NEXUS PROJECT – 3

Name: K.Arunadevi

Project Title : Renal Insufficiency Syndrome

Summary of the data: In the analysis of renal insufficiency syndrome, I employed both K-nearest neighbors (KNN) and decision tree classifiers to predict the presence of the syndrome based on various patient attributes. After preprocessing the dataset to handle missing values, encode categorical variables, and scale numerical features, I applied the KNN algorithm and tuned its hyperparameters, such as the number of neighbors, through cross-validation. Similarly, I utilized a decision tree classifier, optimizing its parameters like maximum depth and minimum samples split. Evaluating the performance of both models using metrics like accuracy, precision, recall, and F1-score allowed for a comprehensive comparison. Through this analysis, I identified important features influencing renal insufficiency syndrome prediction and assessed the strengths and weaknesses of each model. Ultimately, this study provides valuable insights into early diagnosis and treatment planning for renal insufficiency syndrome, potentially guiding future research and clinical decision-making in this domain**.**

Codes done using google colab:

Importing necessary libraries and loading data:

Importing necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

import warnings
warnings.filterwarnings('ignore')

plt.style.use('fivethirtyeight')
%matplotlib inline
pd.set_option('display.max_columns', 26)
```

Loading data

```
df= pd.read_csv('RIS.csv')
df.head()
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	...	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification	Unnamed: 26
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	...	NaN	15.4	44	7800	5.2	yes	yes	no	good	no	no	ckd	NaN
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	...	NaN	11.3	38	6000	NaN	no	no	no	good	no	no	ckd	NaN
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	...	NaN	9.6	31	7500	NaN	no	yes	no	poor	no	yes	ckd	NaN
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	...	2.5	11.2	32	6700	3.9	yes	no	no	poor	yes	yes	ckd	NaN
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	...	NaN	11.6	35	7300	4.6	no	no	no	good	no	no	ckd	NaN

5 rows × 27 columns

```
df.tail()
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	...	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification	Unnamed: 26
395	395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	140.0	49.0	0.5	...	4.9	15.7	47	6700	4.9	no	no	no	good	no	no	notckd	NaN
396	396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	75.0	31.0	1.2	...	3.5	16.5	54	7800	6.2	no	no	no	good	no	no	notckd	NaN
397	397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	100.0	26.0	0.6	...	4.4	15.8	49	6600	5.4	no	no	no	good	no	no	notckd	NaN
398	398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	114.0	50.0	1.0	...	4.9	14.2	51	7200	5.9	no	no	no	good	no	no	notckd	NaN
399	399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	131.0	18.0	1.1	...	3.5	15.8	53	6800	6.1	no	no	no	good	no	no	notckd	NaN

```
df.shape
```

```
(400, 27)
```

Dropping id column

```
df.drop('id', axis = 1, inplace = True)
```

```
df.head()
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification	Unnamed: 26
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	NaN	NaN	15.4	44	7800	5.2	yes	yes	no	good	no	no	ckd	NaN
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN	11.3	38	6000	NaN	no	no	no	good	no	no	ckd	NaN
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN	NaN	9.6	31	7500	NaN	no	yes	no	poor	no	yes	ckd	NaN
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5	11.2	32	6700	3.9	yes	no	no	poor	yes	yes	ckd	NaN
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN	NaN	11.6	35	7300	4.6	no	no	no	good	no	no	ckd	NaN

Replacing all the column names for our better understanding

```
[ ] df.columns = ['Age', 'Blood_Pressure', 'Specific_Gravity', 'Albumin', 'Sugar', 'Red_Blood_Cells', 'Pus_Cell',
                  'Pus_Cell_Clumps', 'Bacteria', 'Blood_Glucose_Random', 'Blood_Urea', 'Serum_Creatinine', 'Sodium',
                  'Potassium', 'Haemoglobin', 'Packed_Cell_Volume', 'White_Blood_Cell_Count', 'Red_Blood_Cell_Count',
                  'Hypertension', 'Diabetes_mellitus', 'Coronary_Artery_Disease', 'Appetite', 'Peda_Edema',
                  'Aanemia', 'Class']
```

```
[ ] df.head()
```

	Age	Blood_Pressure	Specific_Gravity	Albumin	Sugar	Red_Blood_Cells	Pus_Cell	Pus_Cell_Clumps	Bacteria	Blood_Glucose_Random	Blood_Urea	Serum_Creatinine	Sodium	Potassium	Haemoglobin	Packed_Cell
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	NaN	NaN	15.4	
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN	11.3	
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN	NaN	9.6	
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5	11.2	
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN	NaN	11.6	

```
[ ] df.describe()
```

	Age	Blood_Pressure	Specific_Gravity	Albumin	Sugar	Blood_Glucose_Random	Blood_Urea	Serum_Creatinine	Sodium	Potassium	Haemoglobin
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Age                     391 non-null    float64
 1   Blood_Pressure          388 non-null    float64
 2   Specific_Gravity        353 non-null    float64
 3   Albumin                 354 non-null    float64
```

Changing object type of Packed_Cell_Volume,White_Blood_Cell_Count and Red_Blood_Cell_Count to numerical type:

Changing object type of Packed_Cell_Volume,White_Blood_Cell_Count and Red_Blood_Cell_Count to numerical type

```
[ ] df['Packed_Cell_Volume'] = pd.to_numeric(df['Packed_Cell_Volume'], errors='coerce')
    df['White_Blood_Cell_Count'] = pd.to_numeric(df['White_Blood_Cell_Count'], errors='coerce')
    df['Red_Blood_Cell_Count'] = pd.to_numeric(df['Red_Blood_Cell_Count'], errors='coerce')
```

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Age                     391 non-null    float64
 1   Blood_Pressure          388 non-null    float64
 2   Specific_Gravity        353 non-null    float64
 3   Albumin                 354 non-null    float64
 4   Sugar                   351 non-null    float64
 5   Red_Blood_Cells         248 non-null    object
 6   Pus_Cell                335 non-null    object
 7   Pus_Cell_Clumps         396 non-null    object
 8   Bacteria                396 non-null    object
 9   Blood_Glucose_Random    356 non-null    float64
 10  Blood_Urea              381 non-null    float64
 11  Serum_Creatinine        383 non-null    float64
 12  Sodium                  313 non-null    float64
 13  Potassium               312 non-null    float64
```

Extracting categorical and numerical columns

```
[ ]  cat_cols = [col for col in df.columns if df[col].dtype == 'object']
     num_cols = [col for col in df.columns if df[col].dtype != 'object']
```

Unique values in categorical columns

```
[ ]  for col in cat_cols:
         print(f"{col} has {df[col].unique()} values\n")
```

```
Red_Blood_Cells has [nan 'normal' 'abnormal'] values

Pus_Cell has ['normal' 'abnormal' nan] values

Pus_Cell_Clumps has ['notpresent' 'present' nan] values

Bacteria has ['notpresent' 'present' nan] values

Hypertension has ['yes' 'no' nan] values

Diabetes_mellitus has ['yes' 'no' ' yes' '\tno' '\tyes' nan] values

Coronary_Artery_Disease has ['no' 'yes' '\tno' nan] values

Appetite has ['good' 'poor' nan] values

Peda_Edema has ['no' 'yes' nan] values

Aanemia has ['no' 'yes' nan] values
```

Replacing incorrect values

```
[ ]  df['Diabetes_mellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes',' yes':'yes'},inplace=True)
     df['Coronary_Artery_Disease'] = df['Coronary_Artery_Disease'].replace(to_replace = '\tno', value='no')
     df['Class'] = df['Class'].replace(to_replace = {'ckd\t': 'ckd', 'notckd': 'not ckd'})
```

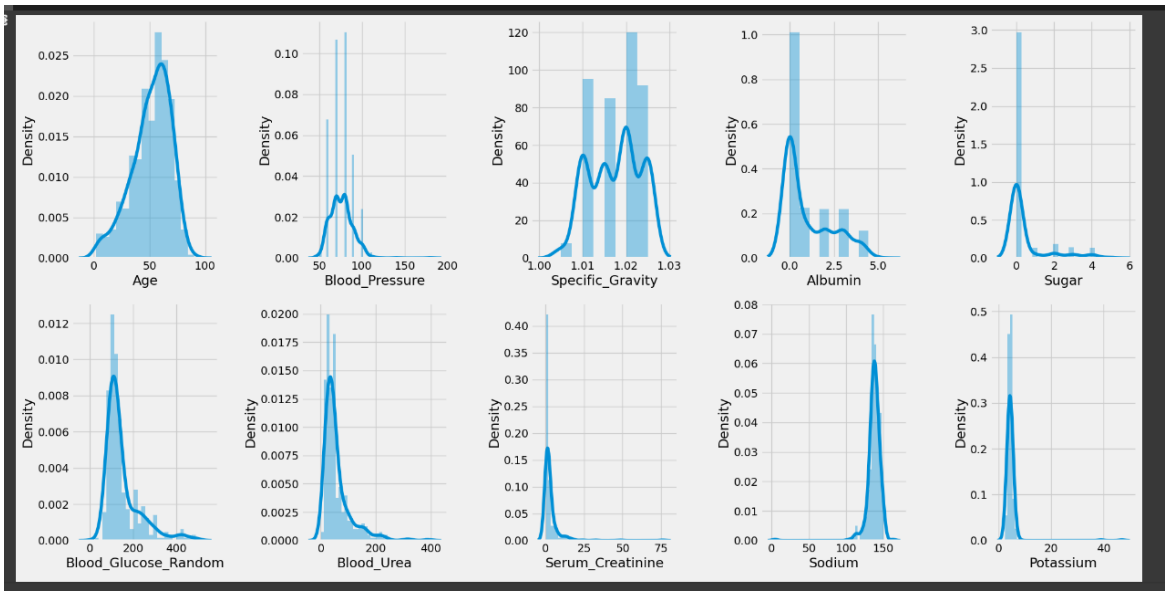
```
[ ]  df['Class'] = df['Class'].map({'ckd': 0, 'not ckd': 1})
     df['Class'] = pd.to_numeric(df['Class'], errors='coerce')
```

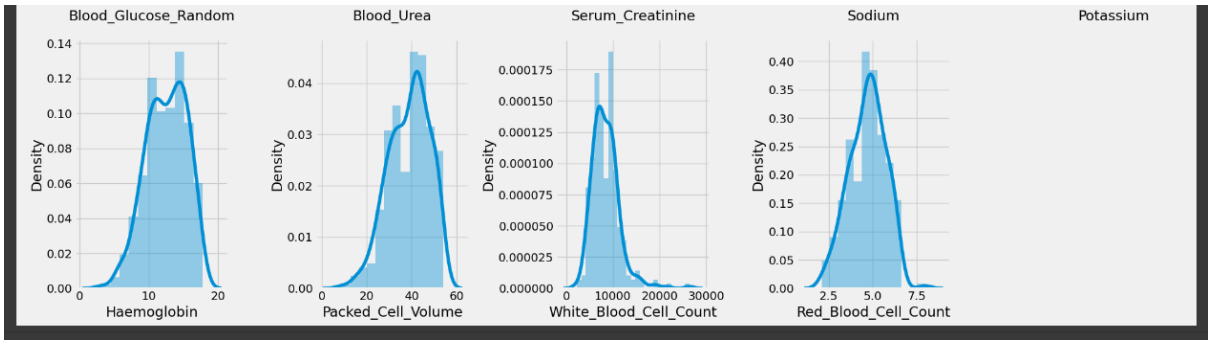
```
cols = ['Diabetes_mellitus', 'Coronary_Artery_Disease', 'Class']

for col in cols:
    print(f"{col} has {df[col].unique()} values\n")
```

```
Diabetes_mellitus has ['yes' 'no' nan] values

Coronary_Artery_Disease has ['no' 'yes' nan] values

Class has [0 1] values
```

Numerical features distribution

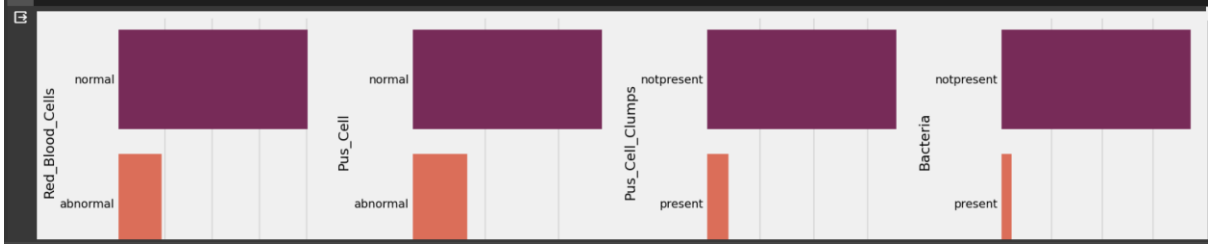
```
plt.figure(figsize = (20, 15))
plotnumber = 1

for column in num_cols:
    if plotnumber <= 14:
        ax = plt.subplot(3, 5, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column)

    plotnumber += 1

plt.tight_layout()
plt.show()
```

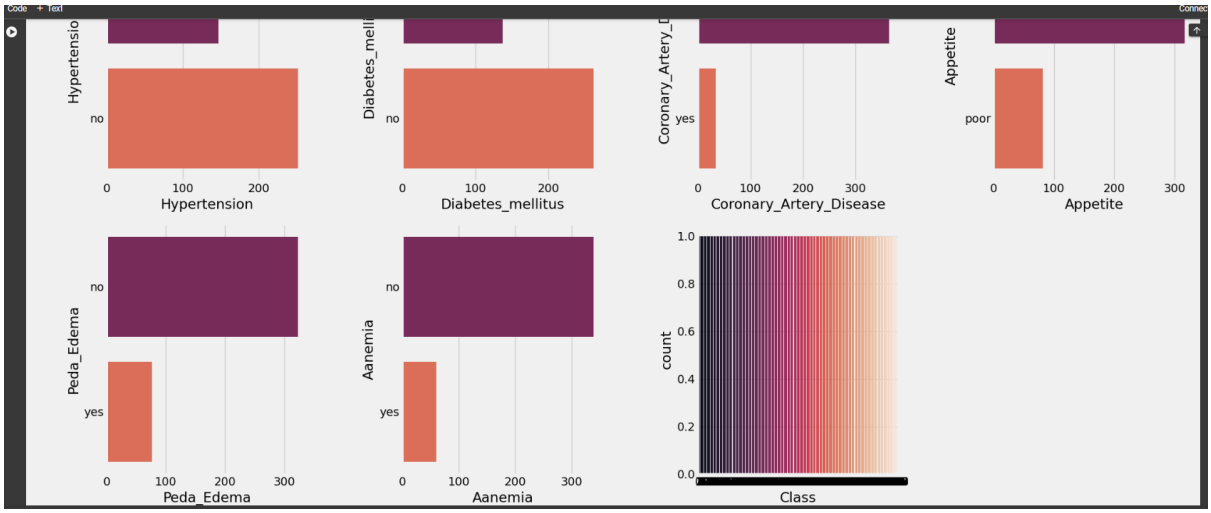




Categorical columns

```
plt.figure(figsize = (20, 15))
plotnumber = 1

for column in cat_cols:
    if plotnumber <= 11:
        ax = plt.subplot(3, 4, plotnumber)
        sns.countplot(df[column], palette = 'rocket')
        plt.xlabel(column)

    plotnumber += 1

plt.tight_layout()
plt.show()
```

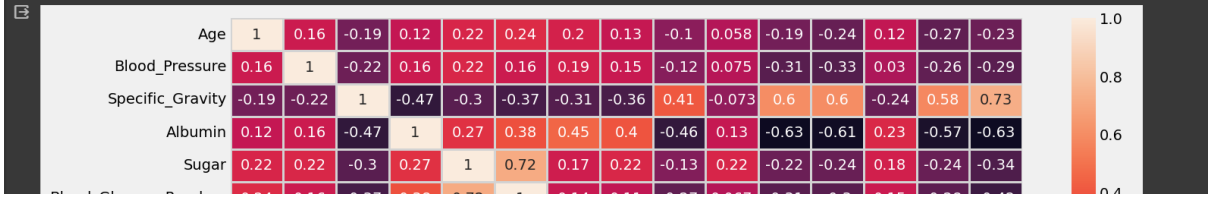


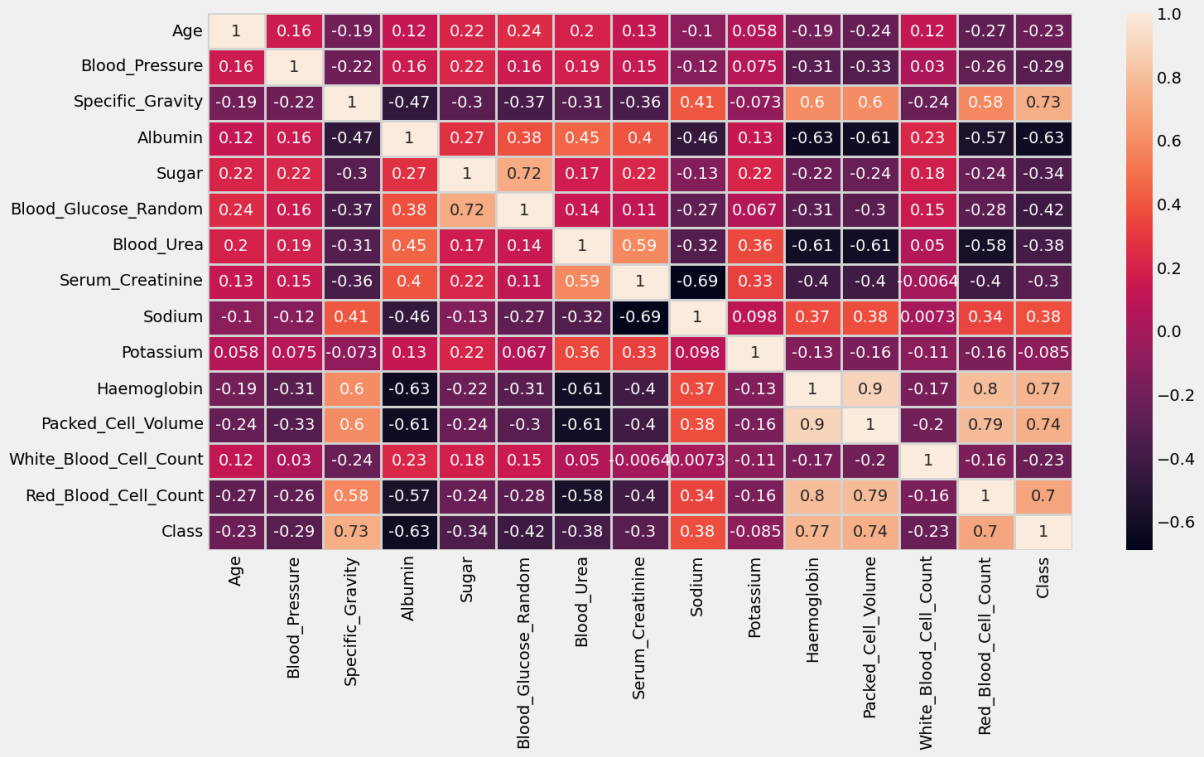


Heatmap of data:

Heatmap of data

```
plt.figure(figsize = (15, 8))

sns.heatmap(df.corr(), annot = True, linewidths = 2, linecolor = 'lightgrey')
plt.show()
```





Data Preprocessing:

Data Preprocessing

```
df.isna().sum().sort_values(ascending = Fa
```

```
Red_Blood_Cells             152
Red_Blood_Cell_Count        131
White_Blood_Cell_Count      106
Potassium                    88
Sodium                       87
Packed_Cell_Volume           71
Pus_Cell                     65
Haemoglobin                  52
Sugar                        49
Specific_Gravity             47
Albumin                      46
Blood_Glucose_Random         44
Blood_Urea                   19
Serum_Creatinine             17
Blood_Pressure               12
Age                           9
Bacteria                      4
Pus_Cell_Clumps               4
Hypertension                  2
Diabetes_mellitus             2
Coronary_Artery_Disease       2
Appetite                      1
Peda_Edema                    1
Aanemia                       1
Class                         0
dtype: int64
```

```
[ ] df[num_cols].isnull().sum()
```

```
Age                   9
Blood_Pressure       12
Specific_Gravity     47
Albumin              46
```

```
[ ]  df[cat_cols].isnull().sum()
```

```
Red_Blood_Cells            152
Pus_Cell                    65
Pus_Cell_Clumps              4
Bacteria                     4
Hypertension                 2
Diabetes_mellitus            2
Coronary_Artery_Disease      2
Appetite                     1
Peda_Edema                   1
Aanemia                      1
Class                        0
dtype: int64
```

For filling null values, we will use two methods, random sampling for higher null values and mean/mode sampling for lower null values

```
[ ]  def random_value_imputation(feature):
         random_sample = df[feature].dropna().sample(df[feature].isna().sum())
         random_sample.index = df[df[feature].isnull()].index
         df.loc[df[feature].isnull(), feature] = random_sample

     def impute_mode(feature):
         mode = df[feature].mode()[0]
         df[feature] = df[feature].fillna(mode)
```

Filling num_cols null values using random sampling method

```
[ ]  for col in num_cols:
         random_value_imputation(col)
```

```
[ ]  df[num_cols].isnull().sum()
```

```
Age                      0
Blood_Pressure           0
Specific_Gravity         0
Albumin                  0
Sugar                    0
Blood_Glucose_Random     0
Blood_Urea               0
Serum_Creatinine         0
Sodium                   0
Potassium                0
Haemoglobin              0
Packed_Cell_Volume       0
White_Blood_Cell_Count   0
Red_Blood_Cell_Count     0
dtype: int64
```

Filling "Red_Blood_Cells" and "Pus_Cell" using random sampling method and rest of cat_cols using mode imputation

```
[ ]  random_value_imputation('Red_Blood_Cells')
     random_value_imputation('Pus_Cell')

     for col in cat_cols:
         impute_mode(col)
```

```
[ ]  df[cat_cols].isnull().sum()
```

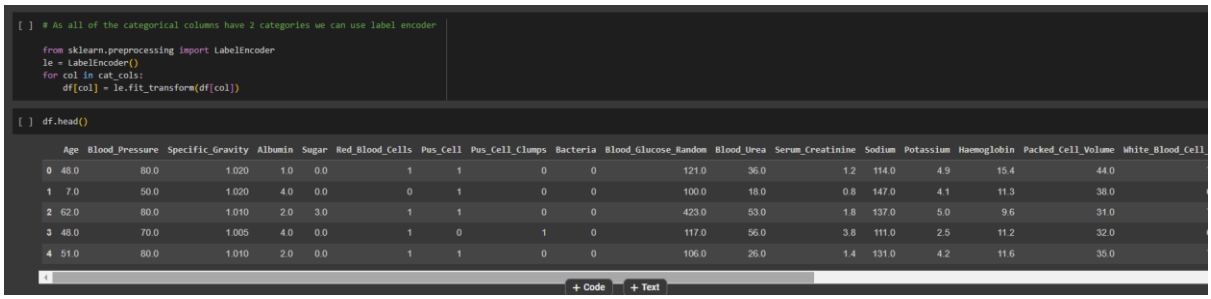
```
Red_Blood_Cells            0
Pus_Cell                   0
Pus_Cell_Clumps            0
Bacteria                   0
Hypertension               0
Diabetes_mellitus          0
Coronary_Artery_Disease    0
Appetite                   0
Peda_Edema                 0
Aanemia                    0
Class                      0
dtype: int64
```

Feature encoding:

Feature Encoding

```
[ ]  for col in cat_cols:
         print(f"{col} has {df[col].nunique()} categories\n")
```

```
Red_Blood_Cells has 2 categories

Pus_Cell has 2 categories

Pus_Cell_Clumps has 2 categories

Bacteria has 2 categories

Hypertension has 2 categories

Diabetes_mellitus has 2 categories

Coronary_Artery_Disease has 2 categories

Appetite has 2 categories

Peda_Edema has 2 categories

Aanemia has 2 categories

Class has 2 categories
```



```
[ ] # As all of the categorical columns have 2 categories we can use label encoder

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in cat_cols:
    df[col] = le.fit_transform(df[col])
```

```
[ ] df.head()
```

	Age	Blood_Pressure	Specific_Gravity	Albumin	Sugar	Red_Blood_Cells	Pus_Cell	Pus_Cell_Clumps	Bacteria	Blood_Glucose_Random	Blood_Urea	Serum_Creatinine	Sodium	Potassium	Haemoglobin	Packed_Cell_Volume	White_Blood_Cell
0	48.0	80.0	1.020	1.0	0.0	1	1	0	0	121.0	36.0	1.2	114.0	4.9	15.4	44.0	
1	7.0	50.0	1.020	4.0	0.0	0	1	0	0	100.0	18.0	0.8	147.0	4.1	11.3	38.0	
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	53.0	1.8	137.0	5.0	9.6	31.0	
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	56.0	3.8	111.0	2.5	11.2	32.0	
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	26.0	1.4	131.0	4.2	11.6	35.0	

+ Code + Text

Model building and train-test split:

MODEL BUILDING Train-Test splitting

```
ind_col = [col for col in df.columns if col != 'Class']
dep_col = 'Class'

X = df[ind_col]
y = df[dep_col]
```

```
[ ]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

KNN:

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

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

```
KNeighborsClassifier
KNeighborsClassifier()
```

```
[ ] # accuracy score, confusion matrix and classification report of knn
    knn_acc = accuracy_score(y_test, knn.predict(X_test))
    print(f"Training Accuracy of KNN is {accuracy_score(y_train, knn.predict(X_train))}")
    print(f"Test Accuracy of KNN is {knn_acc} \n")

    print(f"Confusion Matrix :- \n{confusion_matrix(y_test, knn.predict(X_test))}\n")
    print(f"Classification Report :- \n {classification_report(y_test, knn.predict(X_test))}")
```

```
Training Accuracy of KNN is 0.81875
Test Accuracy of KNN is 0.6

Confusion Matrix :-
[[31 21]
 [11 17]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.74      0.60      0.66        52
           1       0.45      0.61      0.52        28

    accuracy                           0.60        80
   macro avg       0.59      0.60      0.59        80
weighted avg       0.64      0.60      0.61        80
```

Decision Tree classifier:

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
[ ] # accuracy score, confusion matrix and classification report of decision tree

    dtc_acc = accuracy_score(y_test, dtc.predict(X_test))
    print(f"Training Accuracy of Decision Tree Classifier is {accuracy_score(y_train, dtc.predict(X_train))}")
    print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")

    print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(X_test))}\n")
    print(f"Classification Report :- \n {classification_report(y_test, dtc.predict(X_test))}")
```

```
Training Accuracy of Decision Tree Classifier is 1.0
Test Accuracy of Decision Tree Classifier is 0.9625

Confusion Matrix :-
[[52  0]
 [ 3 25]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        52
           1       1.00      0.89      0.94        28

    accuracy                           0.96        80
   macro avg       0.97      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80
```

MODEL COMPARISION:

Accuracy by comparing two techniques:

	Model	Score
1	Decision Tree Classifier	0.9625
0	KNN	0.6000