INDEX

CHRONIC KIDNEY DISEASE

1.INTRODUCT ION

Chronic kidney disease (CKD) is non-communicable disease that has significantly contributed to morbidity, mortality and admission rate of patients worldwide . It is quickly expanding and becoming one of the major causes of death all over the world. A report from indicated that the global yearly life loss caused by CKD increased by and it is the leading cause of death in the world . people throughout the world are likely to have kidney diseases from different factor. Its burden is even higher in low-income countries where detection, prevention and treatment remain low Kidney disease is serious public health problem in Ethiopia effecting hundreds of thousands of people irrespective of age, sex . The lack of safe water, appropriate diet, and physical activities is believed have contributed. Additionally, communities living in rural area have limited knowledge about the CKD.

National kidney foundation classifies stages of CKD into five based on the abnormal kidney function and reduced, a symptoms and is considered as end-stage or kidney failure. The Renal Replacement Therapy (RRT) cost for total kidney failure is very expensive. The treatment is not also available in most developing countries like Ethiopia. As a result, the management of kidney failure and its complications is very difficult in developing countries due to shortage of facilities, physicians, and the high cost to get the treatment . Hence,

early detection of CKD is very essential to minimize the economic burden and maximize the effectiveness of treatments. Predictive analysis using machine learning techniques can be helpful through an early detection of CKD for efficient and timely interventions . In this study, Random Forest (RF), Support Vector Machine (SVM) and Decision Tree (DT) have been used to detect CKD. Most of previous researches focused on two classes, which make treatment recommendations difficult because the type of treatment to be given is based on the severity of CKD.

1.1 Overview

CKD is a condition in which the kidneys are damaged and cannot filter blood as well as they should. Because of this, excess fluid and waste from blood remain in the body and may cause other health problems, such as heart disease and stroke.

Some other health consequences of CKD include:

- ✓ Anemia or low number of red blood cells
- ✓ Increased occurrence of infections
- ✓ Low calcium levels, high potassium levels, and high phosphorus levels in the blood
- ✓ Loss of appetite or eating less
- ✓ Depression or lower quality of life

CKD has varying levels of seriousness. It usually gets worse over time though treatment has been shown to slow progression. If left untreated, CKD can progress to kidney failure and early cardiovascular disease. When the kidneys stop working, dialysis or kidney transplant is needed for survival. Kidney failure treated with dialysis or kidney transplant is called end-stage renal disease (ESRD).

1.2Purpose

The kidneys are two bean-shaped organs. Each kidney is about the size of a fist. Your kidneys filter extra water and wastes out of your blood and make urine. Kidney disease means your kidneys are damaged and can’t filter blood the way they should.You are at greater risk for kidney disease if you have diabetes or high blood pressure. If you experience kidney failure, treatments includekidney transplant or dialysis. Other kidney problems include acute kidney injury, kidney cysts, kidney stones, and kidney infections.

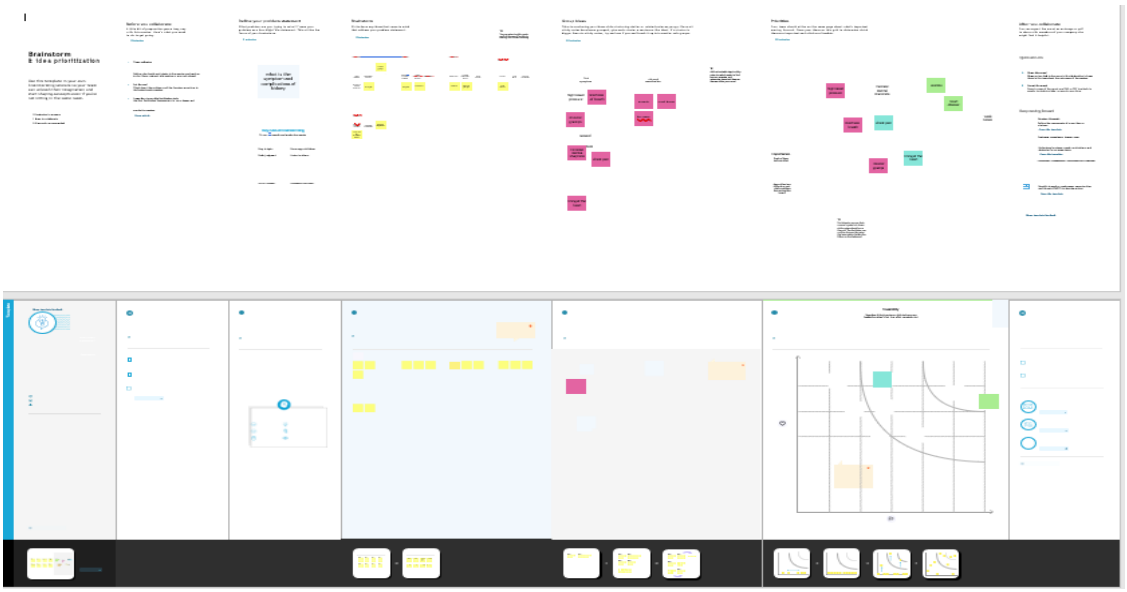
2.PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map

TeamID	NM2023TMID32910
ProjectName	Identifying Patterns and Trends in campusplacementdatausingmachinelearningwith Python
MaximumMarks	5Marks

Chronic kidney disease(ckd) is a major medical problem and can be cured if treated in the early stage.usually,people are not aware that medical tests we take for different purpose could contain valuable information concerning kidney diseases, consequently, attributes of various medical tests are inverstigated to distinguish which attributes may contain helpful information about the disease.The information says that it helps us to measure the severity of the problem, the predicted survival of the patient after the lilness,the pattern of the disease and work for curing the disease.

In todays world as we know most of the people are facing so may disease and as this can be cured if we treat people in early stages this project can use a pretrained model to predict the Chronic Kidney Disease which can help in treatments of people who are suffer from this disease.



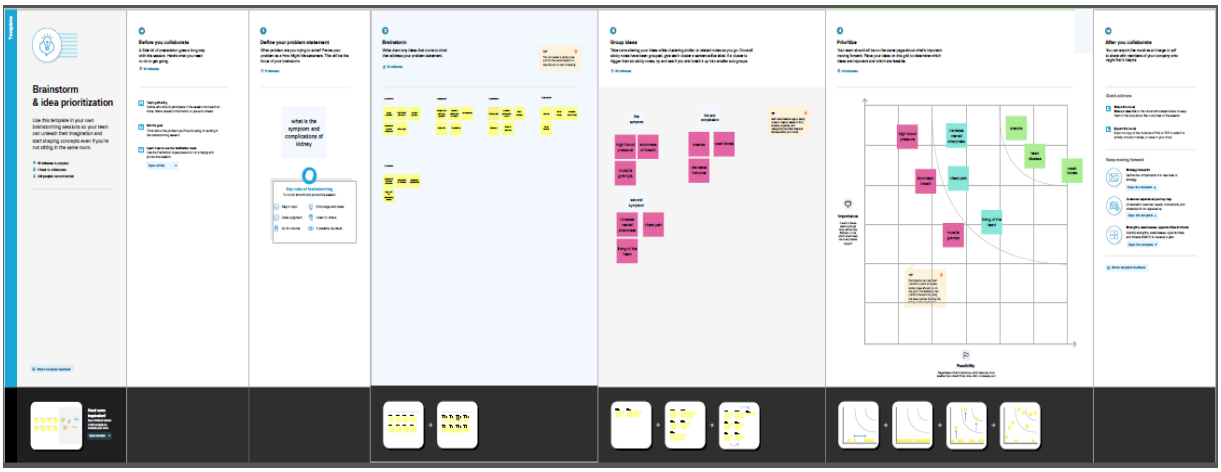
2.2 Brainstorming Map

TeamID	NM2023TMID32910
ProjectName	Identifying Patterns and Trends in campusplacementdatausingmachinelearningwith Python
MaximumMarks	5Marks

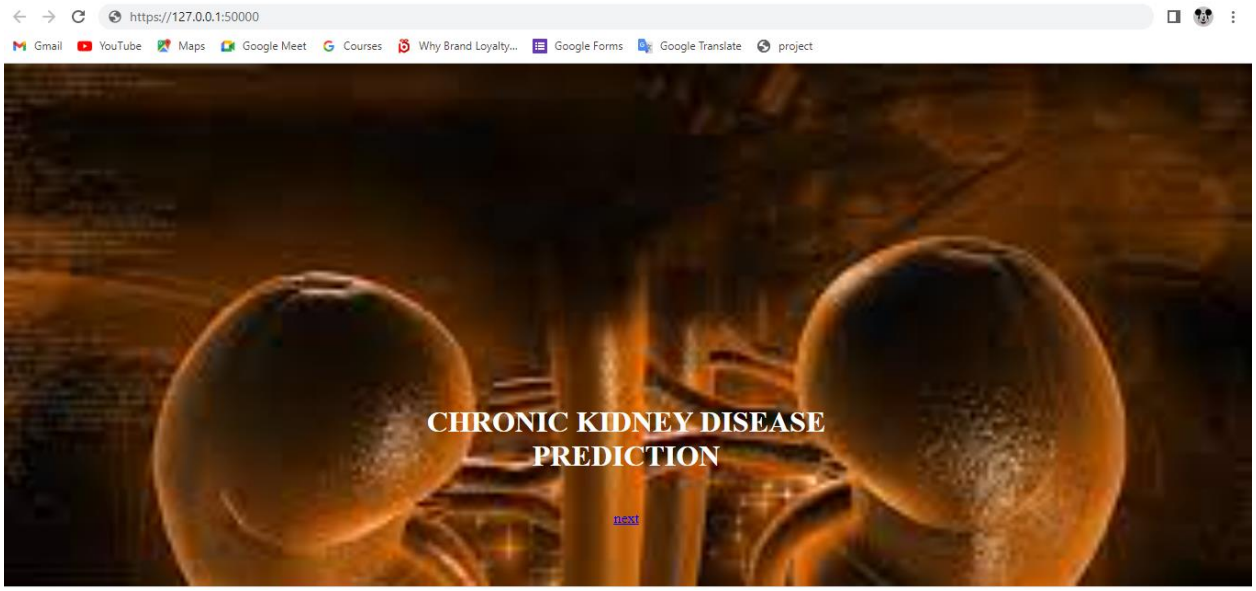
Chronic kidney disease(ckd) is a major medical problem and can be cured if treated in the early stage.usually,people are not aware that medical tests we take for different purpose could contain valuable information concerning kidney diseases, consequently, attributes of various medical tests are inverstigated to distinguish which attributes may contain helpful information about the disease.The information says that it helps us to measure the severity of the problem, the predicted

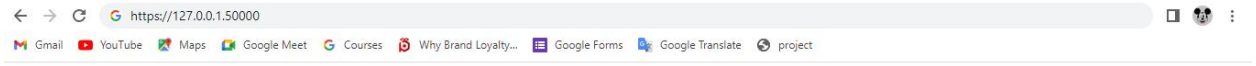
survival of the patient after the lilness,the pattern of the disease and work for curing the disease.

In todays world as we know most of the people are facing so may disease and as this can be cured if we treat people in early stages this project can use a pretrained model to predict the Chronic Kidney Disease which can help in treatments of people who are suffer from this disease.



RESULT

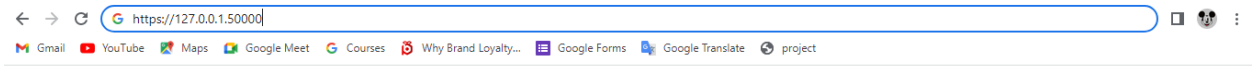




Chronic Kidney Disease
A machine Learning Web app, Built with Flask

Prediction : Oops! You have Chronic Kidney Disease

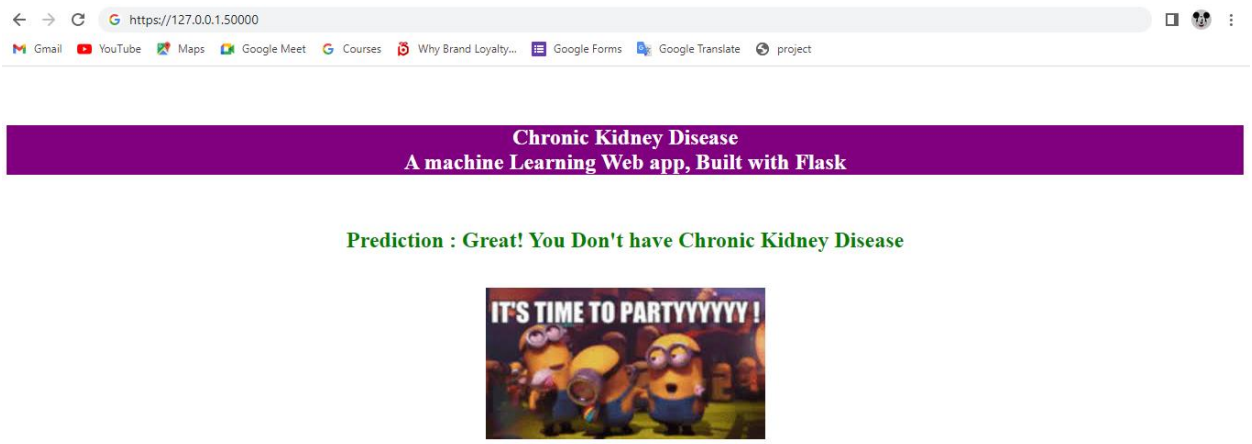
next



Chronic Kidney Disease
A Machine Learning Web App, Built with Flask

No	
Yes	
Normal	
Upnormal	
Yes	
No	

Predict next



4. ADVANTAGES

- Restoration of "normal" renal function.
- Freedom from dialysis .
- Return to " normal" life.
- Reverses pathophysiological changes related to Renal Failure.
- Less expensive than dialysis after 1st year.

DISADVANTAGES

- Life long medications.
- Multiple side effects from medication.
- Increased risk of tumor.
- Increased risk of infection.

- Major surgery.

5. APPLICATIONS

Chronic kidney disease(CKD) is a global public health issue. Mobile technology is pervasive and widely used in chronic disease care. More and more, CKD mobile applications (apps) can be found on popular mobile application platforms, especially in Chinese. We aim to explore current mobile apps for CKD patient care through content analysis to identify the app functions that health professionals can use in CKD patients with self-management.

6.CONCLUSION

Chronic Kidney Disease is a worldwide killer that is under-diagnosed and under-treated. The increased burden of chronic kidney disease in developing countries is due to globalization, low socioeconomic status, and poor access to health care and health care disparities. By early detection, treatment increasing community outreach and access to preventive medicine for high risk population , can decrease the rising burden of CKD.

Chronic Kidney Disease develops indolently, with many patients diagnosed late and a specific cause never established in a significant number of patients. It has various multi-system

complications, significantly impairing the quality of life and shortening the life span of victims. Thus the prevention and early detection of chronic kidney disease is of utmost importance.

Annual screening is recommended for patients at high risk of developing chronic kidney disease. This involves checking blood pressure, urine dipstick testing and estimating the kidney clearance function. In patients with established chronic kidney disease, kidney protective measures are indicated to arrest or slow down the loss of kidney function.

7.FUTURE SCOPE

This would help detect the chances of a person having CKD further on in his life which would. be really helpful and cost-effective people. This model could be integrated with normal blood report generation, which could. automatically flag out if there is a person at risk. Patients would not have to go to a doctor unless they are flagged by the algorithms. This would make it cheaper and easier for the modern busy person.

8.APPENDIX

A. Source code

Attach the code for the solution built

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

plt.style.use('fivethirtyeight')
%matplotlib inline
```

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

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd

```
df.shape
```

```
(400, 26)
```

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

```
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	...	packed_cell_volume	white_blood_cell_count
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	44	7800
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	38	6000
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	31	7500
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	32	6700
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	35	7300

5 rows × 25 columns

```
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
 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
 14  haemoglobin               348 non-null    float64
 15  packed_cell_volume        330 non-null    object 
 16  white_blood_cell_count    295 non-null    object 
 17  red_blood_cell_count      270 non-null    object 
 18  hypertension              398 non-null    object 
 19  diabetes_mellitus         398 non-null    object 
 20  coronary_artery_disease   398 non-null    object 
 21  appetite                  399 non-null    object 
 22  peda_edema                399 non-null    object 
 23  aanemia                   399 non-null    object 
 24  class                     400 non-null    object 
dtypes: float64(11), object(14)
memory usage: 78.2+ KB
```

```
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
 14  haemoglobin               348 non-null    float64
 15  packed_cell_volume        329 non-null    float64
 16  white_blood_cell_count    294 non-null    float64
 17  red_blood_cell_count      269 non-null    float64
 18  hypertension              398 non-null    object 
 19  diabetes_mellitus         398 non-null    object 
 20  coronary_artery_disease   398 non-null    object 
 21  appetite                  399 non-null    object 
 22  peda_edema                399 non-null    object 
 23  aanemia                   399 non-null    object 
 24  class                     400 non-null    object 
dtypes: float64(14), object(11)
memory usage: 78.2+ KB
```

```
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']
```

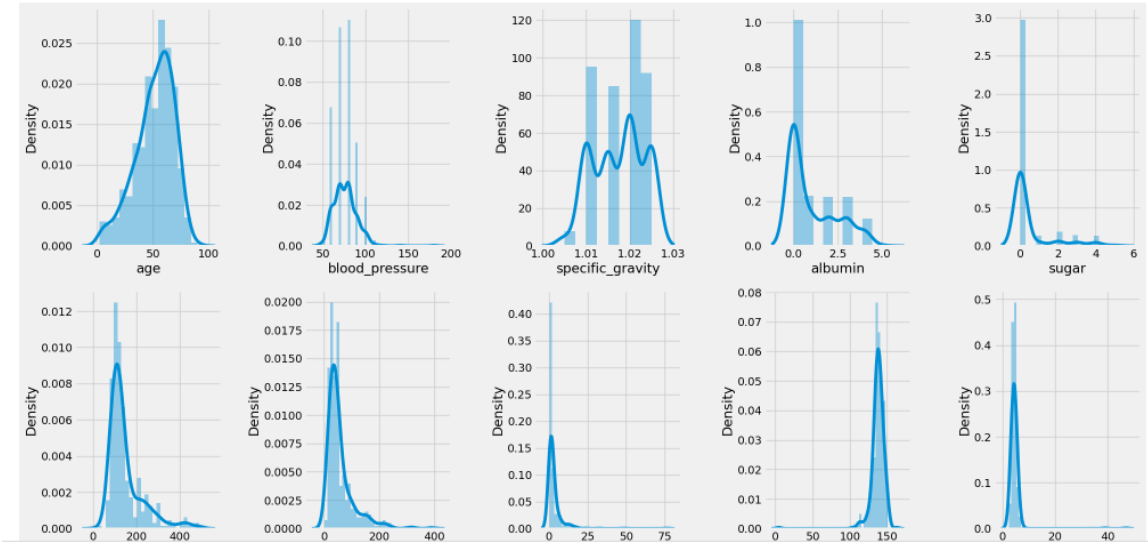
```
for col in cat_cols:
    print(f"{col} has {df[col].unique()} values\n")
```

```
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

class has ['ckd' 'ckd\t' 'notckd'] values
```

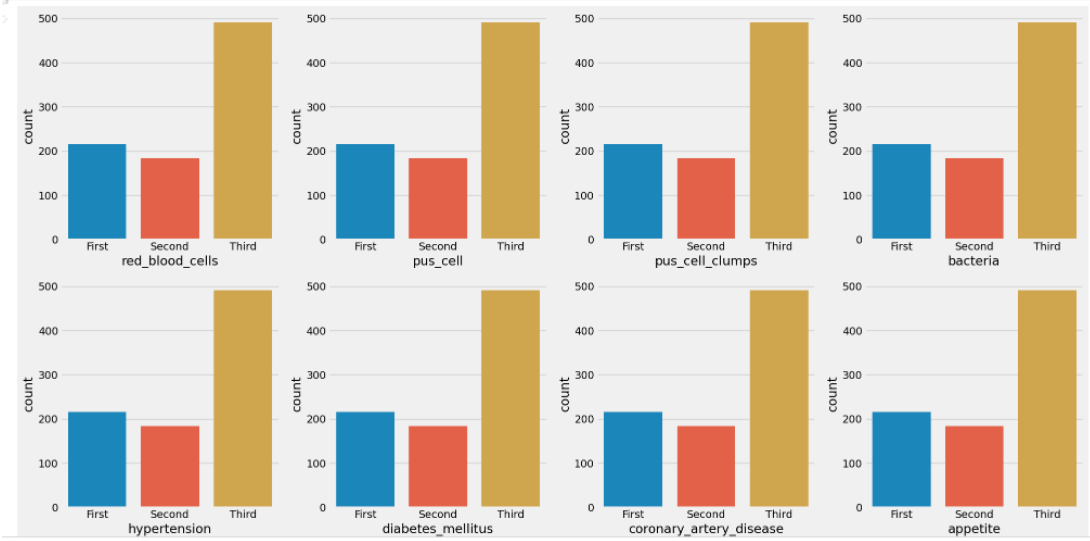
```
[ ]  df['diabetes_mellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes',' yes':'yes'},inplace=True)

     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')
```

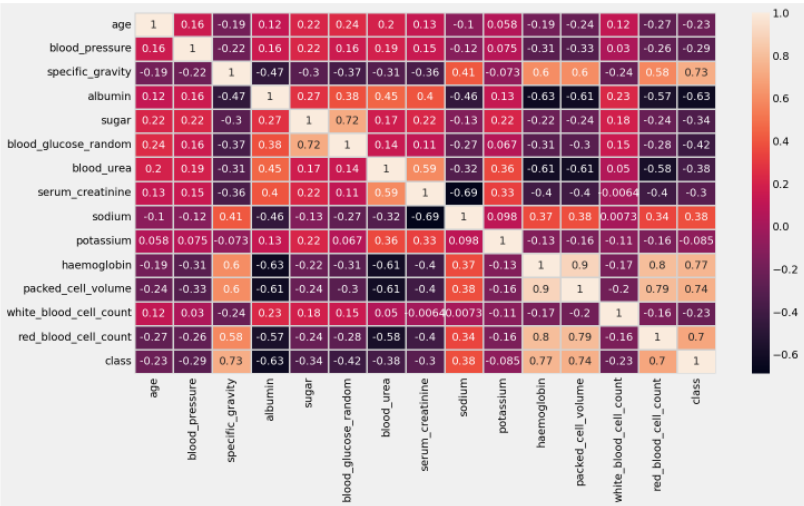
```
[ ]  plt.figure(figsize = (20, 15))
     plotnumber = 1

     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()
```




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

     for column in cat_cols:
         if plotnumber <= 11:
             ax = plt.subplot(3, 4, plotnumber)
             df = sns.load_dataset("titanic")
             sns.countplot(x=df["class"])
             plt.xlabel(column)

         plotnumber += 1

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




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

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

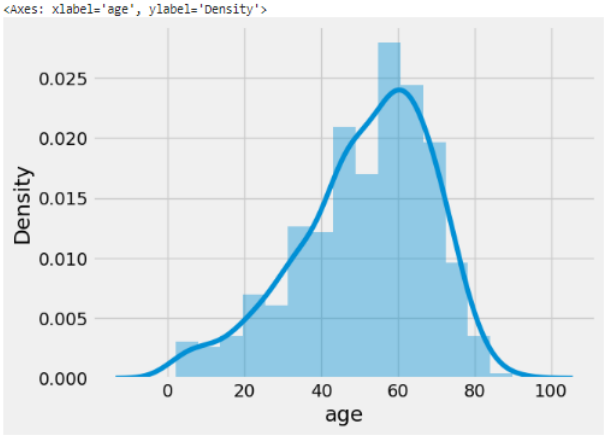



```
[ ]  df.columns
```

```
Index(['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'],
      dtype='object')
```

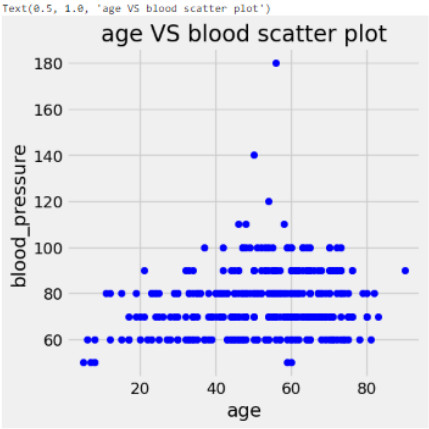
```
[ ] sns.distplot(df.age)
```

```
<Axes: xlabel='age', ylabel='Density'>
```

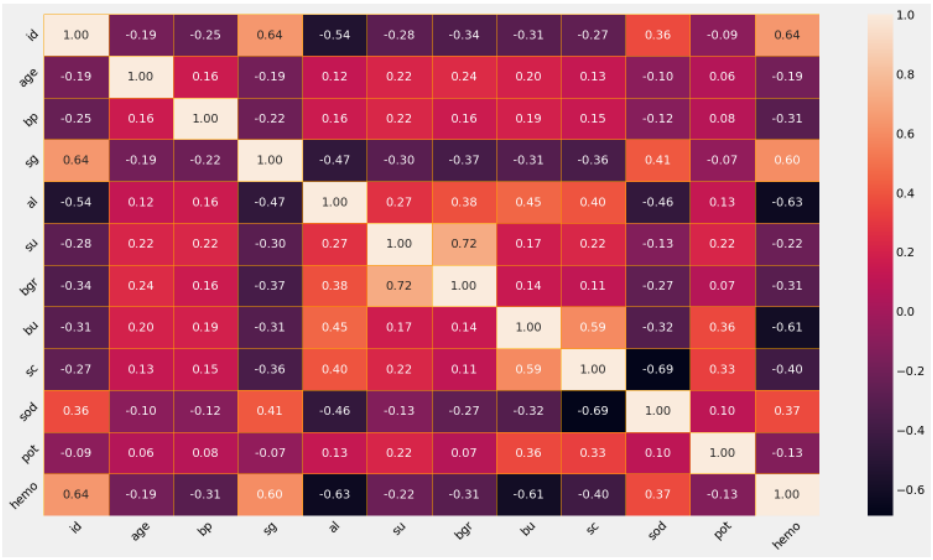



```
[ ] import matplotlib.pyplot as plt
    fig=plt.figure(figsize=(5,5))
    plt.scatter(df['age'],df['bp'],color='blue')
    plt.xlabel('age')
    plt.ylabel('blood_pressure')
    plt.title("age VS blood scatter plot")
```

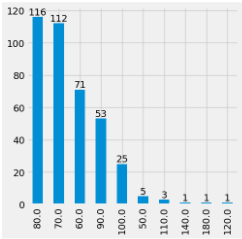
```
Text(0.5, 1.0, 'age VS blood scatter plot')
```




```
[ ] f,ax=plt.subplots(figsize=(18,10))
    sns.heatmap(df.corr(),annot=True,fmt=".2f",ax=ax,linewidths=0.5,linecolor="orange")
    plt.xticks(rotation=45)
    plt.yticks(rotation=45)
    plt.show()
```




```
[ ] plt.figure(figsize=(5,5))
    ax=df.bp.value_counts().plot(kind="bar")
    for i in ax.containers:
      ax.bar_label(i)
```




```
[ ] df.isna().sum().sort_values(ascending = False)
```

```
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
sugar                       49
blood_glucose_random        44
blood_urea                  19
serum_creatinine            17
sodium                      87
potassium                   88
haemoglobin                 52
packed_cell_volume          71
white_blood_cell_count     106
red_blood_cell_count       131
dtype: int64
```

```
[ ] 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
```

```
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)
```

```
[ ] 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
```

```
[ ] 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
```

```
[ ] 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
```

```
[ ] 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	...	packed_cell_volume	white_blood_cell_count	red_blood_cell_count	hypertension	dia
0	48.0	80.0	1.020	1.0	0.0	1	1	0	0	121.0	...	44.0	7800.0	5.2	1	
1	7.0	50.0	1.020	4.0	0.0	1	1	0	0	298.0	...	38.0	6000.0	5.0	0	
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	...	31.0	7500.0	6.2	0	
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	...	32.0	6700.0	3.9	1	
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	...	35.0	7300.0	4.6	0	

5 rows × 25 columns

```
[ ] 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 = 0.30, random_state = 0)
```

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

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
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 0.775
Test Accuracy of KNN is 0.725

Confusion Matrix :-
[[53 19]
 [14 34]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.79      0.74      0.76        72
           1       0.64      0.71      0.67        48

    accuracy                           0.73       120
   macro avg       0.72      0.72      0.72       120
weighted avg       0.73      0.72      0.73       120
```

```
from sklearn.model_selection import GridSearchCV
grid_param = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 10],
    'splitter' : ['best', 'random'],
    'min_samples_leaf' : [1, 2, 3, 5, 7],
    'min_samples_split' : [1, 2, 3, 5, 7],
    'max_features' : ['auto', 'sqrt', 'log2']
}

grid_search_dtc = GridSearchCV(dtc, grid_param, cv = 5, n_jobs = -1, verbose = 1)
grid_search_dtc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 1200 candidates, totalling 6000 fits
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

```
[ ] print(grid_search_dtc.best_params_)
    print(grid_search_dtc.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 3, 'min_samples_split': 5, 'splitter': 'best'}
0.9928571428571429
```

```
[ ] dtc = grid_search_dtc.best_estimator_

    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 0.9964285714285714
Test Accuracy of Decision Tree Classifier is 1.0

Confusion Matrix :-
[[72  0]
 [ 0 48]]

Classification Report :-
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        72
           1       1.00      1.00      1.00        48

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

```
[ ] from sklearn.ensemble import RandomForestClassifier

    rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_features = 'auto', min_samples_leaf = 2, min_samples_split = 3, n_estimators = 130)
    rd_clf.fit(X_train, y_train)

    rd_clf_acc = accuracy_score(y_test, rd_clf.predict(X_test))

    print(f"Training Accuracy of Random Forest Classifier is {accuracy_score(y_train, rd_clf.predict(X_train))}")
    print(f"Test Accuracy of Random Forest Classifier is {rd_clf_acc} \n")

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

```
Training Accuracy of Random Forest Classifier is 1.0
Test Accuracy of Random Forest Classifier is 0.9916666666666667

Confusion Matrix :-
[[71  1]
 [ 0 48]]

Classification Report :-
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        72
           1       0.98      1.00      0.99        48

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120
```

```
[ ] from sklearn.ensemble import AdaBoostClassifier

    ada = AdaBoostClassifier(base_estimator = dtc)
    ada.fit(X_train, y_train)

    ada_acc = accuracy_score(y_test, ada.predict(X_test))

    print(f"Training Accuracy of Ada Boost Classifier is {accuracy_score(y_train, ada.predict(X_train))}")
    print(f"Test Accuracy of Ada Boost Classifier is {ada_acc} \n")

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

```
Training Accuracy of Ada Boost Classifier is 1.0
Test Accuracy of Ada Boost Classifier is 0.9833333333333333

Confusion Matrix :-
[[70  2]
 [ 0 48]]

Classification Report :-
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        72
           1       0.96      1.00      0.98        48

    accuracy                           0.98       120
   macro avg       0.98      0.99      0.98       120
weighted avg       0.98      0.98      0.98       120
```

```
[ ] from sklearn.ensemble import GradientBoostingClassifier

    gb = GradientBoostingClassifier()
    gb.fit(X_train, y_train)

    gb_acc = accuracy_score(y_test, gb.predict(X_test))

    print(f"Training Accuracy of Gradient Boosting Classifier is {accuracy_score(y_train, gb.predict(X_train))}")
    print(f"Test Accuracy of Gradient Boosting Classifier is {gb_acc} \n")

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

```
Training Accuracy of Gradient Boosting Classifier is 1.0
Test Accuracy of Gradient Boosting Classifier is 1.0

Confusion Matrix :-
[[72  0]
 [ 0 48]]

Classification Report :-
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        72
           1       1.00      1.00      1.00        48

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

```
[ ] sgb = GradientBoostingClassifier(max_depth = 4, subsample = 0.90, max_features = 0.75, n_estimators = 200)
    sgb.fit(X_train, y_train)

    sgb_acc = accuracy_score(y_test, sgb.predict(X_test))

    print(f"Training Accuracy of Stochastic Gradient Boosting is {accuracy_score(y_train, sgb.predict(X_train))}")
    print(f"Test Accuracy of Stochastic Gradient Boosting is {sgb_acc} \n")

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

```
Training Accuracy of Stochastic Gradient Boosting is 1.0
Test Accuracy of Stochastic Gradient Boosting is 0.9666666666666667

Confusion Matrix :-
[[71  1]
 [ 3 45]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        72
           1       0.98      0.94      0.96        48

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

```
[ ] from xgboost import XGBClassifier

    xgb = XGBClassifier(objective = 'binary:logistic', learning_rate = 0.5, max_depth = 5, n_estimators = 150)
    xgb.fit(X_train, y_train)

    # accuracy score, confusion matrix and classification report of xgboost

    xgb_acc = accuracy_score(y_test, xgb.predict(X_test))

    print(f"Training Accuracy of XgBoost is {accuracy_score(y_train, xgb.predict(X_train))}")
    print(f"Test Accuracy of XgBoost is {xgb_acc} \n")

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

```
Training Accuracy of XgBoost is 1.0
Test Accuracy of XgBoost is 0.9666666666666667

Confusion Matrix :-
[[71  1]
 [ 3 45]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        72
           1       0.98      0.94      0.96        48

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

```
[ ] from sklearn.ensemble import ExtraTreesClassifier

    etc = ExtraTreesClassifier()
    etc.fit(X_train, y_train)

    etc_acc = accuracy_score(y_test, etc.predict(X_test))

    print(f"Training Accuracy of Extra Trees Classifier is {accuracy_score(y_train, etc.predict(X_train))}")
    print(f"Test Accuracy of Extra Trees Classifier is {etc_acc} \n")

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

```
Training Accuracy of Extra Trees Classifier is 1.0
Test Accuracy of Extra Trees Classifier is 0.9666666666666667

Confusion Matrix :-
[[72  0]
 [ 4 44]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        72
           1       1.00      0.92      0.96        48

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

```
[ ] import pickle
```

```
[ ] from sklearn.linear_model import LogisticRegression
    lgr = LogisticRegression()
```

```
[ ] pickle.dump(lgr,open('CKD.pkl','wb'))
```

```
[ ] from flask import Flask, render_template, request
    import numpy as np
    import pickle
```

```
[ ] app=Flask(__name__)
    model = pickle.load(open('CKD.pkl','rb'))
```

```
[ ] @app.route('/')
    def home():
      return render_template('index.html.html')
```

```
[ ] @app.route('/prediction',methods=['POST','GET'])
    def prediction():
      return render_template('index.html.html')
    @app.route('/Home',methods=['POST','GET'])
    def my_home():
      return render_template('index.html.html')
    @app.route('/predict',methods=['POST'])
    def predict():
      input_features=[float(x) for x in reguest.form.values()]
      features_value=[np.array(input_features)]
      features_name=['blood_urea','blood gulcose random','anemia','cornoary_artery_disease','pus_cell','red_blood_cells',
                     'diabetesmellitus','pedal_edema']
      df=pds.DataFrame(features_value,columns=features_name)
      output = model.predict(df)
      return render_template('index.html.html',prediction_text=output)
```

```
[ ] if(__name__ == '__main__'):
      app.run(debug=True)
```

```
 * Serving Flask app '__main__'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
```