EARLY PREDICTION FOR CHRONIC KIDNEY DISEASE DETECTION: A PROGRESSIVE APPROACH TO HEALTH MANAGEMENT

Submitted by

LEADER

S. KAVYA

TEAM MEMBERS

S. HEMA JANANI

P. MUTHU LAKSHMI

C. SUSMITHA PRICILAL

1 INTROUCTION :

1.1 Overview :

Chronic kidney disease (CKD) is a life-threatening condition that can be difficult to diagnose early because there are no symptoms that when detected the kidney has already lost 25 per cent of its capacity and is under progressive damage. Chronic kidney disease can be detected with regular laboratory tests, and some treatments are present which can prevent development, slow disease progression, reduce complications of decreased Glomerular Filtration Rate (GFR) and risk of cardiovascular disease, and improve survival and quality of life.

CKD can be caused due to lack of water consumption, smoking, improper diet, loss of sleep and many other factors. Majority of the time the disease is detected in its final stage and which sometimes leads to kidney failure.

The main function of the kidney is to filter the blood in the body. Kidney disease is a silent killer because it can cause kidney failure without causing any symptoms or concern. Chronic kidney disease is defined as a decline in kidney function over a period of months or years. Kidney disease is often caused by diabetes and high blood pressure. Chronic kidney disease is a major health problem that affects people worldwide. Not getting the right treatment for chronic kidney disease can have serious consequences, affecting people who can't afford it. Glomerular filtration rate (GFR) is the most accurate test to determine your kidney function and the degree of chronic kidney disease. Blood level, age, gender, and other characteristics can be used to calculate it. In most cases it is better to get sick early. Therefore, it is possible to prevent serious illness.

According to studies, the risk factors for CDK include 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, hemoglobine, packed_cell_volume, white_blood_cell_count, red_blood_cell_count, hypertension, diabetesmellitus, coronary_artery_disease, appetite, pedal_edema, anemia, class on these levels. Thay are separate independent and dependent variables on factors that are red_blood_cells, per_cell, blood_ glucose_randam, blood_urea, pedal_edema, anemia, diabetesmellitus, coronary_artery_disease. And build the model using some multiple algorithms.

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), Logistic Regression 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.

After the model has been developed, it is important to validate its performance using independent datasets. To ensures that the model is accurate, reliable, and generalizable to new data.

Once the model has been validated, it can be deployed and integrated into clinical workflows. This may involve changes to existing processes, such as data collection, storage, and analysis.

By using a progressive approach to early prediction for CKD detection, healthcare providers can improve health management and reduce the burden of CKD. This approach can enable early intervention and treatment, which can slow the progression of the disease and improve patient outcomes. However, it is important to consider the potential limitations and challenges associated with machine learning algorithms, as discussed earlier, and ensure that these models are developed and validated in a responsible and ethical manner.

Keywords: chronic kidney disease(CDK), end-stage disease(ESRD), risk factors, random forest(RF), decision tree(DT).

1.2. PURPOSE:

The purpose of predicting CKD is to evaluate kidney diseases, Which means the need for kidney dialysis or kidney transplant first and this model is to develop and validate predictive models for chronic kidney disease. The main goal will be to evaluate kidney diseases,which means the need for hoe it predict early stages kidney diseases first.

These also teach the patient how to live a healthy life healthy and help the doctor see the risk and severity of the disease, as well as how to proceed with the treatment in the feature. It may possible to identify patterns of data collection using ANN, mining methods, and the future occurrence of certain diseases that may cause harm can be predicted in advance. It is also use to predict whether the patient will suffer or develop chronic kidney disease in the future if they continues their lifestyles. In CKD prediction, the eGFR (glomerular filtration rate) is also helps the doctor for planning the appropriate treatment.

Early prediction and proper treatments can possibly stop, or slow the progression of this chronic disease to end-stage, where dialysis or kidney transplantation is the only way to save patient's life.

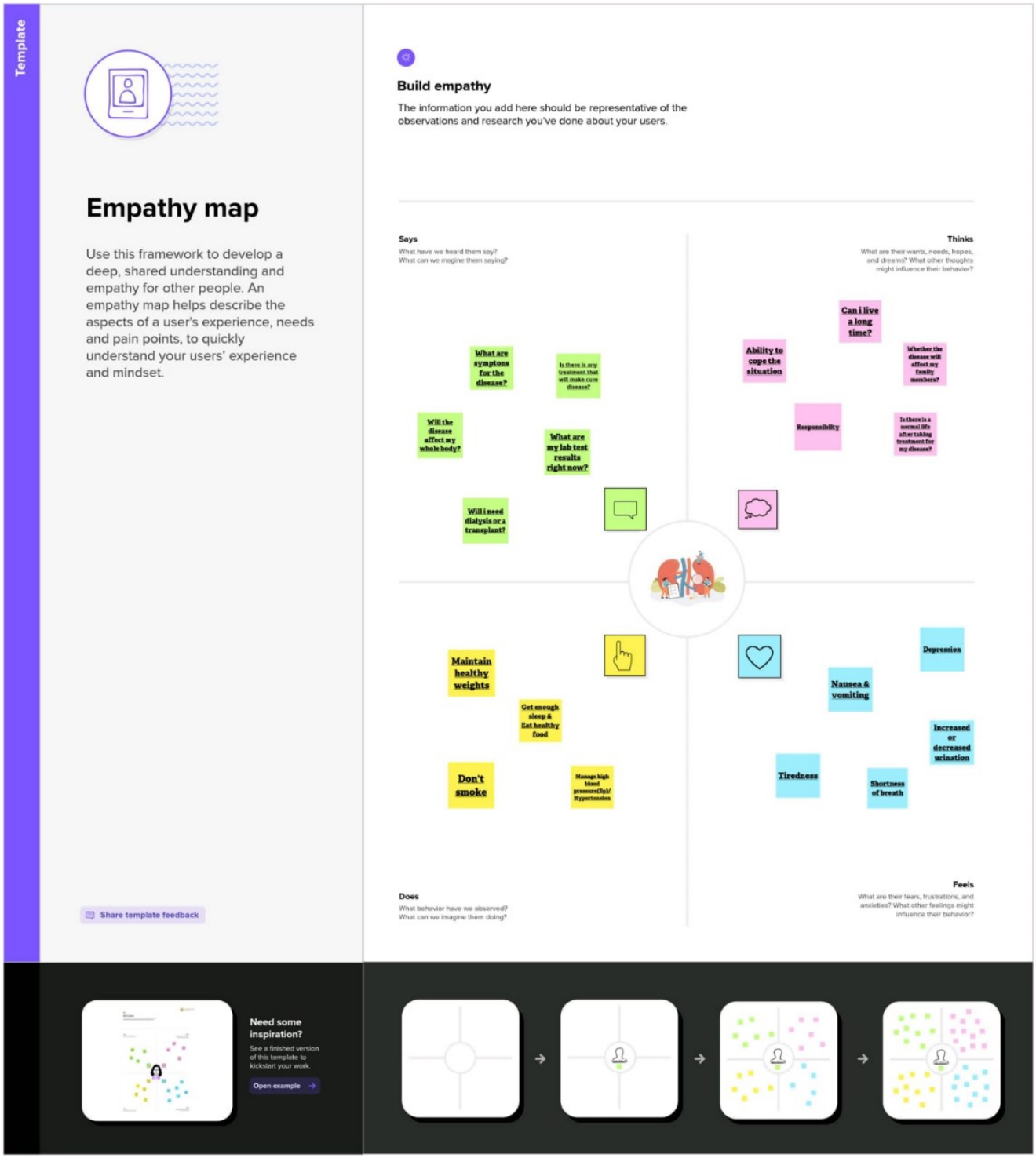
If early stages this project can use pretrained model to predict the Chronic Kidney Disease which can help in treatments of people who are suffer from this diseases**.** This lead to an decreasing burden on the finance and health care systems.

It helps detect trends and find ways to contain the spread of diseases. Using predictive analytics in healthcare can improve the quality of healthcare, collect more clinical data for personalized treatment, and successfully diagnose the medical condition of individual patient.Advanced machine learning approach to build detection models that can receive input data, train a model, and predict the output of future data.

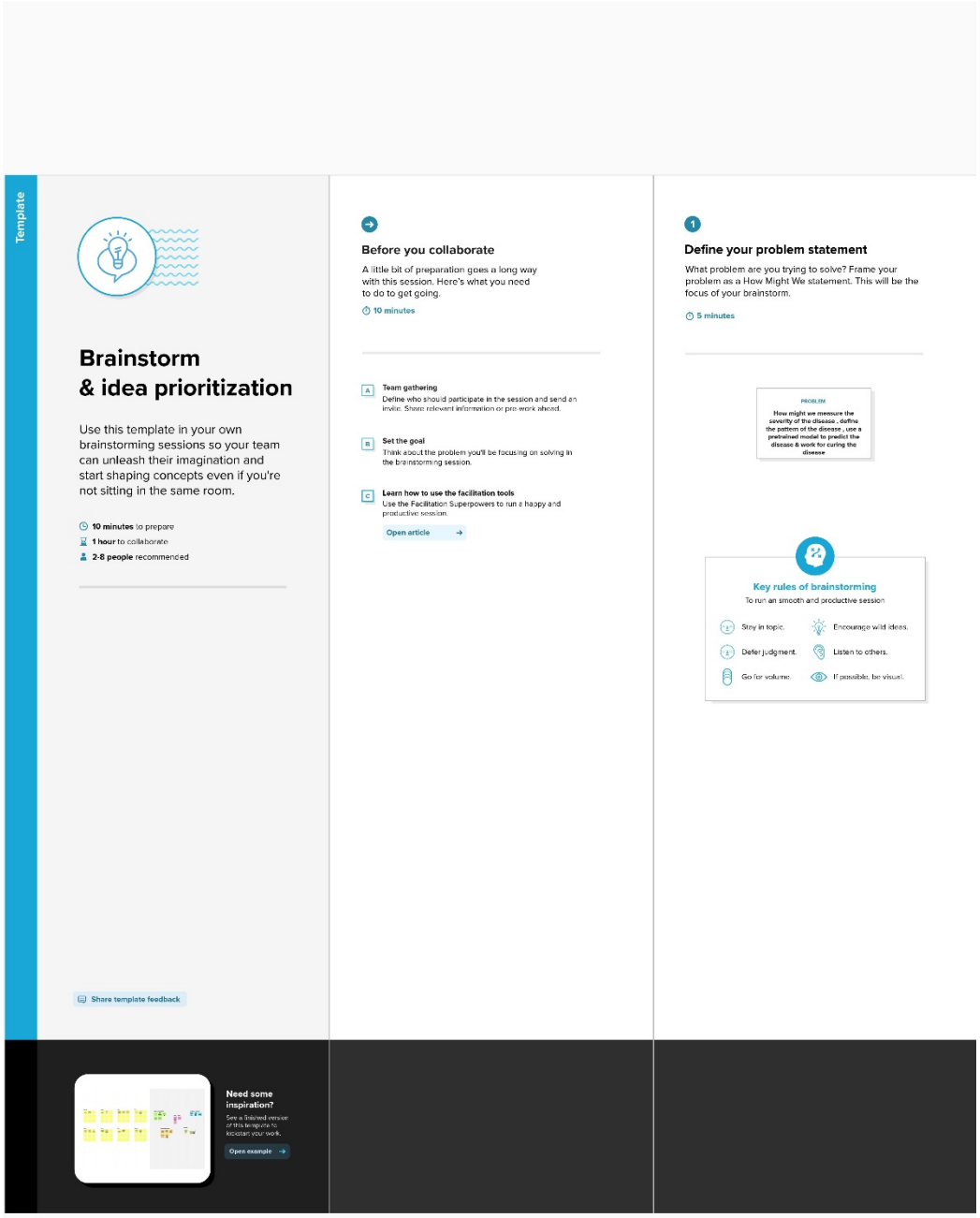
Overall, machine learning has the potential to improve CKD detection and diagnosis by providing accurate predictions and personalized treatment plans. However, it is important to note that these algorithms must be validated using large and diverse datasets before they can be used in clinical practice.

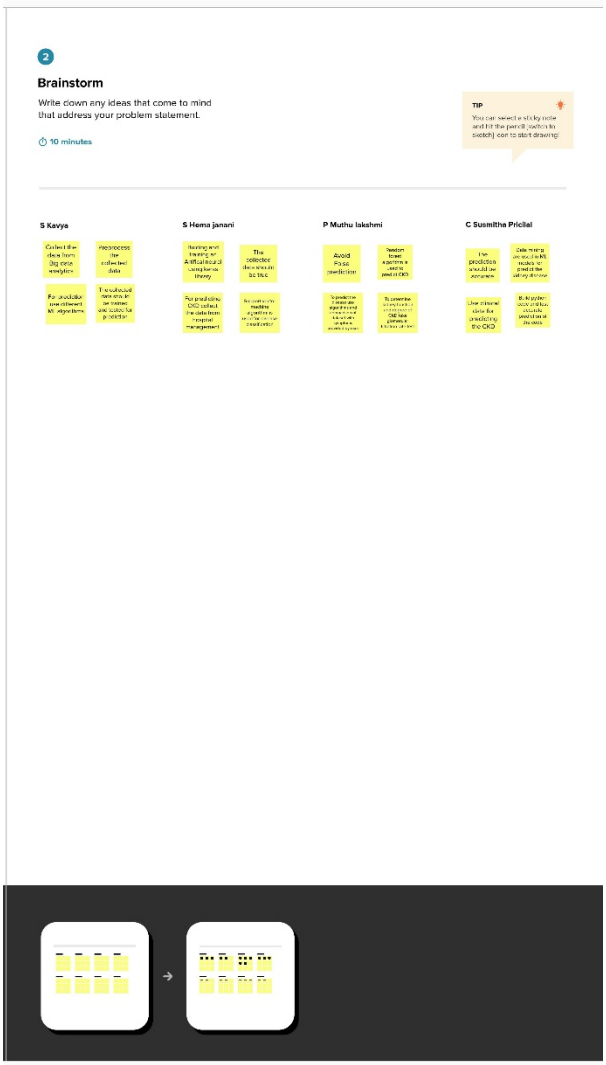
2 Problem Definition & Design Thinking

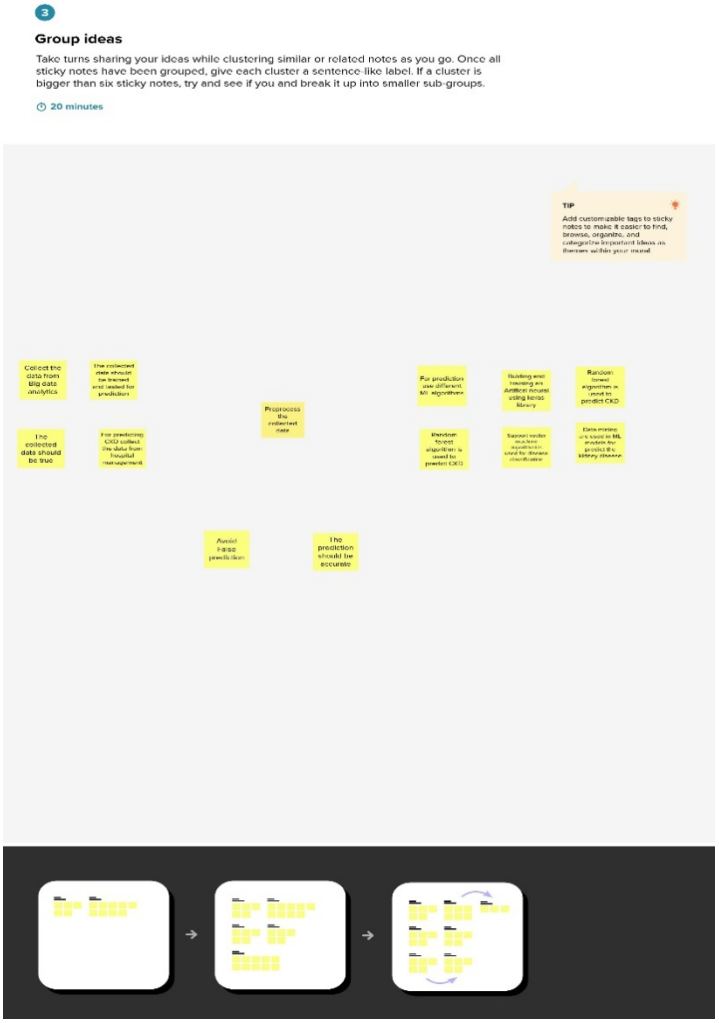
2.1 Empathy Map:



2.2 Ideation& BrainstormingMap







3. RESULT:

Read the Dataset

	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

5 rows × 26 columns

Data preparation

```
data.columns
```

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

```
[4] data.columns=(['Id','Age','Blood_pressure','Specific_gravity','Albumin','Sugar','Red_blood_cells','Pus_cell','Pus_cell_
    data.columns
```

```
Index(['Id', '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', 'Hemoglobin', 'Packed_cell_volume',
       'White_blood_cell_count', 'Red_blood_cell_count', 'Hypertension',
       'Diabetesmellitus', 'Coronary_artery_disease', 'Appetite',
       'Pedal_edema', 'Anemia', 'Class'],
      dtype='object')
```

HANDLING MISSING VALUES

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Id                       400 non-null    int64
 1   Age                      391 non-null    float64
 2   Blood_pressure           388 non-null    float64
 3   Specific_gravity         353 non-null    float64
 4   Albumin                  354 non-null    float64
 5   Sugar                    351 non-null    float64
 6   Red_blood_cells          248 non-null    object
 7   Pus_cell                 335 non-null    object
 8   Pus_cell_clumps          396 non-null    object
 9   Bacteria                 396 non-null    object
 10  Blood glucose random     356 non-null    float64
 11  Blood_urea               381 non-null    float64
 12  Serum_creatinine         383 non-null    float64
 13  Sodium                   313 non-null    float64
 14  Potassium                312 non-null    float64
 15  Hemoglobin               348 non-null    float64
 16  Packed_cell_volume       330 non-null    object
 17  White_blood_cell_count   295 non-null    object
 18  Red_blood_cell_count     270 non-null    object
 19  Hypertension             398 non-null    object
 20  Diabetesmellitus         398 non-null    object
 21  Coronary_artery_disease  398 non-null    object
 22  Appetite                 399 non-null    object
 23  Pedal_edema              399 non-null    object
 24  Anemia                   399 non-null    object
 25  Class                    400 non-null    object
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

Check the columns has any null values

```
[80] data.isnull().any() #it will return true if any columns is having null values
```

```
Id                        False
Age                        True
Blood_pressure             True
Specific_gravity           True
Albumin                    True
Sugar                      True
Red_blood_cells            True
Pus_cell                   True
Pus_cell_clumps            True
Bacteria                   True
Blood glucose random       True
Blood_urea                 True
Serum_creatinine           True
Sodium                     True
Potassium                  True
Hemoglobin                 True
Packed_cell_volume         True
White_blood_cell_count     True
Red_blood_cell_count       True
Hypertension               True
Diabetesmellitus           True
Coronary_artery_disease    True
Appetite                   True
Pedal_edema                True
Anemia                     True
Class                     False
dtype: bool
```

After handling the missing the values check the column has any null values is there

```
data.isnull().sum()
```

```
Id                        0
Age                       0
Blood_pressure            0
Specific_gravity          0
Albumin                   0
Sugar                     0
Red_blood_cells           0
Pus_cell                  0
Pus_cell_clumps           0
Bacteria                  0
Blood glucose random      0
Blood_urea                0
Serum_creatinine          0
Sodium                    0
Potassium                 0
Hemoglobin                0
Packed_cell_volume        0
White_blood_cell_count    0
Red_blood_cell_count      0
Hypertension              0
Diabetesmellitus          0
Coronary_artery_disease   0
Appetite                  0
Pedal_edema               0
Anemia                    0
Class                     0
dtype: int64
```

Replace unwanted classes in column Class

```
[ ] data['Class'] = data['Class'].map({'ckd': 0, 'not ckd': 1})
    c(data['Class'])
```

```
Counter({0: 250, 1: 150})
```

Handling Categorical columns:

Fetch the object types columns

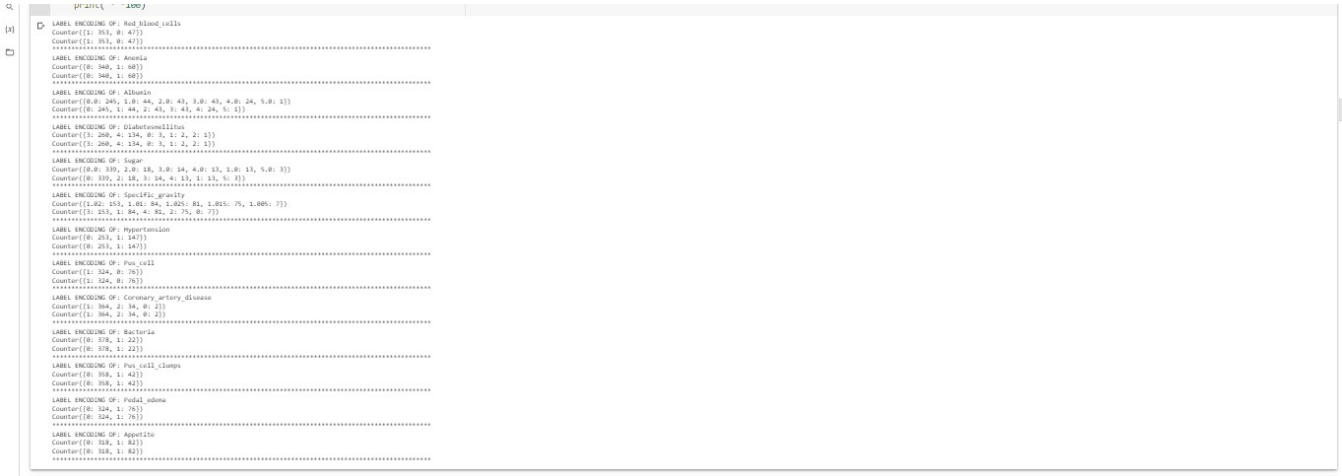
```
#handling categorical columns

catcols=set(data.dtypes[data.dtypes=='O'].index.values) #only fetch object types column
print(catcols)
```

```
{'Red_blood_cells', 'Anemia', 'Red_blood_cell_count', 'Diabetesmellitus', 'Packed_cell_volume', 'Hypertension', 'Pus_cell', 'Coronary_artery_disease', 'Bacteria', 'White_blood_cell_count', 'Pus_cell_clumps', 'Pedal_edema'
```

Check the number of classes in the column

Some column has continuous columns so remove it and add it in a categorical column

Label encoding



Handling Numerical column:

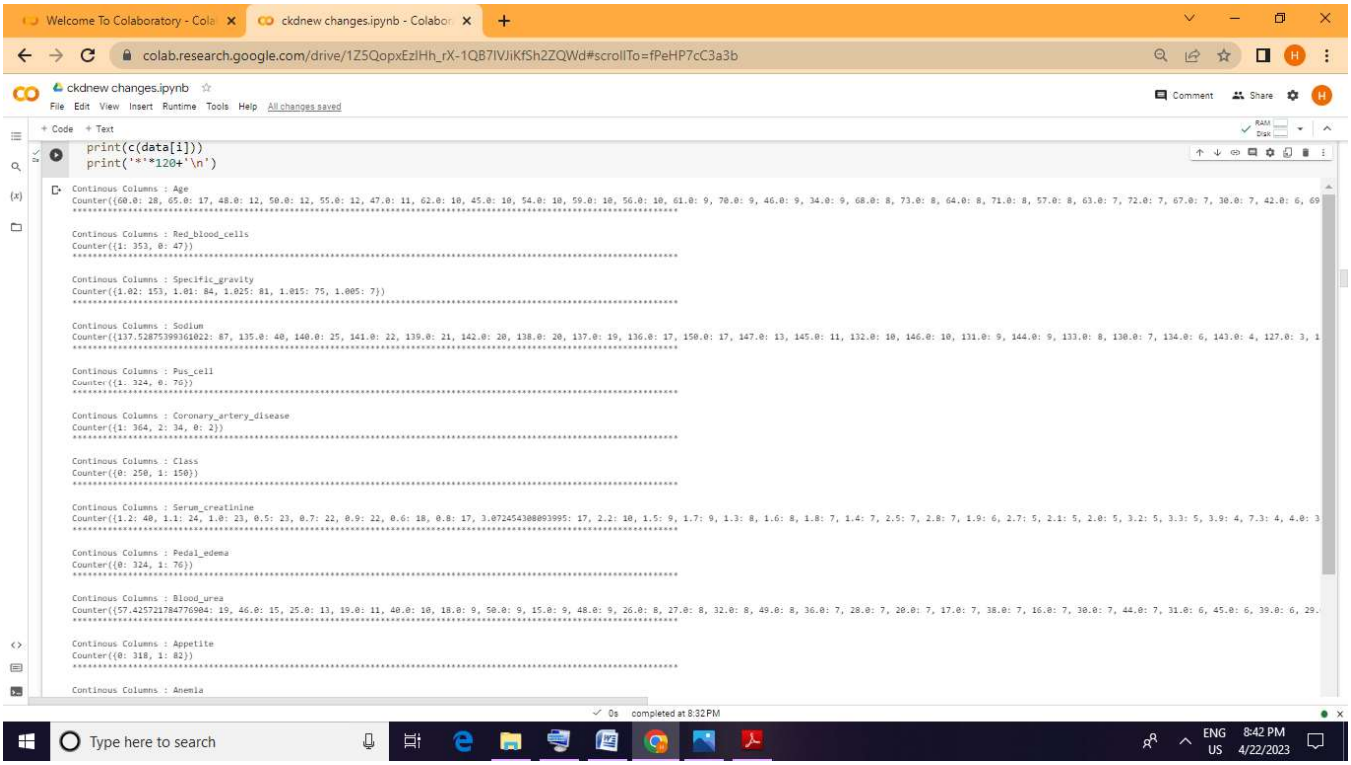
Fetch the text class columns

```
contcols=set(data.dtypes[data.dtypes!='O'].index.values) #only fetch the float and int type columns
print(contcols)
```

```
{'Age', 'Red_blood_cells', 'Specific_gravity', 'Sodium', 'Pus_cell', 'Coronary_artery_disease', 'Class', 'Serum_creatinine', 'Pedal_edema', 'Blood_urea', 'Appetite', 'Anemia', 'Diabetesmellitus', 'Bacteria', 'Hemoglobin', 'Blood_pressure', 'Albumin',
```

```
contcols=set(data.dtypes[data.dtypes!='O'].index.values) #only fetch the float and int type columns
print(contcols)
```

```
disease', 'Class', 'Serum_creatinine', 'Pedal_edema', 'Blood_urea', 'Appetite', 'Anemia', 'Diabetesmellitus', 'Bacteria', 'Hemoglobin', 'Blood_pressure', 'Albumin', 'Sugar', 'Blood glucose random', 'Potassium', 'Hypertension', 'Id', 'Pus_cell_clumps'}
```



Some column has categorical columns so remove it and add it in a continuous column

```
contcols.add('Red_blood_cell_count')
contcols.add('Packed_cell_volume')
contcols.add('White_blood_cell_count')
print(contcols)
```

```
, 'Blood_urea', 'Appetite', 'Anemia', 'Diabetesmellitus', 'Packed_cell_volume', 'Bacteria', 'Hemoglobin', 'White_blood_cell_count', 'Blood_pressure', 'Red_blood_cell_count', 'Blood glucose random', 'Potassium', 'Hypertension', 'Id', 'Pus_cell_clumps'}
```

```
[94] catcols.add('Specific_gravity')
     catcols.add('Albumin')
     catcols.add('Sugar')
     print(catcols)
```

```
{'Red_blood_cells', 'Anenia', 'Albumin', 'Diabetesmellitus', 'Sugar', 'Specific_gravity', 'Hypertension', 'Pus_cell', 'Coronary_artery_disease', 'Bacteria', 'Pus_cell_clumps', 'Pedal_edema', 'Appetite'}
```

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

Replace unwanted classes in some columns so replace

```
[95] data['Diabetesmellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes',' yes':'yes'},inplace=True)
     c(data['Diabetesmellitus'])
```

```
Counter({4: 134, 3: 260, 2: 1, 0: 3, 1: 2})
```

```
[96] data['Coronary_artery_disease'] = data['Coronary_artery_disease'].replace(to_replace = '\tno', value='no')
     c(data['Coronary_artery_disease'])
```

```
Counter({1: 364, 2: 34, 0: 2})
```

Exploratory Data Analysis:

Descriptive statistical analysis

```
#Exploratory Data Analysis
 #Descriptive statistical Analysis


data.describe() #computes summary values for continuous column data
```

	Id	Age	Blood_pressure	Specific_gravity	Albumin	Sugar	Red_blood_cells	Pus_cell	Pus_cell_clumps	Bacteria	...	Sodium	Potassium	Hemoglobin	Hypertension	Diabetesmellitus	Coronary_artery_disease	Appetite	P
count	400.000000	400.000000	400.000000	400.000000	400.00000	400.000000	400.000000	400.000000	400.000000	400.000000	...	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	
mean	199.500000	51.875000	76.469072	1.017712	0.90000	0.395000	0.882500	0.810000	0.105000	0.055000	...	137.528754	4.627244	12.526437	0.387500	3.300000	1.080000	0.205000	
std	115.614301	17.022008	13.475298	0.005434	1.31313	1.040038	0.322418	0.392792	0.306937	0.228266	...	9.204273	2.819783	2.716171	0.482728	0.579517	0.289499	0.404207	
min	0.000000	2.000000	50.000000	1.005000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	...	4.500000	2.500000	3.100000	0.000000	0.000000	0.000000	0.000000	
25%	99.750000	42.000000	70.000000	1.015000	0.00000	0.000000	1.000000	1.000000	0.000000	0.000000	...	135.000000	4.000000	10.875000	0.000000	3.000000	1.000000	0.000000	
50%	199.500000	55.000000	78.234536	1.020000	0.00000	0.000000	1.000000	1.000000	0.000000	0.000000	...	137.528754	4.627244	12.526437	0.000000	3.000000	1.000000	0.000000	
75%	299.250000	64.000000	80.000000	1.020000	2.00000	0.000000	1.000000	1.000000	0.000000	0.000000	...	141.000000	4.800000	14.625000	1.000000	4.000000	1.000000	0.000000	
max	399.000000	90.000000	180.000000	1.025000	5.00000	5.000000	1.000000	1.000000	1.000000	1.000000	...	163.000000	47.000000	17.800000	1.000000	4.000000	2.000000	1.000000	

8 rows × 23 columns

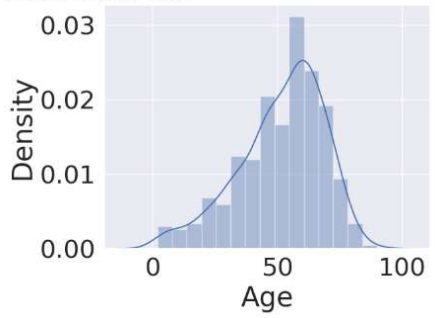
Visual analysis:

Univariate Analysis

The term **univariate analysis** refers to the analysis of one variable

```
[101] #visual analysis
      # 1.Univariate analysis
      #AGE DISTRIBUTION


      sns.distplot(data.Age)
```

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

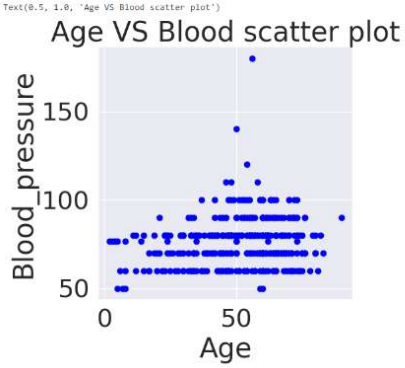


Bivariate analysis

The analysis of two variables.

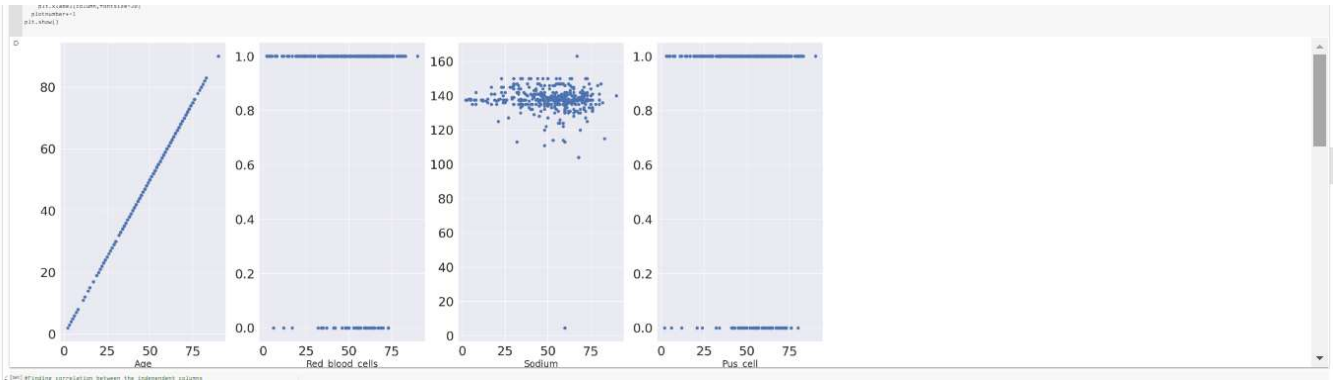
```
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(5,5)) #plot size
plt.scatter(data['Age'],data['Blood_pressure'],color='blue')
plt.xlabel('Age') #set the label for x axis
plt.ylabel('Blood_pressure') #set the labelfor y axis
plt.title("Age VS Blood scatter plot") #Set a title for the axes
```

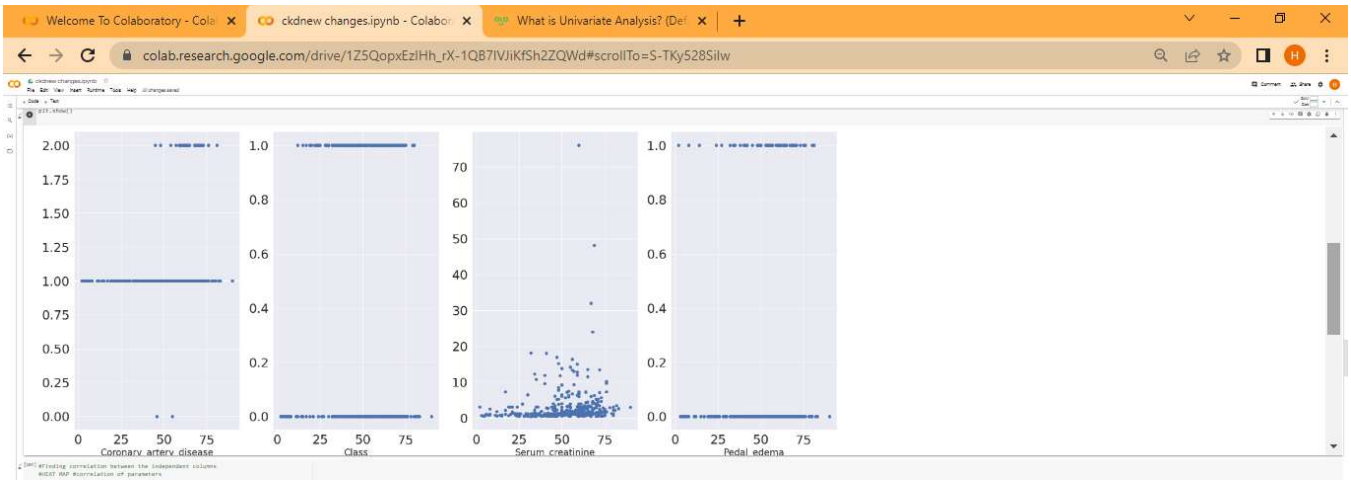
```
Text(0.5, 1.0, 'Age VS Blood scatter plot')
```

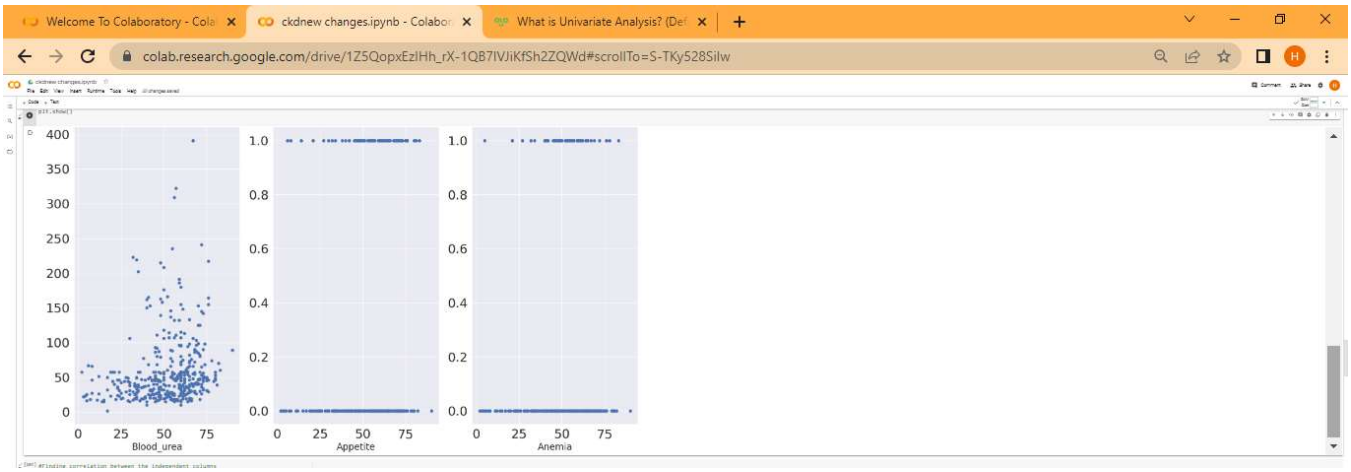


Multivariate analysis

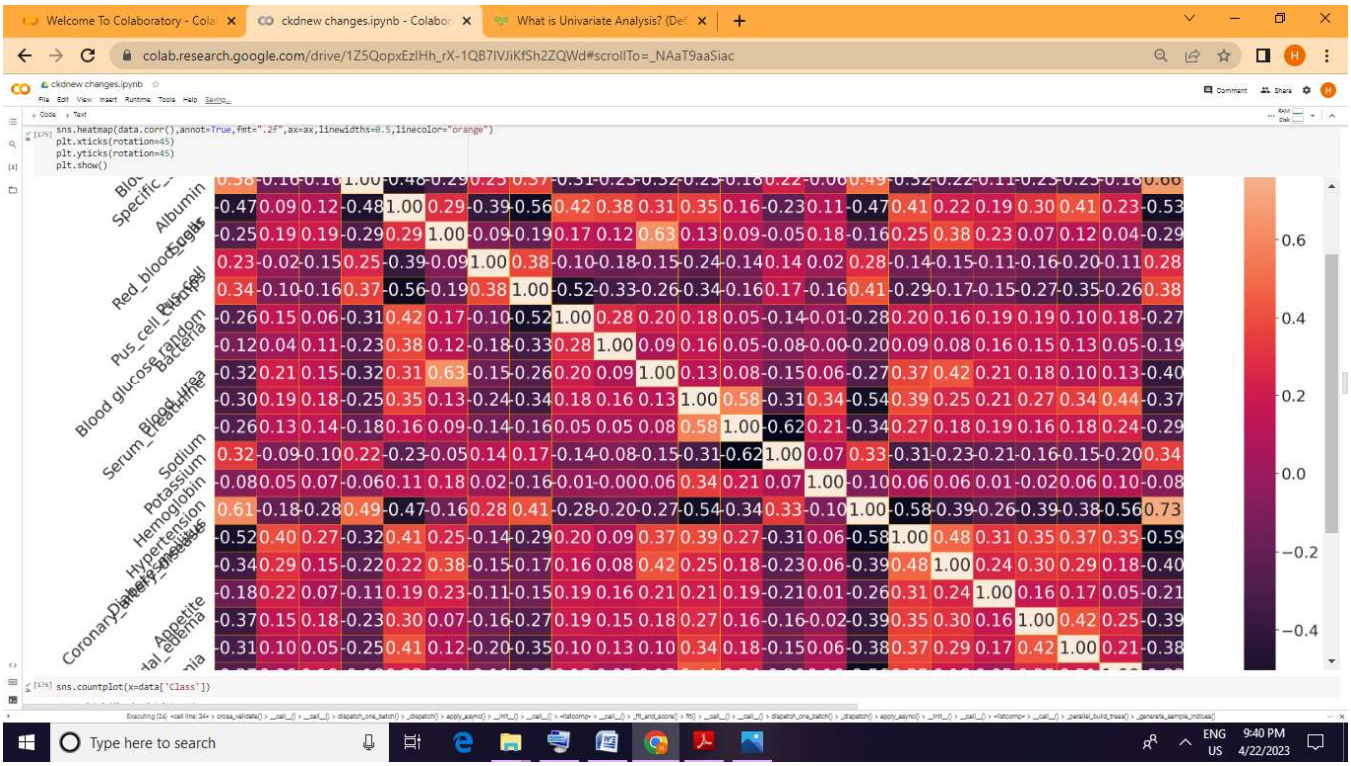
The analysis of two or more variables.



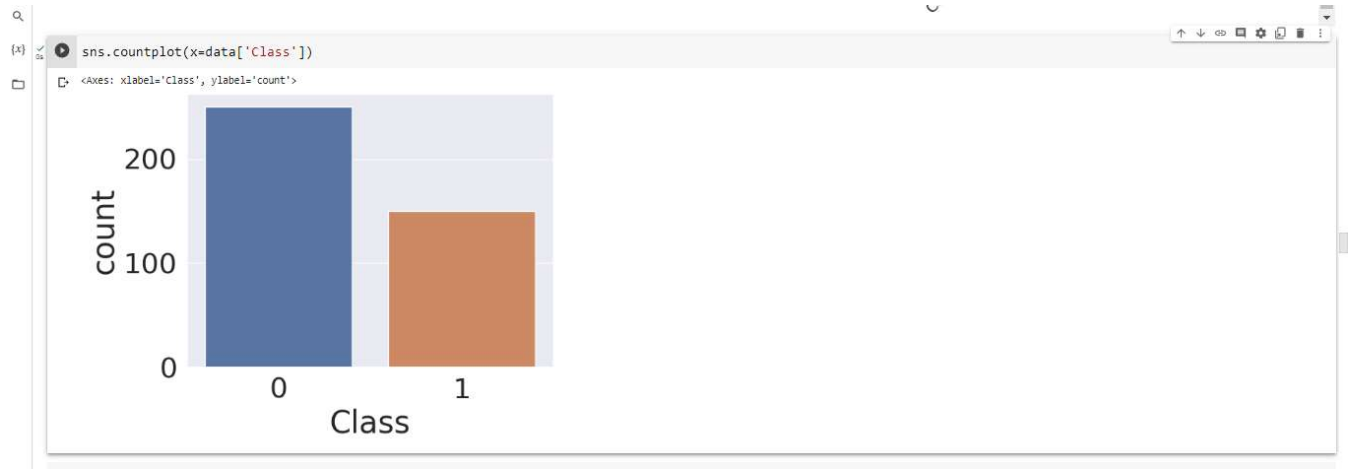




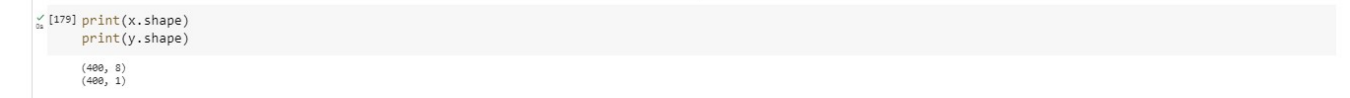
Finding correlation between the independent Columns



The count of target data classes, by using seaborn countplot



Create independent and Dependent



Model building:

ANN Model

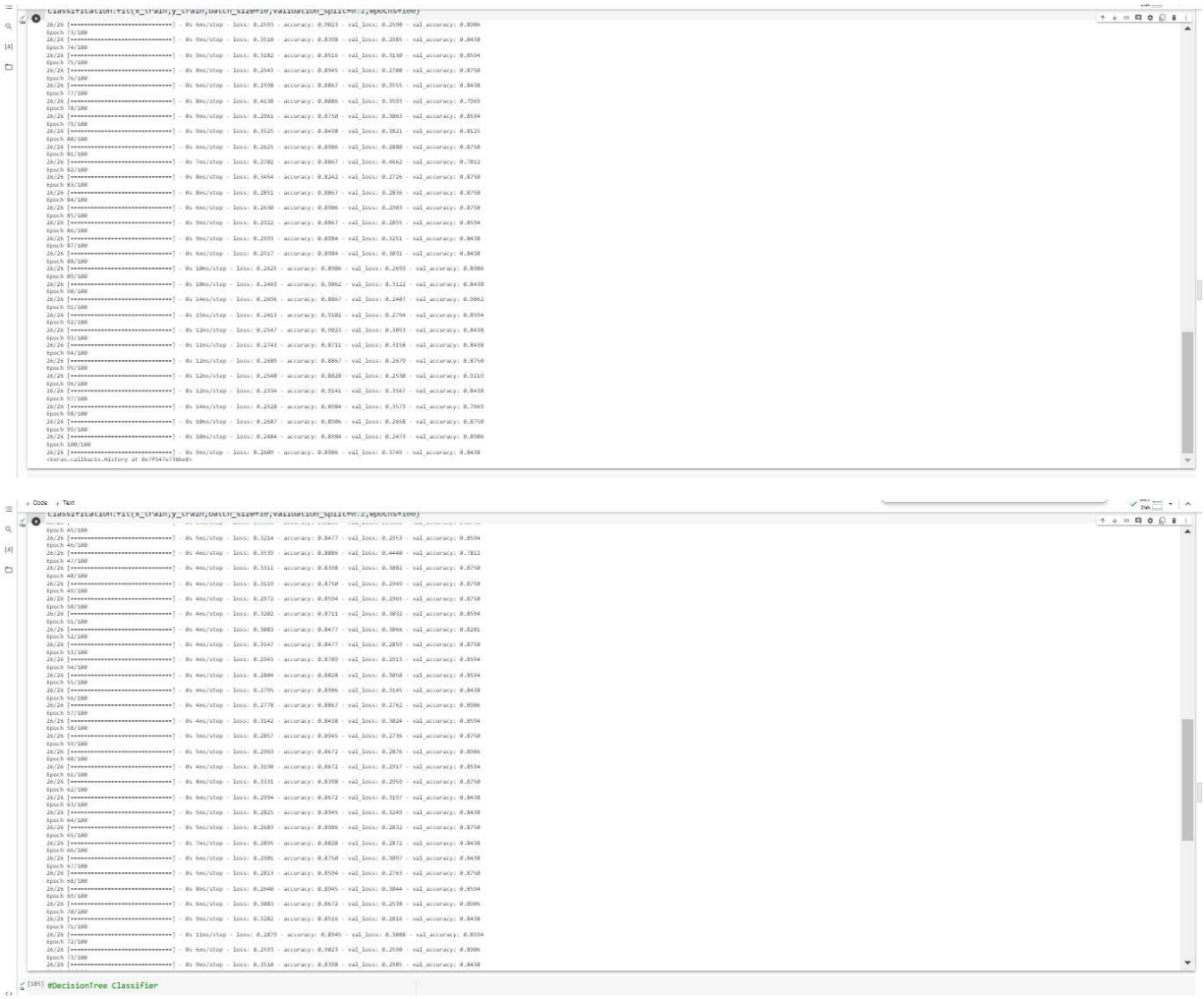
Training the model

```
classification.fit(x_train,y_train,batch_size=10,validation_split=0.2,epochs=100)
```

```
Epoch 1/100
26/26 [==============================] - 1s 10ms/step - loss: 0.6933 - accuracy: 0.6289 - val_loss: 0.5562 - val_accuracy: 0.5938
Epoch 2/100
26/26 [==============================] - 0s 5ms/step - loss: 0.7493 - accuracy: 0.6172 - val_loss: 0.5713 - val_accuracy: 0.5938
Epoch 3/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5477 - accuracy: 0.6406 - val_loss: 0.4904 - val_accuracy: 0.7344
Epoch 4/100
26/26 [==============================] - 0s 4ms/step - loss: 0.5924 - accuracy: 0.6250 - val_loss: 0.4803 - val_accuracy: 0.6562
Epoch 5/100
26/26 [==============================] - 0s 4ms/step - loss: 0.5303 - accuracy: 0.6602 - val_loss: 0.4968 - val_accuracy: 0.7500
Epoch 6/100
26/26 [==============================] - 0s 4ms/step - loss: 0.6332 - accuracy: 0.6445 - val_loss: 0.5463 - val_accuracy: 0.7188
Epoch 7/100
26/26 [==============================] - 0s 6ms/step - loss: 0.6531 - accuracy: 0.6367 - val_loss: 0.9598 - val_accuracy: 0.5625
Epoch 8/100
26/26 [==============================] - 0s 6ms/step - loss: 0.6681 - accuracy: 0.5938 - val_loss: 0.6142 - val_accuracy: 0.5938
Epoch 9/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5550 - accuracy: 0.6641 - val_loss: 0.4993 - val_accuracy: 0.7188
Epoch 10/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5120 - accuracy: 0.6797 - val_loss: 0.5869 - val_accuracy: 0.5938
Epoch 11/100
26/26 [==============================] - 0s 5ms/step - loss: 0.5301 - accuracy: 0.6914 - val_loss: 0.4670 - val_accuracy: 0.6094
Epoch 12/100
26/26 [==============================] - 0s 6ms/step - loss: 0.4740 - accuracy: 0.6992 - val_loss: 0.4436 - val_accuracy: 0.7812
Epoch 13/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4773 - accuracy: 0.6875 - val_loss: 0.4329 - val_accuracy: 0.7500
Epoch 14/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4590 - accuracy: 0.7578 - val_loss: 0.4329 - val_accuracy: 0.8438
Epoch 15/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4573 - accuracy: 0.7812 - val_loss: 0.4297 - val_accuracy: 0.8438
Epoch 16/100
26/26 [==============================] - 0s 7ms/step - loss: 0.4452 - accuracy: 0.7812 - val_loss: 0.4291 - val_accuracy: 0.7969
Epoch 17/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5286 - accuracy: 0.6836 - val_loss: 0.4690 - val_accuracy: 0.6094
Epoch 18/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4710 - accuracy: 0.7500 - val_loss: 0.4161 - val_accuracy: 0.8281
Epoch 19/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4348 - accuracy: 0.7656 - val_loss: 0.4067 - val_accuracy: 0.8438
Epoch 20/100
26/26 [==============================] - 0s 6ms/step - loss: 0.4572 - accuracy: 0.7500 - val_loss: 0.5387 - val_accuracy: 0.7188
Epoch 21/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4402 - accuracy: 0.7734 - val_loss: 0.4010 - val_accuracy: 0.7969
Epoch 22/100
26/26 [==============================] - 0s 6ms/step - loss: 0.4163 - accuracy: 0.7969 - val_loss: 0.3889 - val_accuracy: 0.8438
Epoch 23/100
```

```
classification.fit(x_train,y_train,batch_size=10,validation_split=0.2,epochs=100)
```

```
Epoch 22/100
26/26 [==============================] - 0s 6ms/step - loss: 0.4163 - accuracy: 0.7969 - val_loss: 0.3889 - val_accuracy: 0.8438
Epoch 23/100
26/26 [==============================] - 0s 5ms/step - loss: 0.3936 - accuracy: 0.8477 - val_loss: 0.4124 - val_accuracy: 0.7188
Epoch 24/100
26/26 [==============================] - 0s 6ms/step - loss: 0.4273 - accuracy: 0.7812 - val_loss: 0.4676 - val_accuracy: 0.7656
Epoch 25/100
26/26 [==============================] - 0s 6ms/step - loss: 0.5264 - accuracy: 0.7227 - val_loss: 0.3439 - val_accuracy: 0.8750
Epoch 26/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4406 - accuracy: 0.7578 - val_loss: 0.4024 - val_accuracy: 0.7969
Epoch 27/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3794 - accuracy: 0.8359 - val_loss: 0.3598 - val_accuracy: 0.8125
Epoch 28/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4062 - accuracy: 0.7891 - val_loss: 0.3829 - val_accuracy: 0.8281
Epoch 29/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4458 - accuracy: 0.7617 - val_loss: 0.3721 - val_accuracy: 0.8438
Epoch 30/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4127 - accuracy: 0.7812 - val_loss: 0.3900 - val_accuracy: 0.8125
Epoch 31/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3900 - accuracy: 0.8125 - val_loss: 0.3635 - val_accuracy: 0.8281
Epoch 32/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4819 - accuracy: 0.7070 - val_loss: 0.5357 - val_accuracy: 0.7500
Epoch 33/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3697 - accuracy: 0.8242 - val_loss: 0.3318 - val_accuracy: 0.8594
Epoch 34/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3640 - accuracy: 0.8281 - val_loss: 0.3605 - val_accuracy: 0.7812
Epoch 35/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3591 - accuracy: 0.8359 - val_loss: 0.3117 - val_accuracy: 0.8906
Epoch 36/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3338 - accuracy: 0.8633 - val_loss: 0.3335 - val_accuracy: 0.8125
Epoch 37/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3353 - accuracy: 0.8594 - val_loss: 0.2996 - val_accuracy: 0.8750
Epoch 38/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3823 - accuracy: 0.8438 - val_loss: 0.3332 - val_accuracy: 0.8438
Epoch 39/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3974 - accuracy: 0.7734 - val_loss: 0.3220 - val_accuracy: 0.8594
Epoch 40/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4257 - accuracy: 0.7539 - val_loss: 0.3135 - val_accuracy: 0.8438
Epoch 41/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3444 - accuracy: 0.8594 - val_loss: 0.3468 - val_accuracy: 0.8438
Epoch 42/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3186 - accuracy: 0.8672 - val_loss: 0.4064 - val_accuracy: 0.7969
Epoch 43/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4012 - accuracy: 0.7930 - val_loss: 0.3579 - val_accuracy: 0.8125
Epoch 44/100
```



DecisionTreeClassifier

Fit x_train,y_train

```
#DecisionTree Classifier

from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier(max_depth=4,splitter='best',criterion='entropy')
dtc.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```
[186] y_predict=dtc.predict(x_test)
      y_predict
```

```
array([0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1])
```

Random ForestClassifier

Fit x_train,y_train

```
[259] rfc.fit(x_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

Logistic regression

Fit x_train,y_train

```
[262] #Logistic regression

      from sklearn.linear_model import LogisticRegression
      lgr=LogisticRegression()
      lgr.fit(x_train,y_train)
```

```
LogisticRegression
LogisticRegression()
```

Test the model:

```
[334] #testing the model
      #Logistic regression

      y_pred=lgr.predict([[1,1,121.000000,36.0,0,0,1,0]])
      print(y_pred)
```

```
[1]
```

```
#DecisionTree Classifier

y_pred=dtc.predict([[1,1,121.000000,36.0,0,0,1,0]])
print(y_pred)
```

```
[1]
```

```
[336] #random forest classifier

      y_pred=rfc.predict([[1,1,121.000000,36.0,0,0,1,0]])

      print(y_pred)
      (y_pred)
```

```
[1]
array([1])
```

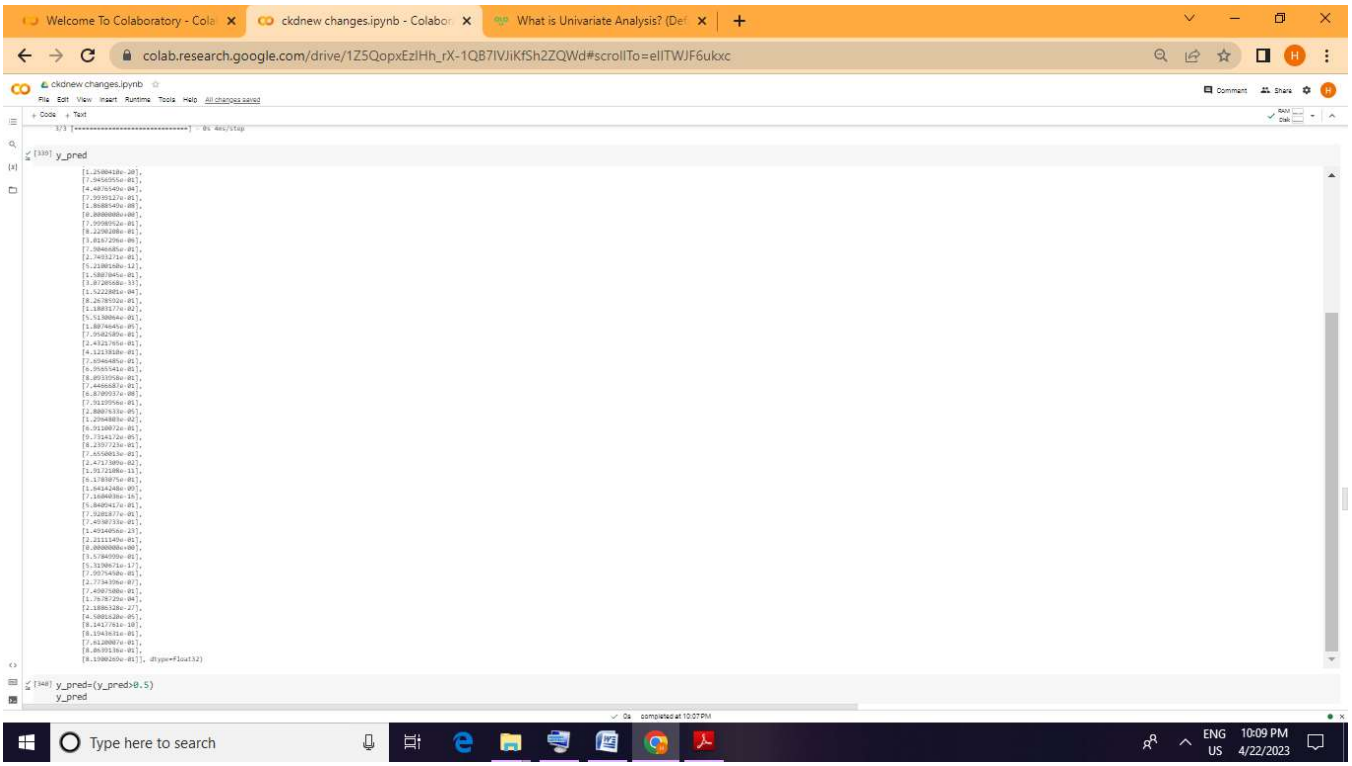
```
array([1])
```

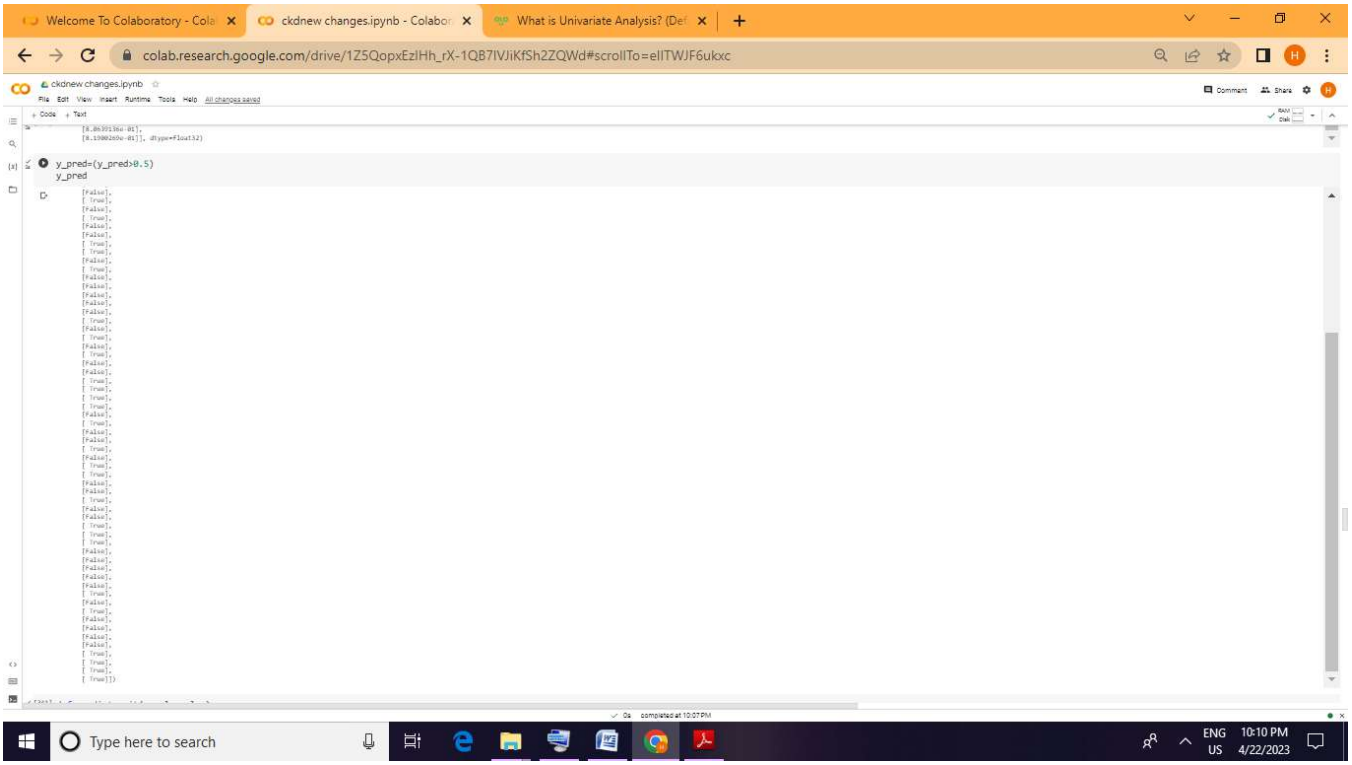
```
classification.save("ckd.h5")   #save the model to test the input
```

```
[338] y_pred=classification.predict(x_test)
```

```
3/3 [==============================] - 0s 4ms/step
```

```
[339] y_pred
```


Welcome To Colaboratory - Cola
ckdnew changes.ipynb - Colabor
What is Univariate Analysis? (Def
colab.research.google.com/drive/1Z5QopxEzlHh_rX-1QB7IVJiKfSh2ZQWd#scrollTo=ellTWJF6ukxc
ckdnew changes.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
3/3 [==============================] - 0s 4ms/step
[330] y_pred
[348] y_pred=(y_pred>0.5)
y_pred
completed at 10:07 PM
Type here to search
ENG
US
10:09 PM
4/22/2023


Welcome To Colaboratory - Cola
ckdnew changes.ipynb - Colabor
What is Univariate Analysis? (Def
colab.research.google.com/drive/1Z5QopxEzlHh_rX-1QB7IVJiKfSh2ZQWd#scrollTo=ellTWJF6ukxc
ckdnew changes.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
y_pred=(y_pred>0.5)
y_pred
completed at 10:07 PM
Type here to search
ENG
US
10:10 PM
4/22/2023

```
[342] test=classification.predict([[1,1,121.000000,36.0,0,0,1,0]])
      if test==1:
        print('Prediction:High chance of ckd!')
      else:
        print('Prediction:Low chance of ckd')

      1/1 [==============================] - 0s 97ms/step
      Prediction:Low chance of ckd
```

Compare the model

```
[343]
          accuracy                           0.90       80
         macro avg       0.89      0.91      0.89       80
      weighted avg       0.91      0.90      0.90       80

      RF
                    precision    recall  f1-score   support

            NO CKD       0.96      0.92      0.94        52
               CKD       0.87      0.93      0.90        28

          accuracy                           0.93       80
         macro avg       0.91      0.93      0.92       80
      weighted avg       0.93      0.93      0.93       80

      DecisionTree
                    precision    recall  f1-score   support

            NO CKD       0.89      0.90      0.90        52
               CKD       0.81      0.79      0.80        28

          accuracy                           0.86       80
         macro avg       0.85      0.84      0.85       80
      weighted avg       0.86      0.86      0.86       80

          fit_time  score_time  test_accuracy  test_precision_weighted  \
      0   0.021826    0.030192       0.921875                 0.936343
      1   0.017584    0.027314       0.906250                 0.914062
      2   0.017719    0.028745       0.906250                 0.908775
      3   0.018189    0.029328       0.843750                 0.880456
      4   0.017540    0.028220       0.828125                 0.827214
      5   0.175050    0.056667       0.968750                 0.968750
      6   0.165188    0.057749       0.921875                 0.926503
      7   0.165749    0.049265       0.890625                 0.890467
      8   0.162104    0.048090       0.921875                 0.937500
      9   0.161900    0.048829       0.906250                 0.906250
      10  0.004289    0.028576       0.937500                 0.938068
      11  0.003214    0.026674       0.859375                 0.860283
      12  0.003163    0.030882       0.859375                 0.859027
      13  0.003265    0.027556       0.859375                 0.861642
      14  0.003505    0.026267       0.859375                 0.858724
```

```
[343]  macro avg       0.85      0.84      0.85       80
       weighted avg       0.86      0.86      0.86       80

          fit_time  score_time  test_accuracy  test_precision_weighted  \
      0   0.021826    0.030192       0.921875                 0.936343
      1   0.017584    0.027314       0.906250                 0.914062
      2   0.017719    0.028745       0.906250                 0.908775
      3   0.018189    0.029328       0.843750                 0.880456
      4   0.017540    0.028220       0.828125                 0.827214
      5   0.175050    0.056667       0.968750                 0.968750
      6   0.165188    0.057749       0.921875                 0.926503
      7   0.165749    0.049265       0.890625                 0.890467
      8   0.162104    0.048090       0.921875                 0.937500
      9   0.161900    0.048829       0.906250                 0.906250
      10  0.004289    0.028576       0.937500                 0.938068
      11  0.003214    0.026674       0.859375                 0.860283
      12  0.003163    0.030882       0.859375                 0.859027
      13  0.003265    0.027556       0.859375                 0.861642
      14  0.003505    0.026267       0.859375                 0.858724

          test_recall_weighted  test_f1_weighted  test_roc_auc         model
      0               0.921875          0.923389      0.968615        LogReg
      1               0.906250          0.906618      0.931548        LogReg
      2               0.906250          0.906622      0.944945        LogReg
      3               0.843750          0.848958      0.929545        LogReg
      4               0.828125          0.827459      0.916410        LogReg
      5               0.968750          0.968750      0.997835            RF
      6               0.921875          0.922163      0.953869            RF
      7               0.890625          0.890322      0.957958            RF
      8               0.921875          0.923862      0.931250            RF
      9               0.906250          0.906250      0.967692            RF
      10              0.937500          0.936739      0.919913  DecisionTree
      11              0.859375          0.859618      0.895833  DecisionTree
      12              0.859375          0.858986      0.860360  DecisionTree
      13              0.859375          0.860282      0.843182  DecisionTree
      14              0.859375          0.858830      0.848718  DecisionTree
```

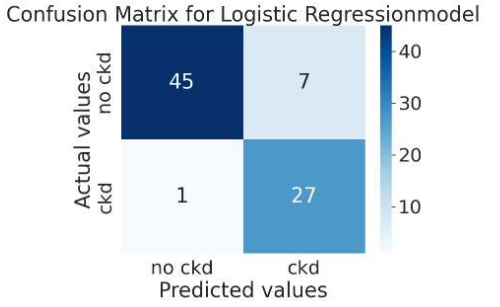
Making confusion matrix

Logistic regression

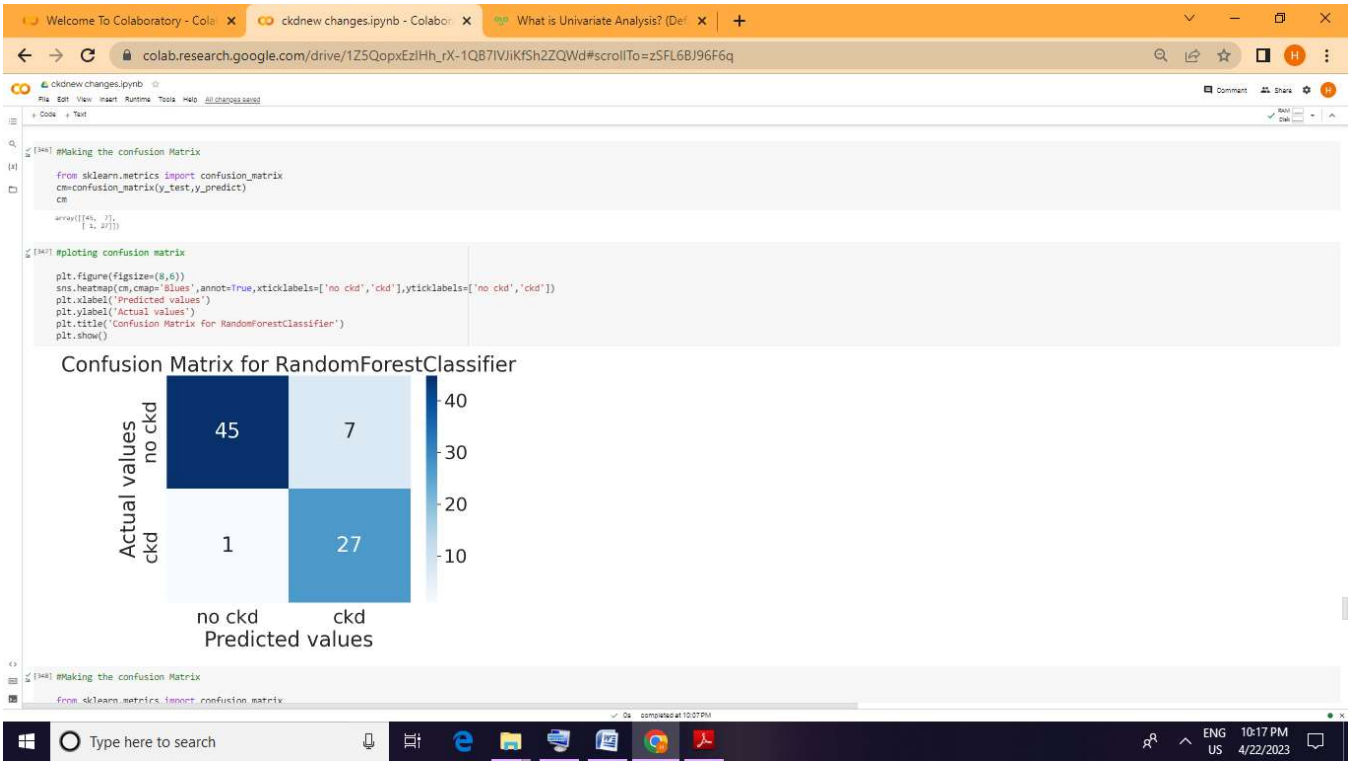
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
```

```
array([[45,  7],
       [ 1, 27]])
```

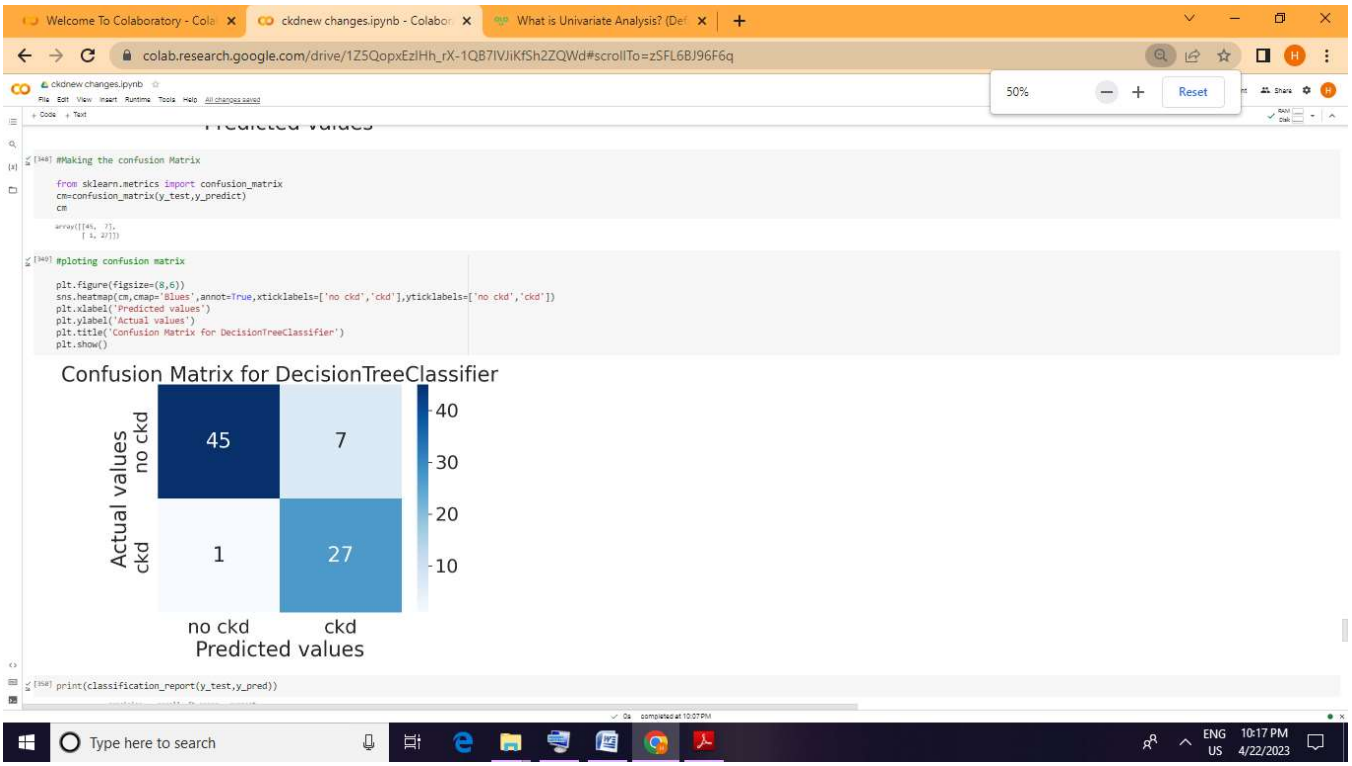
```
#ploting confusion matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,cmap='Blues',annot=True,xticklabels=['no ckd','ckd'],yticklabels=['no ckd','ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for Logistic Regressionmodel')
plt.show()
```



RandomForestClassifier

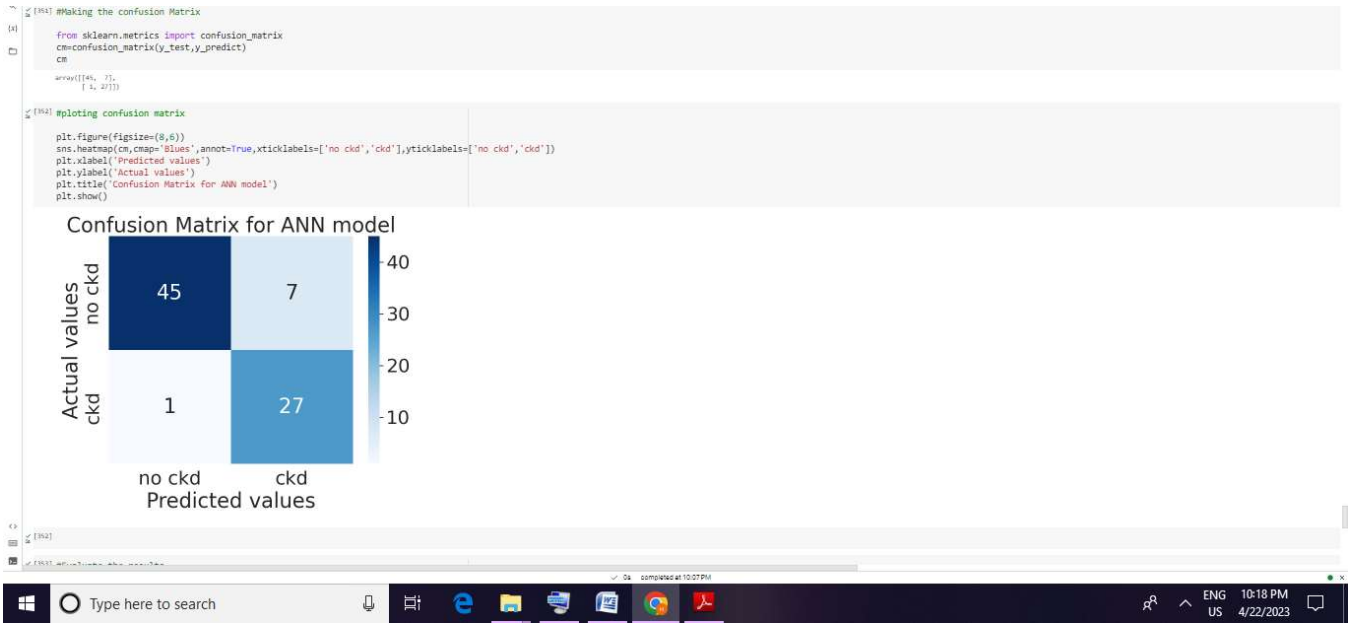


DecisionTreeClassifier




```
[350] print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.89      0.90      0.90        52
           1       0.81      0.79      0.80        28

    accuracy                           0.86        80
   macro avg       0.85      0.84      0.85        80
weighted avg       0.86      0.86      0.86        80
```

ANN



Model deployment

4. ADVANTAGE & DISADVANTAGE:

Advantage:

- The early detection of CKD allows patients to receive timely treatment, slowing the disease's progression. Due to its rapid recognition performance and accuracy, machine learning models can effectively assist physicians in achieving this goal.

- Predictive analysis using machine learning techniques can be helpful through an early detection of CKD for efficient and timely interventions. Machine learning use algorithm and historicaldata to predict its outcome more accurately.

- Prediction in machine learning allows organizations to make predictions about possible outcomes based on historical data.These assumptions allow the organization to make decisions resulting in tangible business results. Predictive analytics can be used to anticipate when users will churn or leave an organization.

- Machine learning prediction accuracy aims to givea good idea of how well a model performs at predicting on unseen data samples.

- Another key advantage is that the predictive evaluation not only identifies usability problems, but actually provides an explanation of them based on the theoretical model underlying the evaluation.

- Machine learning can detect patterns of specific diseases. It can then alert clinicians to any irregularities.

- To detect the Various Diseases through the examining Symptoms of patient's using different techniques of Machine Learning Models**.**

- Disease Prediction using Machine Learning is a system that predicts the disease based on the information provided by the user.

- An hospital usage of the future enhancement of the important on the kidney diseases in full an full understanding in prediction.

Disadvantage:

- Machine can predict diseases but cannot predict the sub types of the diseases caused by occurrence of one disease. It fails to predict all possible conditions of the people, and Existing system handles only structured data. The prediction system are broad and ambiguous.

- That machine learning models are fail silently, which means they will make predictions even if the incoming data looks nothing like the data they were trained against.

- The main disadvantages of machine learning in prediction is high error susceptibility. You end up with biased predictions coming from a biased training set. This leads to irrelevant advertisements being to customers.

- **Data quality**: The accuracy and reliability of CKD prediction models depend heavily on the quality and completeness of the data used to train them. If the data is incomplete or biased, the algorithm may produce inaccurate or unreliable predictions.

- **Limited generalizability**: CKD prediction models are often developed using data from specific populations or geographic regions. This can limit their generalizability to other populations or regions, where different risk factors and comorbidities may be present.

- **Overfitting**: Overfitting occurs when a machine learning algorithm is trained on a limited dataset, resulting in a model that is overly complex and cannot generalize well to new data. This can lead to inaccurate predictions and poor performance.

- **Ethical concerns**: The use of machine learning algorithms in healthcare raises ethical concerns about privacy, security, and equity. For example, biased algorithms may disproportionately affect certain patient groups, leading to health disparities.

- **Cost:** The implementation and maintenance of machine learning algorithms can be costly, requiring specialized expertise and infrastructure. This can limit their accessibility to resource-limited setting.

- **Lack of interpretability**: Machine learning algorithms often work as black boxes, making it difficult for healthcare providers to understand the underlying reasoning behind a prediction. This can be a limitation for clinical decision-making and may result in skepticism or mistrust of the model.

- **Limited incorporation of clinical expertise**: Machine learning algorithms are typically developed using large datasets and mathematical models, without direct input from healthcare providers. This can limit the incorporation of clinical expertise and patient preferences in the development of the model.

- **Uncertainty and error estimation**: Machine learning models often produce a single prediction, without an estimation of the uncertainty or potential errors in the prediction. This can be a limitation for decision-making, particularly in cases where the model's confidence in the prediction is low.

- **Limited data availability**: Machine learning algorithms require large and diverse datasets to train and validate the model. However, data on CKD may be limited or difficult to obtain, particularly in certain regions or patient populations.

- **Integration with clinical workflows**: The implementation of machine learning algorithms in clinical workflows may require changes to existing processes, such as data collection, storage, and analysis. This can be a challenge for healthcare systems and may require additional resources and support.

Overall, while machine learning holds promise for improving CKD prediction and management, it is important to consider the potential limitations and challenges associated with these models. Future research should focus onaddressing these limitations and developing machine learning models that are accurate, interpretable, and ethical.

5. APPLICATION :

For clinical practice, the risk predictions could be used to triage patients or different management procedures.CDK progression are well established, help in research studies, and many studies have used these data to build prediction models.Furthermore, patients may be identified early, provide for public health policy on the “Public health insurance Pre-ESRD preventive program”.

Chronic kidney disease (CKD) is a widespread health problem that affects millions of people worldwide. Early detection and accurate diagnosis of CKD can help improve patient outcomes and reduce the risk of progression to end-stage renal disease (ESRD). Machine learning algorithms have been increasingly used in applications to aid in the detection and diagnosis of CKD.

- **Predictive modelling** : Machine learning models can be trained on clinical and laboratory data to predict the risk of CKD development or progression. These models can be used to identify patients who are at high risk of developing CKD, and to monitor those who have already been diagnosed with the disease.

- **Image analysis**: Machine learning algorithms can be used to analyze medical images such as ultrasound or MRI scans to detect abnormalities in the kidneys. This can help in the early detection and diagnosis of CKD.

- **Biomarker identification**: Machine learning can be used to analyze large amounts of data from blood and urine tests to identify biomarkers that are associated with CKD. These biomarkers can be used to develop new diagnostic tests and to monitor disease progression.

- **Electronic health record analysis**: Machine learning can be used to analyze electronic health records (EHRs) to identify patients who are at high risk of developing CKD. This can help healthcare providers to provide timely interventions and prevent disease progression.

CKD detection and management:

- **Medication management**: Machine learning can be used to predict adverse drug reactions or drug interactions in patients with CKD, who often require multiple medications for the management of their disease. This can help prevent medication errors and improve patient outcomes.

- **Patient stratification**: Machine learning can be used to stratify patients with CKD based on their disease stage, comorbidities, and other clinical factors. This can help healthcare providers to develop personalized treatment plans and provide targeted interventions.

- **Telemedicine:** Machine learning can be used in telemedicine applications to remotely monitor patients with CKD, including the collection of vital signs and other clinical data. This can help to improve patient outcomes and reduce the burden on healthcare providers.

- Clinical decision support: Machine learning can be used to develop clinical decision support systems that can assist healthcare providers in the management of CKD. For example, these systems can provide recommendations for medication dosing or adjustments based on patient-specific data.

In summary, machine learning has the potential to improve the detection, diagnosis, and management of CKD through a variety of applications. However, it is important to

note that these algorithms should be developed and validated in collaboration with healthcare professionals and patients to ensure their safety and effectiveness.

6. CONCLUSION:

Conclusion Machine learning is a powerful tool for making prediction from data. However, it is important to remember that machine it is only as good that is used to train the algorithms. One of the objectives is to reduce premature mortality from non-communicable disease by third in 2030. Chronic kidney disease (CKD) is among the significant contributor to morbidity and mortality from non-communicable diseases that can affected 10–15% of the global population.

This article objects to predict Chronic Kidney Disease based on full features and important features of CKD dataset. For feature selection three different techniques have been applied: correlation-based feature selection, Wrapper method and LASSO regression. In this perception, four classifiers algorithm were applied viz. artificial neural network (ANN), logistic regression, random forest and decision tree. For each classifier, the results were computed based on full features, selected features by CFS, selected features by Wrapper, selected features by LASSO regression, SMOTE with selected features by LASSO, SMOTE with full features.

It was observed that LSVM achieved the highest accuracy of 98.86% in SMOTE with full features. All classifiers algorithms performed well on features selected by LASSO regression with SMOTE and without SMOTE. SMOTE with full features gave the best result for all 5 classifiers. In this research, a total of 7 classifiers were used. However, Logistic and KNN did not give suitable results and it was why they were not used in SMOTE. As per the result, it is concluded that SMOTE is a best technique for balancing a dataset. It is noted that SMOTE gave better results with selected features by LASSO regression as compare to without SMOTE on LASSO regression model. LSVM achieved the highest accuracy in all experiments as compared to other classifiers algorithms.

7. FUTURE SCOPE :

8.APPENDIX

SOURCE CODE:

```
#importing the libraries

import pandas as pd
import numpy as np
from collections import Counter as c
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
from sklearn.metrics import accuracy_score, co
nfusion_matrix
from sklearn.model_selection import train_test
_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegre
ssion
import pickle
```

```
import warnings
warnings.filterwarnings('ignore')

#read the data
data=pd.read_csv("/content/kidney_disease.csv”
)
data.head()

#Data preparation
#Rename the columns

data.columns
data.columns=(['Id','Age','Blood_pressure','Sp
ecific_gravity','Albumin','Sugar','Red_blood_c
ells','Pus_cell','Pus_cell_clumps','Bacteria',
'Blood glucose random','Blood_urea','Serum_cre
atinine','Sodium','Potassium','Hemoglobin','Pa
cked_cell_volume','White_blood_cell_count','Re
d_blood_cell_count','Hypertension','Diabetesme
llitus','Coronary_artery_disease','Appetite','
Pedal_edema','Anemia','Class'])
data.columns

#Handling the missing values
#info will give you a summary of dataset

data.info()
```

```
data.isnull().any() #it will return true ifany columns is having null values
data['Blood glucose random'].fillna(data['Blood glucose random'].mean(),inplace=True)
data['Blood_pressure'].fillna(data['Blood_pressure'].mean(),inplace=True)
data['Blood_urea'].fillna(data['Blood_urea'].mean(),inplace=True)
data['Hemoglobin'].fillna(data['Hemoglobin'].mean(),inplace=True)
data['Potassium'].fillna(data['Potassium'].mean(),inplace=True)
data['Serum_creatinine'].fillna(data['Serum_creatinine'].mean(),inplace=True)
data['Sodium'].fillna(data['Sodium'].mean(),inplace=True)
data['Age'].fillna(data['Age'].mode()[0],inplace=True)
data['Hypertension'].fillna(data['Hypertension'].mode()[0],inplace=True)
data['Pus_cell_clumps'].fillna(data['Pus_cell_clumps'].mode()[0],inplace=True)
data['Appetite'].fillna(data['Appetite'].mode()[0],inplace=True)
data['Albumin'].fillna(data['Albumin'].mode()[0],inplace=True)
data['Pus_cell'].fillna(data['Pus_cell'].mode()[0],inplace=True)
```

```
data['Red_blood_cells'].fillna(data['Red_blood
_cells'].mode()[0],inplace=True)
data['Coronary_artery_disease'].fillna(data['C
oronary_artery_disease'].mode()[0],inplace=Tru
e)
data['Bacteria'].fillna(data['Bacteria'].mode(
)[0],inplace=True)
data['Anemia'].fillna(data['Anemia'].mode()[0]
,inplace=True)
data['Sugar'].fillna(data['Sugar'].mode()[0],i
nplace=True)
data['Diabetesmellitus'].fillna(data['Diabetes
mellitus'].mode()[0],inplace=True)
data['Pedal_edema'].fillna(data['Pedal_edema']
.mode()[0],inplace=True)
data['Specific_gravity'].fillna(data['Specific
_gravity'].mode()[0],inplace=True)
data['Packed_cell_volume'].fillna(data['Packed
_cell_volume'].mode()[0],inplace=True)
data['Red_blood_cell_count'].fillna(data['Red_
blood_cell_count'].mode()[0],inplace=True)
data['White_blood_cell_count'].fillna(data['Wh
ite_blood_cell_count'].mode()[0],inplace=True)
data.isnull().sum()

#replaces the unwanted classes
```

```
data['Class'] = data['Class'].replace(to_repla
ce = {'ckd\t': 'ckd', 'notckd': 'not ckd'})
data['Class'] = data['Class'].map({'ckd': 0, '
not ckd': 1})
c(data['Class'])

#handling categorical columns

catcols=set(data.dtypes[data.dtypes=='O'].inde
x.values) #only fetch object types column
print(catcols)

for i in catcols:
  print("Columns:",i)
  print(c(data[i]))   #using counter for check
ing the number of classes in the column
  print('*'*120+'\n')

catcols.remove('Red_blood_cell_count')#remove
is used for removing particular column
catcols.remove('Packed_cell_volume')
catcols.remove('White_blood_cell_count')
print(catcols)

from sklearn.preprocessing import LabelEncoder
for i in catcols:
    print("LABEL ENCODING OF:",i)
    LEi=LabelEncoder()
```

```
    print(c(data[i]))
    data[i]=LEi.fit_transform(data[i])
    print(c(data[i]))
    print("*"*100)

# Handling numerical columns

contcols=set(data.dtypes[data.dtypes!='O'].ind
ex.values) #only fetch the float and int type
columns
print(contcols)

for i in contcols:
  print("Continous Columns :",i)
  print(c(data[i]))
  print('*'*120+'\n')

contcols.remove('Specific_gravity')
contcols.remove('Albumin')
contcols.remove('Sugar')
print(contcols)

contcols.add('Red_blood_cell_count')
contcols.add('Packed_cell_volume')
contcols.add('White_blood_cell_count')
print(contcols)

catcols.add('Specific_gravity')
```

```
catcols.add('Albumin')
catcols.add('Sugar')
print(catcols)

data['Diabetesmellitus'].replace(to_replace =
{'\tno':'no','\tyes':'yes',' yes':'yes'},inpla
ce=True)
c(data['Diabetesmellitus'])
data['Coronary_artery_disease'] = data['Corona
ry_artery_disease'].replace(to_replace = '\tno
', value='no')
c(data['Coronary_artery_disease'])

def clean_dataset(df):
    assert isinstance(data, pd.DataFrame), "df
 needs to be a pd.DataFrame"
    data.dropna(inplace=True)
    indices_to_keep = ~data.isin([np.nan, np.i
nf, -np.inf]).any(axis=1)
    return data[indices_to_keep].astype(np.flo
at64)
data.replace([np.inf,-
np.inf], np.nan, inplace=True)

#Exploratory Data Analysis
 #Descriptive statistical Analysis
```

```
data.describe()


#visual analysis
# 1.Univariate analysis
#AGE DISTRIBUTION


sns.distplot(data.Age)
#2.Bivariate analysis
#Age vs Pressure


import matplotlib.pyplot as plt
fig=plt.figure(figsize=(5,5)) #plot size
plt.scatter(data['Age'],data['Blood_pressure']
,color='blue')
plt.xlabel('Age') #set the label for x axis
plt.ylabel('Blood_pressure') #set the labelfor
 y axis
plt.title("Age VS Blood scatter plot") #Set a
title for the axes

#Multivariate analysis
#Age Vs all continuous columns


plt.figure(figsize=(30,40),facecolor='white')
plotnumber=1
```

```
for column in contcols:
  if plotnumber<=11:  #as there are 11 continous columns in the data
    ax=plt.subplot(3,4,plotnumber) #3,4 is refer to 3X4 matrix
    plt.scatter(data['Age'],data[column]) #plotting scatter plot
    plt.xlabel(column,fontsize=20)
  plotnumber+=1
plt.show()

#Finding correlation between the independent columns
#HEAT MAP #correlation of parameters
f,ax=plt.subplots(figsize=(38,20))
sns.heatmap(data.corr(),annot=True,fmt=".2f",ax=ax,linewidths=0.5,linecolor="orange")
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.show()

sns.countplot(x=data['Class'])

#creating Independent and Dependent


selcols=['Red_blood_cells','Pus_cell','Blood glucose random','Blood_urea','Pedal_edema','Ane
```

```
mia','Diabetesmellitus','Coronary_artery_disea
se']
x=pd.DataFrame(data,columns=selcols)
y=pd.DataFrame(data,columns=['Class'])

#creating Independent and Dependent
selcols=['Red_blood_cells','Pus_cell','Blood g
lucose random','Blood_urea','Pedal_edema','Ane
mia','Diabetesmellitus','Coronary_artery_disea
se']
x=pd.DataFrame(data,columns=selcols)
y=pd.DataFrame(data,columns=['Class'])
print(x.shape)
print(y.shape)

#Splitting the data into train and test

from sklearn.model_selection import train_test
_split
x_train,x_test,y_train,y_test=train_test_split
(x,y,test_size=0.2,random_state=0)

#model  building
#Train the model in multiple algorithm
#ANN algorithm
#importint the keras libraries and packages


import tensorflow
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf

#creating ANN skleton view

classification = Sequential()
classification.add(Dense(30,activation='relu')
)
classification.add(Dense(128,activation='relu'
))
classification.add(Dense(64,activation='relu')
)
classification.add(Dense(32,activation='relu')
)
classification.add(Dense(1 ,activation='sigmoi
d'))

#compiling the ANN model
classification.compile(optimizer='adam',loss='
binary_crossentropy',metrics=['accuracy'])
#training the model

classification.fit(x_train,y_train,batch_size=
10,validation_split=0.2,epochs=100)

#DecisionTree Classifier
```

```
from sklearn.tree import DecisionTreeClassifie
r
dtc=DecisionTreeClassifier(max_depth=4,splitte
r='best',criterion='entropy')
dtc.fit(x_train,y_train)


y_predict=dtc.predict(x_test)
y_predict
y_predict_train=dtc.predict(x_train)

#Random forest model

from sklearn.ensemble import RandomForestClass
ifier
rfc=RandomForestClassifier(n_estimators=10,cri
terion='entropy')
rfc.fit(x_train,y_train)
y_predict_train=rfc.predict(x_train)
y_predict=rfc.predict(x_test)

#Logistic regression

from sklearn.linear_model import LogisticRegre
ssion
lgr=LogisticRegression()
lgr.fit(x_train,y_train)
```

```
from sklearn.metrics import accuracy_score,cla
ssification_report
y_predict=lgr.predict(x_test)
#testing the model
#Logistic regression

y_pred=lgr.predict([[1,1,121.000000,36.0,0,0,1
,0]])
print(y_pred)

#DecisionTree Classifier

y_pred=dtc.predict([[1,1,121.000000,36.0,0,0,1
,0]])
print(y_pred)

#random forest classifier

y_pred=rfc.predict([[1,1,121.000000,36.0,0,0,1
,0]])

print(y_pred)
(y_pred)

classification.save("ckd.h5")  #save the model
 to test the input
y_pred=classification.predict(x_test)
y_pred=(y_pred>0.5)
```

```
y_pred

def predict_exit(sample_value):
  sample_value=np.array(sample_value)
  sample_value=sample_value(1,-1)
  sample_value=sc.transform(sample_value)
  return classifier.predict(sample_value)


test=classification.predict([[1,1,121.000000,3
6.0,0,0,1,0]])
if test==1:
  print('Prediction:High chance of ckd!')
else:
  print('Prediction:Low chance of ckd')


#compare model
from sklearn.model_selection import train_test
_split #splits data in random train and test
from sklearn.model_selection import train_test
_split


from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasR
egressor
```

```
from sklearn.model_selection import cross_val_
score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScal
er
from sklearn.pipeline import Pipeline
from sklearn import model_selection
dfs=[]
models = [('LogReg',LogisticRegression()),
        ('RF',RandomForestClassifier()),
        ('DecisionTree',DecisionTreeClassifier
())]
df_mean=pd.DataFrame(models)
df_mean['models'] = df_mean.index
results=[]
names=[]
scoring=['accuracy','precision_weighted','reca
ll_weighted','f1_weighted','roc_auc']
target_names=['NO CKD','CKD']
for name, model in models:
  kfold =model_selection.KFold(n_splits=5,shuf
fle=True,random_state=90210)
  cv_results= model_selection.cross_validate(m
odel,x_train,y_train,cv=kfold,scoring=scoring)
  clf=model.fit(x_train,y_train)
  y_pred=clf.predict(x_test)
  print(name)
```

```
  print(classification_report(y_test,y_pred,ta
rget_names=target_names))
  results.append(cv_results)
  names.append(name)
  this_df=pd.DataFrame(cv_results)
  this_df['model']=name
  dfs.append(this_df)
final=pd.concat(dfs,ignore_index=True)
print(final)

#Making the confusion Matrix

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm

#ploting confusion matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,cmap='Blues',annot=True,xtickla
bels=['no ckd','ckd'],yticklabels=['no ckd','c
kd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for Logistic Regre
ssionmodel')
plt.show()
```

```
#Making the confusion Matrix

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm

#ploting confusion matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,cmap='Blues',annot=True,xtickla
bels=['no ckd','ckd'],yticklabels=['no ckd','c
kd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for RandomForestCl
assifier')
plt.show()

#Making the confusion Matrix

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm

#ploting confusion matrix

plt.figure(figsize=(8,6))
```

```
sns.heatmap(cm,cmap='Blues',annot=True,xtickla
bels=['no ckd','ckd'],yticklabels=['no ckd','c
kd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for DecisionTreeCl
assifier')
plt.show()

print(classification_report(y_test,y_pred))

#Making the confusion Matrix

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm

#ploting confusion matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,cmap='Blues',annot=True,xtickla
bels=['no ckd','ckd'],yticklabels=['no ckd','c
kd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for ANN model')
plt.show()
```

```
#Evaluate the results
import pandas as pd

bootstraps=[]
bootstraps= pd.DataFrame(bootstraps)
for model in list(set(final.model.values)):
  model_df=final.loc[final.model==model]
  bootstrap=model_df.sample(n=30,replace=True)
  bootstraps.append(bootstrap)
bootstrap_df=pd.concat([bootstraps],ignore_ind
ex=True)
results_long=pd.melt(bootstrap_df,var_name='me
trics',value_name='values')
time_metrics=['fit_time','score_time']   #fit
time metrics

#PERFORMANCE METRICS

results_long_nofit=results_long.loc[results_lo
ng['metrics'].isin(time_metrics)]   #get df wi
thout fit data
results_long_nofit=results_long_nofit.sort_val
ues(by='values')

#TIME METRICS
```

```
results_long_fit=results_long.loc[results_long
['metrics'].isin(time_metrics)]   #get df with
 fit data
results_long_fit=results_long_fit.sort_values(
by='values')

import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(20,12))
sns.set(font_scale=2.5)
df_mean=pd.DataFrame(models)
df_mean['models'] = df_mean.index
g=sns.boxplot(x=models,y="values",hue="metrics
",data=results_long_nofit,palette="Set3")
plt.legend(bbox_to_anchor=(1.05,1),loc=2,borde
raxespad=0.)
plt.title("Comparision of Model by classificat
ion Metric")
plt.savefig('.\benchmark_models_performance.pn
g',dpi=300)

import pickle

pickle.dump(lgr,open('ckd.pkl','wb'))
```

Creating template:

Home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Chronic Kidney disease Prediction</title>
</head>
<body background="IMG-20230422-WA0009.jpg" style="background-repeat:no-repeat;background-size:100%">
<h3 align="right">
   <font face="sans serif" size="3">
      <a href="#">HOME</a>
      <a href="predict.html">PREDICTION</a>
   </font>
</h3>
<h1 align="center">
   <font face="Italic" size="8">
      CHRONIC KIDNEY DISEASE PREDICTION
   </font>
</h1>
<h4 align="center">
   <font face="Lato" size="3" color="#fffff">
```

Chronic kidney disease(CKD)is one of the most critical health problems due to its increasing prevalence.</br>

In this project we aim to test the ability of machine learning algorithms for the prediction of </br>chronic kidney

disease using the smallest subset of features.

****

</h4></body>

</html>

Predict.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Chronic Kidney Disease Prediction</title>
</head>
<body background="img.jpg" style="background-repeat:no-repeat;background-size:100%">
<h1 align="right">
        <a href="home.html">HOME</a>
</h1>

<h1 align="center">

   CHRONIC KIDNEY DISEASE PREDICTION
```

```
</h1>
<h4 align="center">

    A Machine Learning Web App Built With Flask

</h4>
<form action="output.html" method="post">

  <table border="1" width="45%" align="center">

    <tr>
      <td>Enter your Blood_urea</td>
      <td><input type="text" value="" name="blood_urea"
title="Enter your Blood_urea"/></td>
    </tr>

    <tr>
      <td>Enter your Blood_glucose_random</td>
      <td><input type="text" value=""
name="blood_glucose_random" title="Enter your
Blood_glucose_random" id="bloo_glucose_random"/></td>
    </tr>
    <tr>
```

```
        <td>Select Anemia or not</td>
        <td><select name="anemia">
            <option value="-1" selected>

            </option>
            <option value="1">Yes</option>
            <option value="2">No</option>
        </select></td>
    </tr>

      <tr>
        <td>Select Coronary_artery_disease or not</td>
        <td><select name="coronary_artery_disease">
            <option value="-1" selected>

            </option>
            <option value="1">Yes</option>
            <option value="2">No</option>
        </select></td>
    </tr>

<tr>
        <td>Select Pus_cell or not</td>
```

```
        <td><select name="pus_cell">
            <option value="-1" selected>

            </option>
            <option value="1">Normal</option>
            <option value="2">AbNormal</option>
        </select></td>
    </tr>

<tr>
        <td>Select Red_blood_cells or not</td>
        <td><select name="red_blood_cells">
            <option value="-1" selected>

            </option>
            <option value="1">Normal</option>
            <option value="2">AbNormal</option>
        </select></td>
    </tr>

<tr>
        <td>Select Diabetesmellitus or not</td>
        <td><select name="diabetesmellitus">
```

```
            <option value="-1" selected>

            </option>
            <option value="1">Yes</option>
            <option value="2">No</option>
          </select></td>
        </tr>

<tr>
          <td>Select Pedal_edema or not</td>
          <td><select name="pedal_edema">
            <option value="-1" selected>

            </option>
            <option value="1">Yes</option>
            <option value="2">No</option>
          </select></td>
        </tr>
    </table>


<table align="center" border="1">
```

```
    <tr>
        <td><input type="submit" value="Predict"/><td>
    </tr>

</table>
</form>
</body>
</html>
```

Output.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Chronic Kidney Disease Prediction</title>
</head>
<body background="IMG-20230422-WA0009.jpg" style="background-repeat:no-repeat;background-size:100%">

<a href="home.html">Back to Home</a>

</body>
</html>
```

App.py

```
from flask import Flask, render_template, request
import pickle
import sklearn
import pandas as pd
import numpy as np



app = Flask(_name_)
model = pickle.load(open('ckd.pk1', 'rb'))



@app.route('/')
def home():
    return render_template('home.html')



@app.route('/predict.html', methods=['POST', 'GET'])
def prediction():
    return render_template('predict.html')
```

```
@app.route('/home.html', methods=['POST', 'GET'])
def my_home():
    return render_template('home.html')



@app.route('/output.html', methods=['POST'])
def predict():
    input_features = [float(x) for x in request.form.values()]
    features_value = [np.array(input_features)]


    features_name = ['blood_urea', 'blood_glucose_random', 'aanemia',
'coronary_artery_disease',
                     'pus_cell', 'red_blood_cells', 'diabetesmellitus',
'Pedal_edema']
    df = pd.DataFrame(features_value, columns=features_name)


    model.predict(df)
    if output == 1:
        return render_template('output.html', prediction_text="Great! you
don't have CKD ")
    else:
        return render_template('output.html', prediction_text="You have
CKD")
```

```
if _name_ == '_main_':
    app.run(debug=True)
```