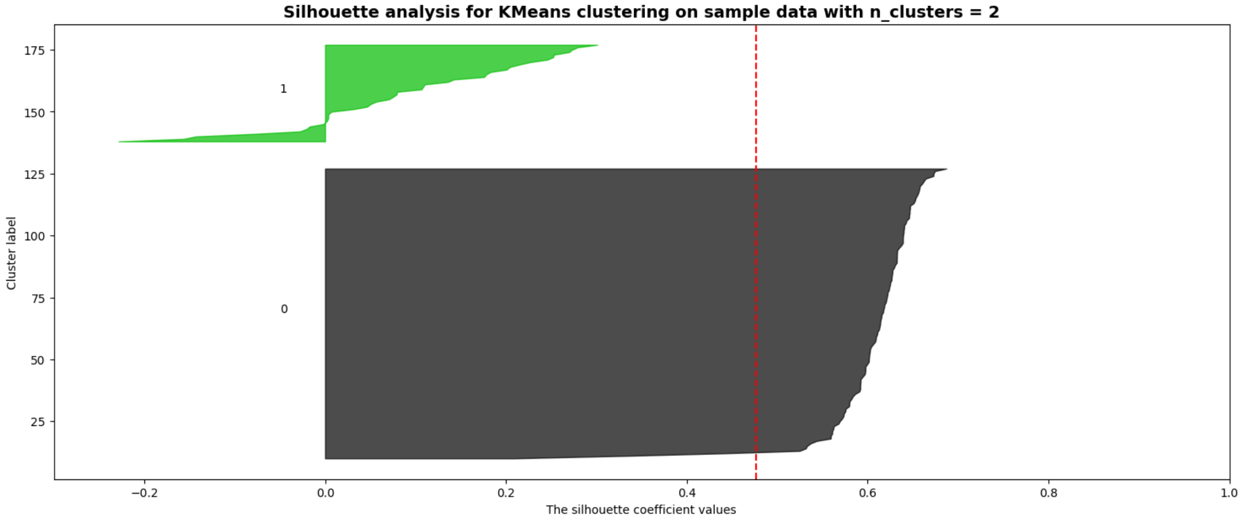
```
data = data.dropna(ignore_index=True) # drop observations with missing values
```

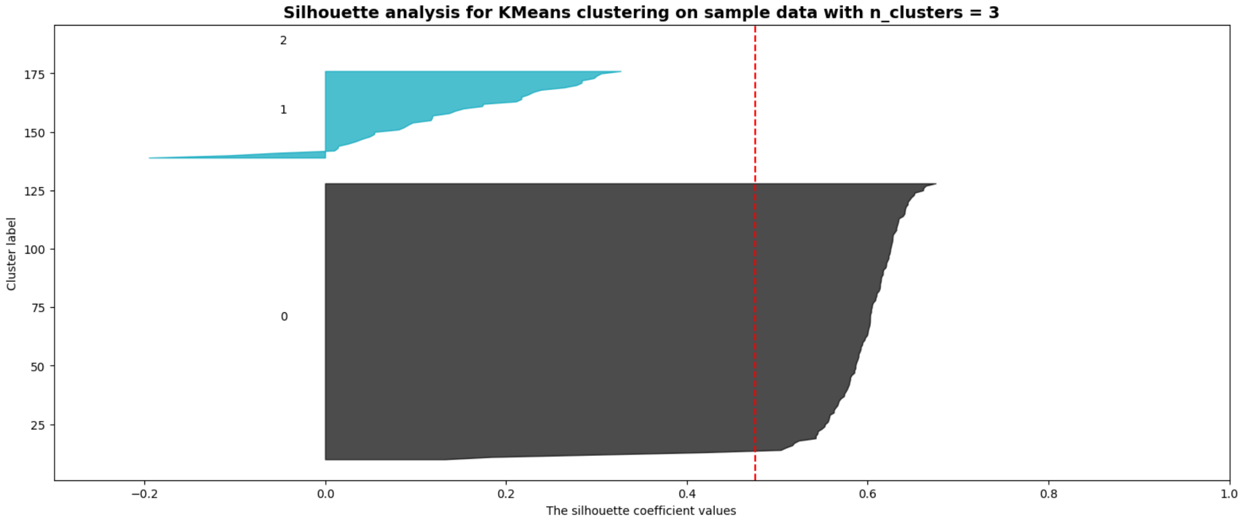
While this greatly reduces the data's sample size, it avoids adding bias by entering other values for missing values (eg. mean, median, or mode), or filling them in another way.

6. We do not conduct outlier analysis. This is because outliers can be very important in diagnosing diseases in health data. By removing outliers, we would miss the opportunity to see how extreme values can help in finding disease risks.

7. We now conduct using K-means clustering on the data, after scaling and creating dummy variables, to conduct sub-group analysis.

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
from sklearn.preprocessing import scale
scaled_data = data.copy()
scaled_data[num_col] = pd.DataFrame(scale(data[num_col]))
dummy = pd.get_dummies(data = scaled_data, columns = cat_vars)
scaled_data = scaled_data.drop(cat_vars, axis = 1)
scaled_data = pd.concat([scaled_data, dummy], axis = 1)
scaled_data = scaled_data.drop(['class_ckd','class_notckd'], axis = 1)
range_n_clusters = [2, 3]
for n_clusters in range_n_clusters:
    km = KMeans(n_clusters = n_clusters, n_init = 20, random_state=0)
    cluster_labels_km = km.fit_predict(scaled_data)
    # average silhouette score
    silhouette_avg_km = silhouette_score(scaled_data, cluster_labels_km)
    # compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(scaled_data, cluster_labels_km)
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.3, 1])# change this based on the silhouette range
    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels_km == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            y=np.arange(y_lower, y_upper),
            x1=0,
            x2=ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )
        # label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        # Compute the new y_lower for next cluster silhouette scores
        y_lower = y_upper + 10
    ax1.set_title("The silhouette plot for various cluster")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
    # vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg_km, color="red", linestyle="--")
    plt.title(
        "Silhouette analysis for KMeans clustering on sample data with n_clusters = %d"
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )
plt.show()
```

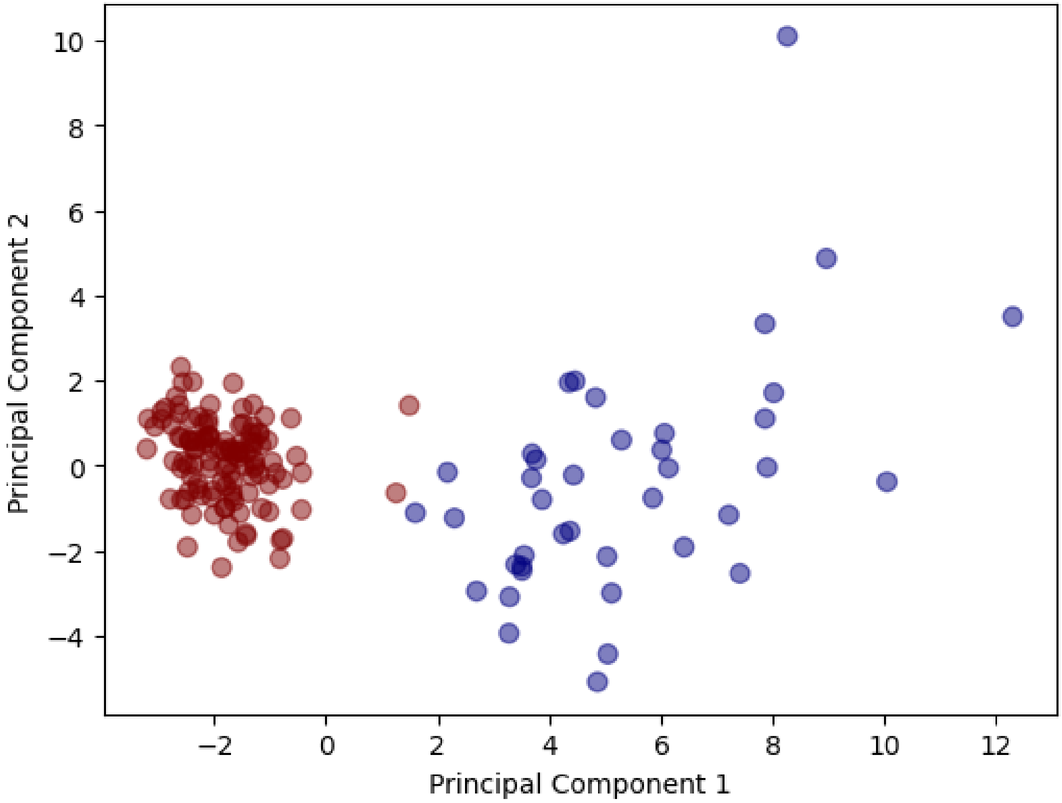




There appears to be two subgroups, we now visualize them using PCA with two PCs.

```
km1 = KMeans(n_clusters=2, n_init=20, random_state=0)
km1.fit(scaled_data)
cluster_labels_km1 = km1.fit_predict(scaled_data)
```

```
from sklearn.decomposition import PCA, TruncatedSVD, FactorAnalysis
pca = PCA()
pc_scores = pd.DataFrame(pca.fit_transform(scaled_data), index=scaled_data.index)
color_idx = pd.factorize(cluster_labels_km1)[0]
cmap = plt.cm.jet
scatter = plt.scatter(pc_scores.iloc[:,0], pc_scores.iloc[:,1], c=color_idx, cmap=cmap, alpha=
plt.ylabel('Principal Component 2')
plt.xlabel('Principal Component 1')
plt.show()
```



From this analysis, we can see there is a clearly defined sub-group in the bottom left corner based on the data.

8. We now split the unscaled, original data into a 70/30 training/test split for testing, using a random seed of 1 and stratified sampling.

```
from sklearn.model_selection import train_test_split
y = data['class']
x = data.drop('class', axis = 1)
X_train, X_test, y_train, y_test = train_test_split(
x, y, test_size=0.3, random_state=1, stratify=y)
```

9. We choose two classifiers to address the classification problem. First we use k-nearest neighbours (KNN) then we use a decision tree. We choose these classifiers as they are supervised learning methods and decision trees are interpretable. Also, classic k-nearest neighbours is used for numerical variables only while decision trees can handle mixed variable types, allowing us to see how including categorical variables can affect performance.

10. We will use accuracy and sensitivity to compare the performance of the classifiers. Accuracy compares the overall performance while sensitivity will compare how good the classifier is at correctly identifying an individual with chronic kidney disease.

11. We use lasso regression to determine important features for KNN while decision trees automatically perform feature selection. We use the previous data which was scaled and uses dummy variables and select the corresponding testing and training values.

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet, RidgeCV, LassoCV, ElasticNetCV
x_train_num = scaled_data.iloc[X_train.index]
y_train_num = y_train.cat.codes
x_test_num = scaled_data.iloc[X_test.index]
lasso_cv = LassoCV(alphas=np.logspace(-4, 4, 100), cv=5, max_iter=1500)
lasso_cv.fit(x_train_num, y_train_num)
m_lasso = Lasso(alpha=lasso_cv.alpha_)
m_lasso.fit(x_train_num, y_train_num)
m_lasso_pre = m_lasso.predict(x_test_num)
pd.DataFrame({'Feature': x_train_num.columns, 'Coefficient': m_lasso.coef_.reshape(len(x_train
```

	Feature	Coefficient
0	age	-0.000000

	Feature	Coefficient
1	bp	-0.000000
2	bgr	-0.000000
3	bu	-0.000000
4	sc	-0.000000
5	sod	0.000000
6	pot	-0.000000
7	hemo	0.000000
8	pcv	0.000000
9	wc	-0.000000
10	rc	0.000000
11	age	-0.000000
12	bp	-0.000000
13	bgr	-0.000092
14	bu	-0.000000
15	sc	-0.000000
16	sod	0.000101
17	pot	-0.000000
18	hemo	0.000135
19	pcv	0.000262
20	wc	-0.000062
21	rc	0.000000
22	sg_1.005	0.000000
23	sg_1.01	-0.000000
24	sg_1.015	-0.000000
25	sg_1.02	0.000000
26	sg_1.025	0.000000
27	al_0.0	0.998354
28	al_1.0	-0.000000
29	al_2.0	-0.000000
30	al_3.0	-0.000000

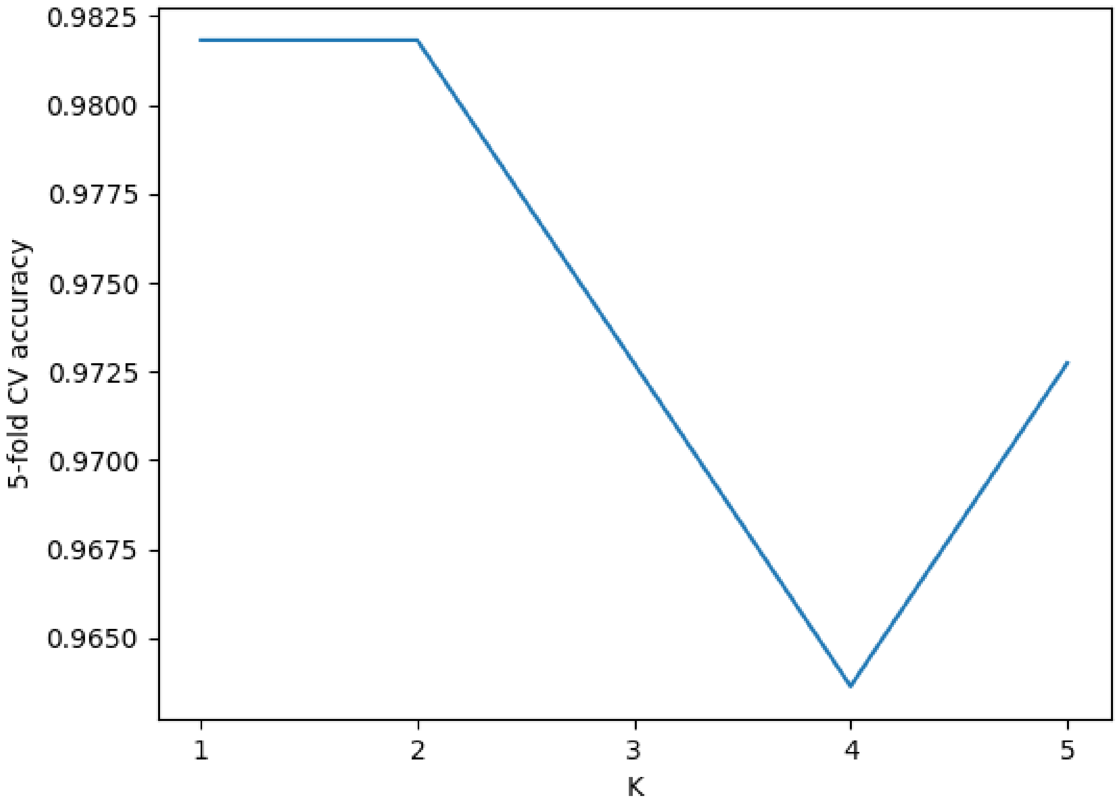
	Feature	Coefficient
31	al_4.0	-0.000000
32	al_5.0	0.000000
33	su_0.0	0.000000
34	su_1.0	-0.000000
35	su_2.0	-0.000000
36	su_3.0	-0.000000
37	su_4.0	-0.000000
38	su_5.0	-0.000000
39	rbc_abnormal	-0.000000
40	rbc_normal	0.000000
41	pc_abnormal	-0.000000
42	pc_normal	0.000000
43	pcc_notpresent	0.000000
44	pcc_present	-0.000000
45	ba_notpresent	0.000000
46	ba_present	-0.000000
47	htn_no	0.000000
48	htn_yes	-0.000000
49	dm_no	0.000000
50	dm_yes	-0.000000
51	cad_no	0.000000
52	cad_yes	-0.000000
53	appet_good	0.000000
54	appet_poor	-0.000000
55	pe_no	0.000000
56	pe_yes	-0.000000
57	ane_no	0.000000
58	ane_yes	-0.000000

Using lasso regression, we see that many variables have been given a coefficient of zero. Specifically all coefficients have been shrunk to zero except blood glucose random, sodium, hemoglobin, packed cell volume, white blood cell count, and albumin. Note that albumin received by far the largest coefficient.

12. We now implement 5-fold cross validation for KNN with the scaled numerical variables not given a coefficient of 0 in the lasso regression process, and decision tree with all the variables. First we petform the cross-validation using the test set to select the best value of k.

```
## KNN Implementation
from sklearn.model_selection import cross_val_score
X_train_red = X_train.drop(['age', 'bp', 'sg','al','su','rbc','pc','pcc','ba','bu','sc','pot',
X_test_red =  X_test.drop(['age', 'bp', 'sg','al','su','rbc','pc','pcc','ba','bu','sc','pot','
X_test_red = scale(X_test_red)
X_train_red = scale(X_train_red)

from sklearn import neighbors
from sklearn import metrics
k_range = range(1, 6)
cv_scores = []
for k in k_range:
    knn_cv = neighbors.KNeighborsClassifier(n_neighbors=k)
    cv_scores_k = cross_val_score(knn_cv, X_train_red, y_train, cv=5)
    cv_scores.append(np.mean(cv_scores_k))
plt.plot(k_range, cv_scores)
plt.xlabel('K')
plt.ylabel('5-fold CV accuracy')
plt.xticks(range(1,6))
plt.show()
```



Selecting K = 2 to maximize accuracy and minimize variance:

```
knn = neighbors.KNeighborsClassifier(n_neighbors = 2)
knn.fit(X_train_red,y_train)
pred = knn.predict(X_test_red)
from sklearn.metrics import mean_squared_error, confusion_matrix, classification_report
cm = pd.DataFrame(confusion_matrix(y_test, pred), index=['No CKD', 'CKD'], columns=['No CKD',
sensitivity = cm.iloc[1,1]/(cm.iloc[1,0]+cm.iloc[1,1])
print('Sensitivity : ', sensitivity)
print('Accuracy : ', round(metrics.accuracy_score(y_test, pred),2))
```

```
Sensitivity :  0.9714285714285714
Accuracy :  0.98
```

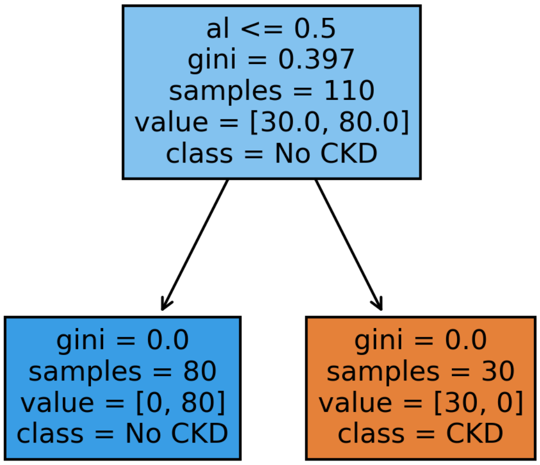
Now using a decision tree using the original data with dummy variables for categorical variables (all variables, unscaled).

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
for col in cat_vars.drop('class'):
    X_train[col] = pd.Categorical(X_train[col]).codes
    X_test[col] = pd.Categorical(X_test[col]).codes
cs_dt = DecisionTreeClassifier(
    max_depth = 15,
    random_state=1
)
cs_dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=15, random_state=1)
```

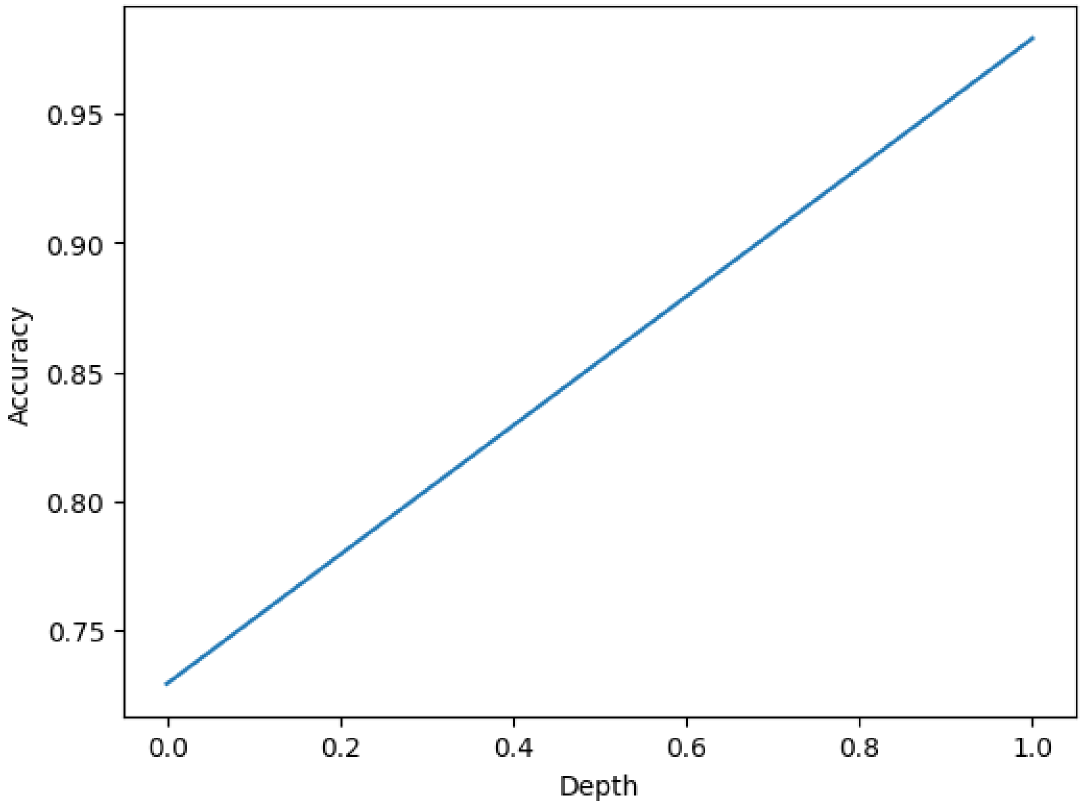
```
## Decision Tree
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
cs_dt = DecisionTreeClassifier(
    max_depth = 15,
    random_state=1
    )
cs_dt.fit(X_train, y_train)
fig, axes = plt.subplots(
    nrows = 1,ncols = 1,figsize = (4,4), dpi=300
    )
plot_tree(
    cs_dt,
    max_depth= 5,
    feature_names = X_train.columns.tolist(),
    class_names=['CKD', 'No CKD'],
    filled = True
    )
```

```
[Text(0.5, 0.75, 'al <= 0.5\ngini = 0.397\nsamples = 110\nvalue = [30.0, 80.0]\nclass = No CKD
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 80\nvalue = [0, 80]\nclass = No CKD'),
Text(0.75, 0.25, 'gini = 0.0\nsamples = 30\nvalue = [30, 0]\nclass = CKD')]
```



Using a pruning tree to maximize accuaracy:

```
path = cs_dt.cost_complexity_pruning_path(
    x_train_num,
    y_train
)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
clfs = [] # save fitted trees with different alphas
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(
        ccp_alpha=ccp_alpha
        )
    clf.fit(x_train_num, y_train)
    clfs.append(clf)
depth = [clf.tree_.max_depth for clf in clfs]
test_score = [clf.score(x_test_num, y_test) for clf in clfs]
plt.plot(depth, test_score)
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.show()
```



Here, it appears the variable albumin is such a prominent variable in detecting chronic kidney disease that no other variables are considered in the process.

```
cs_dt = DecisionTreeClassifier(
    max_depth = 1,
    random_state=1
)
cs_dt.fit(X_train, y_train)
pred = cs_dt.predict(X_test)
cm = pd.DataFrame(confusion_matrix(y_test, pred), index=['No CKD', 'CKD'], columns=['No CKD',
cm.index.name = 'True'
cm.columns.name = 'Predicted'
sensitivity = cm.iloc[1,1]/(cm.iloc[1,0]+cm.iloc[1,1])
```

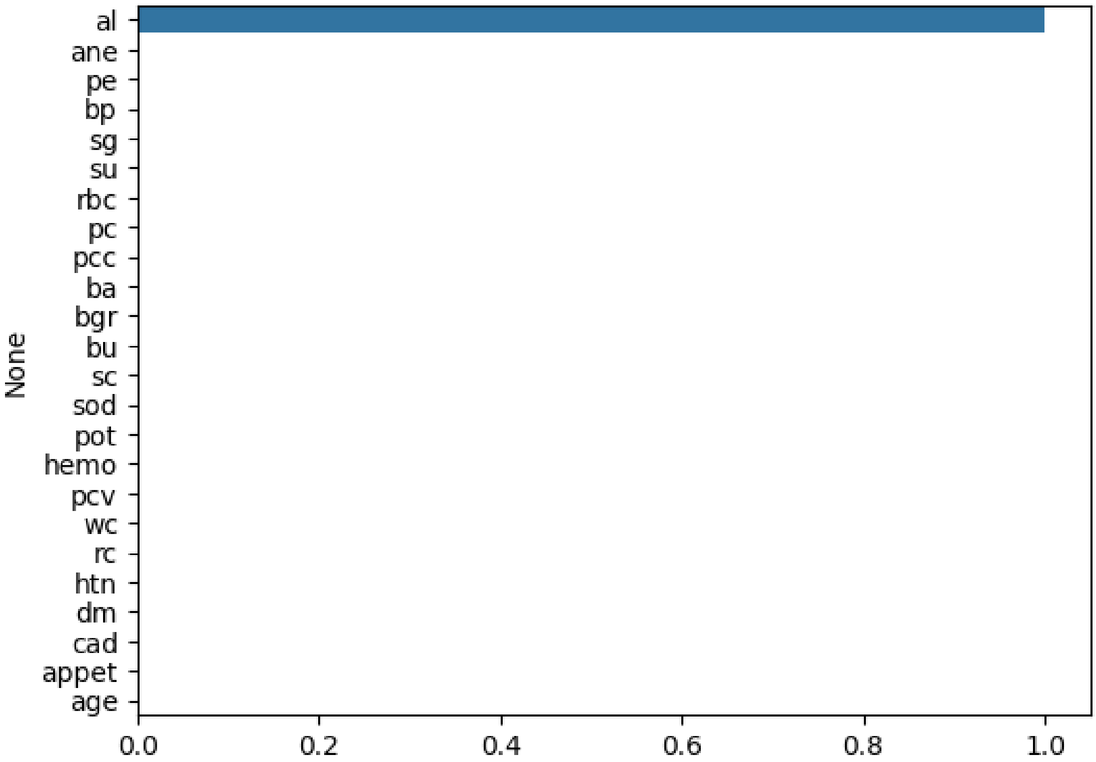
```
print('Sensitivity : ', sensitivity)
print('Accuracy : ', cs_dt.score(X_test, y_test))
```

```
Sensitivity :  1.0
Accuracy :  0.9791666666666666
```

We can see KNN using scaled variables blood glucose random, sodium, hemoglobin, packed cell volume, and white blood cell count gives an accuracy of 0.98 and a sensitivity of 0.97 in the classification of chronic kidney disease and no kidney disease. The decision tree, which had access to all variables, unscaled, only used albumin and had an accuracy of 0.98 and a sensitivity of 1, with one case of no chronic kidney disease being labelled a chronic kidney disease. Overall, the decision tree performed better despite only using one variable.

13. We re-train the interpretable classifier (decision tree) using all the data and analyze and interpret the significance of the predictor variables.

```
## Decision Tree
for col in cat_vars.drop('class'):
    x[str(col)] = pd.Categorical(x[str(col)]).codes
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
cs_dt_best = DecisionTreeClassifier(
    max_depth = 1,
    random_state=0
    )
cs_dt_best.fit(x, y)
fea_imp = cs_dt_best.feature_importances_
sorted_indices = fea_imp.argsort()[::-1]
sorted_feature_names = X_train.columns[sorted_indices]
sorted_importances = fea_imp[sorted_indices]
sns.barplot(x = sorted_importances, y = sorted_feature_names)
plt.show()
```



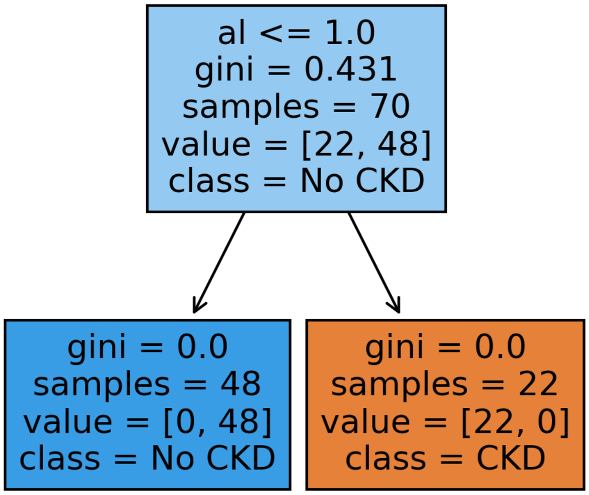
From this plot, we can see albumin is the only important variable in determining an individual's risk of chronic kidney disease according to the decision tree. Specifically, if albumin is 0, the patient is very unlikely to have chronic kidney disease, otherwise they likely do have ckd. This could be due to the sample size and/or test train split. Using random forests or dropping albumin may reveal other important variables.

14. Now split the sub-groups identified in question 7 to improve the decision tree accuracy.

```
xtrain1 = []
xtrain2 = []
ytrain1 = []
ytrain2 = []
for i in range(len(X_train)):
    if cluster_labels_km1[i] == 0:
        xtrain1.append(X_train.iloc[i])
        ytrain1.append(y_train.iloc[i])
    else:
        xtrain2.append(X_train.iloc[i])
        ytrain2.append(y_train.iloc[i])
xtest1 = []
xtest2 = []
ytest1 = []
ytest2 = []
for i in range(len(X_test)):
    if cluster_labels_km1[i] == 0:
        xtest1.append(X_test.iloc[i])
        ytest1.append(y_test.iloc[i])
    else:
        xtest2.append(X_test.iloc[i])
        ytest2.append(y_test.iloc[i])
```

```
## Using decsion tree
x1_dt = DecisionTreeClassifier(
    max_depth = 1,
    random_state=1
)
x1_dt.fit(xtrain1,ytrain1)
fig, axes = plt.subplots(
    nrows = 1,ncols = 1,figsize = (4,4), dpi=300
    )
plot_tree(
    x1_dt,
    max_depth= 1,
    feature_names = X_train.columns.tolist(),
    class_names=['CKD', 'No CKD'],
    filled = True
    )
```

```
[Text(0.5, 0.75, 'al <= 1.0\ngini = 0.431\nsamples = 70\nvalue = [22, 48]\nclass = No CKD'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 48\nvalue = [0, 48]\nclass = No CKD'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 22\nvalue = [22, 0]\nclass = CKD')]
```



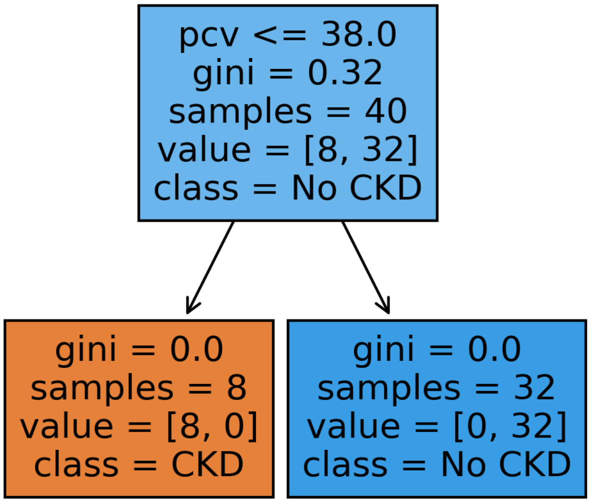
```
x1_dt.fit(xtrain1, ytrain1)
pred = x1_dt.predict(xtest1)
cm = pd.DataFrame(confusion_matrix(ytest1, pred), index=['No CKD', 'CKD'], columns=['No CKD',
cm.index.name = 'True'
cm.columns.name = 'Predicted'
sensitivity = cm.iloc[1,1]/(cm.iloc[1,0]+cm.iloc[1,1])
print('Sensitivity : ', sensitivity)
print('Accuracy : ', x1_dt.score(xtest1, ytest1))
```

```
Sensitivity :  1.0
Accuracy :  1.0
```

```
## Using decsion tree
x2_dt = DecisionTreeClassifier(
    max_depth = 1,
    random_state=1
)
```

```
x2_dt.fit(xtrain2,ytrain2)
fig, axes = plt.subplots(
    nrows = 1,ncols = 1,figsize = (4,4), dpi=300
    )
plot_tree(
    x2_dt,
    max_depth= 1,
    feature_names = X_train.columns.tolist(),
    class_names=['CKD', 'No CKD'],
    filled = True
    )
```

```
[Text(0.5, 0.75, 'pcv <= 38.0\ngini = 0.32\nsamples = 40\nvalue = [8, 32]\nclass = No CKD'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]\nclass = CKD'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 32\nvalue = [0, 32]\nclass = No CKD')]
```



```
x2_dt.fit(xtrain2, ytrain2)
pred = x2_dt.predict(xtest2)
cm = pd.DataFrame(confusion_matrix(ytest2, pred), index=['No CKD', 'CKD'], columns=['No CKD',
cm.index.name = 'True'
cm.columns.name = 'Predicted'
sensitivity = cm.iloc[1,1]/(cm.iloc[1,0]+cm.iloc[1,1])
print('Sensitivity : ', sensitivity)
print('Accuracy : ', x2_dt.score(xtest2, ytest2))
```

```
Sensitivity :  1.0
Accuracy :  1.0
```

The accuracy and sensitivity for the first and second subgroup is 1. In both question 12 (without sub-group considerations), the accuracy is 0.98 and the sensitivity is 1. Therefore, the accuracy is slightly improved after first splitting the data based on K-means clustering, with K=2.

14. Entirely individual submission.
15. Here is the link to the public GitHub repository https://github.com/seebachl/3DA3-Asgmt-6.

Bibliography

Rubini, L., Soundarapandian P., and P. Eswaran. 2015. “Chronic Kidney Disease.” UCI Machine Learning Repository.