STATS 3DA3

Homework Assignment 6

Xinyi Chen (400326045), Huixuan Wu (400332004), Ruiqi Xiong (400344851)

2024-04-04

Submission Deadline

- All submissions must be made before 10:00 PM on Thursday, April 18, 2024.

Submission Guidelines

- Format: Submissions are to be made in PDF format via Avenue to Learn, either individually or as a group of up to three members.
  - GitHub Repository: Your submission must include a link to a public GitHub repository containing the assignment.
  - Team Submissions: For group submissions, Question 15 must detail each member's contributions. Note that while there are no points allocated to Question 15, failure to provide this information will result in the assignment not being graded.

Late Submissions

- 15% will be deducted from assignments each day after the due date (rounding up).
- Assignments won't be accepted after 48 hours after the due date.

Assignment Standards

Please ensure your assignment adheres to the following standards for submission:

- **Title Page Requirements:** Each submission must include a title page featuring your group members' names and student numbers. Assignments lacking a title page will not be considered for grading.
- **Individual Work:** While discussing homework problems with peers and group is permitted, the final written submission must be your group work.
- **Formatting Preferences:** The use of LaTeX for document preparation is highly recommended.
- **Font and Spacing:** Submissions must utilize an eleven-point font (Times New Roman or a similar font) with 1.5 line spacing. Ensure margins of at least 1 inch on all sides.

- **Submission Content:** Do not include the assignment questions within your PDF. Instead, clearly mark each response with the corresponding question number. Screenshots are not an acceptable form of submission under any circumstances.
- **Academic Writing:** Ensure that your writing and any references used are appropriate for an undergraduate level of study.
- **Originality Checks:** Be aware that the instructor may use various tools, including those available on the internet, to verify the originality of submitted assignments.
- Assignment policy on the use of generative AI:
  - Students are not permitted to use generative AI in this assignment. In alignment with McMaster academic integrity policy, it "shall be an offence knowingly to … submit academic work for assessment that was purchased or acquired from another source". This includes work created by generative AI tools. Also state in the policy is the following, "Contract Cheating is the act of"outsourcing of student work to third parties" (Lancaster & Clarke, 2016, p. 639) with or without payment." Using Generative AI tools is a form of contract cheating. Charges of academic dishonesty will be brought forward to the Office of Academic Integrity.

Chronic Kidney Disease Classification Challenge

Overview

Engage with the dataset from the Early Stage of Indians Chronic Kidney Disease (CKD) project, which comprises data on 250 early-stage CKD patients and 150 healthy controls.

For foundational knowledge on the subject, refer to "Predict, diagnose, and treat chronic kidney disease with machine learning: a systematic literature review" by Sanmarchi et al., (2023).

Objectives

Analyze the dataset using two classification algorithms, focusing on exploratory data analysis, feature selection, engineering, and especially on handling missing values and outliers. Summarize your findings with insightful conclusions.

Classifier Requirement: Ensure at least one of the classifiers is interpretable, to facilitate in-depth analysis and inference.

Guidelines

- **Teamwork:** Group submissions should compile the workflow (Python codes and interpretations) into a single PDF, including a GitHub repository link. The contributions listed should reflect the GitHub activity.
- **Content:** Address the following questions in your submission, offering detailed insights and conclusions from your analysis.

Assignment Questions

1. **Classification Problem Identification:** Define and describe a classification problem based on the dataset.
2. **Variable Transformation:** Implement any transformations chosen or justify the absence of such modifications.
3. **Dataset Overview:** Provide a detailed description of the dataset, covering variables, summaries, observation counts, data types, and distributions (at least three statements).

4. **Association Between Variables:** Analyze variable relationships and their implications for feature selection or extraction (at least three statements).
5. **Missing Value Analysis and Handling:** Implement your strategy for identifying and addressing missing values in the dataset, or provide reasons for not addressing them.
6. **Outlier Analysis:** Implement your approach for identifying and managing outliers, or provide reasons for not addressing them.
7. **Sub-group Analysis:** Explore potential sub-groups within the data, employing appropriate data science methods to find the sub-groups of patients and visualize the sub-groups. The sub-group analysis must not include the labels (for CKD patients and healthy controls).
8. **Data Splitting:** Segregate 30% of the data for testing, using a random seed of 1. Use the remaining 70% for training and model selection.
9. **Classifier Choices:** Identify the two classifiers you have chosen and justify your selections.
10. **Performance Metrics:** Outline the two metrics for comparing the performance of the classifiers.
11. **Feature Selection/Extraction:** Implement methods to enhance the performance of at least one classifier in (9). The answer for this question can be included in (12).
12. **Classifier Comparison:** Utilize the selected metrics to compare the classifiers based on the test set. Discuss your findings (at least two statements).
13. **Interpretable Classifier Insight:** After re-training the interpretable classifier with all available data, analyze and interpret the significance of predictor variables in the context of the data and the challenge (at least two statements).
14. [**Bonus**] Sub-group Improvement Strategy: If sub-groups were identified, propose and implement a method to improve one classifier performance further. Compare the performance of the new classifer with the results in (12).
15. **Team Contributions:** Document each team member's specific contributions related to the questions above.
16. **Link** to the public GitHub repository.

Notes

- This assignment encourages you to apply sophisticated machine learning methods to a vital healthcare challenge, promoting the development of critical analytical skills, teamwork, and

practical problem-solving abilities in the context of chronic kidney disease diagnosis and treatment.

- Students can choose one classifer not covered in the lectures.

Solution

Import dataset:

(Rubini and Eswaran 2015)

```
import pandas as pd
from scipy.io.arff import loadarff
from scipy.io import arff
```

```
from sklearn import neighbors
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
```

```
data = []
with open('chronic_kidney_disease_full.arff') as f:
    for line in f:
        line = line.replace('\n', '')
        data.append(line.split(','))



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

df = pd.DataFrame(data[145:-2], columns=names)
```

```
df
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	wbcc	rbcc	htn	dr
0	48	80	1.020	1	0	?	normal	notpresent	notpresent	121	...	7800	5.2	yes	ye
1	7	50	1.020	4	0	?	normal	notpresent	notpresent	?	...	6000	?	no	no
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent	423	...	7500	?	no	ye
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	...	6700	3.9	yes	no
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent	106	...	7300	4.6	no	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	55	80	1.020	0	0	normal	normal	notpresent	notpresent	140	...	6700	4.9	no	no
396	42	70	1.025	0	0	normal	normal	notpresent	notpresent	75	...	7800	6.2	no	no
397	12	80	1.020	0	0	normal	normal	notpresent	notpresent	100	...	6600	5.4	no	no
398	17	60	1.025	0	0	normal	normal	notpresent	notpresent	114	...	7200	5.9	no	no
399	58	80	1.025	0	0	normal	normal	notpresent	notpresent	131	...	6800	6.1	no	no

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

Q1

The dataset includes medical records with a label indicating the presence or absence of chronic kidney disease (CKD). The objective of our classification problem is to develop a predictive model that accurately identifies whether a patient has CKD, based on a set of features present in the dataset. This classification model aims to assist in the early detection of chronic kidney disease.

Q2

```
for col in df.columns:
    if df[col].dtype == 'object' and any(df[col].str.contains(r'\d', na=False)):
        df[col] = pd.to_numeric(df[col], errors='coerce')



tr = {
    'rbc' : {'normal': 1, 'abnormal':0},
    'pc' : {'normal': 1, 'abnormal':0},
    'pcc' : {'present': 1, 'notpresent':0, 'no':0, 'yes': 1},
    'ba' : {'present': 1, 'notpresent':0, 'no':0, 'yes': 1},
    'htn' : {'yes': 1, 'no':0},
    'dm' : {'yes': 1, 'no':0},
    'cad' : {'yes': 1, 'no':0},
    'pe' : {'yes': 1, 'no':0, 'good': 1},
    'ane' : {'yes': 1, 'no':0},
    'appet' : {'good': 1, 'poor':0, 'no':0, 'yes': 1},
    'class' : {'ckd':1, 'notckd':0}

}

for col, tr in tr.items():
    df[col] = df[col].replace(tr)
```

```
C:\Users\Rick\AppData\Local\Temp\ipykernel_41836\2639493903.py:22: FutureWarning: Downcasting
  df[col] = df[col].replace(tr)
```

We transformed categorical variables into numerical values.

Q3

```
features = df.shape[1]
observations = df.shape[0]
print("\n numbers of features: ", features)
print("\n numbers of observations: ", observations)
```

```
 numbers of features:  26

 numbers of observations:  400
```

(1).There are 25 features and 400 observations.

```
types = df.dtypes
print(types)
```

```
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        int64
no_name     object
dtype: object
```

(2).There are 14 categorical variables with dtype of 'object': Specific Gravity, Albumin, Sugar, Red Blood Cells, Pus Cell, Pus Cell Clumps, Bacteria, Hypertension, Diabetes Mellitus, Coronary Artery Disease, Appetite, Pedal Edema, Anemia, Class.

(3).There are 11 numerical variables with dtype of 'float64': Age, Blood Pressure, Blood Glucose Random, Blood Urea, Serum Creatinine, Sodium, Potassium, Hemoglobin, Packed Cell Volume, White Blood Cell Count, Red Blood Cell Count.

```
summary = df.describe()
print(summary)
```

```
              age          bp          sg          al          su          bgr  \
count  391.000000  388.000000  353.000000  354.000000  351.000000  356.000000
mean    51.483376   76.469072    1.017408    1.016949    0.450142  148.036517
std     17.169714   13.683637    0.005717    1.352679    1.099191   79.281714
min      2.000000   50.000000    1.005000    0.000000    0.000000   22.000000
25%     42.000000   70.000000    1.010000    0.000000    0.000000   99.000000
50%     55.000000   80.000000    1.020000    0.000000    0.000000  121.000000
75%     64.500000   80.000000    1.020000    2.000000    0.000000  163.000000
max     90.000000  180.000000    1.025000    5.000000    5.000000  490.000000

               bu          sc         sod         pot        hemo         pcv  \
```

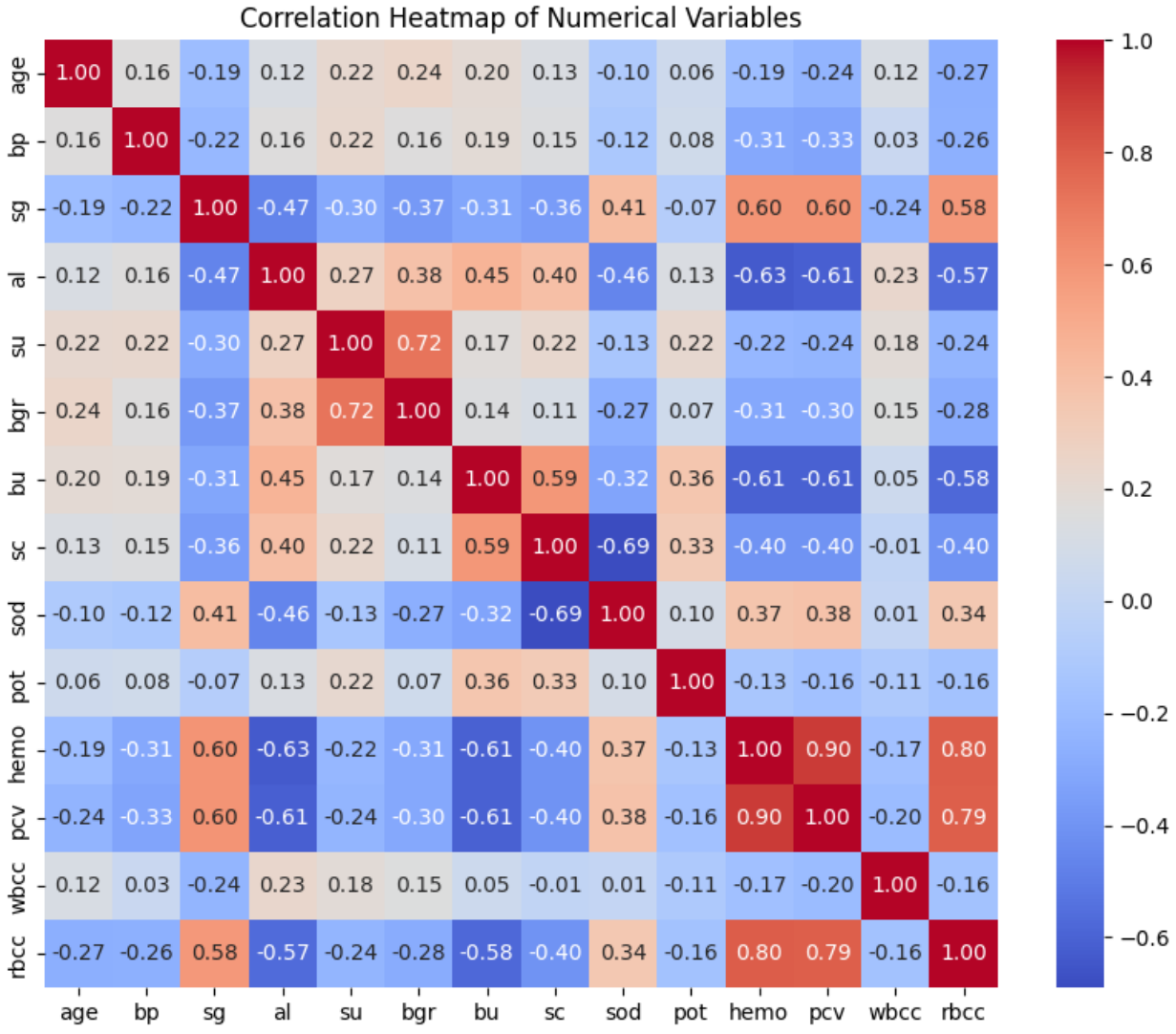
count	381.000000	383.000000	313.000000	312.000000	348.000000	329.000000
mean	57.425722	3.072454	137.528754	4.627244	12.526437	38.884498
std	50.503006	5.741126	10.408752	3.193904	2.912587	8.990105
min	1.500000	0.400000	4.500000	2.500000	3.100000	9.000000
25%	27.000000	0.900000	135.000000	3.800000	10.300000	32.000000
50%	42.000000	1.300000	138.000000	4.400000	12.650000	40.000000
75%	66.000000	2.800000	142.000000	4.900000	15.000000	45.000000
max	391.000000	76.000000	163.000000	47.000000	17.800000	54.000000

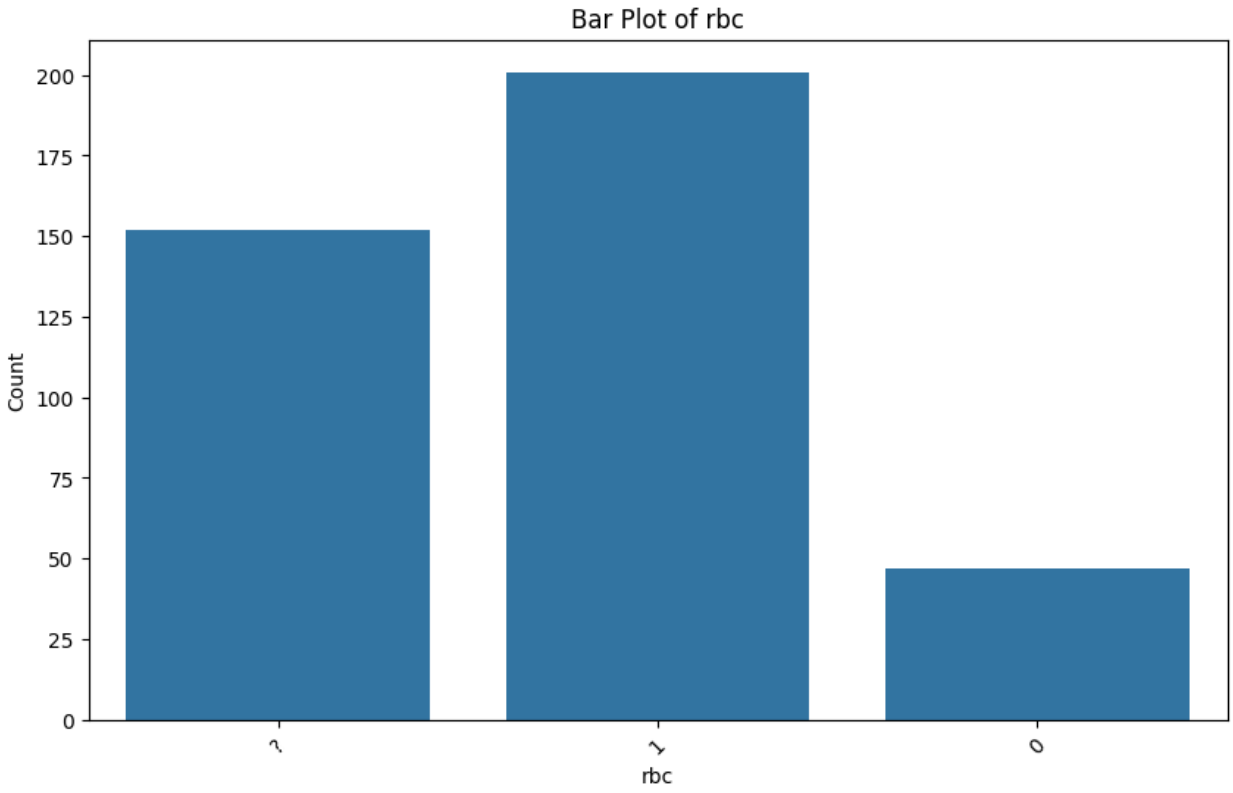
	wbcc	rbcc	class
count	294.000000	269.000000	400.000000
mean	8406.122449	4.707435	0.625000
std	2944.474190	1.025323	0.484729
min	2200.000000	2.100000	0.000000
25%	6500.000000	3.900000	0.000000
50%	8000.000000	4.800000	1.000000
75%	9800.000000	5.400000	1.000000
max	26400.000000	8.000000	1.000000

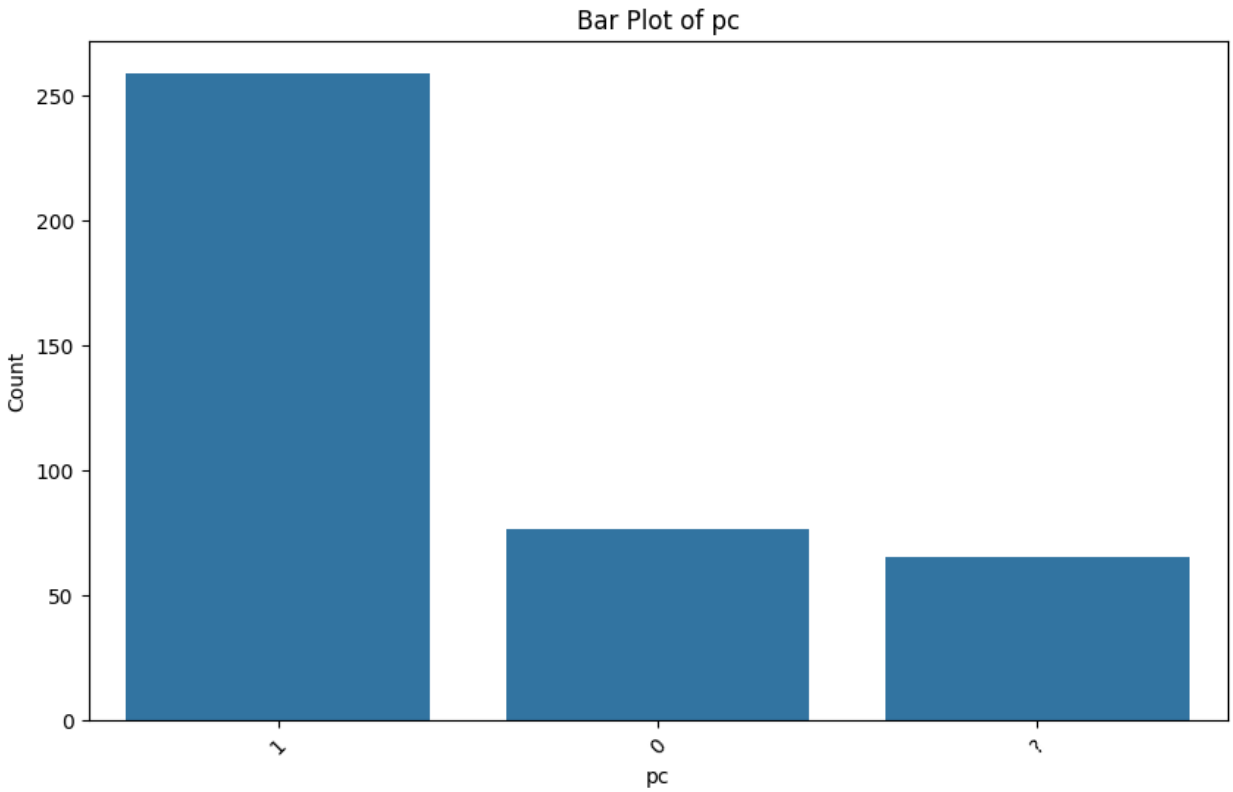
Q4

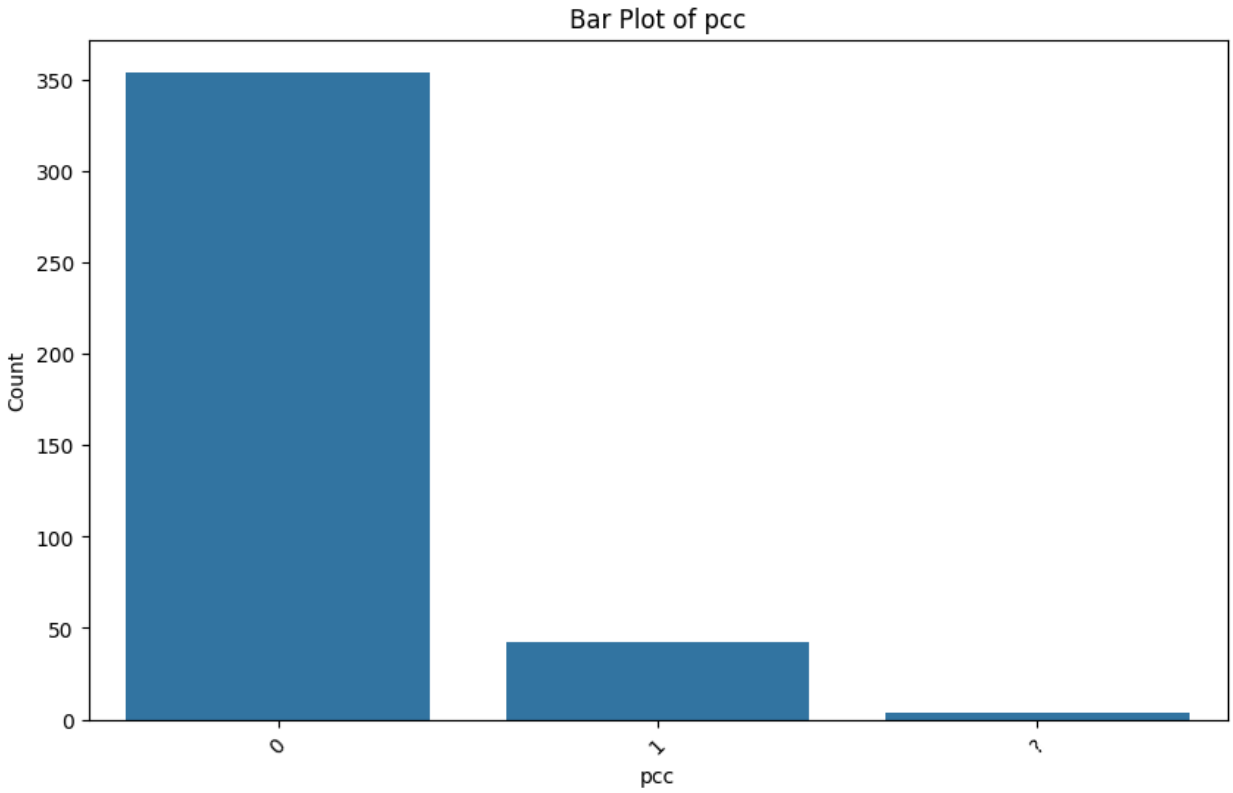
```
numerical_columns = df.select_dtypes(include=['float64'])

plt.figure(figsize=(10, 8))
sns.heatmap(numerical_columns.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Variables')
plt.show()

categorical_columns = df.select_dtypes(include=['object']).drop(columns=['no_name'])

for col in categorical_columns.columns:
```

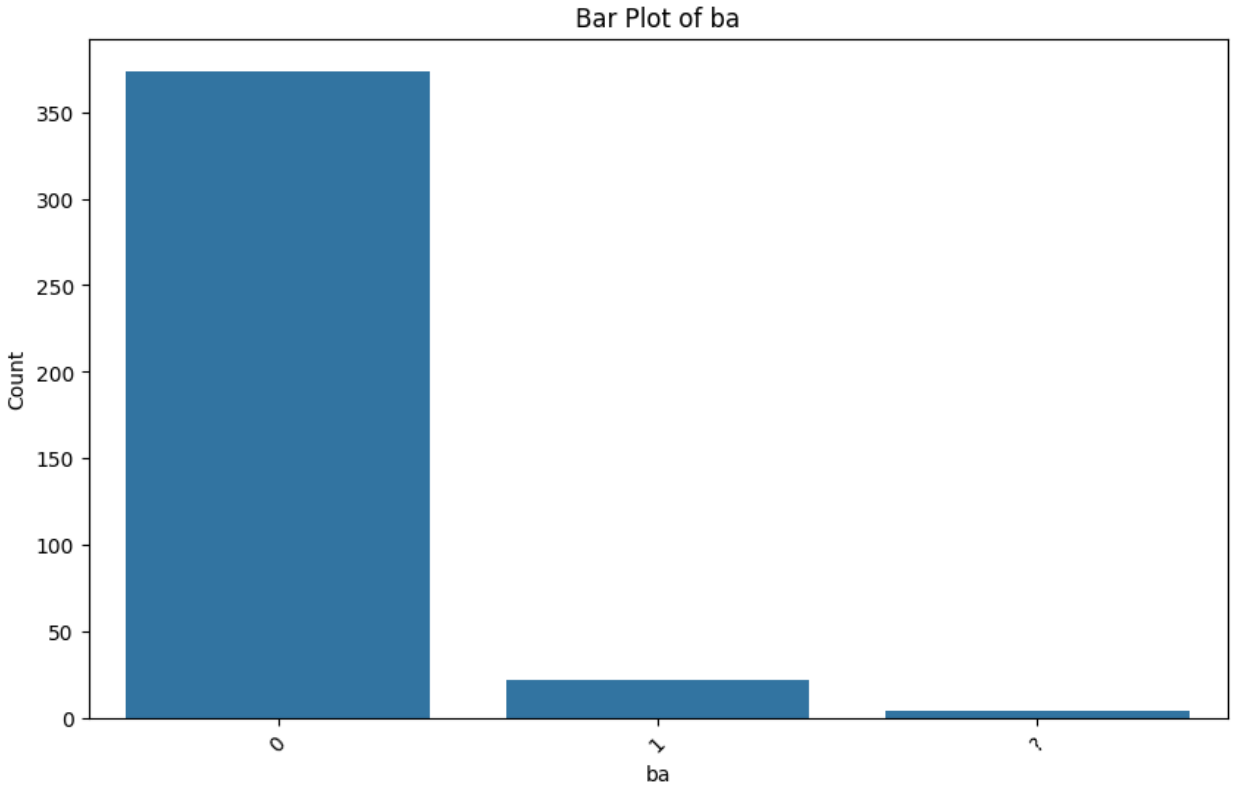
```
    plt.figure(figsize=(10, 6))
    sns.countplot(x=col, data=df)
    plt.title(f'Bar Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45)
    plt.show()
```

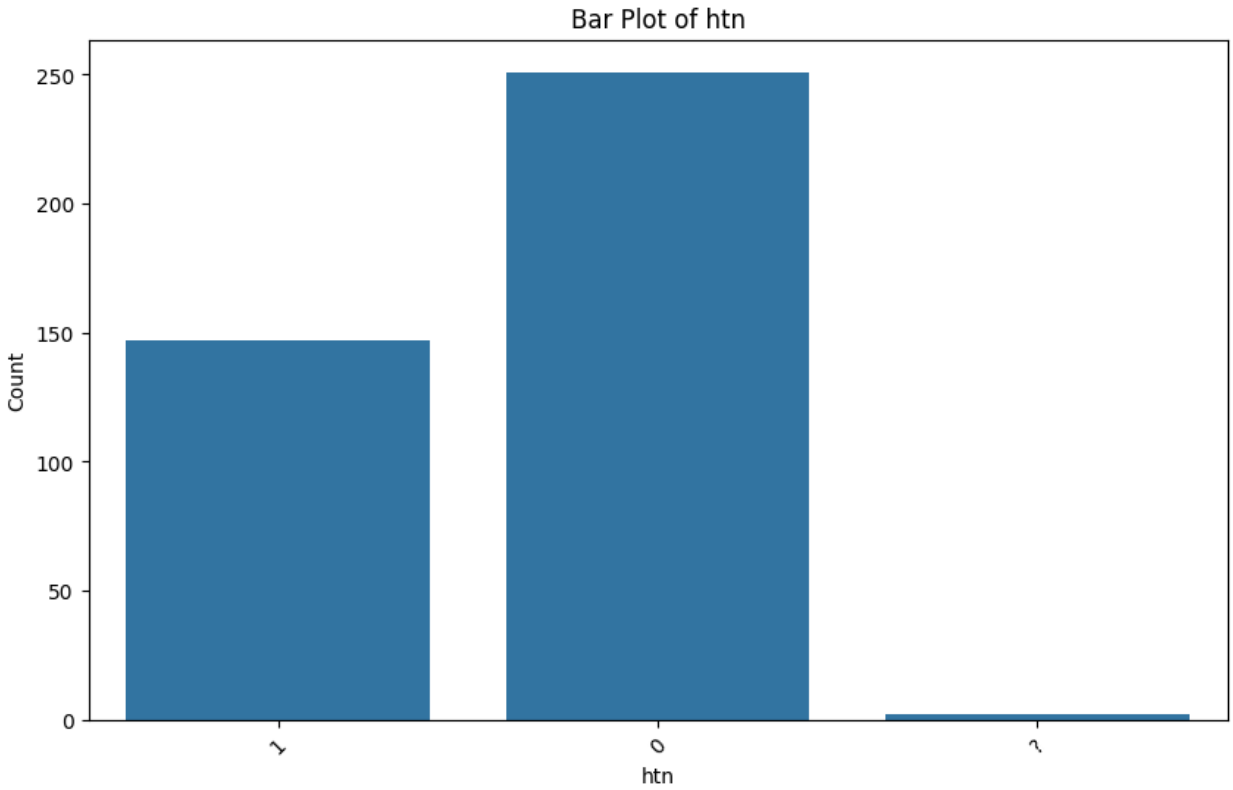


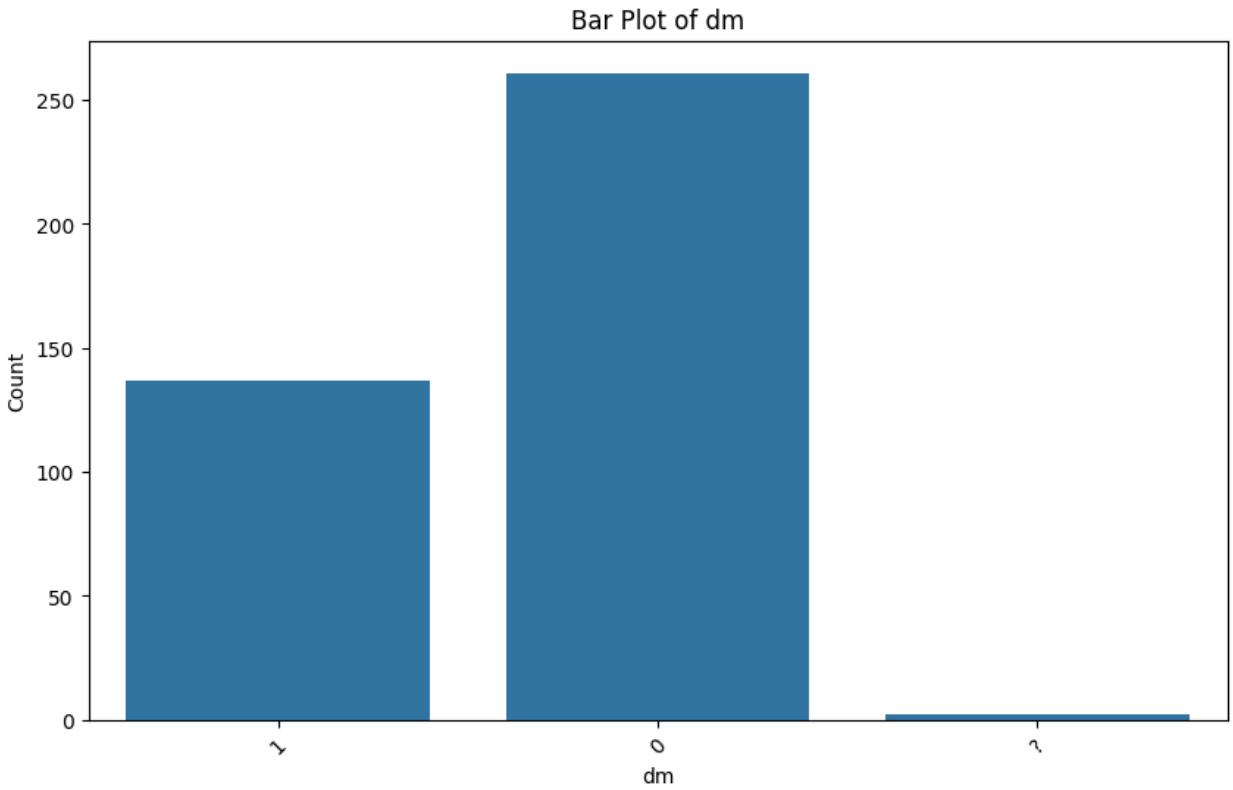

Bar Plot of rbc
Count
200
175
150
125
100
75
50
25
0
?
1
0
rbc

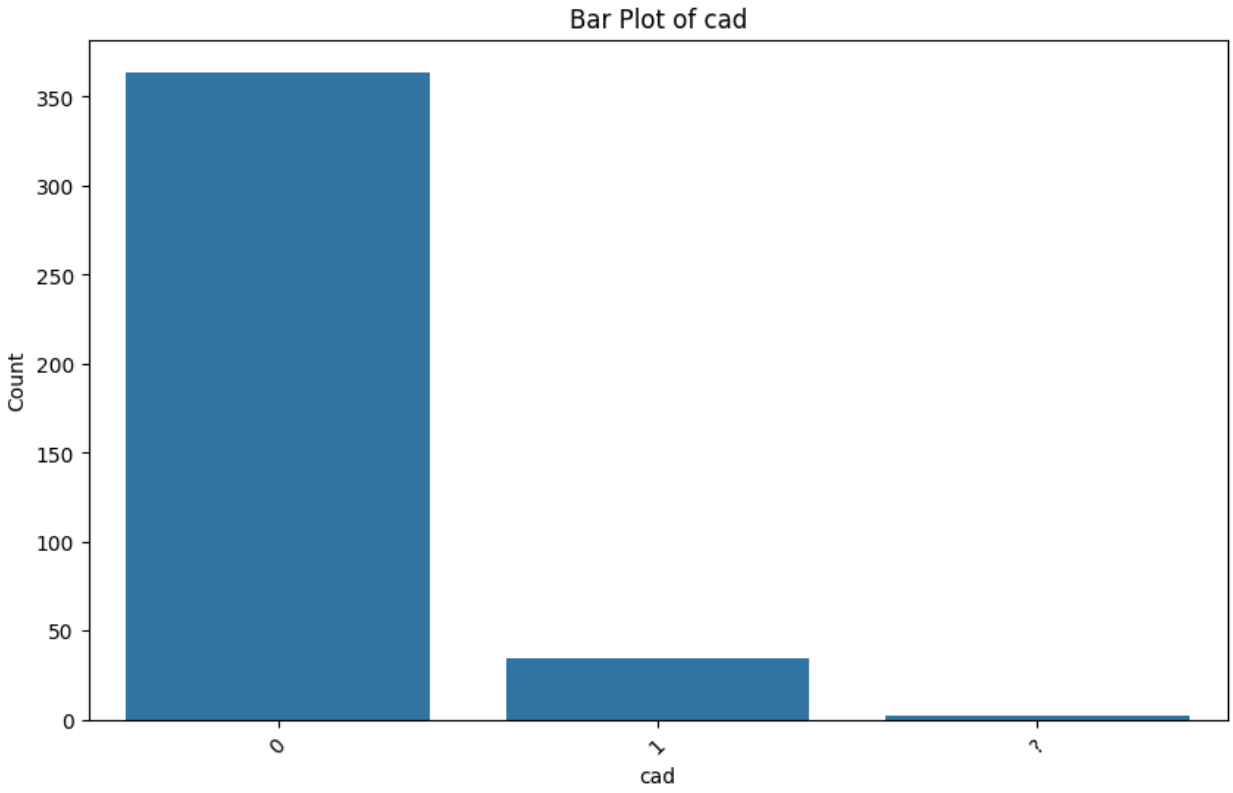

Bar Plot of pc
Count
250
200
150
100
50
0
1
0
?
pc

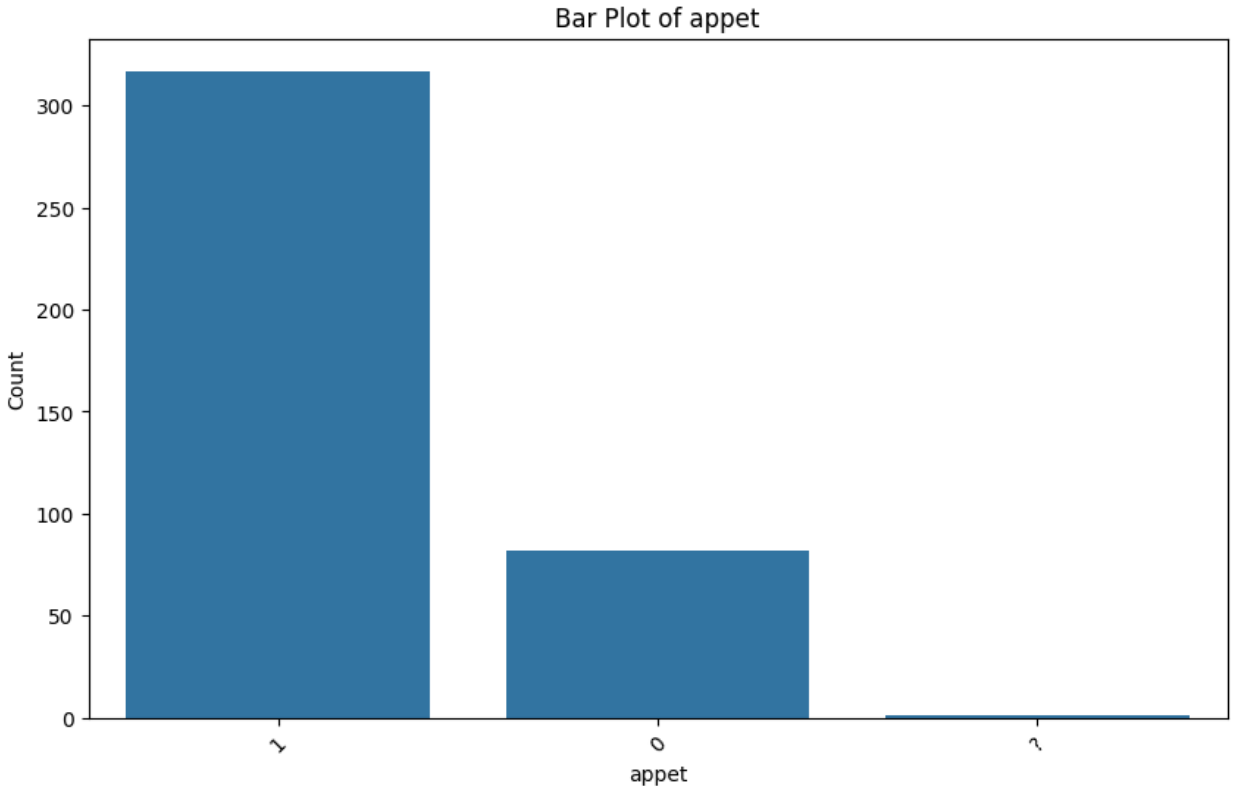

Bar Plot of pcc
Count
350
300
250
200
150
100
50
0
0
1
?
pcc

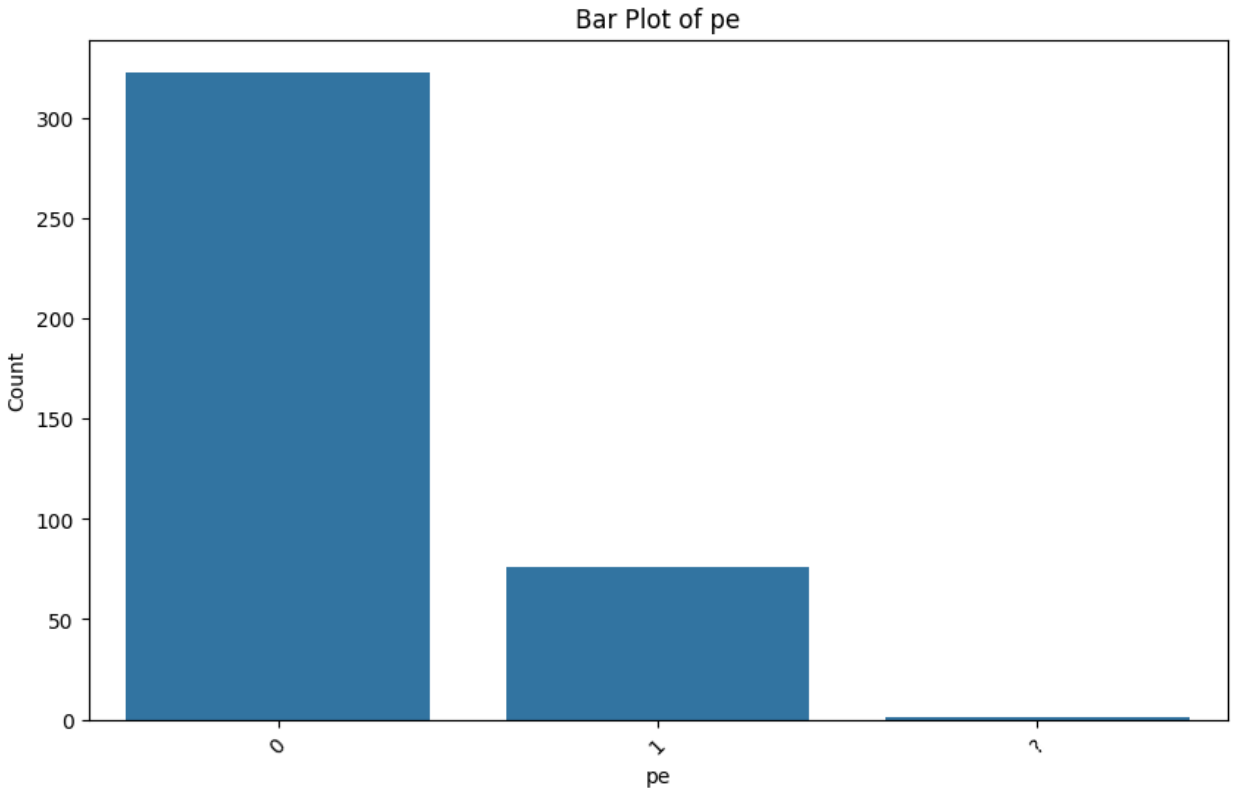

Bar Plot of ba
Count
350
300
250
200
150
100
50
0
0
1
?
ba

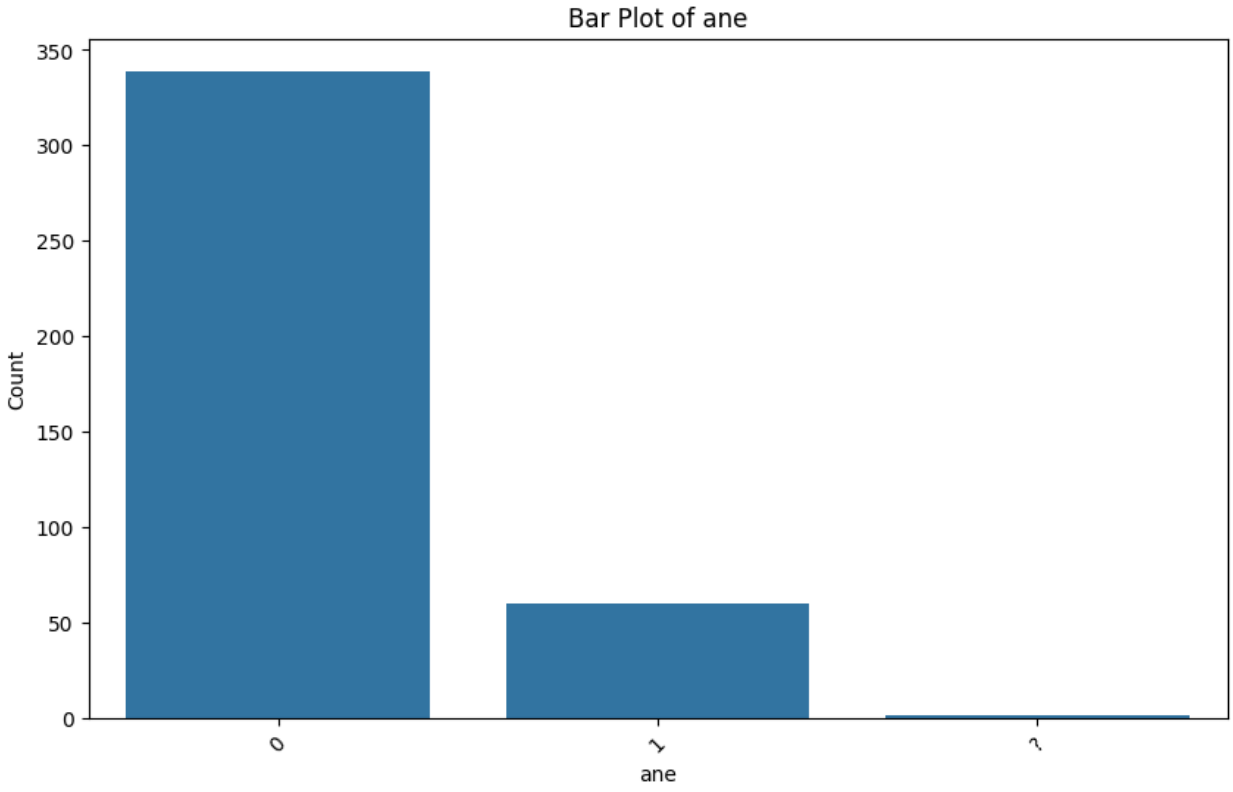

Bar Plot of htn
Count
250
200
150
100
50
0
1
0
?
htn


Bar Plot of dm
Count
250
200
150
100
50
0
1
0
?
dm


Bar Plot of cad
Count
0
50
100
150
200
250
300
350
0
1
?
cad


Bar Plot of appet
Count
0
50
100
150
200
250
300
1
0
?
appet


Bar Plot of pe
Count
300
250
200
150
100
50
0
0
1
?
pe


Bar Plot of ane
Count
350
300
250
200
150
100
50
0
0
1
?
ane

Commemts:

(1) `hemo (hemoglobin)` has a highest positive correlation(0.90) with `pcv (packed cell volume)`, meaning that when hemoglobin is high in the body, the packed cell volume is also high.
(2) `sc (serum creatinine)` has the highest negative correlation(-0.69) with `sod (sodium)`, which means with high hemoglobin in the body, the packed cell volume is low.
(3) From Bar Plot, we see that there are almost 150 missing values in the variable `rbc(Red Blood Cells)`, so we would like to drop this variable.

Q5

```
missing_values = df.isnull().sum()
missing_values
```

```
age          9
bp          12
sg          47
al          46
su          49
rbc          0
pc           0
pcc          0
ba           0
bgr         44
bu          19
sc          17
sod         87
pot         88
hemo        52
pcv         71
wbcc       106
rbcc       131
htn          0
dm           0
```

```
cad           0
appet         0
pe            0
ane           0
class         0
no_name     398
dtype: int64
```

```
total_missing_value = df.isnull().sum().sum()
total_missing_value
```

```
1176
```

```
drop_numerical_columns = missing_values[missing_values > 50].index
df.drop(columns = drop_numerical_columns, inplace = True)
df.drop(columns=['rbc', 'pc'], inplace = True)
df
```

	age	bp	sg	al	su	pcc	ba	bgr	bu	sc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1.0	0.0	0	0	121.0	36.0	1.2	1	1	0	1	0	0	1
1	7.0	50.0	1.020	4.0	0.0	0	0	NaN	18.0	0.8	0	0	0	1	0	0	1
2	62.0	80.0	1.010	2.0	3.0	0	0	423.0	53.0	1.8	0	1	0	0	0	1	1
3	48.0	70.0	1.005	4.0	0.0	1	0	117.0	56.0	3.8	1	0	0	0	1	1	1
4	51.0	80.0	1.010	2.0	0.0	0	0	106.0	26.0	1.4	0	0	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	55.0	80.0	1.020	0.0	0.0	0	0	140.0	49.0	0.5	0	0	0	1	0	0	0
396	42.0	70.0	1.025	0.0	0.0	0	0	75.0	31.0	1.2	0	0	0	1	0	0	0
397	12.0	80.0	1.020	0.0	0.0	0	0	100.0	26.0	0.6	0	0	0	1	0	0	0
398	17.0	60.0	1.025	0.0	0.0	0	0	114.0	50.0	1.0	0	0	0	1	0	0	0
399	58.0	80.0	1.025	0.0	0.0	0	0	131.0	18.0	1.1	0	0	0	1	0	0	0

```
df = df.replace('?', float('nan'))
missing_values = df.isnull().sum()
missing_values
```

```
C:\Users\Rick\AppData\Local\Temp\ipykernel_41836\779703511.py:1: FutureWarning: Downcasting bel
  df = df.replace('?', float('nan'))
```

```
age       9
bp       12
sg       47
al       46
su       49
pcc       4
ba        4
bgr      44
bu       19
sc       17
htn       2
dm        2
cad       2
appet     1
pe        1
ane       1
class     0
dtype: int64
```

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

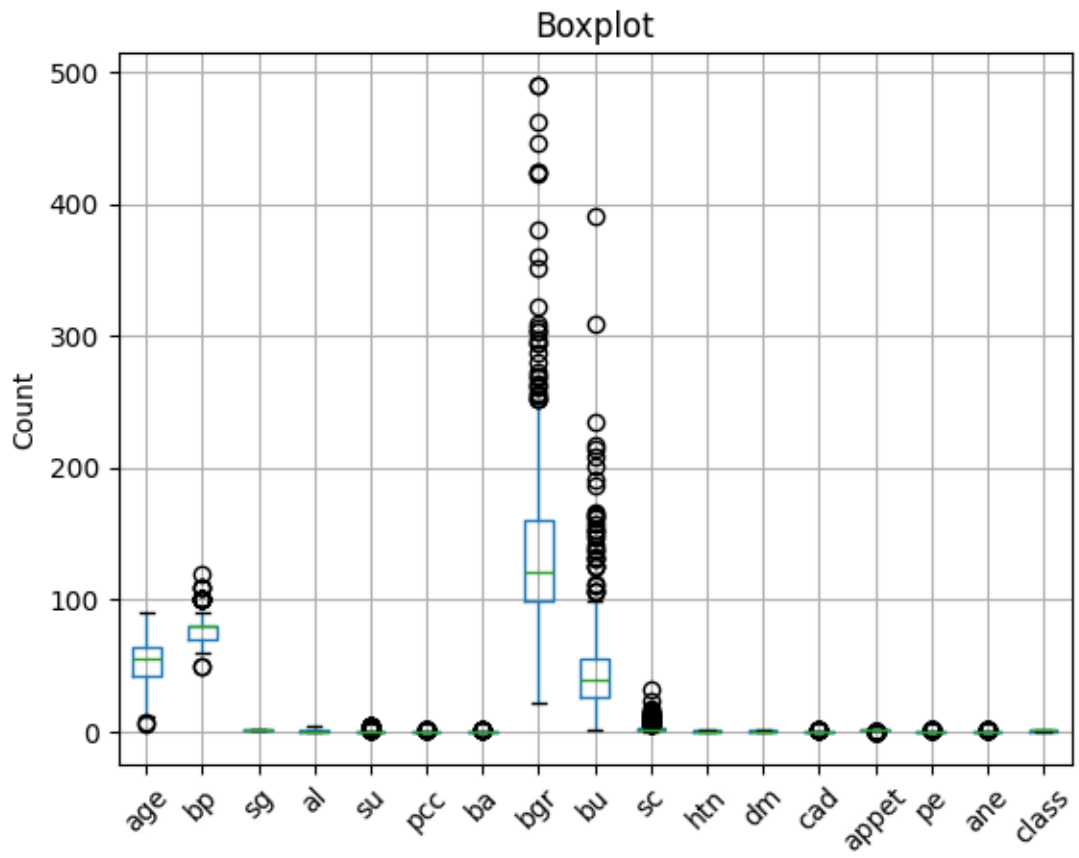
```
missing_values = df.isnull().sum()
missing_values
```

```
age      0
```

```
bp       0
sg       0
al       0
su       0
pcc      0
ba       0
bgr      0
bu       0
sc       0
htn      0
dm       0
cad      0
appet    0
pe       0
ane      0
class    0
dtype: int64
```

Q6

```
df.boxplot()


plt.xticks(rotation=45)
plt.title('Boxplot')
plt.ylabel('Count')


plt.show()
```




```
from scipy.stats import boxcox


selected_features = ['age', 'bp', 'bu', 'sc']


X_to_transform = df[selected_features]


for feature in selected_features:
    X_to_transform[feature], _ = boxcox(abs(X_to_transform[feature]))


for feature in selected_features:
    df[feature] = X_to_transform[feature]
```

```
C:\Users\Rick\AppData\Local\Temp\ipykernel_41836\1488133554.py:11: 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/i
  X_to_transform[feature], _ = boxcox(abs(X_to_transform[feature]))
```

```
df.dtypes
```

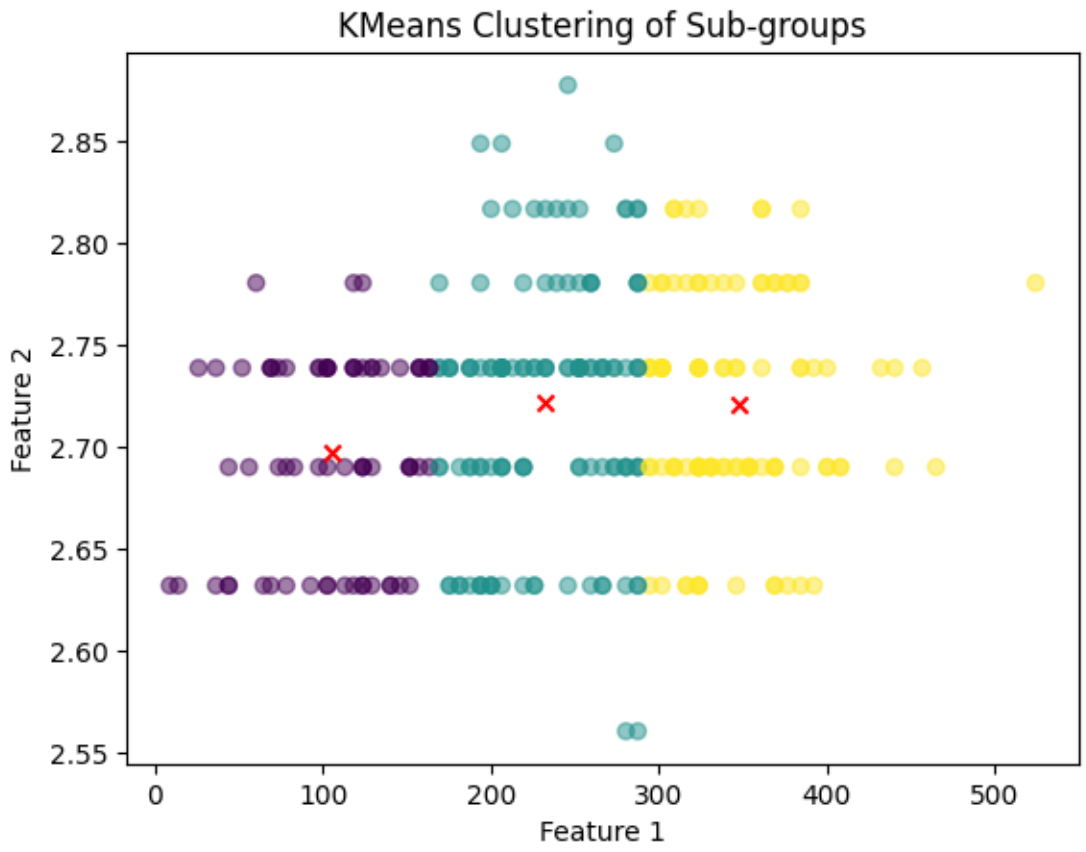
```
age      float64
bp       float64
sg       float64
al       float64
su       float64
pcc      float64
ba       float64
bgr      float64
bu       float64
sc       float64
htn      float64
dm       float64
cad      float64
appet    float64
pe       float64
ane      float64
class      int64
dtype: object
```

Outliers in chronic kidney disease data may represent crucial medical cases. If we removing them might omit significant clinical insights and reduce the model's ability to reflect real-world variability. So, we decided to use Box-Cox transformation to adjust them.

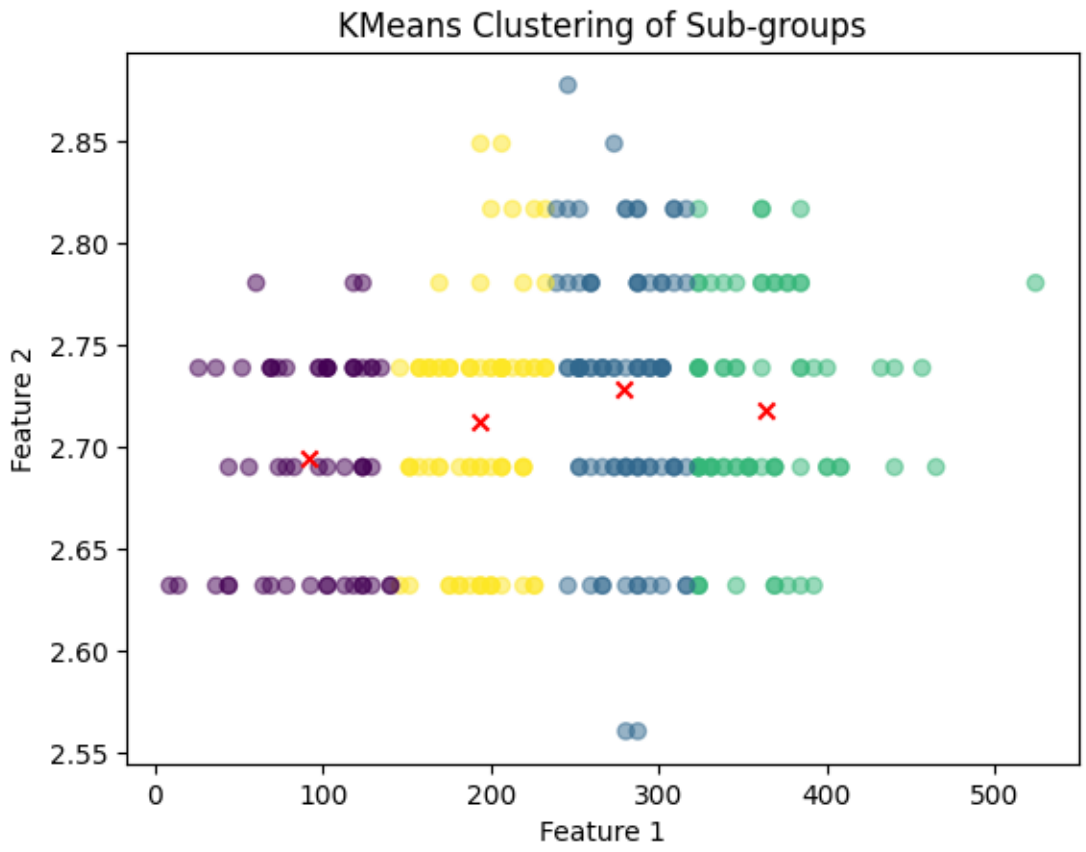
Q7

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder



X_numeric = df[['age', 'bp', 'bu', 'sc']]

X_categorical = df[['pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane']]
encoder = LabelEncoder()
X_categorical_encoded = X_categorical.apply(encoder.fit_transform)

X_processed = pd.concat([X_numeric, X_categorical_encoded], axis=1)

k = 2

kmeans = KMeans(n_clusters=k, random_state=1)
kmeans.fit(X_processed)

cluster_labels = kmeans.labels_

plt.scatter(X_processed.iloc[:, 0], X_processed.iloc[:, 1], c=cluster_labels, cmap='viridis', a
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', marker='x')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans Clustering of Sub-groups')
plt.show()
```

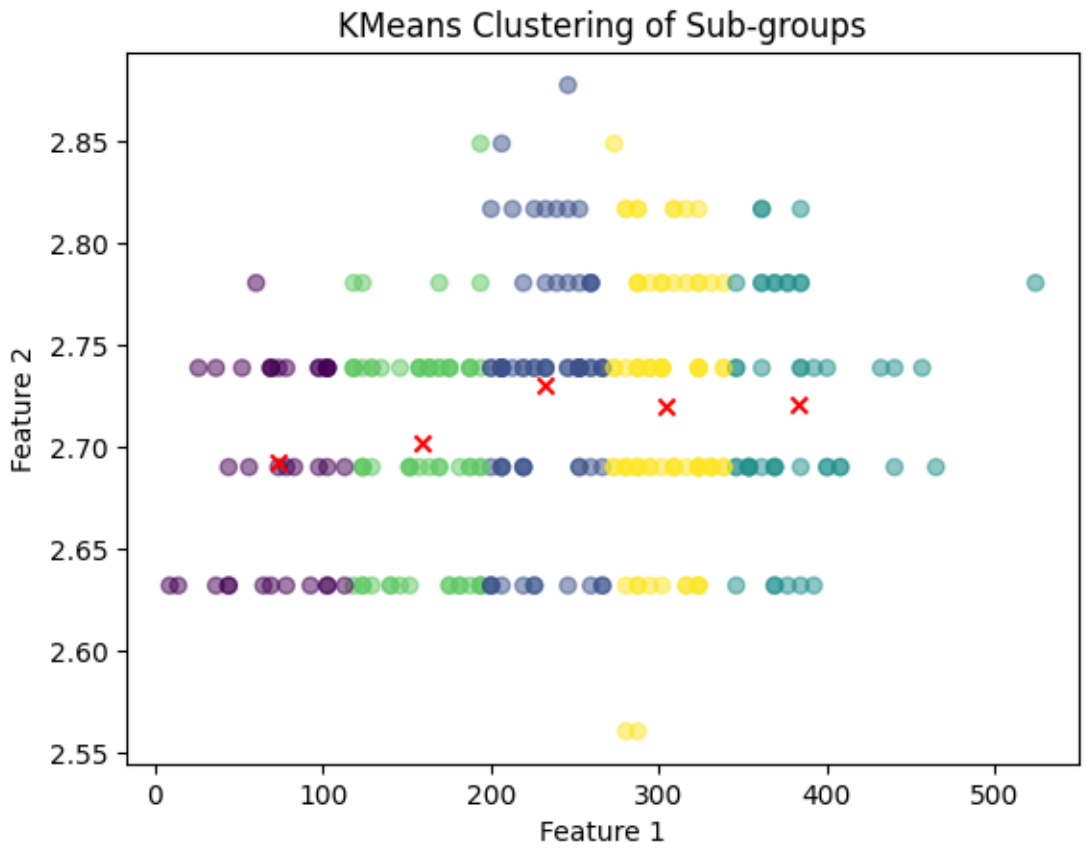

```
k = 3


kmeans = KMeans(n_clusters=k, random_state=1)
kmeans.fit(X_processed)


cluster_labels = kmeans.labels_


plt.scatter(X_processed.iloc[:, 0], X_processed.iloc[:, 1], c=cluster_labels, cmap='viridis', a
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', marker='x')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans Clustering of Sub-groups')
plt.show()
```




```
k = 4


kmeans = KMeans(n_clusters=k, random_state=1)
kmeans.fit(X_processed)


cluster_labels = kmeans.labels_


plt.scatter(X_processed.iloc[:, 0], X_processed.iloc[:, 1], c=cluster_labels, cmap='viridis', a
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', marker='x')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans Clustering of Sub-groups')
plt.show()
```




```
k = 5


kmeans = KMeans(n_clusters=k, random_state=1)
kmeans.fit(X_processed)


cluster_labels = kmeans.labels_


plt.scatter(X_processed.iloc[:, 0], X_processed.iloc[:, 1], c=cluster_labels, cmap='viridis', a
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', marker='x')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans Clustering of Sub-groups')
plt.show()
```

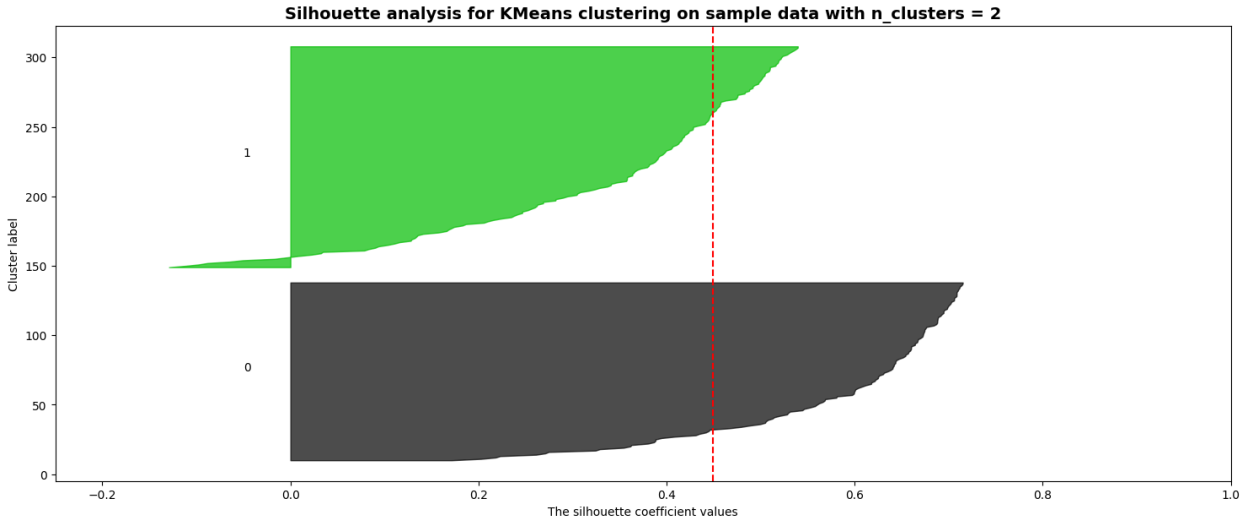


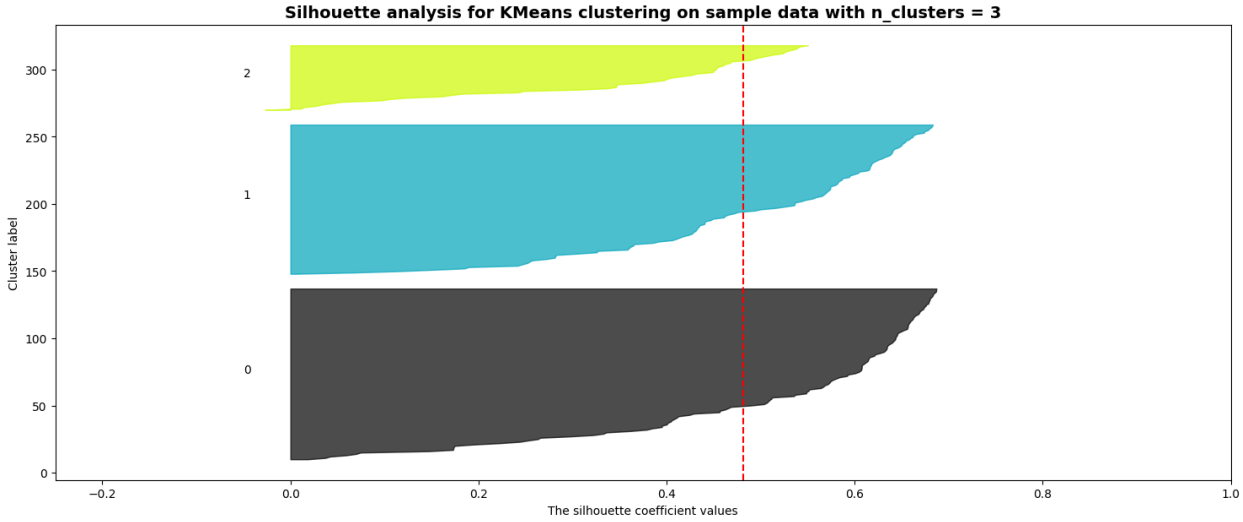

```
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.cluster import KMeans
from scipy.cluster import hierarchy
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.metrics.cluster import rand_score
```

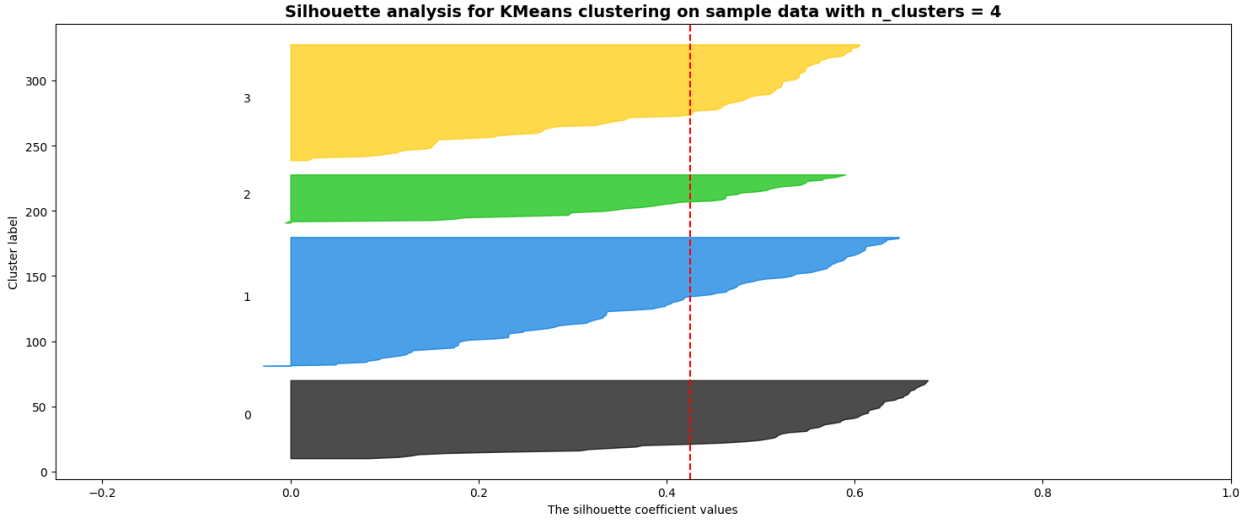
```
x=df
range_n_clusters = [2, 3, 4, 5]
for n_clusters in range_n_clusters:
    km = KMeans(n_clusters = n_clusters, n_init = 20, random_state=1)
    cluster_labels_km = km.fit_predict(x)
    silhouette_avg_km = silhouette_score(x, cluster_labels_km)
    # Compute the silhouette scores for each sample
```

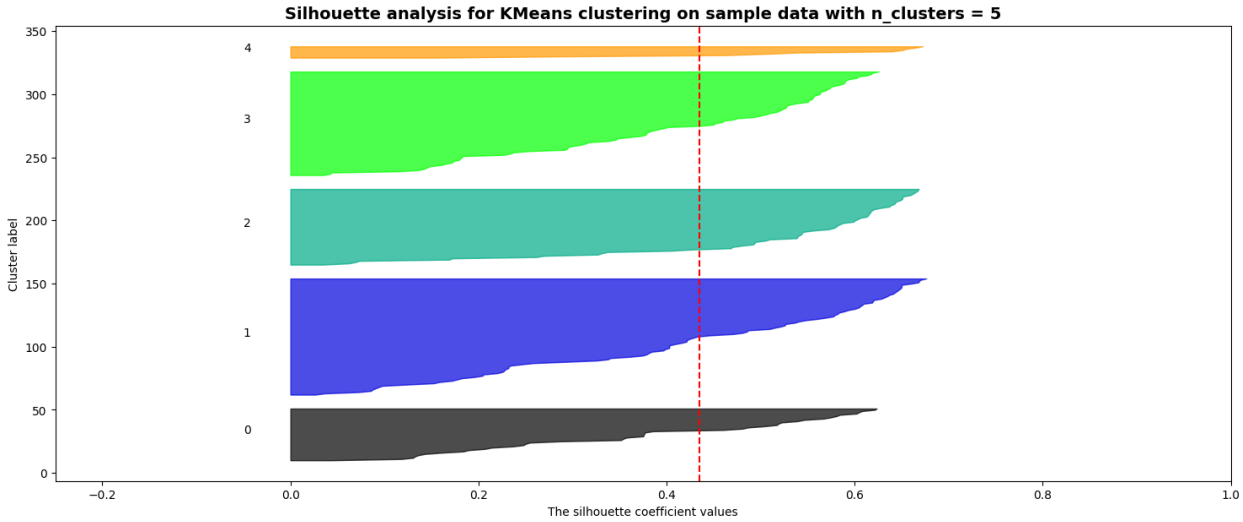
```
    sample_silhouette_values = silhouette_samples(x, cluster_labels_km)
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.25, 1])# change this based on the silhouette range

    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(
            np.arange(y_lower, y_upper),
            0,
            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 plot
        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")

    # The 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",
    )
```









The plot with 2 clusters shows good silhouette scores and shape, so we choose K=2.

Q8

```
x = df.drop(['class'], axis=1)
y = df['class']
```

```
x_train, x_test, y_train, y_test = train_test_split(
    x,
    y,
    test_size=0.3,
    random_state=1,
    stratify=y
    )
```

Q9

We have selected Decision Trees and Logistic Regression as our classifiers. Decision Trees provide a highly interpretable model structure that can easily capture non-linear patterns. They are also robust to outliers and can handle both numerical and categorical data effectively. Logistic Regression is chosen for its efficiency and simplicity, especially when dealing with binary classification problems. It performs well when the decision boundary is linear and can provide probabilities for outcomes.

Q10

(1) Accuracy

Accuracy is the simplest and most direct metric in the classification problem and is a balanced evaluation of the test results. For a given test dataset, the ratio of the number of samples correctly classified by the classifier to the total number of samples. In general, the higher the correct rate, the better the classifier.

(2) F1 Score

F1 Score is a metric used to measure the precision of a binary classification model, it is a weighted harmonic mean of precision and recall.

Q11

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression



dt_model = DecisionTreeClassifier(random_state=1)
dt_model.fit(x_train, y_train)
```

```
DecisionTreeClassifier(random_state=1)
```

```
from sklearn.feature_selection import SequentialFeatureSelector

glm_model = LogisticRegression(random_state=1)

def select_features_sfs(logreg, x_train, y_train, num_features):

    sfs = SequentialFeatureSelector(estimator=logreg,
                                    n_features_to_select=num_features,
                                    direction='forward',
                                    scoring='accuracy',
                                    cv=5)



    sfs.fit(x_train, y_train)



    sel_features_train = x_train.columns[sfs.support_]

    return sel_features_train
```

```
selected_features_train = select_features_sfs(glm_model, x_train, y_train, num_features=5)

glm_model.fit(x_train[selected_features_train], y_train)
```

```
LogisticRegression(random_state=1)
```

Q12

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
```

Decision Tree:

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

```
k = 2
kmeans = KMeans(n_clusters=k, random_state=1)
kmeans.fit(X_processed)

cluster_labels = kmeans.labels_

X_processed_df = pd.DataFrame(X_processed, columns=x.columns)
X_processed_df['Cluster_Labels'] = cluster_labels

X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_processed_df, y, test_size=0.3,
```

```
df.fit(X_train_d, y_train_d)
```

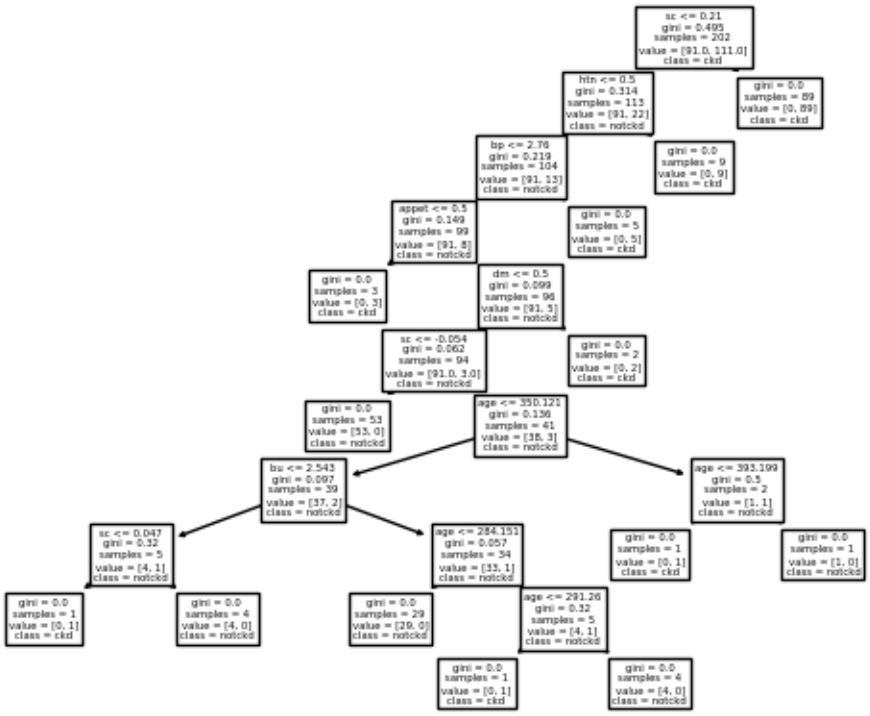
```
DecisionTreeClassifier(max_depth=30, random_state=1)
```

```
plot_tree(
    df,
    max_depth= 30,
    feature_names = X_train_d.columns.tolist(),
    class_names=['notckd', 'ckd']
)

pred = df.predict(X_test_d)
pred[:5]

print(classification_report(y_test_d, pred))
cm = pd.DataFrame(confusion_matrix(y_test_d, pred), index=['No', 'Yes'], columns=['No', 'Yes'])
cm.index.name = 'True'
cm.columns.name = 'Predicted'
cm

df.score(X_test_d, y_test_d)

print(classification_report(y_test_d, pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96        39
           1       0.98      0.96      0.97        48

    accuracy                           0.97        87
   macro avg       0.96      0.97      0.97        87
```

```
weighted avg       0.97      0.97      0.97        87
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96        39
           1       0.98      0.96      0.97        48

    accuracy                           0.97        87
   macro avg       0.96      0.97      0.97        87
weighted avg       0.97      0.97      0.97        87
```



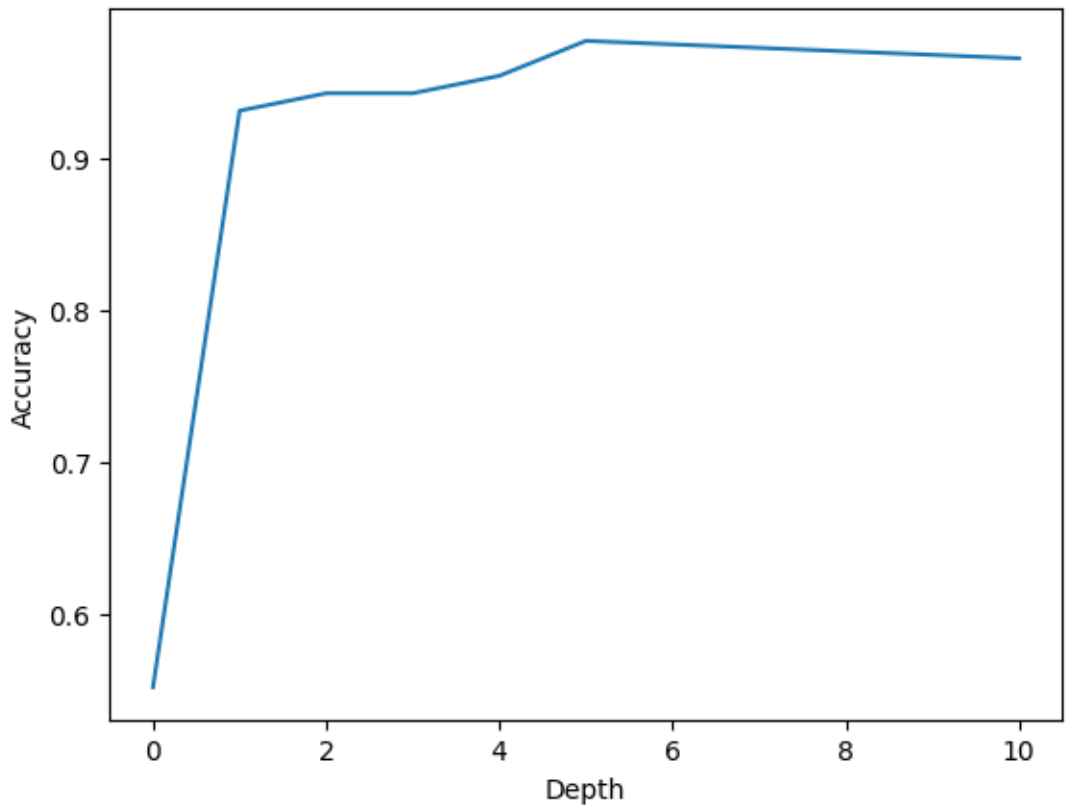

```
path = df.cost_complexity_pruning_path(
    X_train_d,
    y_train_d
)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

```
clfs = [] # save fitted trees with different alphas
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(
        random_state=1,
        ccp_alpha=ccp_alpha
        )
    clf.fit(X_train_d, y_train_d)
    clfs.append(clf)
```

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

```
[10, 5, 4, 3, 2, 1, 0]
```

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



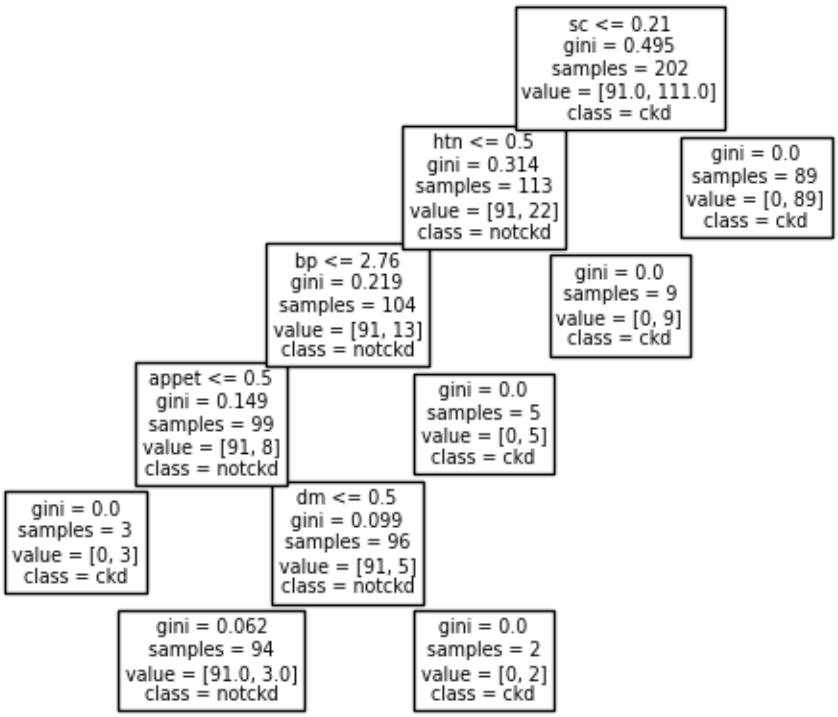

```
df_best = DecisionTreeClassifier(
    max_depth = 5,
    random_state=1
    )
df_best.fit(x, y)
```

```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

```
df_best.fit(X_train_d, y_train_d)


plot_tree(
    df_best,
    max_depth= 5,
    feature_names = X_train_d.columns.tolist(),
    class_names=['notckd', 'ckd']
)
```

```
pred = df_best.predict(X_test_d)
pred[:5]

print(classification_report(y_test_d, pred))
cm = pd.DataFrame(confusion_matrix(y_test_d, pred), index=['No', 'Yes'], columns=['No', 'Yes'])
cm.index.name = 'True'
cm.columns.name = 'Predicted'
cm

df_best.score(X_test_d, y_test_d)

print(classification_report(y_test_d, pred))
```

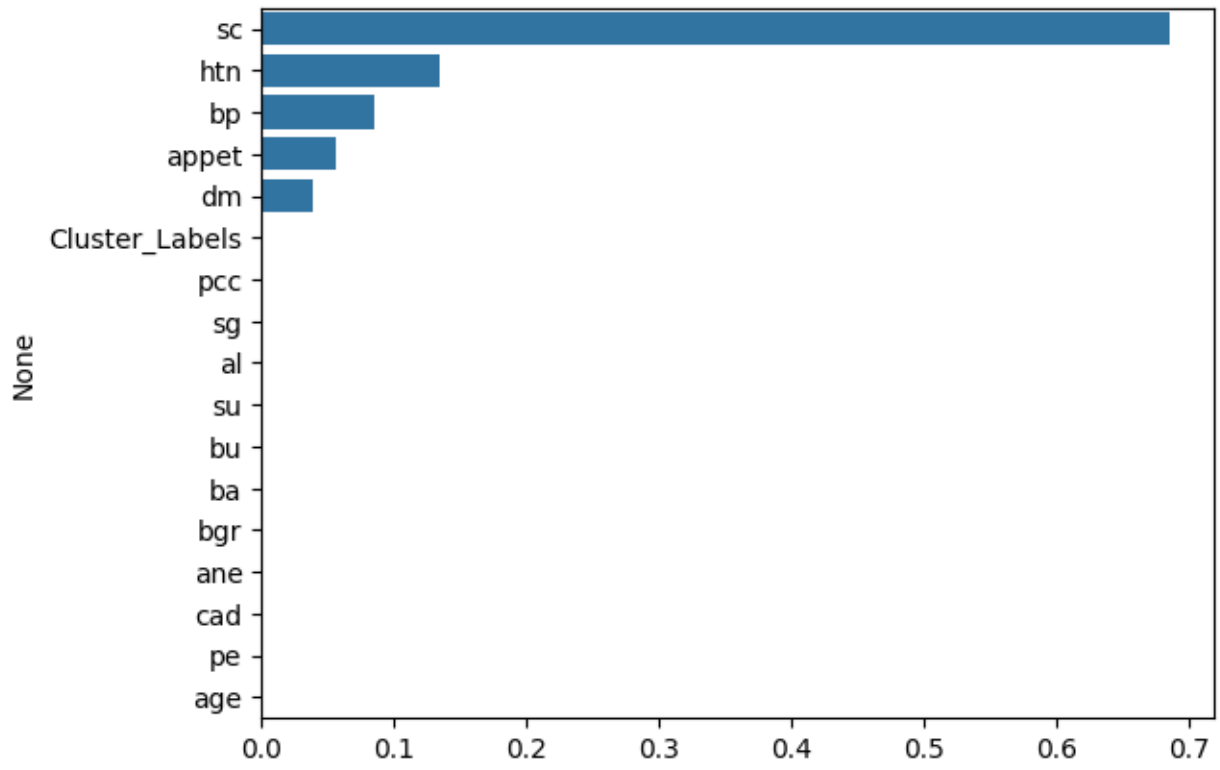
```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        39
           1       1.00      0.96      0.98        48

    accuracy                           0.98        87
   macro avg       0.98      0.98      0.98        87
weighted avg       0.98      0.98      0.98        87

              precision    recall  f1-score   support

           0       0.95      1.00      0.97        39
           1       1.00      0.96      0.98        48

    accuracy                           0.98        87
   macro avg       0.98      0.98      0.98        87
weighted avg       0.98      0.98      0.98        87
```




```
fea_imp = df_best.feature_importances_
```

```
# Sort the feature importances from greatest to
# least using the sorted indices
sorted_indices = fea_imp.argsort()[::-1]# read from the tail of the argsort to get greatest to
sorted_feature_names = X_train_d.columns[sorted_indices]
sorted_importances = fea_imp[sorted_indices]
```

```
sns.barplot(x = sorted_importances, y = sorted_feature_names)
plt.show()
```



Logistic Regression:

```
from sklearn.linear_model import LogisticRegression


pred_prob = glm_model.predict_proba(x_test[selected_features_train])
pred_prob
```

```
array([[8.76659006e-01, 1.23340994e-01],
       [2.80556291e-07, 9.99999719e-01],
       [1.69140198e-05, 9.99983086e-01],
       [1.12114497e-04, 9.99887886e-01],
       [7.96751835e-01, 2.03248165e-01],
       [4.87079267e-03, 9.95129207e-01],
       [7.55087917e-01, 2.44912083e-01],
       [9.42400214e-01, 5.75997858e-02],
       [9.83485610e-01, 1.65143901e-02],
       [1.48506297e-04, 9.99851494e-01],
```

[4.48405825e-03, 9.95515942e-01],

[8.37720932e-01, 1.62279068e-01],

[9.83485610e-01, 1.65143901e-02],

[8.88878662e-03, 9.91111213e-01],

[7.96751835e-01, 2.03248165e-01],

[2.45449738e-04, 9.99754550e-01],

[7.96751835e-01, 2.03248165e-01],

[6.77143636e-04, 9.99322856e-01],

[6.35077888e-01, 3.64922112e-01],

[7.96751835e-01, 2.03248165e-01],

[7.55087917e-01, 2.44912083e-01],

[2.06749718e-04, 9.99793250e-01],

[6.24412515e-04, 9.99375587e-01],

[1.13529586e-07, 9.99999886e-01],

[9.83485610e-01, 1.65143901e-02],

[1.12735768e-04, 9.99887264e-01],

[6.60383166e-05, 9.99933962e-01],

[3.15514146e-04, 9.99684486e-01],

[9.12056375e-01, 8.79436247e-02],

[1.48644252e-03, 9.98513557e-01],

[1.04312591e-04, 9.99895687e-01],

[8.37720932e-01, 1.62279068e-01],

[9.66435355e-01, 3.35646454e-02],

[1.68363789e-02, 9.83163621e-01],

[1.83556845e-02, 9.81644315e-01],

[4.79100813e-05, 9.99952090e-01],

[9.83485610e-01, 1.65143901e-02],

[9.83485610e-01, 1.65143901e-02],

[1.90959704e-01, 8.09040296e-01],

[9.66435355e-01, 3.35646454e-02],

[1.12518952e-05, 9.99988748e-01],

[3.06971268e-04, 9.99693029e-01],

[2.21776937e-05, 9.99977822e-01],

[3.62711483e-07, 9.99999637e-01],

[5.40991168e-03, 9.94590088e-01],

[9.66435355e-01, 3.35646454e-02],

[6.71350356e-04, 9.99328650e-01],

[1.94030521e-03, 9.98059695e-01],

[1.77059560e-02, 9.82294044e-01],

[9.83485610e-01, 1.65143901e-02],

[9.83485610e-01, 1.65143901e-02],

[7.82757049e-05, 9.99921724e-01],

[7.55087917e-01, 2.44912083e-01],

[1.99421239e-03, 9.98005788e-01],

[9.42400214e-01, 5.75997858e-02],

[7.55087917e-01, 2.44912083e-01],

[8.37720932e-01, 1.62279068e-01],

[4.64167887e-01, 5.35832113e-01],

[8.37720932e-01, 1.62279068e-01],

[2.92451695e-04, 9.99707548e-01],

[9.42400214e-01, 5.75997858e-02],

[7.55087917e-01, 2.44912083e-01],

[7.55087917e-01, 2.44912083e-01],

[7.70690033e-05, 9.99922931e-01],

[1.17963746e-03, 9.98820363e-01],

[8.76659006e-01, 1.23340994e-01],

[9.83485610e-01, 1.65143901e-02],

[1.20356752e-01, 8.79643248e-01],

[6.20084119e-02, 9.37991588e-01],

[7.55087917e-01, 2.44912083e-01],

[2.00488052e-01, 7.99511948e-01],

[9.83485610e-01, 1.65143901e-02],

[7.55087917e-01, 2.44912083e-01],

[7.96751835e-01, 2.03248165e-01],

```
       [9.12056375e-01, 8.79436247e-02],
       [1.13749957e-06, 9.99998863e-01],
       [2.42057212e-06, 9.99997579e-01],
       [1.10543428e-02, 9.88945657e-01],
       [8.37720932e-01, 1.62279068e-01],
       [7.55087917e-01, 2.44912083e-01],
       [1.37281400e-03, 9.98627186e-01],
       [9.66435355e-01, 3.35646454e-02],
       [9.20219142e-02, 9.07978086e-01],
       [8.76659006e-01, 1.23340994e-01],
       [6.81941944e-03, 9.93180581e-01],
       [1.88913667e-01, 8.11086333e-01],
       [9.12056375e-01, 8.79436247e-02]])
```

```
df_new = pd.DataFrame(data = {'prob1': pred_prob[:,1], 'y_test': y_test})
df_new.head()
##
```

	prob1	y_test
273	0.123341	0
242	1.000000	1
93	0.999983	1
133	0.999888	1
311	0.203248	0

```
df_new['y_test_pred'] = df_new.prob1.map(lambda x: 1 if x>0.5 else 0)
df_new.head()
```

	prob1	y_test	y_test_pred
273	0.123341	0	0
242	1.000000	1	1

	prob1	y_test	y_test_pred
93	0.999983	1	1
133	0.999888	1	1
311	0.203248	0	0

```
from sklearn.metrics import confusion_matrix, classification_report

cf_matrix = confusion_matrix(df_new.y_test, df_new.y_test_pred)
print('Confusion Matrix : \n', cf_matrix)

total = sum(sum(cf_matrix))

accuracy = (cf_matrix[0,0]+cf_matrix[1,1])/total
print ('Accuracy : ', accuracy)

print(classification_report(df_new.y_test, df_new.y_test_pred))
```

```
Confusion Matrix :
 [[39  0]
 [ 3 45]]
Accuracy :  0.9655172413793104
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        39
           1       1.00      0.94      0.97        48

    accuracy                           0.97        87
   macro avg       0.96      0.97      0.97        87
weighted avg       0.97      0.97      0.97        87
```

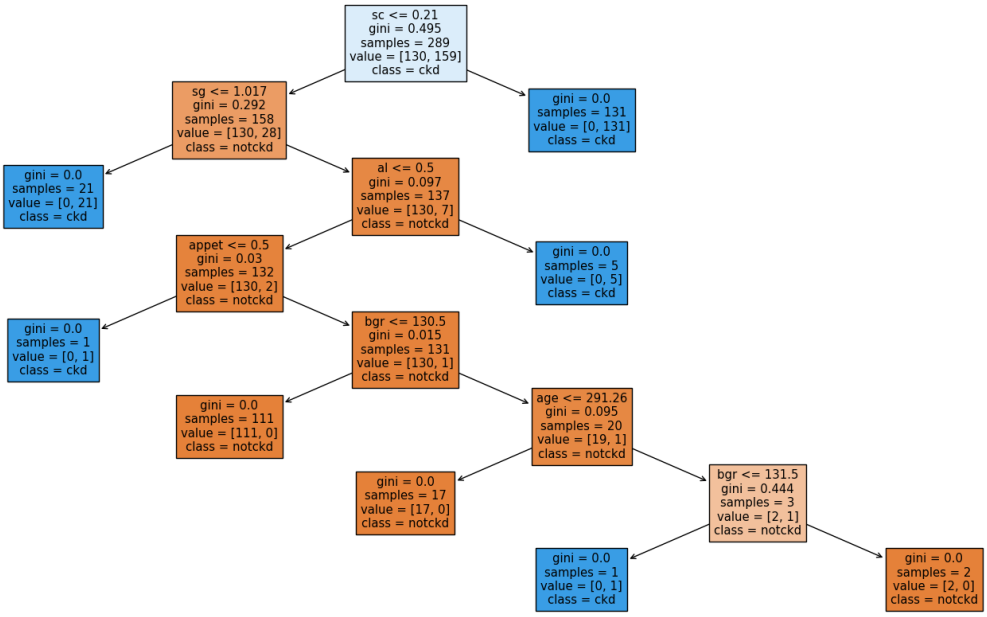
By comparing the accuracy of both the classifiers, the accuracy of the decision tree is 0.98, and the logistic regression is 0.97, so the decision tree is better in predicting correctly.

The result also shows that the F1 score of the decision tree is higher than the logistic regression, which shows that the model with metrics of decision tree has a higher precision and recall, a good balance of precision and recall and handles the unbalanced dataset more efficiently.

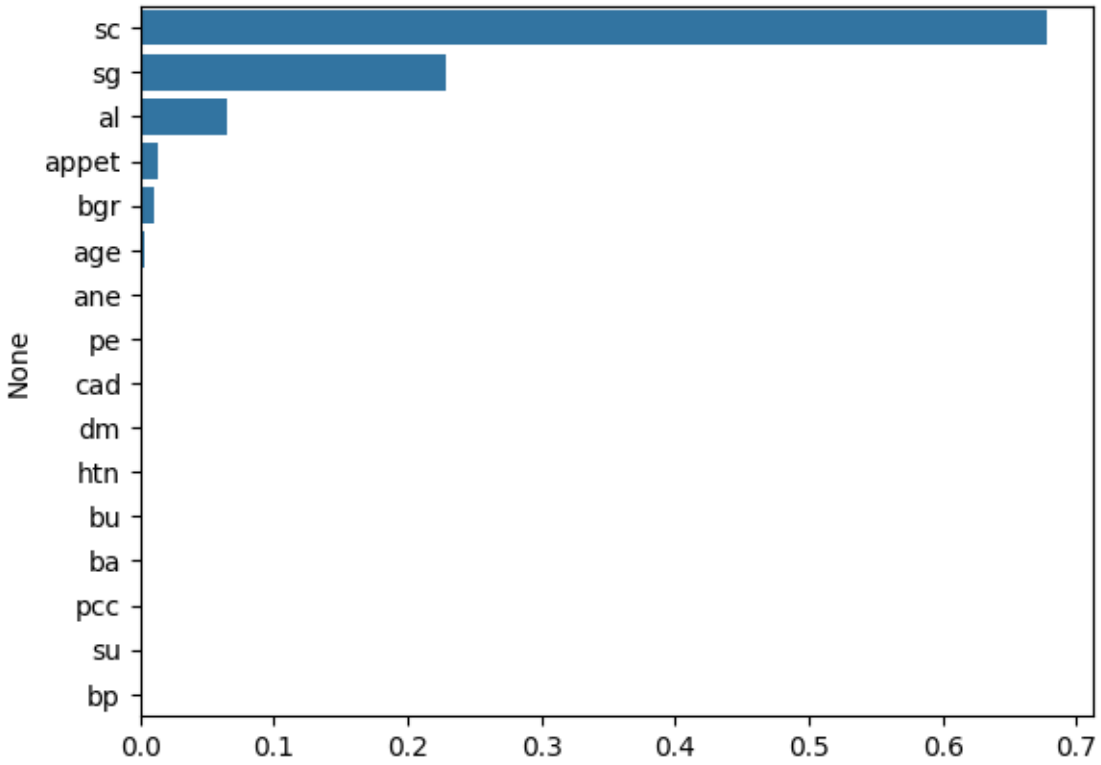
Q13

```
from sklearn import tree
decision_tree = DecisionTreeClassifier(max_depth=30, random_state=1)

# Retrain the model on the entire dataset
decision_tree.fit(x, y)

importances = decision_tree.feature_importances_
print("Feature importances:", importances)



plt.figure(figsize=(20, 10))
tree.plot_tree(decision_tree, filled=True, feature_names=x.columns, class_names=['notckd', 'ck
plt.show()
```

```
Feature importances: [0.00396146 0.         0.22923753 0.06533104 0.         0.
 0.         0.00991343 0.         0.67789189 0.         0.
 0.         0.01366466 0.         0.         ]
```




```
sorted_indices = importances.argsort()[::-1]

sorted_feature_names = x.columns[sorted_indices]

sorted_importances = importances[sorted_indices]


sns.barplot(x = sorted_importances, y = sorted_feature_names)

plt.show()
```



By visualizing the importances of predictor variables, we observed that sc(Serum Creatinine) exhibits the highest importance value, indicating that the most important effect in predicting the classification results. Serum Creatinine plays a strong role in whether or not a person has chronic kidney disease, and higher levels of Serum Creatinine may predict problems with kidney function.

sg (Specific Gravity) and al (Albumin) are two other important predictor variables because the values of sg(Specific Gravity) and al(Albumin) are obvious in the plot.

Understanding these variables' importance can help adapt preventive measures against chronic kidney disease to meet the challenges effectively.

Q14

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

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



model_poly = LogisticRegression()



model_poly.fit(X_train_poly, y_train)



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

```
Improved Model Accuracy: 0.9885057471264368
```

```
d:\workshop\stats3da\assign1\.venv\Lib\site-packages\sklearn\linear_model\_logistic.py:469: Co
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

We choose to use the polynomial features to increase the complexity of features and then use the logistic regression for prediction. According to k-means, we know that clustering works best when k=2. So we choose the polynomial degree of 2. Then the new accuracy increases.

Q15

1.Xinyi add question 1 and dataset

2.Ruiqi add code, Xinyi add comment

3.Huixuan add code and comment, Ruiqi add code

4.Huixuan add comment, Ruiqi add code, Xinyi add comment

5.Xinyi, Huixuan, and Ruiqi all add code

6.Ruiqi add code

7.Ruiqi add code, Xinyi add code and comment 8.Xinyi add code

9.Xinyi add answer

10.Huixuan add answer

11.Ruiqi add code

12.Huixuan add logistic regression, Xinyi add decision tree

13.Huixuan add code, Xinyi add comment

14.Ruiqi add code, Huixuan add comment

Q16

https://github.com/chenx333/Chronic-Kidney-Disease-Classification-Challenge

Grading scheme

1.	Answer [1]
2.	Codes [2]
	OR answer [2]
3.	Codes [3] and answer [3]
4.	Codes [2] and answer [3]
5.	Codes [2]
	OR answer [2]
6.	Codes [2]
	OR answer [2]
7.	Codes [3] and Plot [1]
8.	Codes [1]
9.	Answers [2]
10.	Describe the two metrics [2]
11.	Codes [2]
	these codes can be included in (12)
12.	Codes (two classifiers training,
	model selection for each classifier,
	classifiers comparisons) [5] and answer [2]
13.	Codes [1] and answers [2]
14.	Codes and comparison will
	give **bonus 2 points for the final grade**.

The maximum point for this assignment is 39. We will convert this to 100%.

All group members will receive the same grade if they contribute to the same.

References

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