STATS 3DA3

Homework Assignment 6

Chengdai Xu(400397089), Tianmu Li(400371751), Jiajun Zhang (400359213)

2024-04-04

Github URL: https://github.com/Xxxxcd/STATS3DA3.git

```
from ucimlrepo import fetch_ucirepo

import matplotlib.pyplot as plt

import matplotlib.cm as cm

import pandas as pd

import numpy as np

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.datasets import make_circles

from sklearn.impute import SimpleImputer

from sklearn.metrics import mean_squared_error, confusion_matrix, classification_report

from sklearn.metrics import accuracy_score, classification_report

from sklearn.metrics import silhouette_samples, silhouette_score

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import scale

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree

from sklearn.tree import plot_tree



# fetch dataset

chronic_kidney_disease = fetch_ucirepo(id=336)

# data (as pandas dataframes)

X = chronic_kidney_disease.data.features

y = chronic_kidney_disease.data.targets




for i in range(len(y)):
    if y.loc[i, "class"] == "ckd\t":
        y.loc[i, "class"] = "ckd"

X["dm"].isna().sum()





for i in range(len(y)):
    if X.loc[i, "dm"] == "\tno":
        X.loc[i, "dm"] = "no"

distinct = set(y["class"])
print(distinct)
```

```
{'notckd', 'ckd'}
```

The dataset has some issues converting it into Pandas dataframe as it contains TAB/ spaces in the data which I correct them in by remove "" which represents tab.

1.

This dataset contains medical information for each record, including whether an individual has chronic kidney disease ("ckd" or "notckd"). We would like to utilize this dataset to predict the presence of chronic kidney disease, In generial, we want to use model for classfication, essentially classifying groups of individuals who may potentially have the condition of chronic kidney disease.

```
X = pd.DataFrame(X)
y = pd.DataFrame(y)
X.head(5)
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbc
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	15.4	44.0	780
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	11.3	38.0	600
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	9.6	31.0	750
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	11.2	32.0	670
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	11.6	35.0	730

2.

We decide to use Decision Tree and Support Vector Machine(SVM) to classify the data. For Decision Tree, it is unnecessary to transform the variables, because they are not sensitive to feature scaling. Doing so will not change the result. However, normalize the data could be useful for SVM, we will standardlize numerical columns after we have an overview of the dataset and use them when we fit SVM model for the data in later codes.

```
print(X.dtypes)
print(y.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
dtype: object
class    object
dtype: object
```

```
X.isna().sum()
```

```
age        9
bp        12
sg        47
al        46
su        49
rbc      152
pc        65
pcc        4
ba         4
bgr       44
bu        19
sc        17
sod       87
pot       88
hemo      52
```

```
pcv       71
wbcc     106
rbcc     131
htn        2
dm         2
cad        2
appet      1
pe         1
ane        1
dtype: int64
```

```
cat = ['rbc', 'pc', 'pcc','ba','htn','dm','cad','appet','pe','ane','al','su']
#al and su are categorical variables, though their type is float64.

for col in cat:
    X[col] = pd.Categorical(X[col]).codes
```

```
#After the previous processing, categorical variables are now switched into numbers for model p

for col in cat:
    X[col] = X[col].replace(-1, np.nan)


X.isna().sum()
```

```
age        9
bp        12
sg        47
al        46
su        49
rbc      152
pc        65
pcc        4
```

```
ba          4
bgr        44
bu         19
sc         17
sod        87
pot        88
hemo       52
pcv        71
wbcc      106
rbcc      131
htn         2
dm          2
cad         2
appet       1
pe          1
ane         1
dtype: int64
```

3. Dataset Overview

```
print(y)
```

```
      class
0       ckd
1       ckd
2       ckd
3       ckd
4       ckd
..      ...
395  notckd
396  notckd
```

```
397    notckd
398    notckd
399    notckd

[400 rows x 1 columns]
```

We manually check the dictionary from the data source webstite. For variable “al” and “su”, although they have a type of “float”, they are actually discreate numbers, so we treat them as categorical datas.

```
categorcal_cols = ['rbc', 'pc', 'pcc','ba','htn','dm','cad','appet','pe','ane','al','su']
numerical_cols = ["age","bp","sg","bgr","bu","sc","sod","pot","hemo","pcv","wbcc","rbcc"]
```

```
print(X.shape)
print(y.shape)
```

```
(400, 24)
(400, 1)
```

```
X[numerical_cols].describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	391.0	51.483376	17.169714	2.000	42.00	55.00	64.50	90.000
bp	388.0	76.469072	13.683637	50.000	70.00	80.00	80.00	180.000
sg	353.0	1.017408	0.005717	1.005	1.01	1.02	1.02	1.025
bgr	356.0	148.036517	79.281714	22.000	99.00	121.00	163.00	490.000
bu	381.0	57.425722	50.503006	1.500	27.00	42.00	66.00	391.000
sc	383.0	3.072454	5.741126	0.400	0.90	1.30	2.80	76.000
sod	313.0	137.528754	10.408752	4.500	135.00	138.00	142.00	163.000
pot	312.0	4.627244	3.193904	2.500	3.80	4.40	4.90	47.000
hemo	348.0	12.526437	2.912587	3.100	10.30	12.65	15.00	17.800

	count	mean	std	min	25%	50%	75%	max
pcv	329.0	38.884498	8.990105	9.000	32.00	40.00	45.00	54.000
wbcc	294.0	8406.122449	2944.474190	2200.000	6500.00	8000.00	9800.00	26400.000
rbcc	269.0	4.707435	1.025323	2.100	3.90	4.80	5.40	8.000

```
X[categorcal_cols]
```

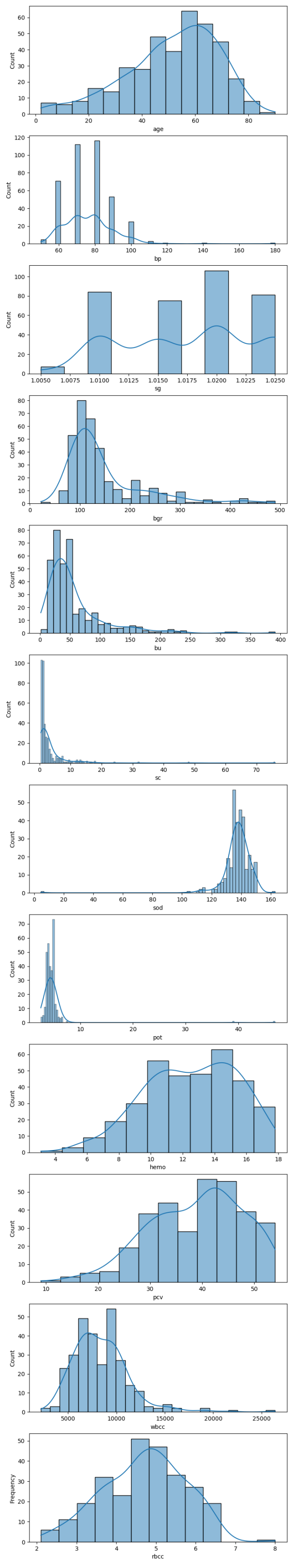
	rbc	pc	pcc	ba	htn	dm	cad	appet	pe	ane	al	su
0	NaN	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
1	NaN	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0
2	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	2.0	3.0
3	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	4.0	0.0
4	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
395	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
396	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
397	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
398	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
399	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
numeric_X = X.select_dtypes(include=['float'])
```

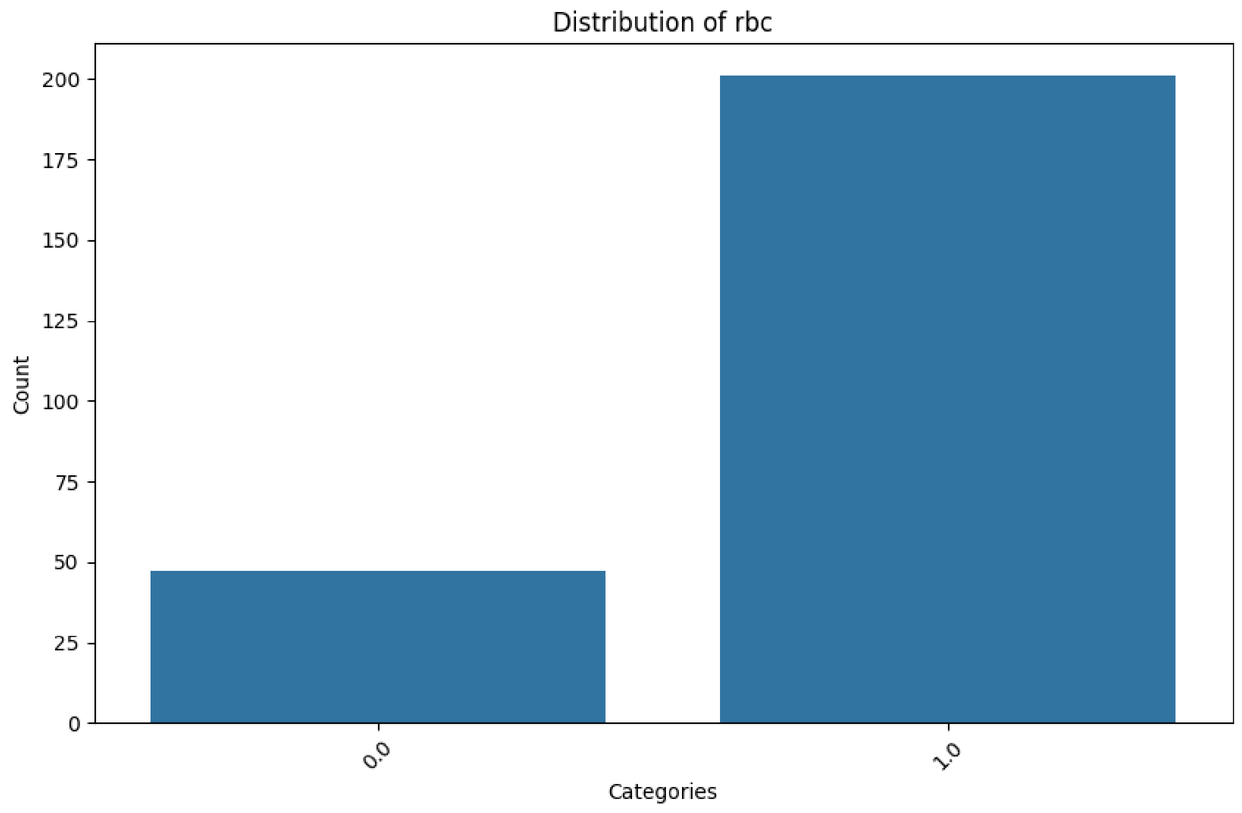
```
def plot_dist(data, features):
    fig, axes = plt.subplots(nrows=len(features), ncols=1, figsize=(8, 4 * len(features)))
    if len(features) == 1:
        axes = [axes]
    for ax, feature in zip(axes, features):
        sns.histplot(data[feature], kde=True, ax=ax)
    ax.set_xlabel(feature)
    ax.set_ylabel('Frequency')
```

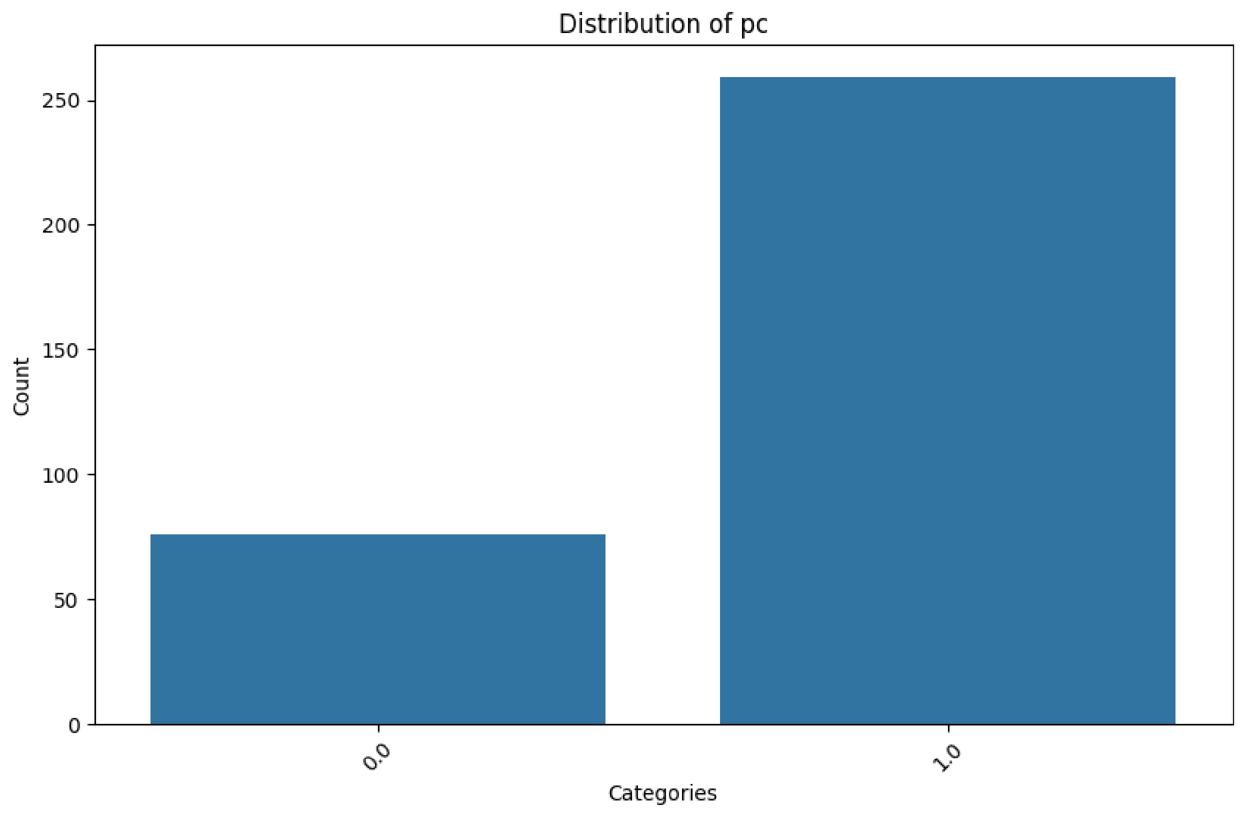
```
plt.show()
```

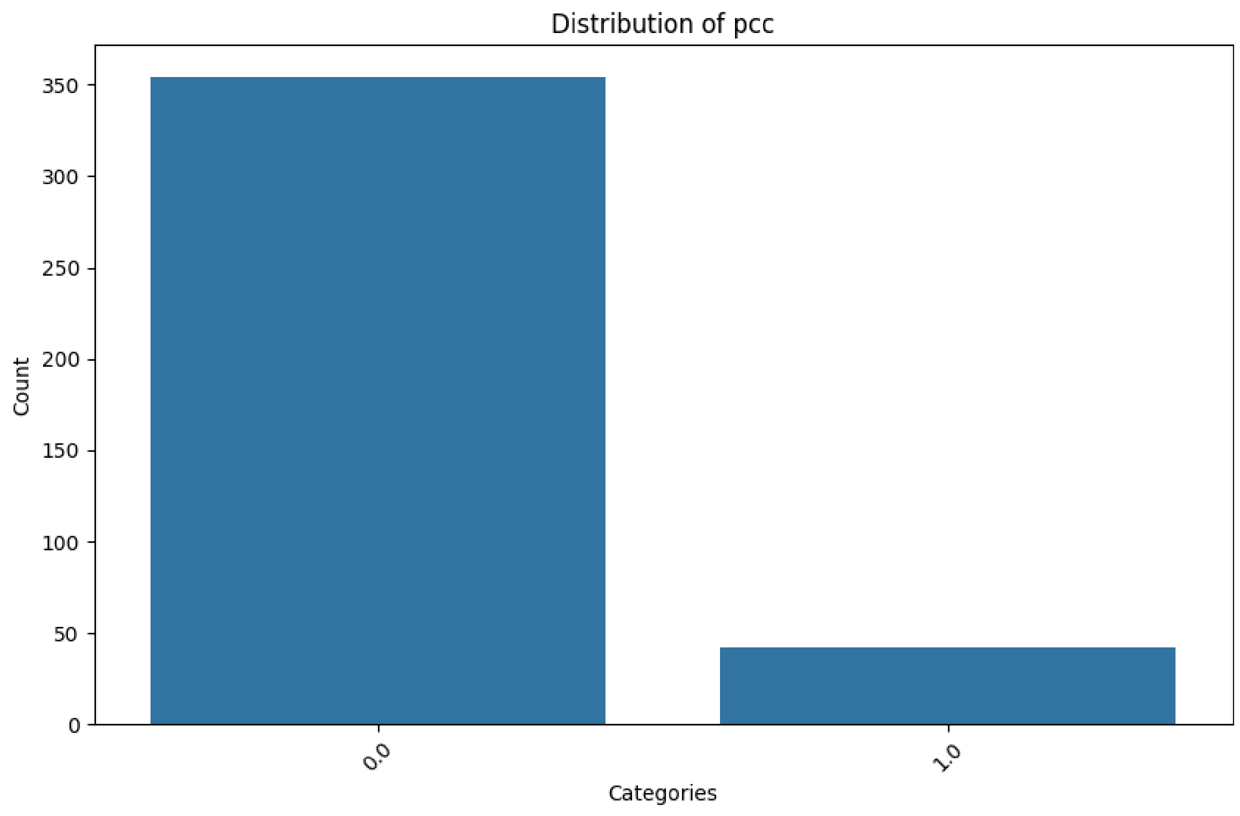
```
plot_dist(X,numerical_cols)
```

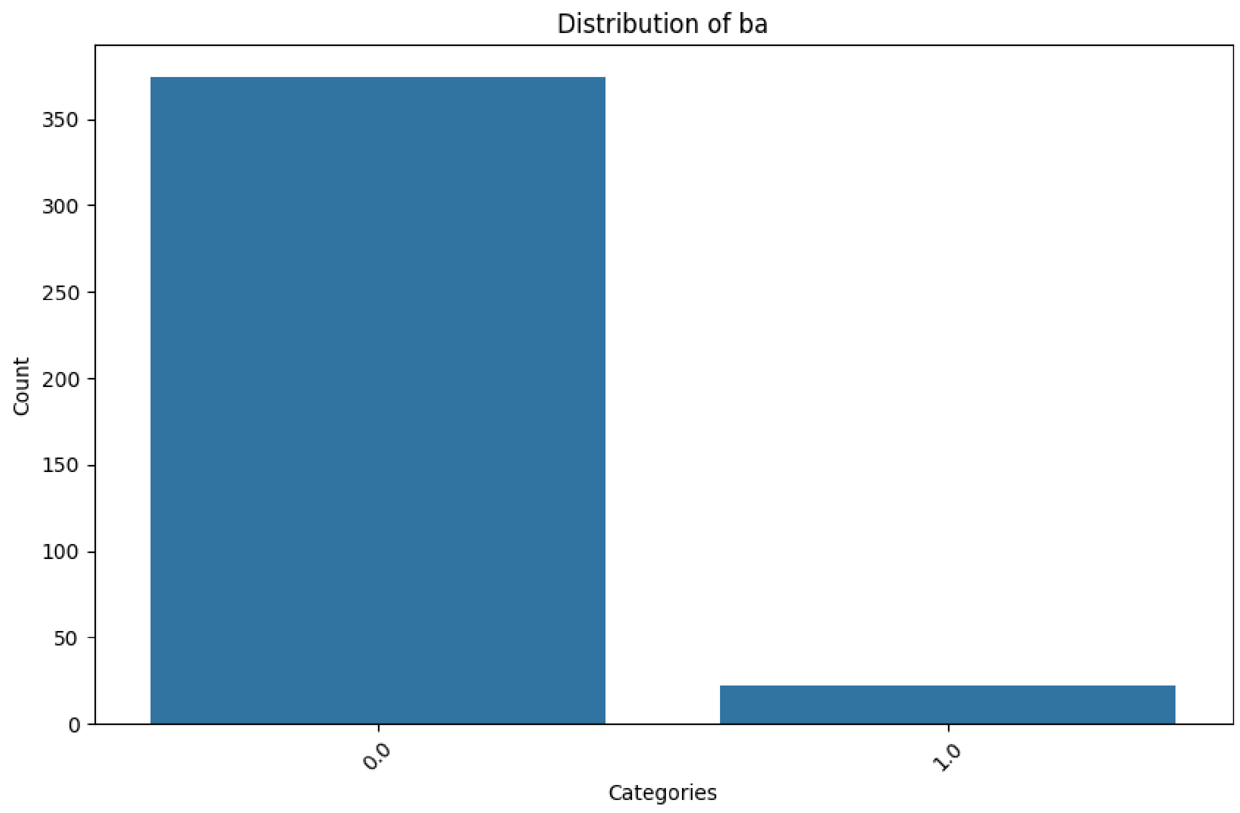

Count
age
bp
sg
bgr
bu
sc
sod
pot
hemo
pcv
wbcc
Frequency
rbcc

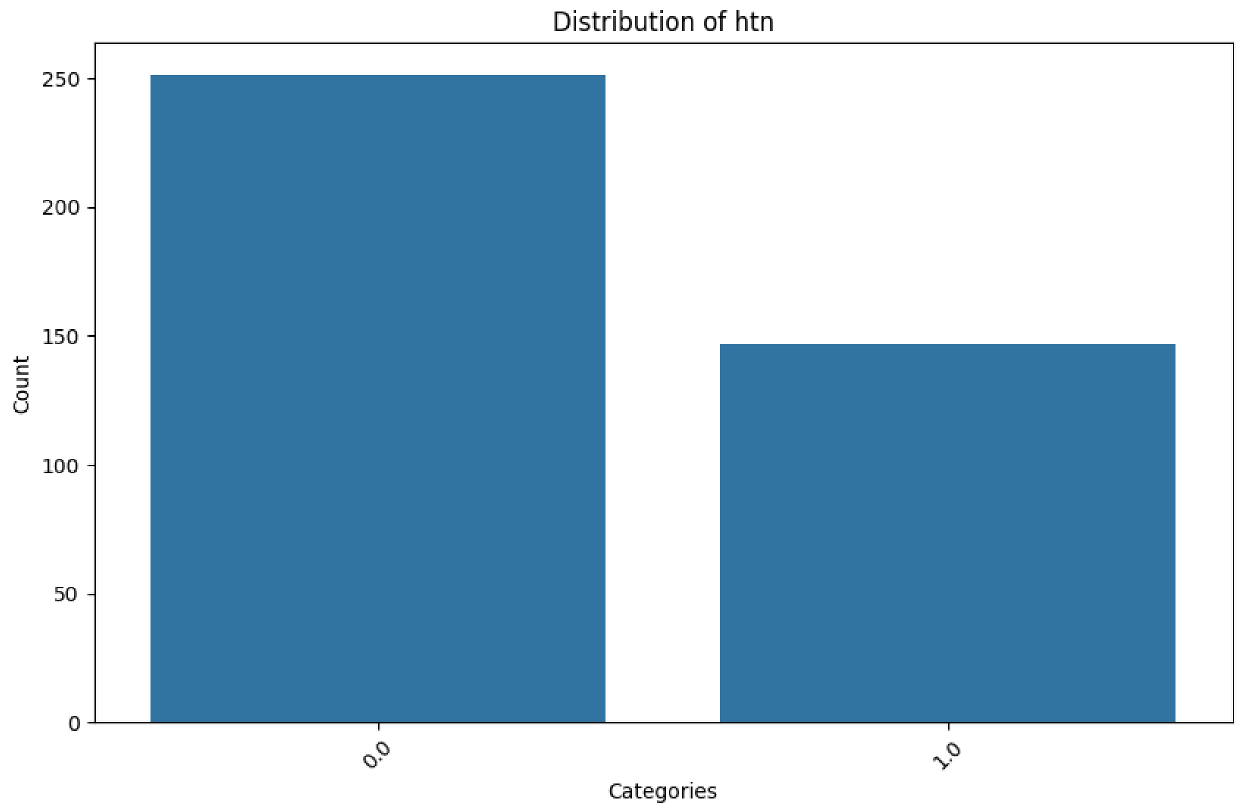
```python
for i in range(len(categorcal_cols)):
    plt.figure(figsize=(10, 6))
    sns.countplot(x=categorcal_cols[i], data=X)
    plt.title('Distribution of '+ categorcal_cols[i])
    plt.xlabel('Categories')
    plt.ylabel('Count')
    plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed
    plt.show()
```

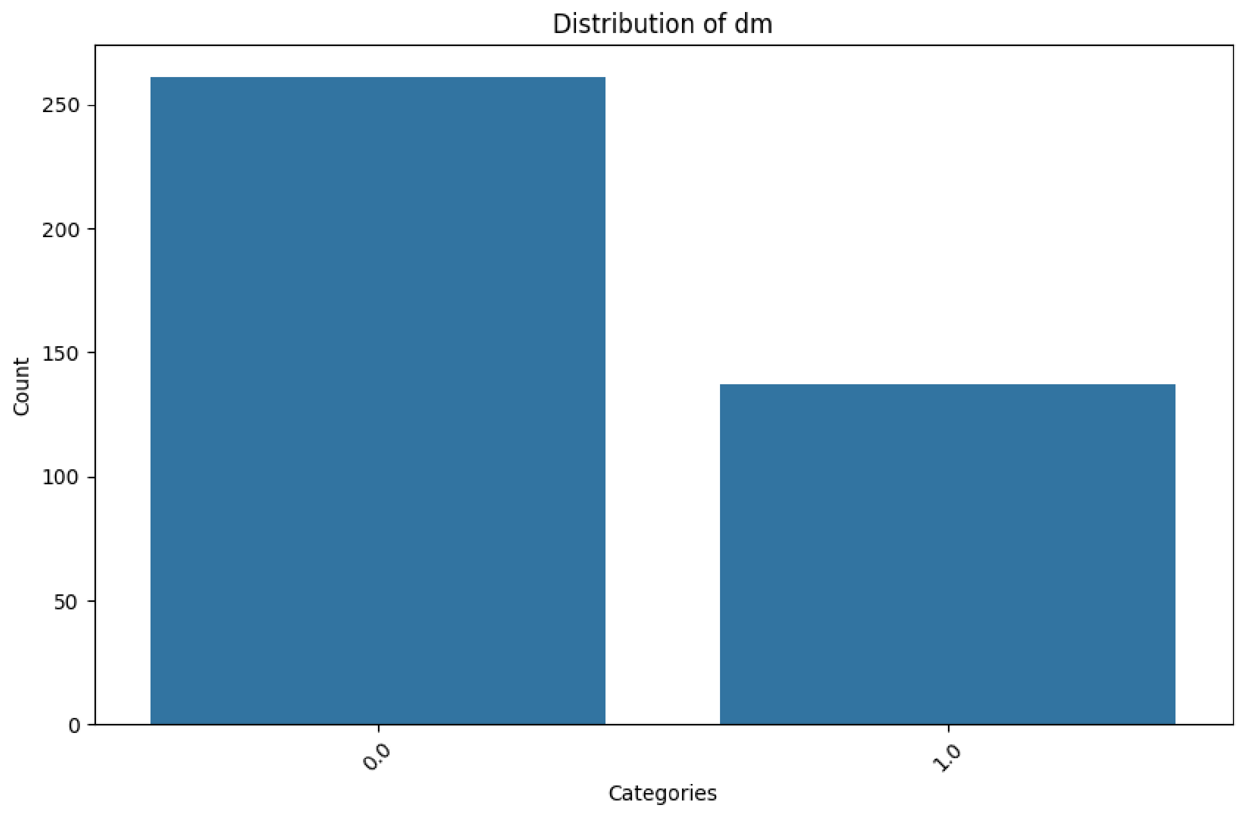


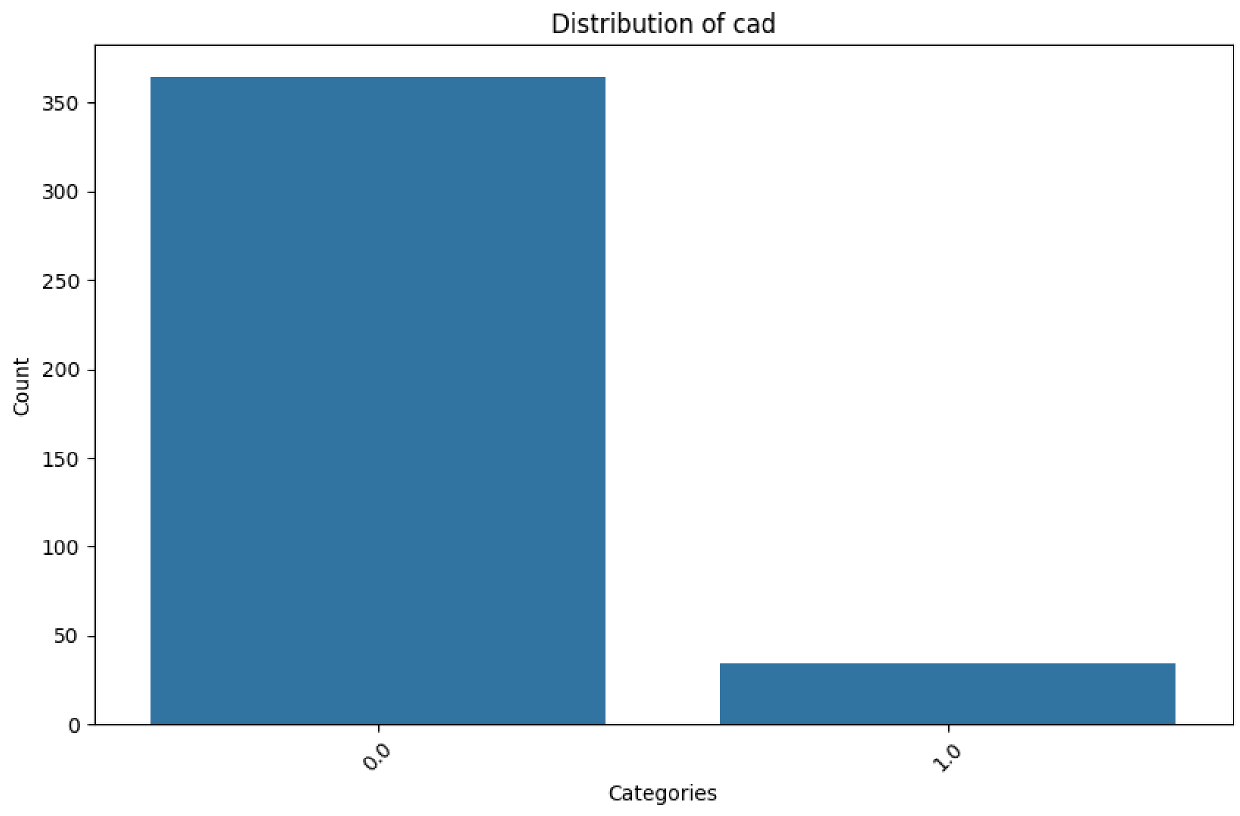

Distribution of pc
Count
250
200
150
100
50
0
0.0
1.0
Categories

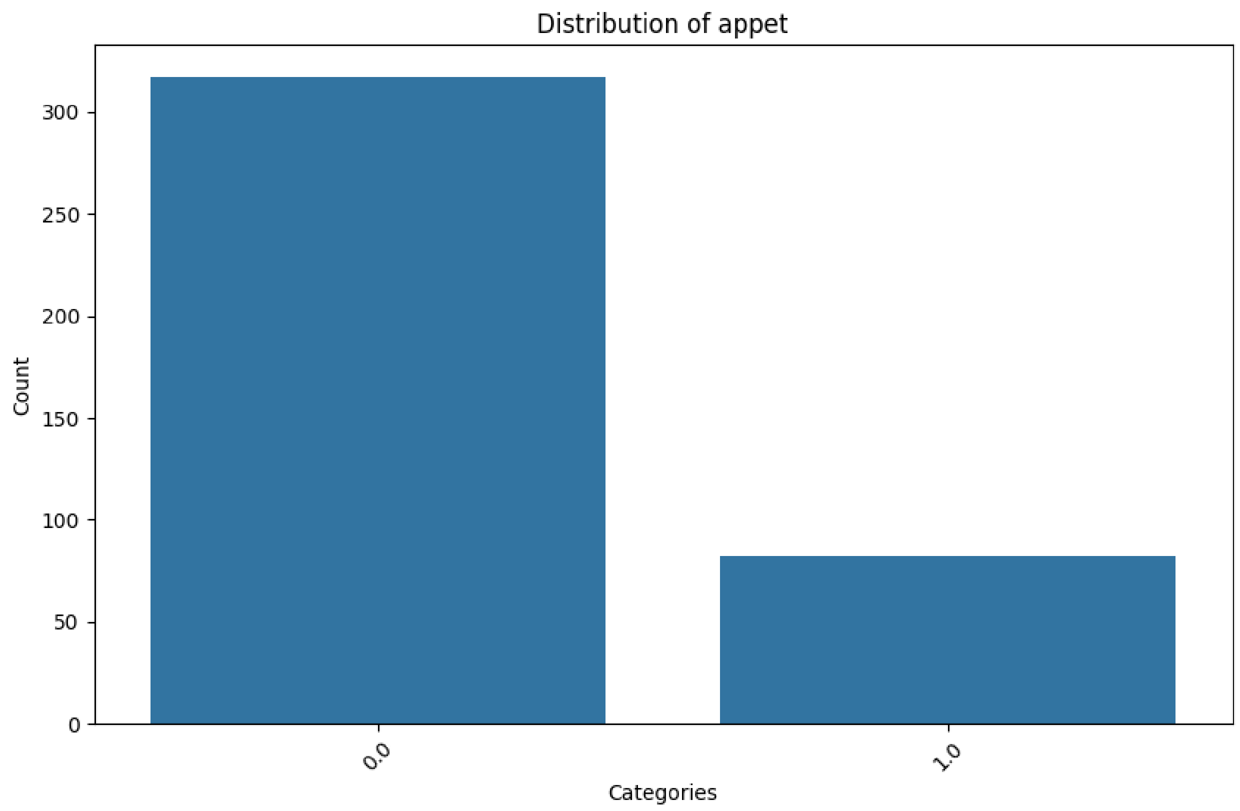

Distribution of pcc
Count
350
300
250
200
150
100
50
0
0.0
1.0
Categories

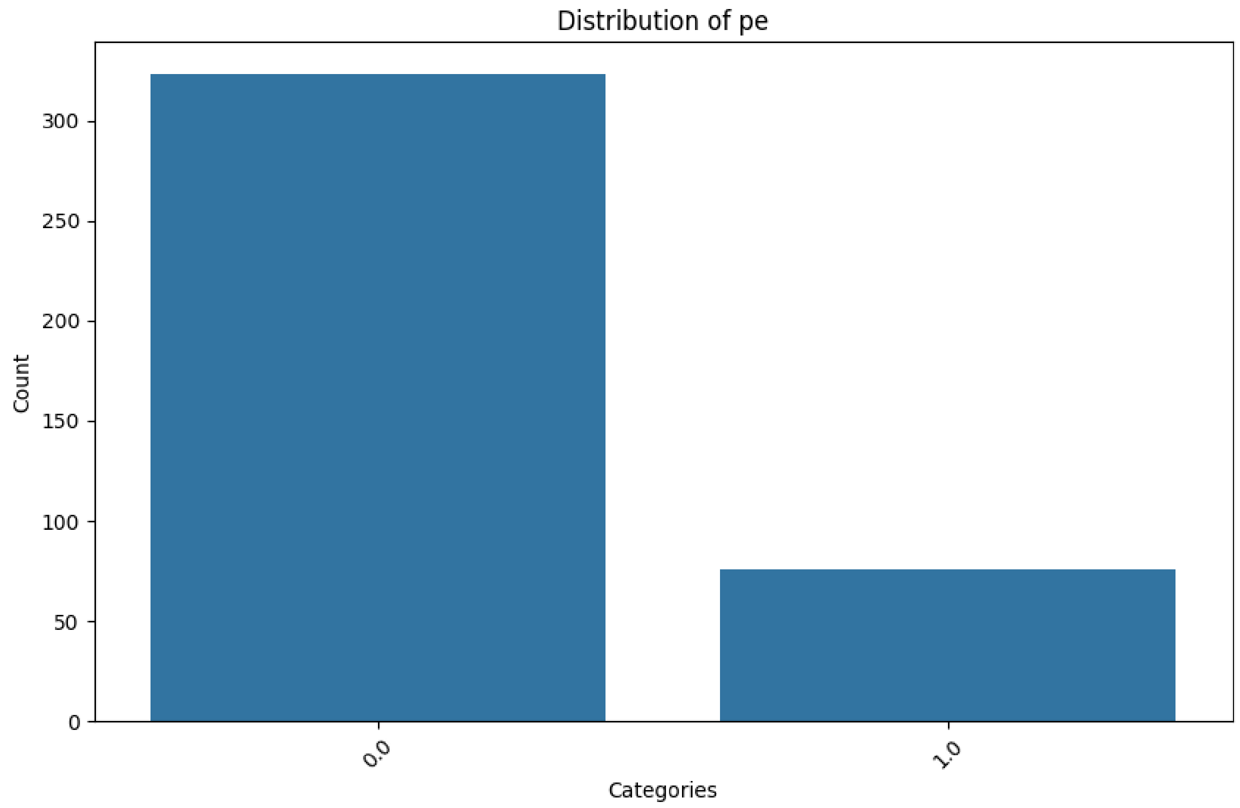

Distribution of ba
Count
Categories
0.0
1.0
0
50
100
150
200
250
300
350

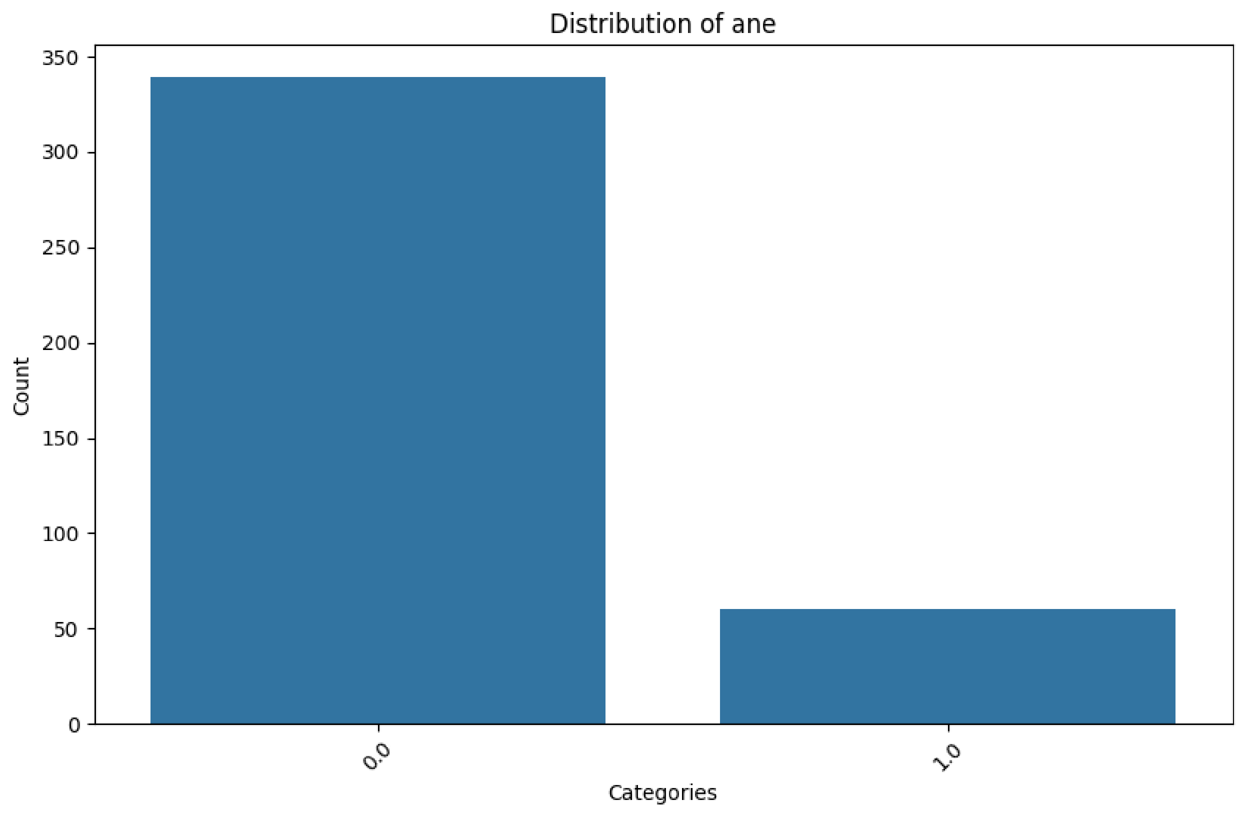

Distribution of htn
Count
Categories
0.0
1.0
0
50
100
150
200
250

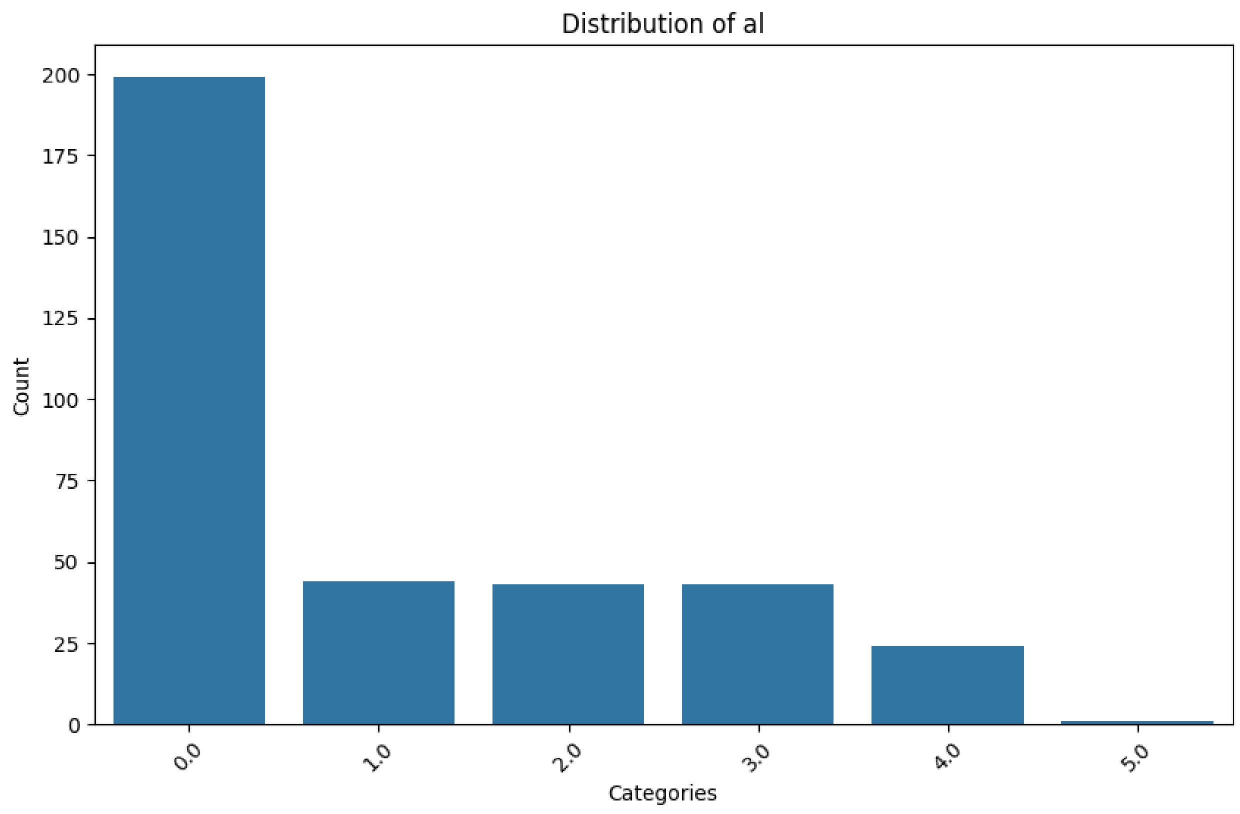

Distribution of dm
Count
Categories
0.0
1.0
0
50
100
150
200
250

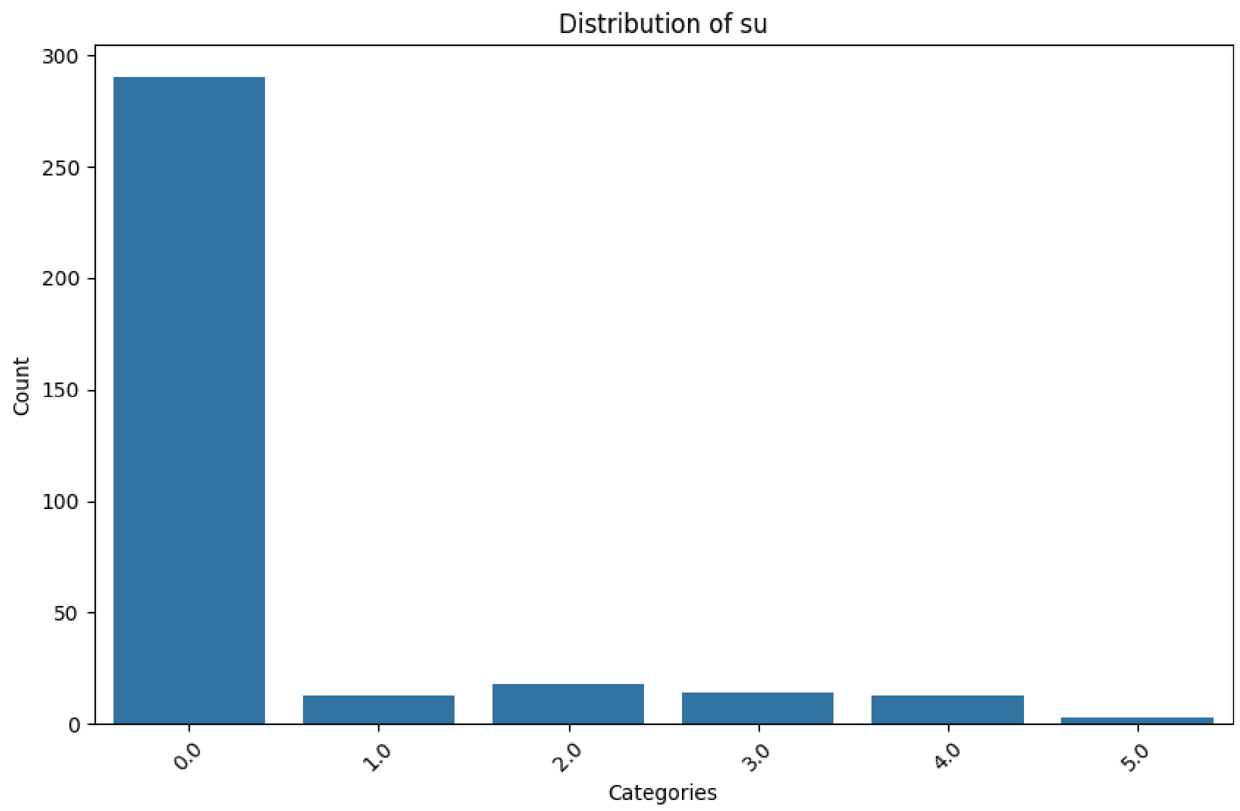

Distribution of cad
Count
Categories
0.0
1.0
0
50
100
150
200
250
300
350


Distribution of appet
Count
300
250
200
150
100
50
0
0.0
1.0
Categories


Distribution of pe
Count
300
250
200
150
100
50
0
0.0
1.0
Categories


Distribution of ane
Count
Categories
0.0
1.0
0
50
100
150
200
250
300
350


Distribution of al
Count
Categories
0.0
1.0
2.0
3.0
4.0
5.0
0
25
50
75
100
125
150
175
200



We standardlize numerical columns here, and we will not use them untill we train our SVM model.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
StandardCol=X
StandardCol[numerical_cols] = scaler.fit_transform(StandardCol[numerical_cols])
StandardCol[numerical_cols].describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	391.0	9.994847e-17	1.001281	-2.885708	-0.553039	0.205078	0.759087	2.246163
bp	388.0	-2.380684e-16	1.001291	-1.936857	-0.473370	0.258373	0.258373	7.575807
sg	353.0	2.415443e-15	1.001419	-2.173584	-1.297699	0.454071	0.454071	1.329955
bgr	356.0	-1.796316e-16	1.001407	-1.591967	-0.619380	-0.341498	0.189004	4.319341
bu	381.0	-3.729883e-17	1.001315	-1.108830	-0.603246	-0.305843	0.170001	6.613723
sc	383.0	0.000000e+00	1.001308	-0.466102	-0.378897	-0.309133	-0.047519	12.719271

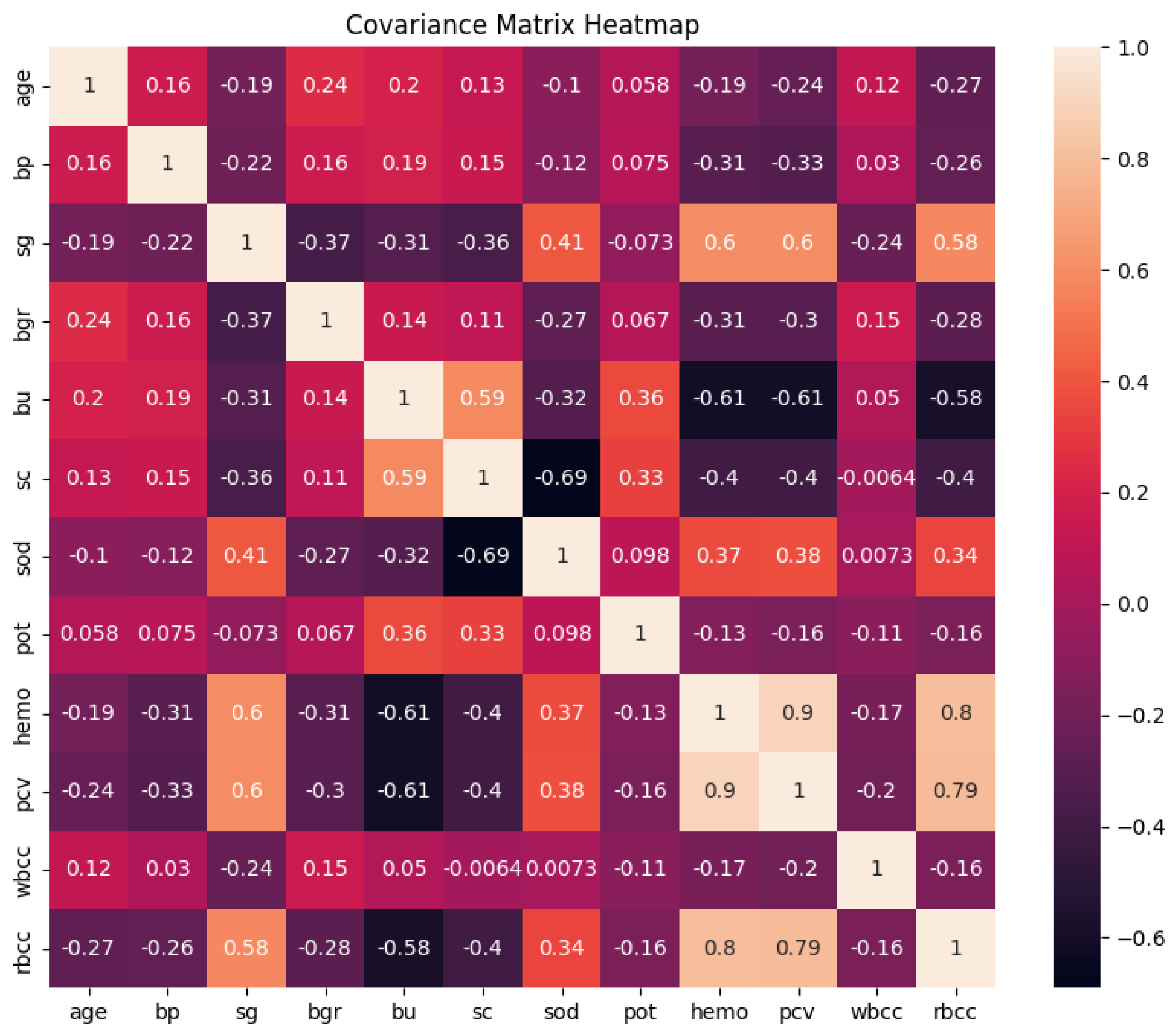
	count	mean	std	min	25%	50%	75%	max
sod	313.0	2.270105e-17	1.001601	-12.800936	-0.243334	0.045347	0.430254	2.451017
pot	312.0	-7.970832e-17	1.001606	-0.667102	-0.259423	-0.071263	0.085536	13.288071
hemo	348.0	4.083579e-17	1.001440	-3.241109	-0.765520	0.042485	0.850490	1.813219
pcv	329.0	1.295823e-16	1.001523	-3.329218	-0.766953	0.124270	0.681284	1.683910
wbcc	294.0	1.450087e-16	1.001705	-2.111312	-0.648460	-0.138162	0.474195	6.121486
rbcc	269.0	8.452553e-16	1.001864	-2.547777	-0.788961	0.090447	0.676719	3.217231

There are a total of 400 records. The age of these records are mostly distributed among 40 to 80. For variable “sc”,“bu”,“bgr” and “wbcc”, they are left-skewed. For variable “pcv”, and “hemo”, they are right-skewed.

4

The heatmap below shows the corrlations of the combinations of the variables.

```python
# Plot heatmap of covariance matrix
plt.figure(figsize=(10, 8))
sns.heatmap(X[numerical_cols].corr(),annot=True)
plt.title('Covariance Matrix Heatmap')
plt.show()
```



Relation between “sod” and “sc” has the most negative corrleation;

Relation between “hemo” and “rbcc” has the most positive corrleation.;

Relation between “wbcc” and “bp” has the smallest corrleation, which means they are close to a independent relation.

5

For quantitative variables, we replace the missing values by the mean of each column. For categorical variables, we replace the missing values by the most frequently appeared value in each column.

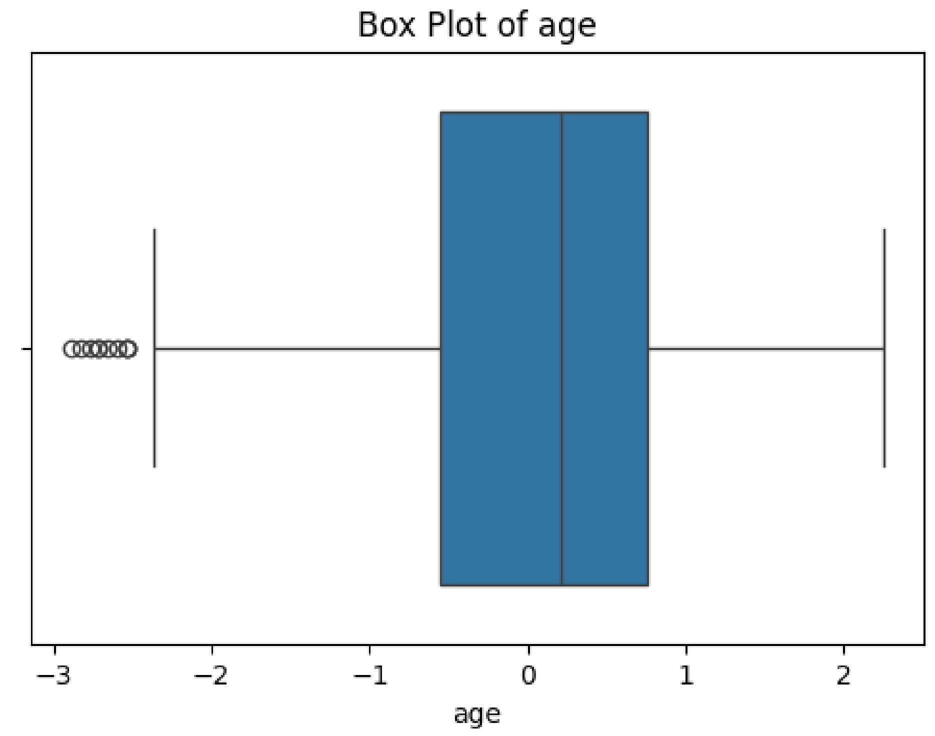
However, we will not process the missing values now since Decision Tree can accept missing values. We will impute missing values before we use SVM.

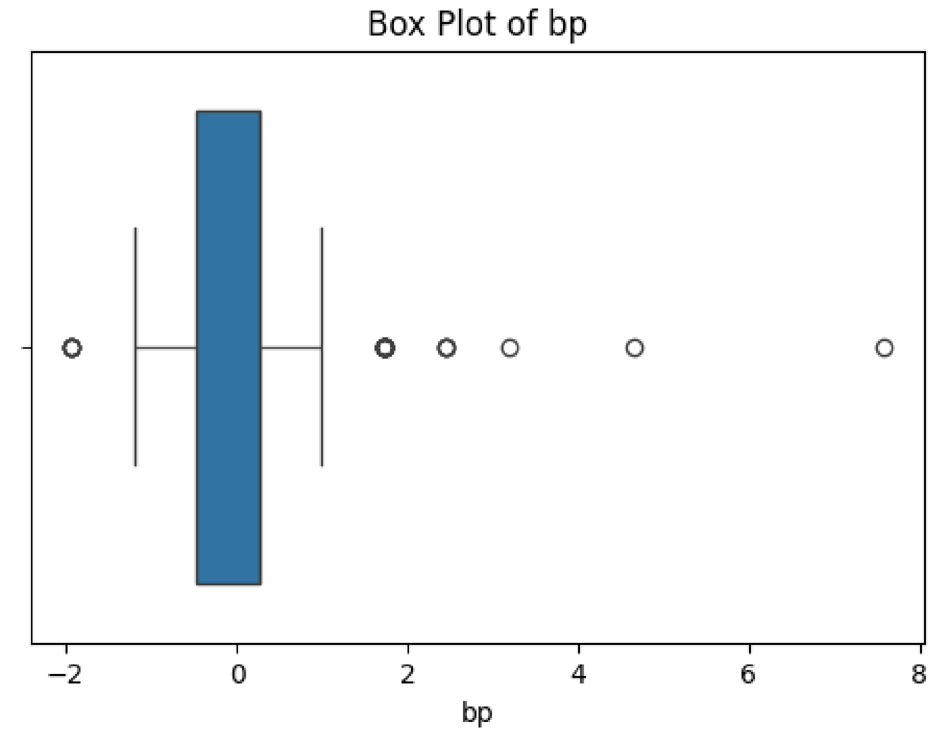
The related code is at page 41.

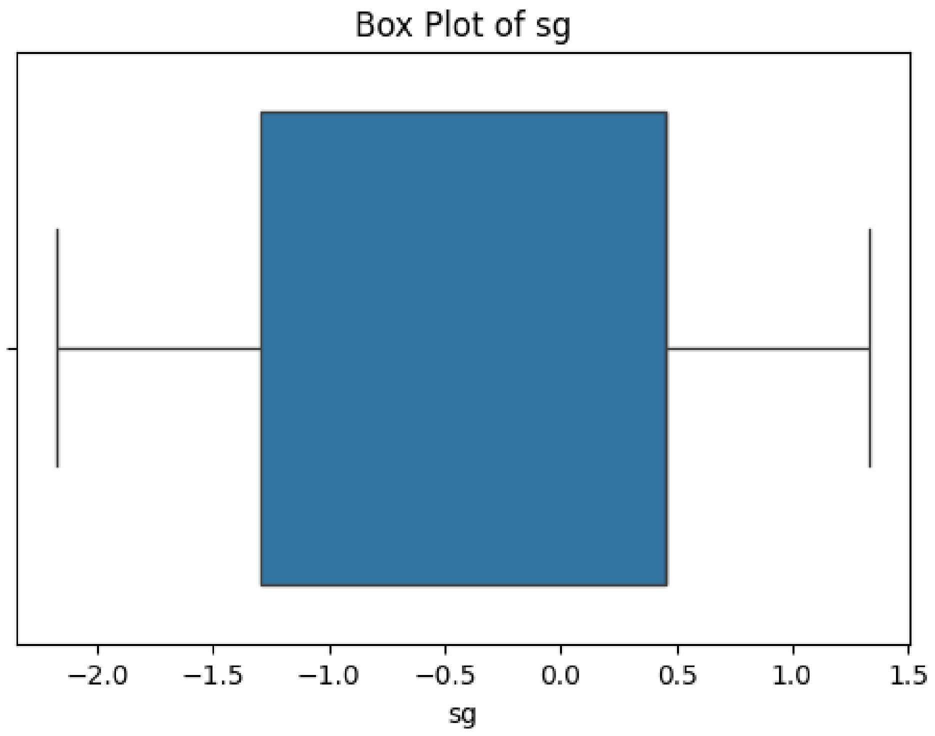
6 Outlier Analysis

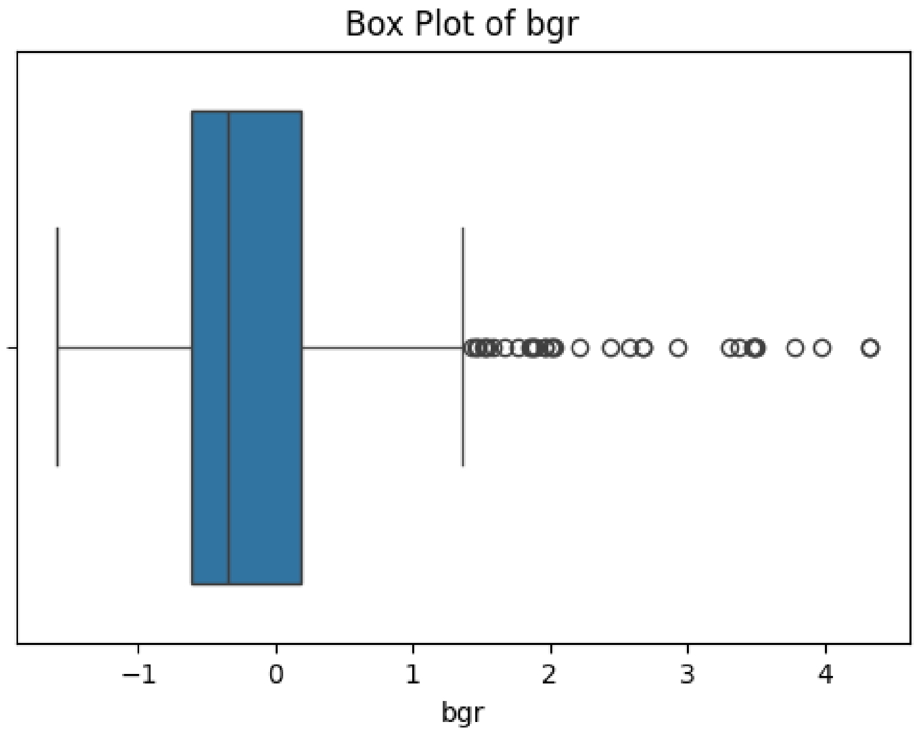
From the histogrms below we obtain that some outliers do exist. However, we decide not to remove them because a certain value in a certain column may be influenced by other variables, we cannot find a clear method or rule to remove the outliers though we got a heatmap. Also, some of the medical data may be conclusive in determing the disease, so we think keep these outlier here is not necessary.

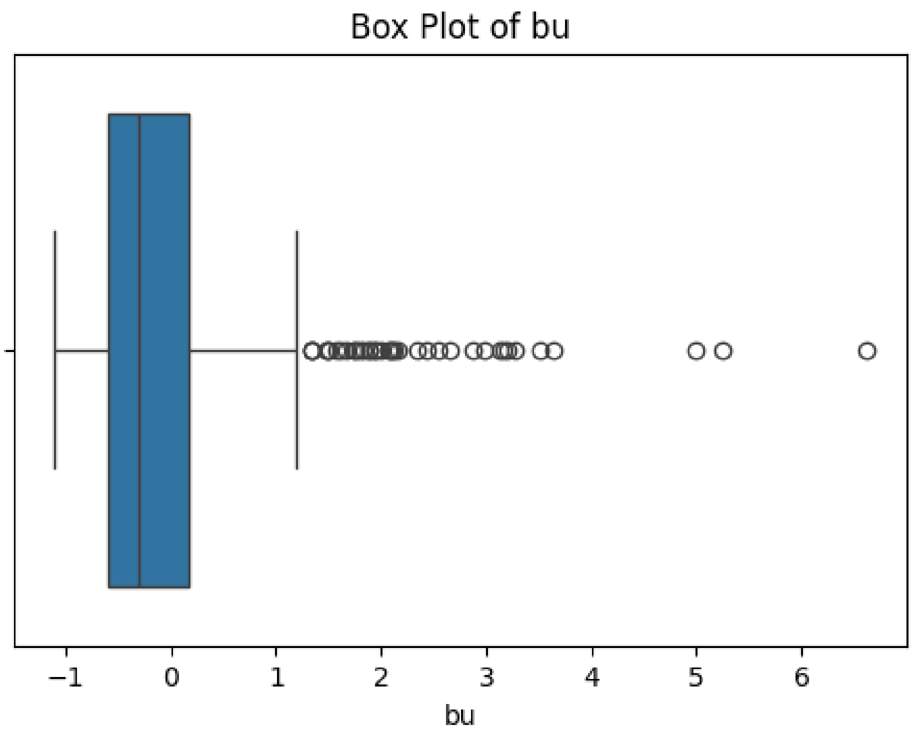
```
for col in numerical_cols:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=X[col])
    plt.title(f'Box Plot of {col}')
    plt.xlabel(col)
    plt.show()
```

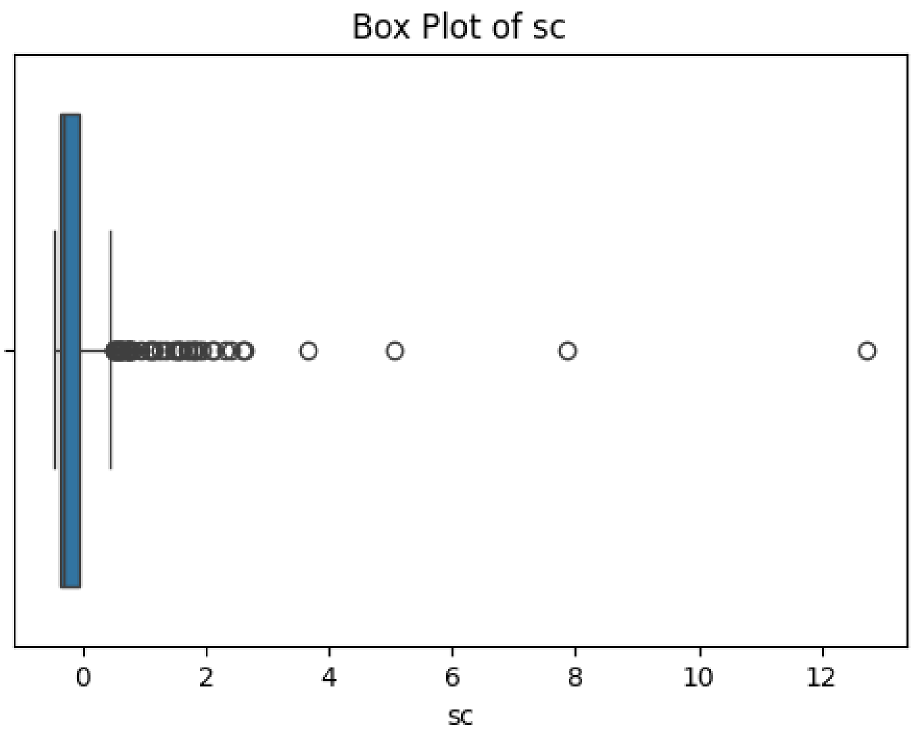

Box Plot of age
−3
−2
−1
0
1
2
age

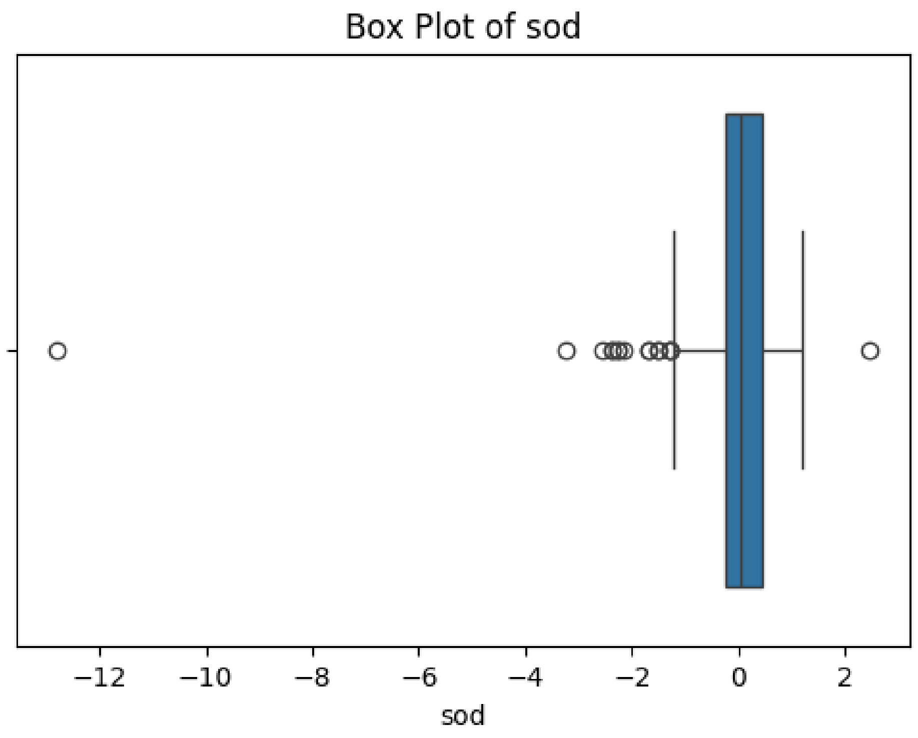

Box Plot of bp
−2
0
2
4
6
8
bp

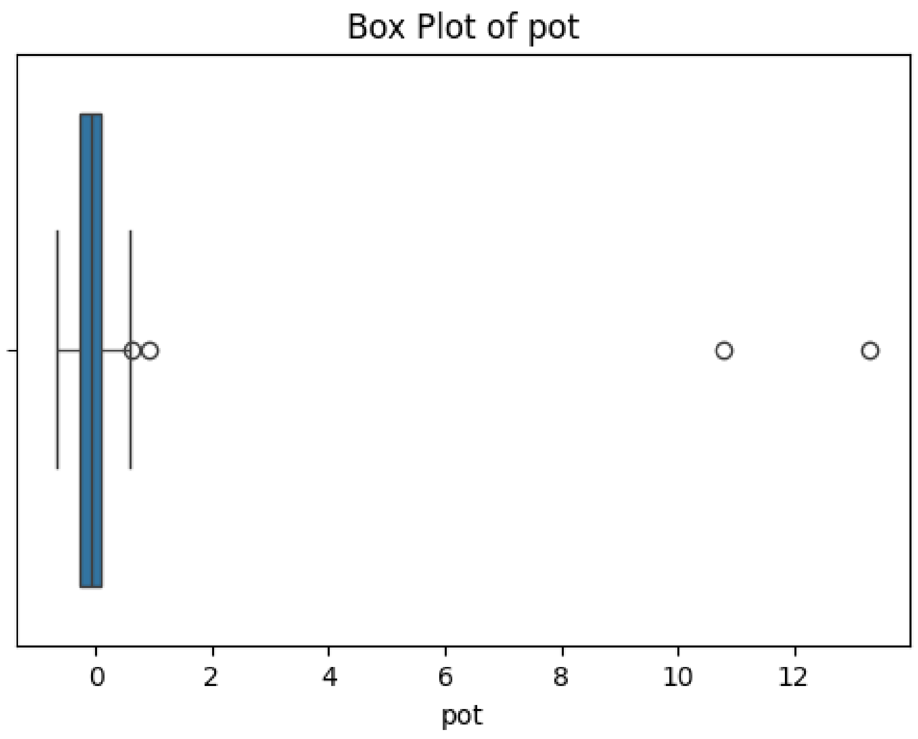

Box Plot of sg
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
sg

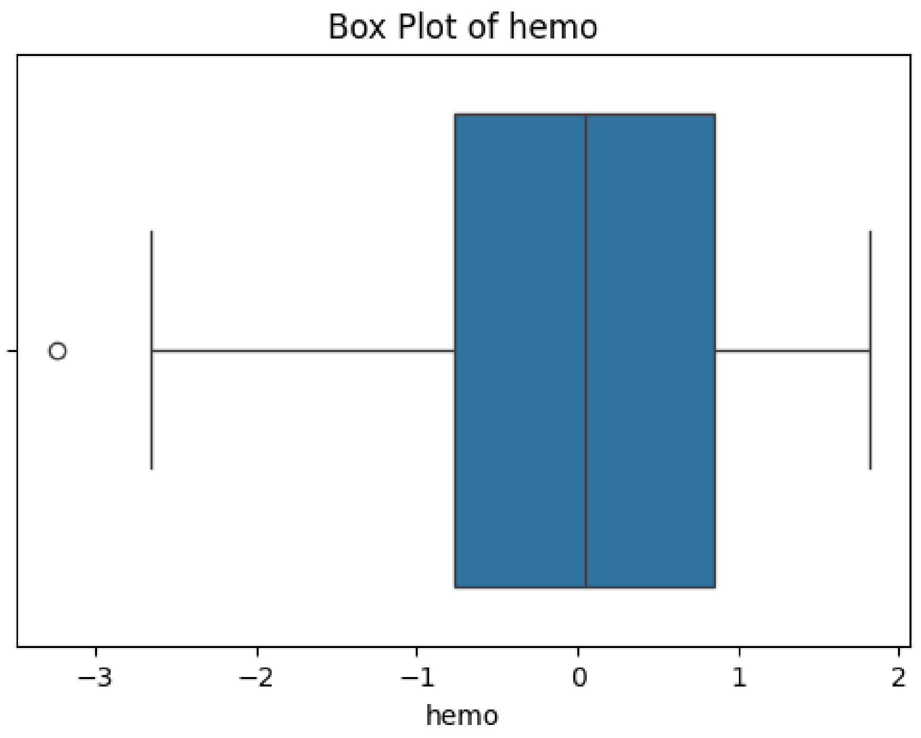

Box Plot of bgr
−1
0
1
2
3
4
bgr

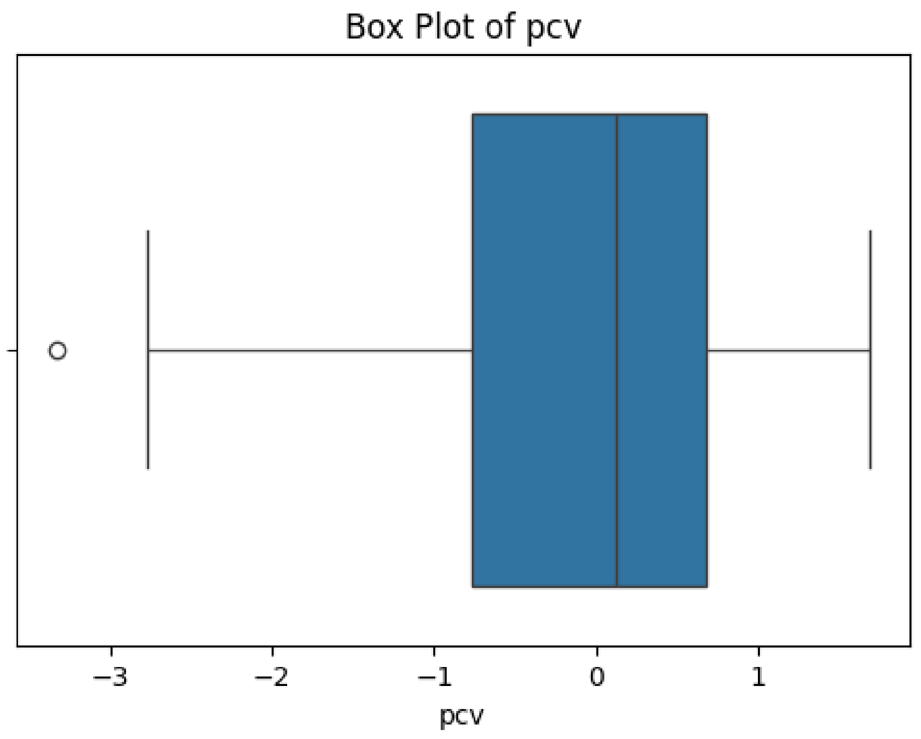

Box Plot of bu
−1
0
1
2
3
4
5
6
bu

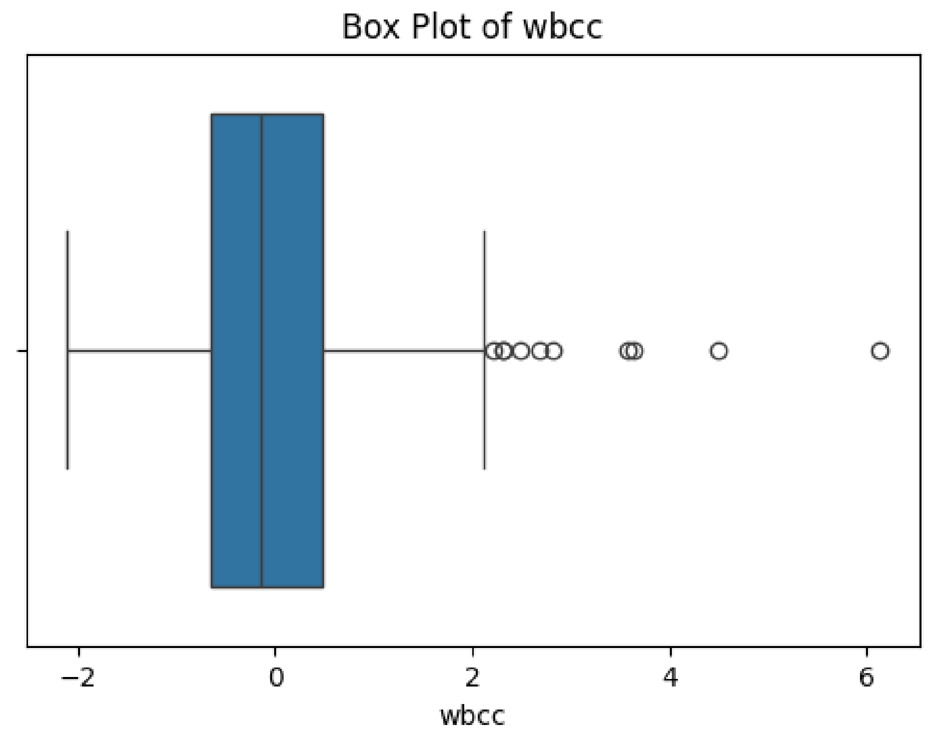

Box Plot of sc
0
2
4
6
8
10
12
sc

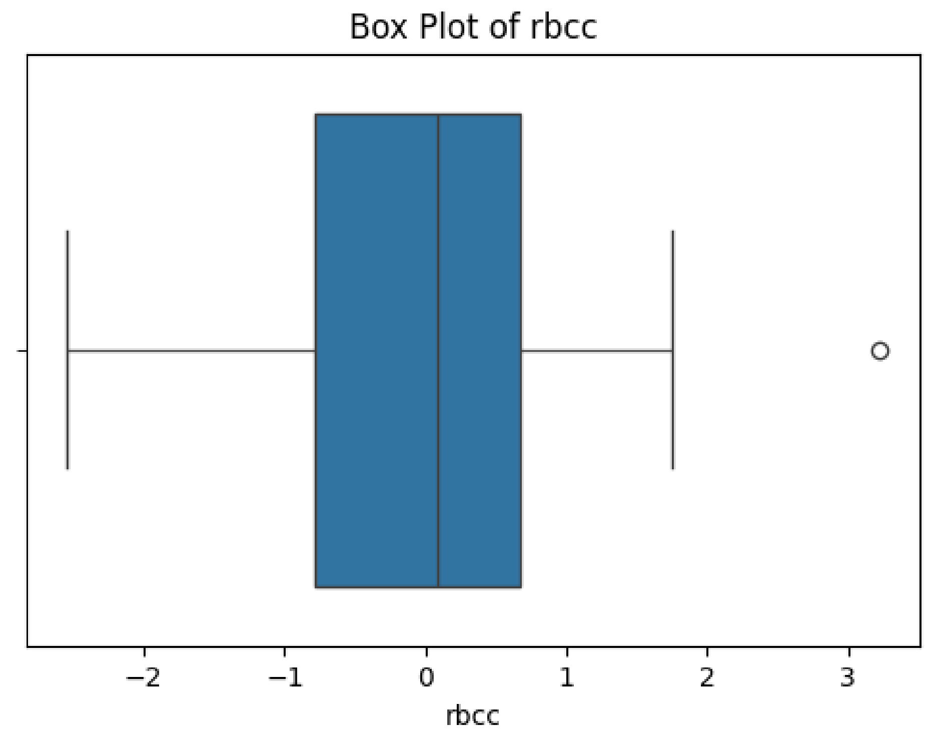

Box Plot of sod
−12
−10
−8
−6
−4
−2
0
2
sod


Box Plot of pot
0
2
4
6
8
10
12
pot


Box Plot of hemo
−3
−2
−1
0
1
2
hemo


Box Plot of pcv
−3
−2
−1
0
1
pcv


Box Plot of wbcc
−2
0
2
4
6
wbcc


Box Plot of rbcc
−2
−1
0
1
2
3
rbcc

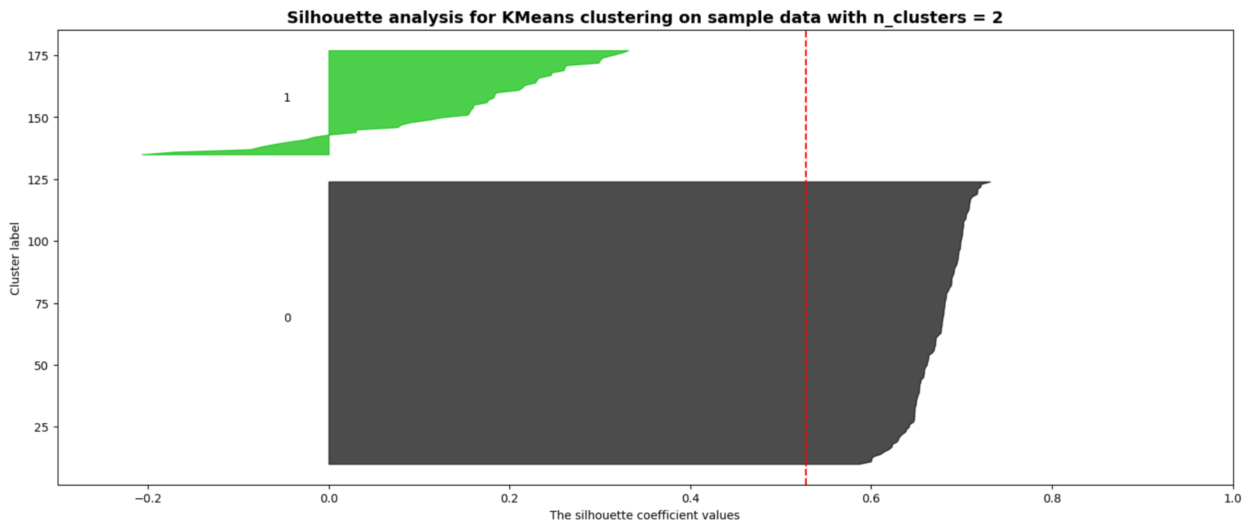
7. Subgrouping

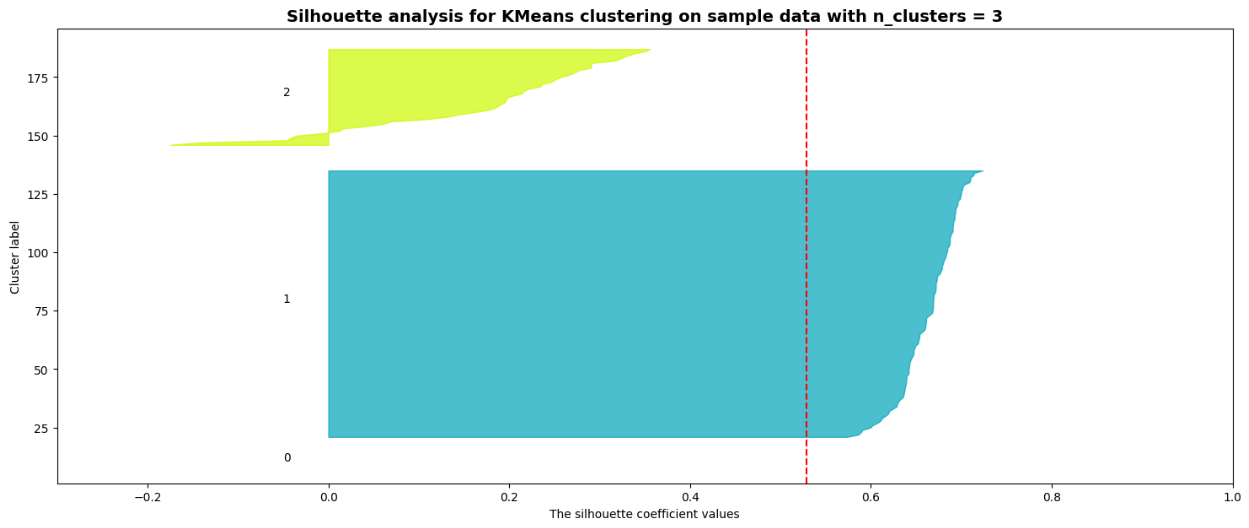
```
combined_df = pd.concat([X, y], axis=1)


# Drop rows with missing values
combined_df = combined_df.dropna()


new_y=combined_df["class"]
new_X = combined_df.drop(columns=["class"])
```

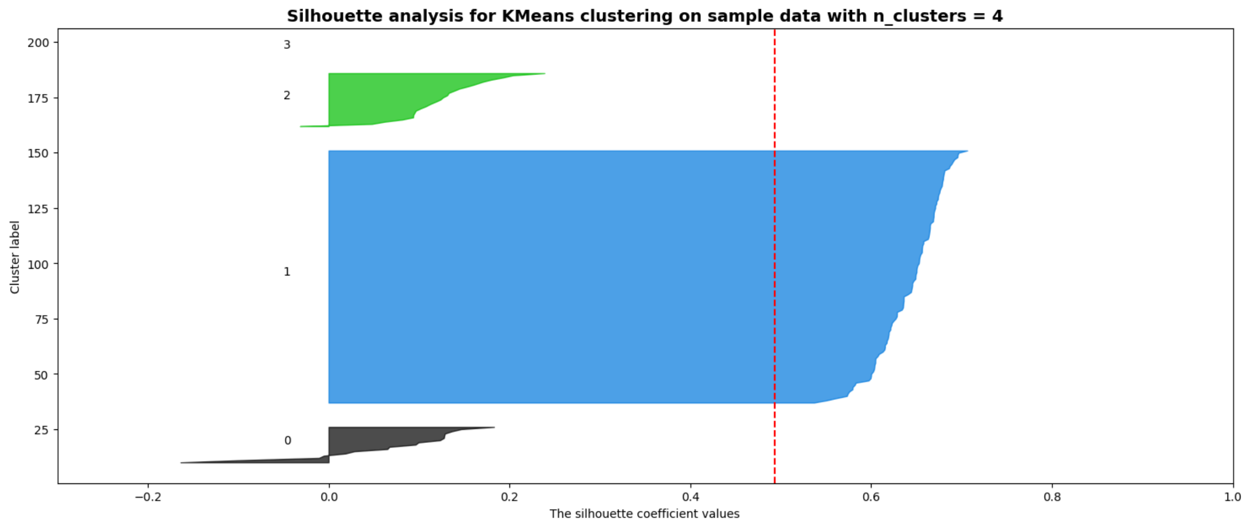
```
print(new_y.shape)
print(new_X.shape)
```

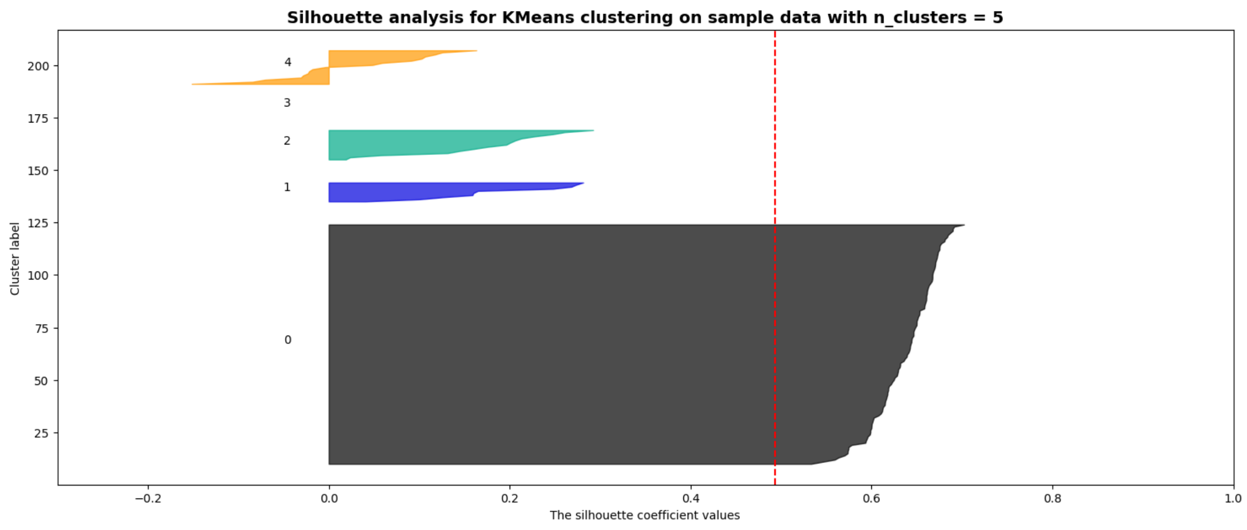
```
(158,)
(158, 24)
```

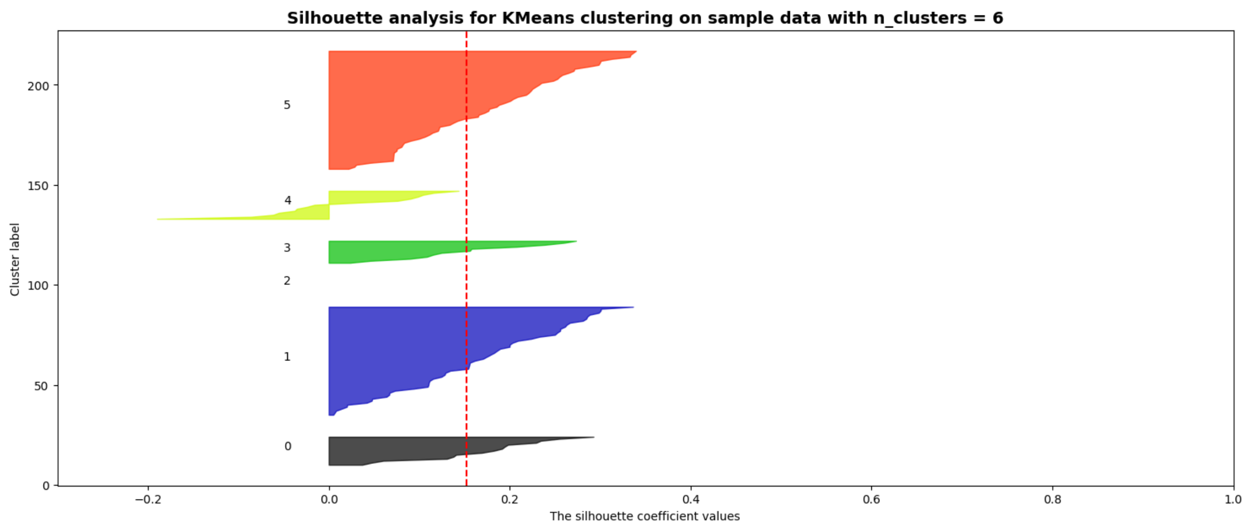
```
range_n_clusters = [2, 3, 4, 5, 6]
for n_clusters in range_n_clusters:
    km = KMeans(n_clusters=n_clusters, n_init=20, random_state=0)
    labels = km.fit_predict(new_X)

    silhouette_avg = silhouette_score(new_X, labels)
    sample_silhouette_values = silhouette_samples(new_X, labels)
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.3, 1])

    y_lower = 10

    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_values[labels == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        # Use the colormap for coloring
        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,
        )

        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        y_lower = y_upper + 10

    ax1.set_title("The silhouette plot for various clusters")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    ax1.axvline(x=silhouette_avg, 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()
```


Silhouette analysis for KMeans clustering on sample data with n_clusters = 2
Cluster label
The silhouette coefficient values
0
1


Silhouette analysis for KMeans clustering on sample data with n_clusters = 3
Cluster label
The silhouette coefficient values
0
1
2

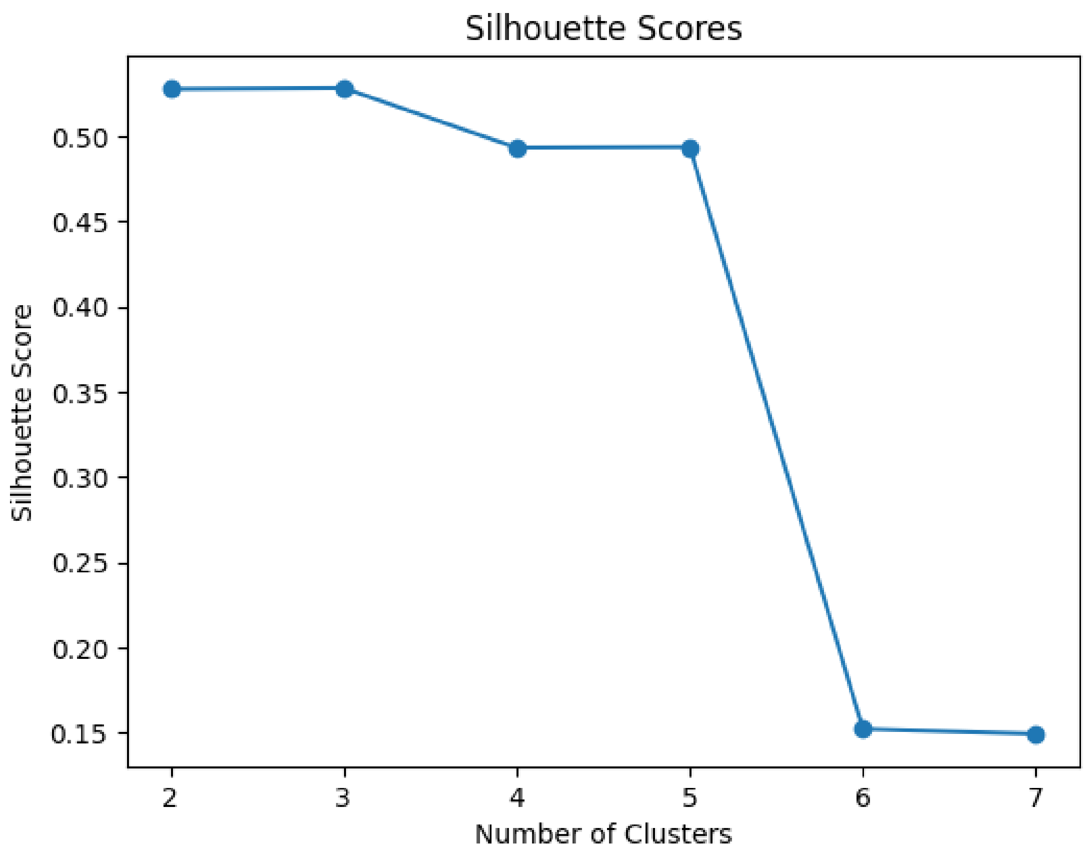

Silhouette analysis for KMeans clustering on sample data with n_clusters = 4
Cluster label
The silhouette coefficient values
0
1
2
3





```
k_values = range(2, 8) # You can adjust this range as needed
# Initialize lists to store silhouette scores
silhouette_scores = []
# Iterate through different values of k
for k in k_values:
# Fit KMeans clustering to the data
    kmeans = KMeans(n_clusters=k, n_init = 20,random_state=0)
    kmeans.fit(new_X)
# Compute the silhouette score
    silhouette_avg = silhouette_score(new_X, kmeans.labels_)
    silhouette_scores.append(silhouette_avg)
```

```
# Plot silhouette scores against k
plt.plot(k_values, silhouette_scores, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Scores')
plt.show()
```



We observe that when k=2, it has the best silhouette score. This result also aligns with the common sense the dataset.

```
combined_df = pd.concat([X, y], axis=1)


# Drop rows with missing values
combined_df = combined_df.dropna()
```

```
combined_df["class"]
```

```
3         ckd
9         ckd
11        ckd
14        ckd
20        ckd
        ...
395    notckd
396    notckd
397    notckd
398    notckd
399    notckd
Name: class, Length: 158, dtype: object
```

8. Data Splitting

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

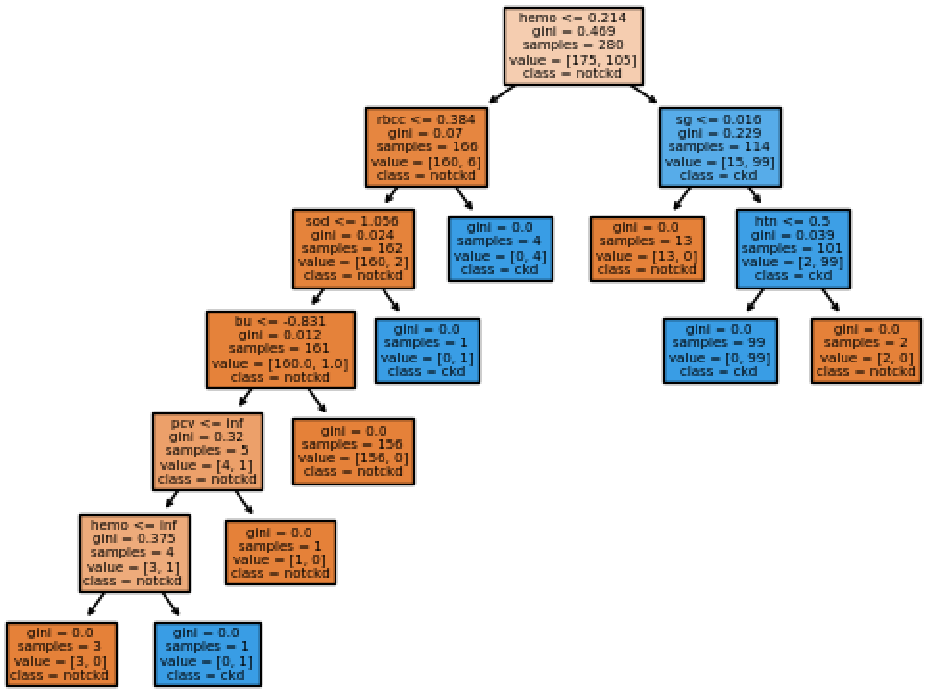
9.

Like we mentioned before, we will be using Decision Tree and SVM for classification. We choose these two methods after we explore the whole dataset.

```
cs_dt = DecisionTreeClassifier(
    max_depth =30,
    random_state=1
)
```

```
cs_dt.fit(X_train, y_train)


plot_tree(
    cs_dt,
    max_depth= 30,
    feature_names = X_train.columns.tolist(),
    class_names=['notckd', 'ckd'],
    filled=True
)


pred_DT = cs_dt.predict(X_test)
pred_DT[:5]
```

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



10. Performance Metrics

We use the confucsion matrix as well as classification_report to measure our accuracy.

```
cm_DT = pd.DataFrame(confusion_matrix(y_test, pred_DT), index=['No', 'Yes'], columns=['No', 'Ye
cm_DT.index.name = 'True'
cm_DT.columns.name = 'Predicted'



print(cm_DT)
print(cs_dt.score(X_test, y_test))
print(classification_report(y_test, pred_DT))
```

```
Predicted  No  Yes
True
No         73    2
Yes         3   42
0.9583333333333334
              precision    recall  f1-score   support

         ckd       0.96      0.97      0.97        75
      notckd       0.95      0.93      0.94        45

    accuracy                           0.96       120
   macro avg       0.96      0.95      0.96       120
weighted avg       0.96      0.96      0.96       120
```

11.

Here we use two methods to improve the performance on decision tree. First method we prune the tree. Second Method, we fit our data with criterion='entropy', we did some research, and we agree on using entropy instead of “gini” is more suitable for this dataset, and its performance also proves that.

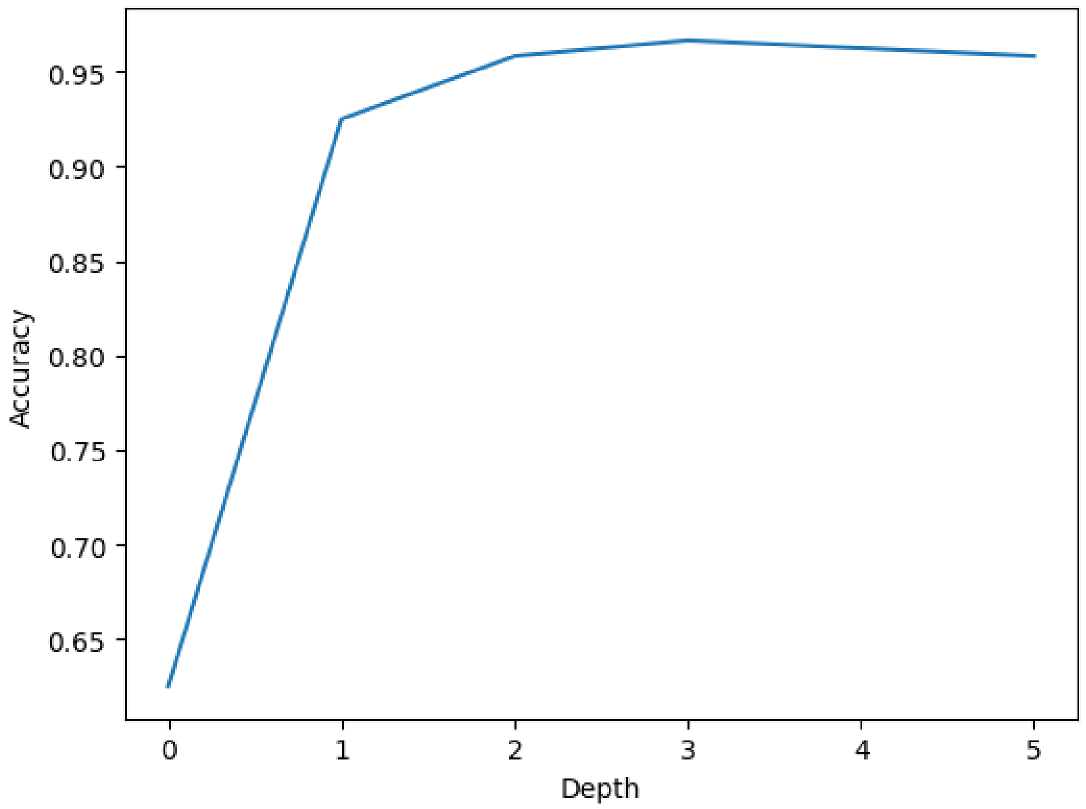
```
path = cs_dt.cost_complexity_pruning_path(
    X_train,
    y_train
)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

```
clfs = [] # save fitted trees with different alphas
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(
        random_state=0,
        ccp_alpha=ccp_alpha
        )
    clf.fit(X_train, y_train)
    clfs.append(clf)
```

```
depth = [clf.tree_.max_depth for clf in clfs]
depth
```

```
[5, 5, 3, 2, 2, 1, 0]
```

```
test_score = [clf.score(X_test, y_test) for clf in clfs]
plt.plot(depth, test_score)
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.show()
```



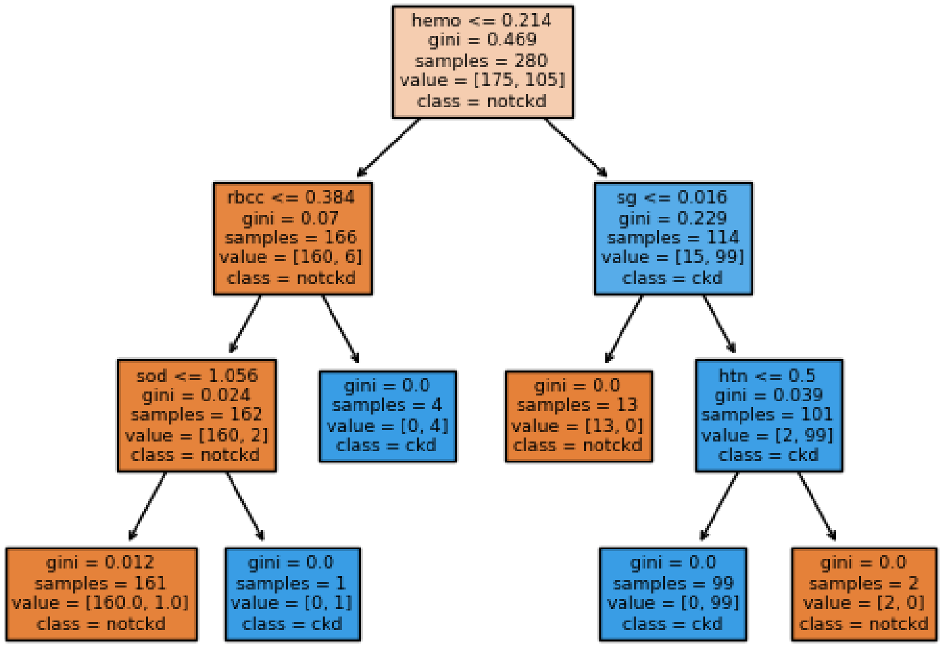
```
cs_dt_best = DecisionTreeClassifier(
    max_depth = 3,
    random_state=1
    )
cs_dt_best.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=3, random_state=1)
```

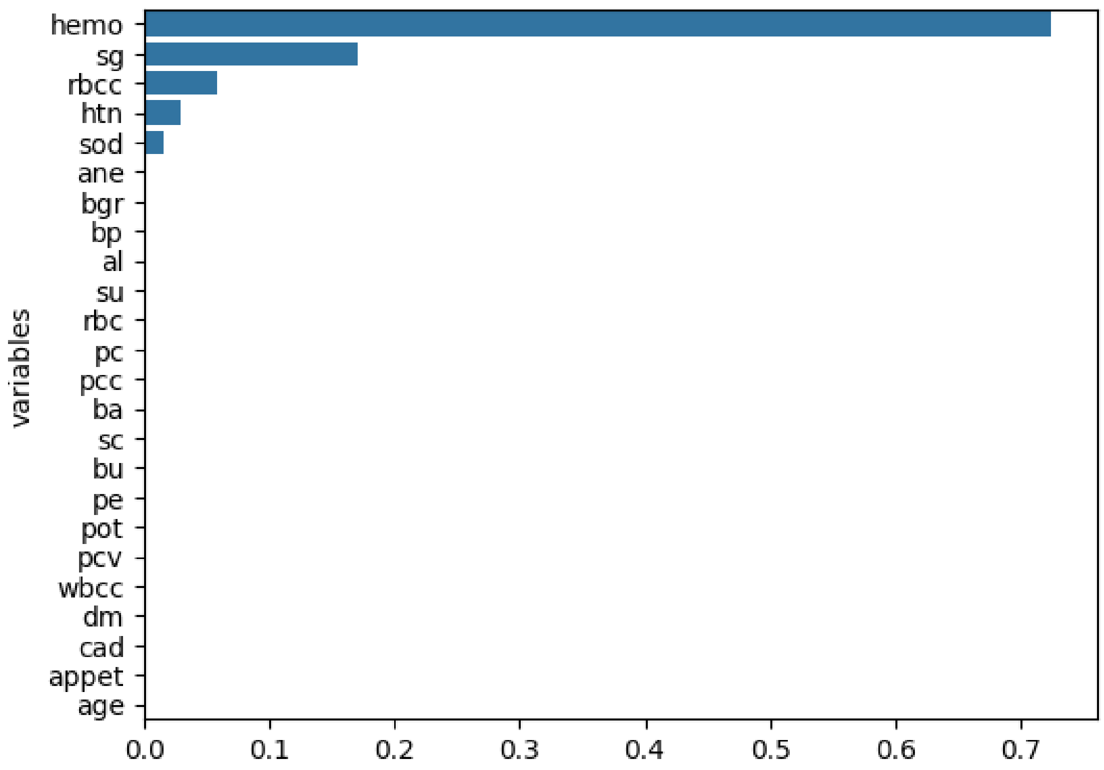
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y)
plot_tree(
    cs_dt_best,
    max_depth= 3,
    feature_names = X_train.columns.tolist(),
    filled=True,
    class_names=['notckd', 'ckd']
```

```
)

pred_BDT = cs_dt.predict(X_test)

print(classification_report(y_test, pred_BDT))
cm_BDT = pd.DataFrame(confusion_matrix(y_test, pred_BDT), index=['No', 'Yes'], columns=['No',
cm_BDT.index.name = 'True'
cm_BDT.columns.name = 'Predicted'

print(cm_BDT)
cs_dt.score(X_test, y_test)
print("Accuracy:", accuracy_score(y_test, pred_BDT))
```

```
              precision    recall  f1-score   support

         ckd       0.96      0.97      0.97        75
      notckd       0.95      0.93      0.94        45

    accuracy                           0.96       120
   macro avg       0.96      0.95      0.96       120
weighted avg       0.96      0.96      0.96       120

Predicted  No  Yes
True
No         73    2
Yes         3   42
Accuracy: 0.9583333333333334
```



```
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.ylabel("variables")


plt.show()
```

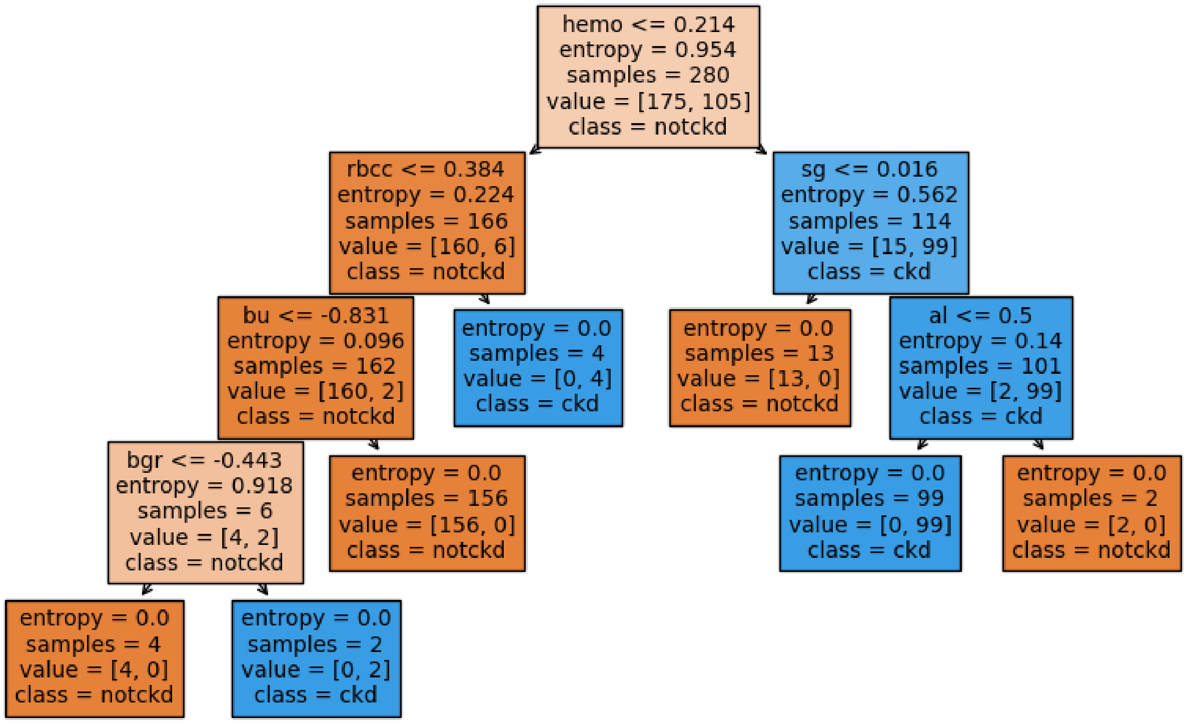


From above plot, we can observe variable “hemo” has a significant influce on the model, next is “sg” and “rbcc”. “htn” and “sod” also have some influence. Rest of the variables do not have a crucial impact on the model. After pruning the tree, we can see the acuracy did not increase. That is mostly because the previous one has a really large depth, but even with a max depth of 3, the model can achieve the same accuracy. Simplifing the model can also be seen as an improvement.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y)
clf = DecisionTreeClassifier(max_depth =30,
    random_state=1,criterion='entropy')
clf.fit(X_train, y_train)



pred_clf = clf.predict(X_test)
```

```
print("Accuracy:", accuracy_score(y_test, pred_clf))
print(classification_report(y_test, pred_clf))



plt.figure(figsize=(10,6))
plot_tree(clf, filled=True,feature_names = X_train.columns.tolist(),class_names=['notckd', 'ck
plt.show()
```

```
Accuracy: 0.9666666666666667
              precision    recall  f1-score   support

         ckd       0.96      0.99      0.97        75
      notckd       0.98      0.93      0.95        45

    accuracy                           0.97       120
   macro avg       0.97      0.96      0.96       120
weighted avg       0.97      0.97      0.97       120
```



The next method we used for improving the model is to change the crierion. The improvement of accuracy scores indicates a better result, which implies we have a better model.

```
cm_clf = pd.DataFrame(confusion_matrix(y_test, pred_clf), index=['No', 'Yes'], columns=['No',
cm_clf.index.name = 'True'
cm_clf.columns.name = 'Predicted'
cm_clf
```

Predicted	No	Yes
True		
No	74	1
Yes	3	42

Like we mentioned above, we will use mean for missing numerical datas, and most frequent more categorical data.

```
float_col = X[numerical_cols].columns.tolist()
num_imputer = SimpleImputer(missing_values=np.nan,strategy='mean')
X.loc[:, float_col] = num_imputer.fit_transform(X.loc[:, float_col])

# For categorical features
obj_col = X[categorcal_cols].columns.tolist()
cat_imputer = SimpleImputer(strategy='most_frequent',missing_values=np.nan)
X.loc[:, obj_col] = cat_imputer.fit_transform(X.loc[:, obj_col])
```

```
X.isna().sum()
```

```
age      0
bp       0
sg       0
al       0
su       0
```

```
rbc      0
pc       0
pcc      0
ba       0
bgr      0
bu       0
sc       0
sod      0
pot      0
hemo     0
pcv      0
wbcc     0
rbcc     0
htn      0
dm       0
cad      0
appet    0
pe       0
ane      0
dtype: int64
```

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

```python
# Initialize and train SVM model
svm = SVC(kernel='linear')
y_train = np.ravel(y_train)
svm.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

12

```
cm = confusion_matrix(y_test, pred_clf)
print('Confusion Matrix from Decicion Tree: \n', cm)
accuracyDT = accuracy_score(y_test, pred_clf)
print("Accuracy from SVM:", accuracyDT)
```

```
Confusion Matrix from Decicion Tree:
 [[74  1]
 [ 3 42]]
Accuracy from SVM: 0.9666666666666667
```

```
y_pred_SVM = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_SVM)
cm_SVM = confusion_matrix(y_test, y_pred_SVM)
print('Confusion Matrix from Decicion Tree: \n', cm_SVM)
print("Accuracy from SVM:", accuracy)
```

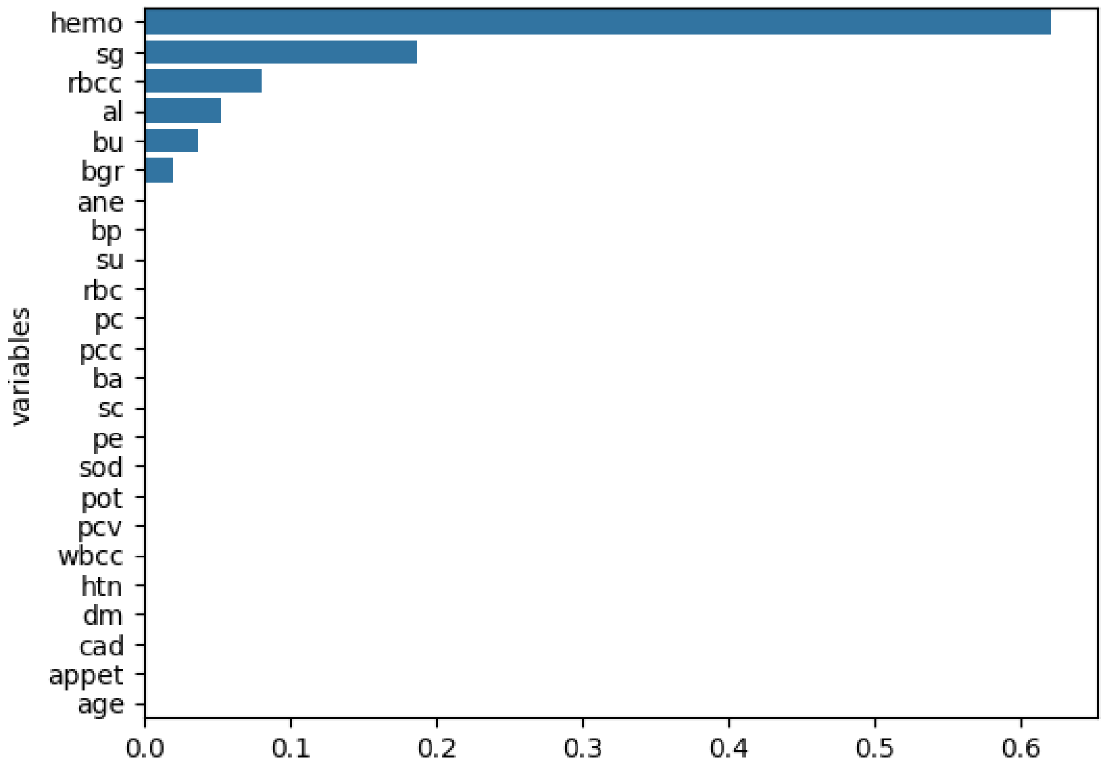
```
Confusion Matrix from Decicion Tree:
 [[75  0]
 [ 2 43]]
Accuracy from SVM: 0.9833333333333333
```

By comparing the confusion matrix and accuarcy score, we can conclude that linear SVC is a more suitable model for this dataset.

13

```
fea_imp = clf.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.ylabel("variables")

plt.show()
```



14

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import accuracy_score


X_train, X_test, y_train, y_test = train_test_split(
    new_X, new_y, test_size=0.3, random_state=1, stratify=new_y)
```

```
poly=PolynomialFeatures(degree=2, interaction_only=True)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)


DTC = DecisionTreeClassifier(max_depth =3,
    random_state=1,criterion='entropy')


DTC.fit(X_train_poly, y_train)


y_pred_poly = DTC.predict(X_test_poly)
accuracy_poly = accuracy_score(y_test, y_pred_poly)
print("New Model Accuracy:", accuracy_poly)
```

```
New Model Accuracy: 0.9791666666666666
```

The improved model does not bring us a better result. It is may because polynomial features introduce interaction terms between features, and sometimes there may be cases these interactions may not be relevant or may even introduce noise, leading to overfitting. It also increase the dimension of the model where a overly complicated model may not be as accurate as a simplier one.

15 Distribution

Tianmu Li (400371751): Question 2, 3, 4, 5, 6, 11

Chengdai Xu (400397089): Question 5, 7, 8, 9, 10, 11,14

Jiajun Zhang (400359213) : Question 1, 3, 7, 11, 12, 13

We have done a lot of work together by sharing the jupyter file itself instead of using github, but all of the questions are done under continuously disscussion and communication in the group. The contribution of each team members are equal and fair.