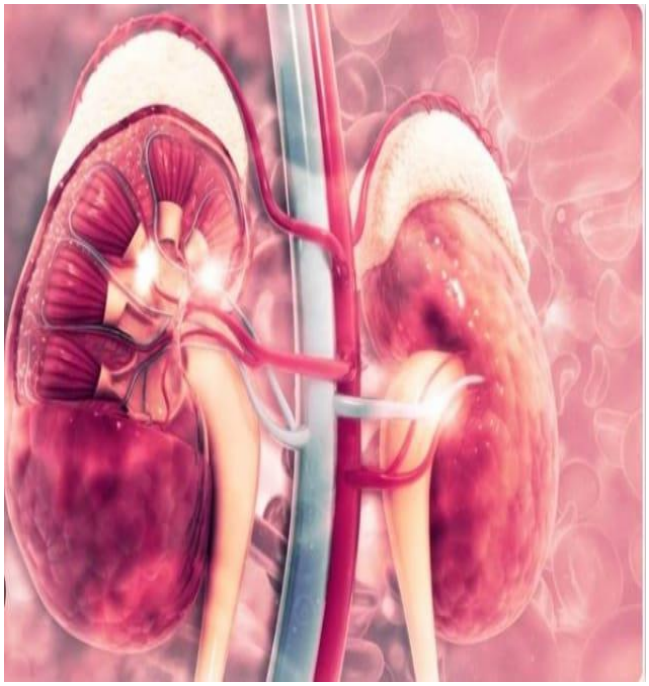
EARLY PREDICTION FOR CHRONIC KIDNEY DISEASE DETECTION: A PROGRESSIVE APPROACH TO HEALTH MANAGEMENT

Project Based Experiential Learning Program

Handbook

SmartBridge Educational Services Pvt. Ltd.

Early Prediction for Chronic Kidney Disease Detection: A Progressive Approach to Health Management



A project submitted by

Leader

1.Karthiga S

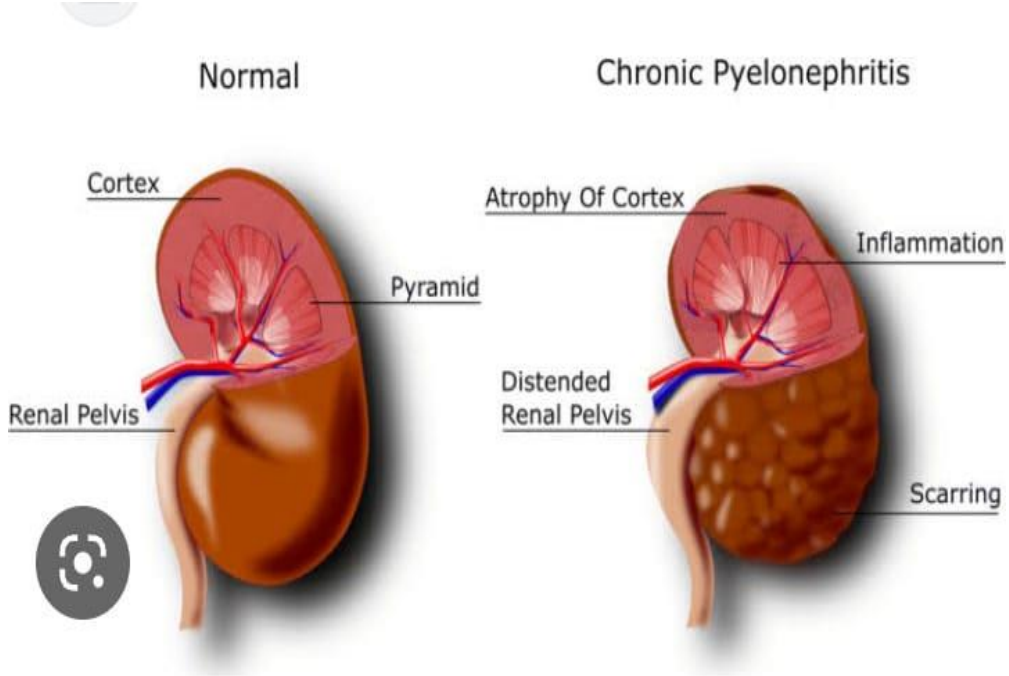
Team Members

1.Karthika S

2.Sabitha M

3.Sumithra M

1 INTRODUCTION



1.1 Overview

- Chronic Kidney Disease (CKD) is a major medical problem and can be cured if treated in the early
- stages. Usually, people are not aware that medical tests we take for different purposes could contain
- valuable information concerning kidney diseases. Consequently, attributes of various medical tests
- are investigated 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 illness, the pattern of the disease and work for curing the disease.
- In today's world as we know most of the people are facing so many disease and as this can be cured
- if we treat people in early stages this project can use a pertained model to predict the Chronic Kidney
- Diseases which can help in treatments of peoples who are suffer from this disease

.

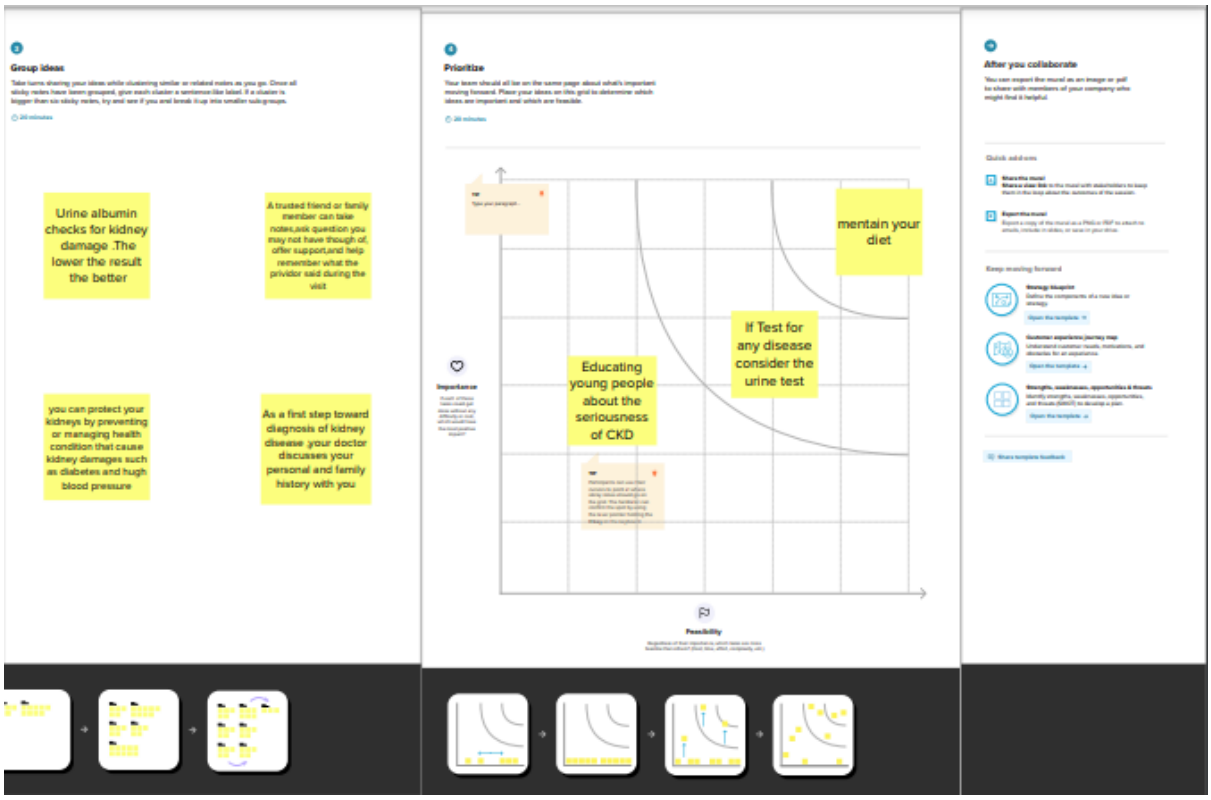
1.2 Purpose

- Predictive analysis using machine learning techniques can be helpful through an early detection of CKD for efficient and timely interventions. Using Random Forest (RF), Support Vector Machine (SVM) and Decision Tree (DT) have been used to detect CKD.
- 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.
- The purpose of this model is to develop and validate predictive models for chronic kidney disease.
- 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

2 Problem Definition & Design Thinking

2.1 Empathy Map

2.2 Ideation & Brainstorming Map



3 RESULT

```
[3]  plt.style.available
```

```
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-v0_8',
 'seaborn-v0_8-bright',
 'seaborn-v0_8-colorblind',
 'seaborn-v0_8-dark',
 'seaborn-v0_8-dark-palette',
 'seaborn-v0_8-darkgrid',
 'seaborn-v0_8-deep',
 'seaborn-v0_8-muted',
 'seaborn-v0_8-notebook',
 'seaborn-v0_8-paper',
 'seaborn-v0_8-pastel',
 'seaborn-v0_8-poster',
 'seaborn-v0_8-talk',
 'seaborn-v0_8-ticks',
 'seaborn-v0_8-white',
 'seaborn-v0_8-whitegrid',
 'tableau-colorblind10']
```

Read the Dataset

```
[5]  dataset.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

5 rows × 26 columns

Rename the column

```
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')
```

```
[7] 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', 'hypertenstion',
       'diabetesmellitus', 'coronary_artery_disease', 'appetite',
       'pedal_edema', 'anemia', 'class'],
      dtype='object')
```

```
 25  class                     400 non-null    object
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

Handling missing values

```
[8] <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  hypertenstion            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
```

```
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
hypertenstion              True
diabetesmellitus           True
coronary_artery_disease    True
appetite                   True
pedal_edema                True
anemia                     True
class                     False
dtype: bool
```

Handling Categorical columns

```
{'class', 'bacteria', 'red_blood_cells', 'coronary_artery_disease', 'pedal_edema', 'pus_cell_clumps', 'anemia', 'packed_cell_volume', 'diabetesmellitus', 'pus_cel
```

```
Counter({'bacteria': 1})
**********

columns: red_blood_cells
Counter({'red_blood_cells': 1})
**********

columns: coronary_artery_disease
Counter({'coronary_artery_disease': 1})
**********

columns: pedal_edema
Counter({'pedal_edema': 1})
**********

columns: pus_cell_clumps
Counter({'pus_cell_clumps': 1})
**********

columns: anemia
Counter({'anemia': 1})
**********

columns: packed_cell_volume
Counter({'packed_cell_volume': 1})
**********

columns: diabetesmellitus
Counter({'diabetesmellitus': 1})
**********
```

```
columns: pus_cell
Counter({'pus_cell': 1})
**********

columns: white_blood_cell_count
Counter({'white_blood_cell_count': 1})
**********

columns: appetite
Counter({'appetite': 1})
**********

columns: red_blood_cell_count
Counter({'red_blood_cell_count': 1})
**********

columns: hypertenstion
Counter({'hypertenstion': 1})
**********
```

```
{'class', 'bacteria', 'red_blood_cells', 'coronary_artery_disease', 'pedal_edema', 'pus_cell_clumps', 'anemia', 'diabetesmellitus', 'pus_cell', 'appetite', 'hyper
```

Label Encoding for categorical columns

```
LABLE ENCODING: anemia
Counter({'no': 340, 'yes': 60})
Counter({0: 340, 1: 60})
**********************************************************************************************
LABLE ENCODING: pedal_edema
Counter({'no': 324, 'yes': 76})
Counter({0: 324, 1: 76})
**********************************************************************************************
LABLE ENCODING: appetite
Counter({'good': 318, 'poor': 82})
Counter({0: 318, 1: 82})
**********************************************************************************************
LABLE ENCODING: bacteria
Counter({'notpresent': 378, 'present': 22})
Counter({0: 378, 1: 22})
**********************************************************************************************
LABLE ENCODING: class
Counter({'ckd': 248, 'notckd': 150, 'ckd\t': 2})
Counter({0: 248, 2: 150, 1: 2})
**********************************************************************************************
LABLE ENCODING: coronary_artery_disease
Counter({'no': 364, 'yes': 34, '\tno': 2})
Counter({1: 364, 2: 34, 0: 2})
**********************************************************************************************
LABLE ENCODING: diabetesmellitus
Counter({'no': 260, 'yes': 134, '\tno': 3, '\tyes': 2, ' yes': 1})
Counter({3: 260, 4: 134, 0: 3, 1: 2, 2: 1})
**********************************************************************************************
```

```
LABLE ENCODING: hypertenstion
Counter({'no': 253, 'yes': 147})
Counter({0: 253, 1: 147})
**********************************************************************************************
LABLE ENCODING: pus_cell
Counter({'normal': 324, 'abnormal': 76})
Counter({1: 324, 0: 76})
**********************************************************************************************
LABLE ENCODING: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
Counter({0: 358, 1: 42})
**********************************************************************************************
LABLE ENCODING: red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
Counter({1: 353, 0: 47})
**********************************************************************************************
```

Handling Numrical columns

```
{'class', 'potassium', 'coronary_artery_disease', 'pedal_edema', 'age', 'bacteria', 'red_blood_cells', 'blood glucose random', 'diabetesmellitus', 'id', 'serum_cr
```

```
Continous Columns: class
Counter({0: 248, 2: 150, 1: 2})
**********************************************************************************************************************

Continous Columns: potassium
Counter({4.62724358974359: 88, 5.0: 30, 3.5: 30, 4.9: 27, 4.7: 17, 4.8: 16, 4.0: 14, 4.2: 14, 4.1: 14, 3.8: 14, 3.9: 14, 4.4: 14, 4.5: 13, 3.7: 12, 4.3: 12, 3.6
**********************************************************************************************************************

Continous Columns: coronary_artery_disease
Counter({1: 364, 2: 34, 0: 2})
**********************************************************************************************************************

Continous Columns: pedal_edema
Counter({0: 324, 1: 76})
**********************************************************************************************************************

Continous Columns: age
Counter({60.0: 28, 65.0: 17, 48.0: 12, 50.0: 12, 55.0: 12, 47.0: 11, 62.0: 10, 45.0: 10, 54.0: 10, 59.0: 10, 56.0: 10, 61.0: 9, 70.0: 9, 46.0: 9, 34.0: 9, 68.0
**********************************************************************************************************************

Continous Columns: bacteria
Counter({0: 378, 1: 22})
**********************************************************************************************************************

Continous Columns: red_blood_cells
Counter({1: 353, 0: 47})
**********************************************************************************************************************
```

```
Continous Columns: blood glucose random
Counter({148.0365168539326: 44, 99.0: 10, 100.0: 9, 93.0: 9, 107.0: 8, 117.0: 6, 140.0: 6, 92.0: 6, 109.0: 6, 131.0: 6, 130.0: 6, 70.0: 5, 114.0: 5, 95.0: 5, 1
**********************************************************************************************************************

Continous Columns: diabetesmellitus
Counter({3: 260, 4: 134, 0: 3, 1: 2, 2: 1})
**********************************************************************************************************************

Continous Columns: id
Counter({0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1, 11: 1, 12: 1, 13: 1, 14: 1, 15: 1, 16: 1, 17: 1, 18: 1, 19: 1, 20: 1, 21: 1, 22: 1,
**********************************************************************************************************************

Continous Columns: serum_creatinine
Counter({1.2: 40, 1.1: 24, 1.0: 23, 0.5: 23, 0.7: 22, 0.9: 22, 0.6: 18, 0.8: 17, 3.072454308093995: 17, 2.2: 10, 1.5: 9, 1.7: 9, 1.3: 8, 1.6: 8, 1.8: 7, 1.4: 7,
**********************************************************************************************************************

Continous Columns: pus_cell_clumps
Counter({0: 358, 1: 42})
**********************************************************************************************************************

Continous Columns: specific_gravity
Counter({1.02: 153, 1.01: 84, 1.025: 81, 1.015: 75, 1.005: 7})
**********************************************************************************************************************

Continous Columns: blood_pressure
Counter({80.0: 116, 70.0: 112, 60.0: 71, 90.0: 53, 100.0: 25, 76.46907216494846: 12, 50.0: 5, 110.0: 3, 140.0: 1, 180.0: 1, 120.0: 1})
**********************************************************************************************************************
```

```
Continous Columns: blood_urea
Counter({57.425721784776904: 19, 46.0: 15, 25.0: 13, 19.0: 11, 40.0: 10, 18.0: 9, 50.0: 9, 15.0: 9, 48.0: 9, 26.0: 8, 27.0: 8, 32.0: 8, 49.0: 8, 36.0: 7, 28.0:
**********

Continous Columns: pus_cell
Counter({1: 324, 0: 76})
**********

Continous Columns: appetite
Counter({0: 318, 1: 82})
**********

Continous Columns: sodium
Counter({137.52875399361022: 87, 135.0: 40, 140.0: 25, 141.0: 22, 139.0: 21, 142.0: 20, 138.0: 20, 137.0: 19, 136.0: 17, 150.0: 17, 147.0: 13, 145.0: 11, 132.0
**********

Continous Columns: anemia
Counter({0: 340, 1: 60})
**********

Continous Columns: albumin
Counter({0.0: 245, 1.0: 44, 2.0: 43, 3.0: 43, 4.0: 24, 5.0: 1})
**********

Continous Columns: sugar
Counter({0.0: 339, 2.0: 18, 3.0: 14, 4.0: 13, 1.0: 13, 5.0: 3})
**********
Continous Columns: hemoglobin
Counter({12.526436781609195: 52, 15.0: 16, 10.9: 8, 9.8: 7, 11.1: 7, 13.0: 7, 13.6: 7, 11.3: 6, 10.3: 6, 12.0: 6, 13.9: 6, 15.4: 5, 11.2: 5, 10.8: 5, 9.7: 5, 1
**********

Continous Columns: hypertenstion
Counter({0: 253, 1: 147})
**********
```

```
{'class', 'potassium', 'coronary_artery_disease', 'pedal_edema', 'age', 'bacteria', 'red_blood_cells', 'blood glucose random', 'diabetesmellitus', 'id', 'serum_cr
```

```
{'class', 'potassium', 'coronary_artery_disease', 'pedal_edema', 'packed_cell_volume', 'white_blood_cell_count', 'age', 'bacteria', 'red_blood_cells', 'blood gluc
```

```
0      1
1      1
2      1
3      1
4      1
       ..
395    1
396    1
397    1
398    1
399    1
Name: coronary_artery_disease, Length: 400, dtype: int64
```

```
0      4
1      3
2      4
3      3
4      3
      ..
395    3
396    3
397    3
398    3
399    3
Name: diabetesmellitus, Length: 400, dtype: int64
```

Exploratory Data Analys

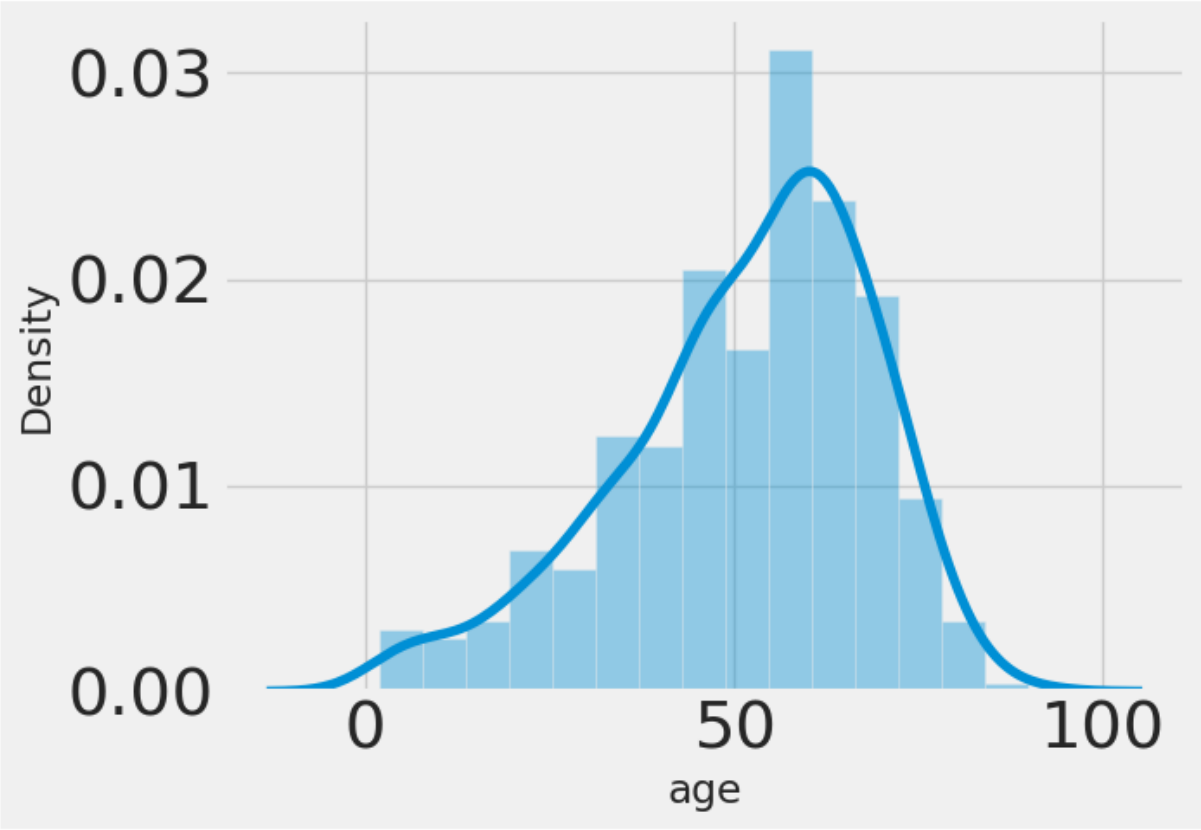
	id	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell	pus_cell_clumps	bacteria	...	sodium	po
count	400.000000	400.000000	400.000000	400.000000	400.00000	400.000000	400.000000	400.000000	400.000000	400.000000	...	400.000000	400
mean	199.500000	51.675000	76.469072	1.017712	0.90000	0.395000	0.882500	0.810000	0.105000	0.055000	...	137.528754	4
std	115.614301	17.022008	13.476298	0.005434	1.31313	1.040038	0.322418	0.392792	0.306937	0.228266	...	9.204273	2
min	0.000000	2.000000	50.000000	1.005000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	...	4.500000	2
25%	99.750000	42.000000	70.000000	1.015000	0.00000	0.000000	1.000000	1.000000	0.000000	0.000000	...	135.000000	4
50%	199.500000	55.000000	78.234536	1.020000	0.00000	0.000000	1.000000	1.000000	0.000000	0.000000	...	137.528754	4
75%	299.250000	64.000000	80.000000	1.020000	2.00000	0.000000	1.000000	1.000000	0.000000	0.000000	...	141.000000	4
max	399.000000	90.000000	180.000000	1.025000	5.00000	5.000000	1.000000	1.000000	1.000000	1.000000	...	163.000000	47

8 rows × 23 columns

Visual analysis

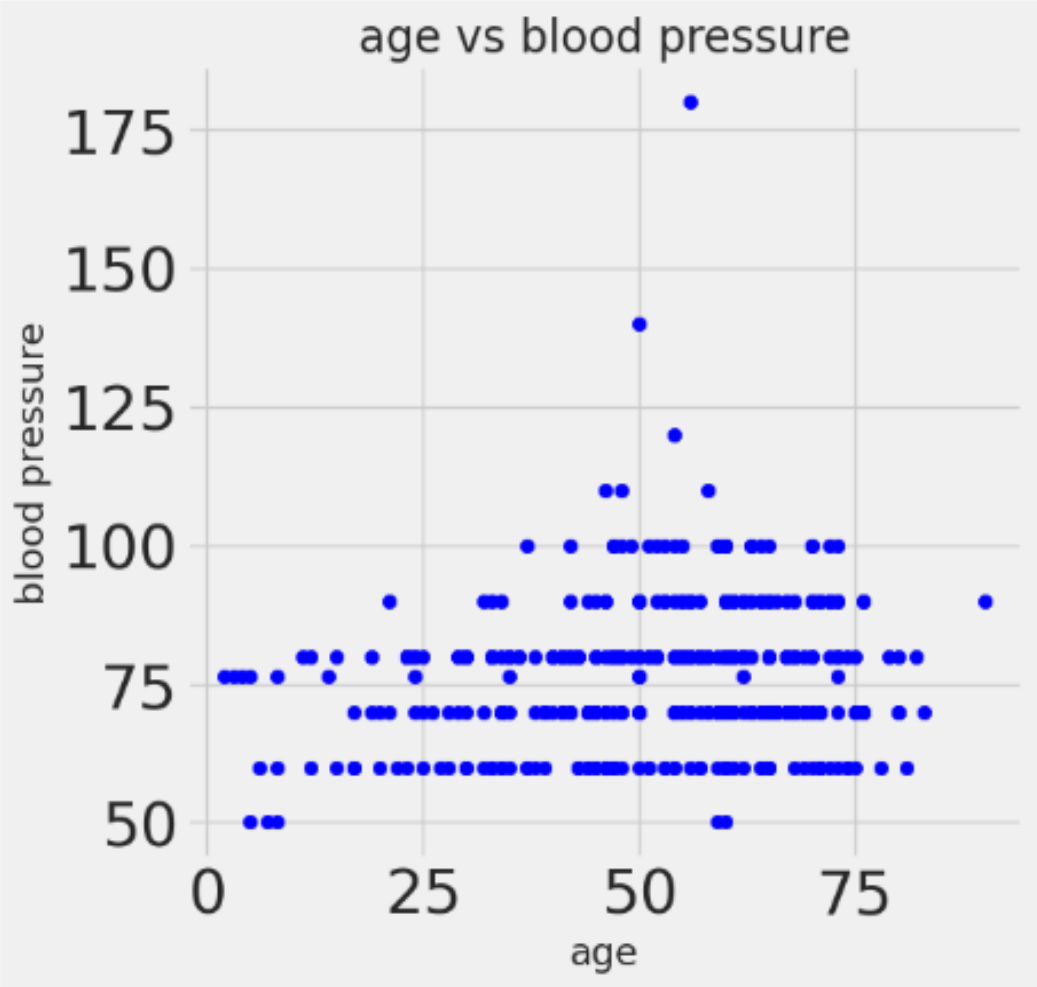
Univariate analysis

```
<ipython-input-560-3323bb223b46>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data.age)
<Axes: xlabel='age', ylabel='Density'>
```



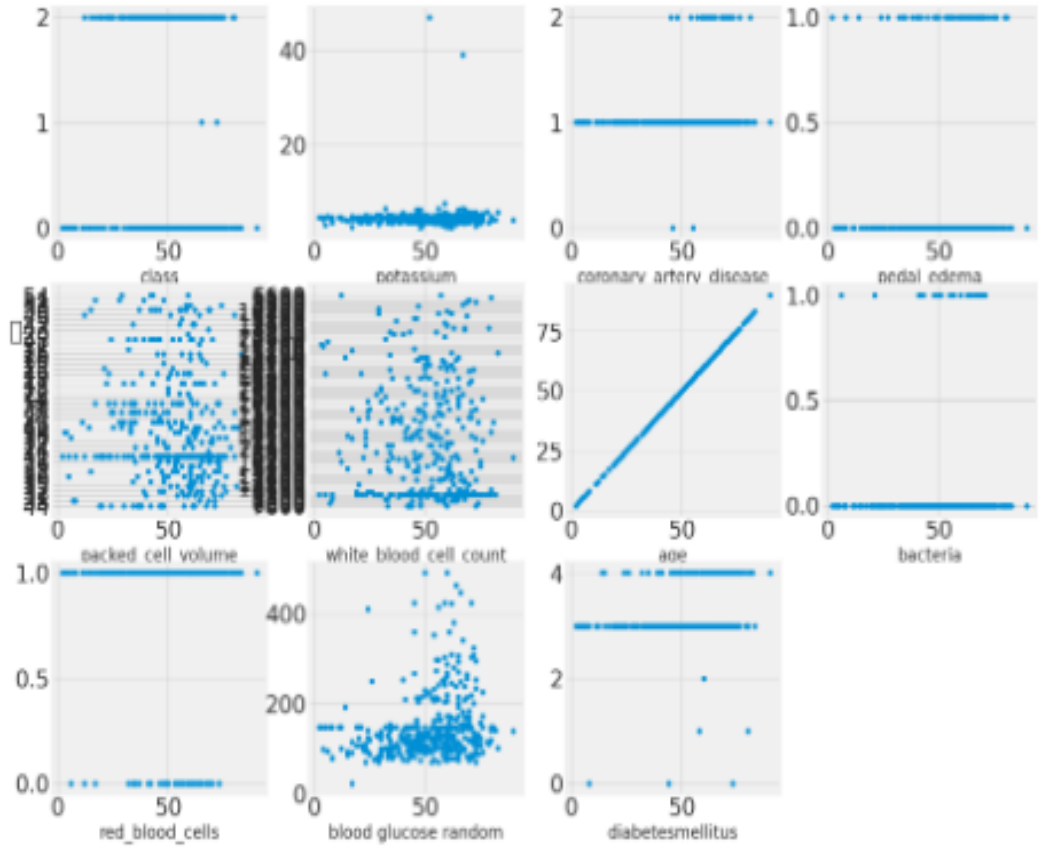
Bivariate analysis

```
Text(0.5, 1.0, 'age vs blood pressure')
```



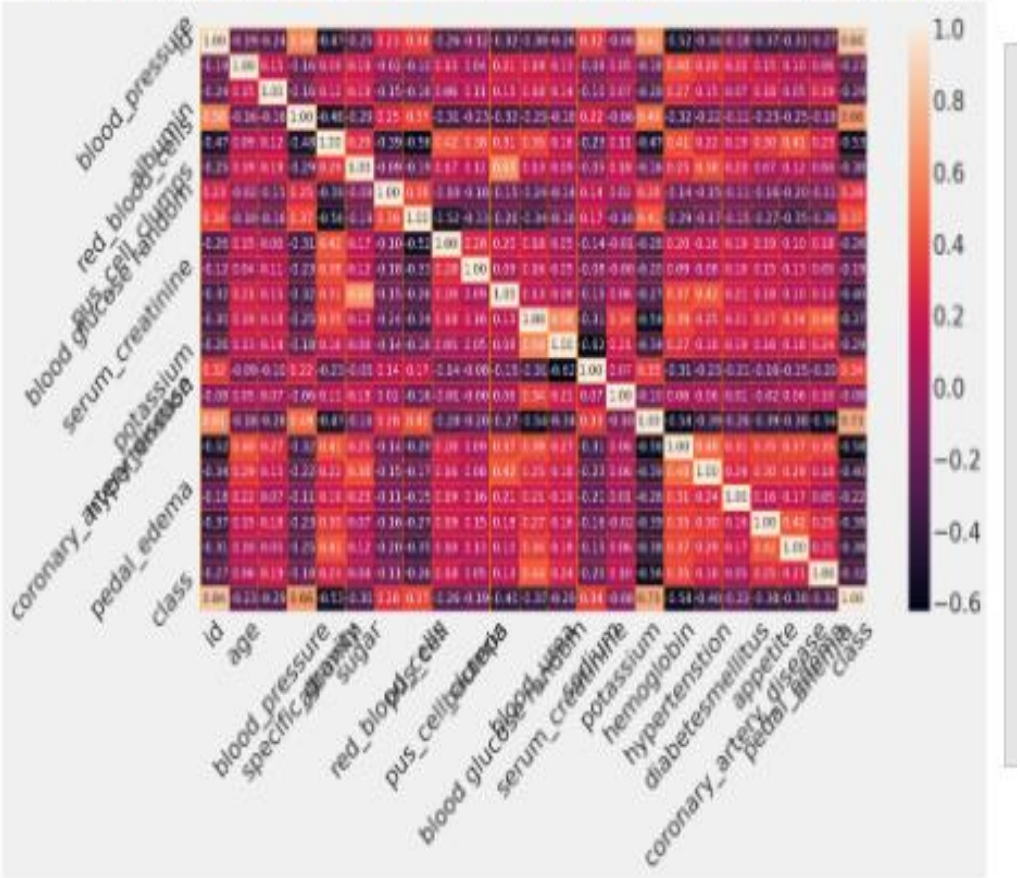
Multivariate analysis

```
/usr/local/lib/python3.9/dist-packages/IPython/core/pylabto
  fig.canvas.print_figure(bytes_io, **kw)
```

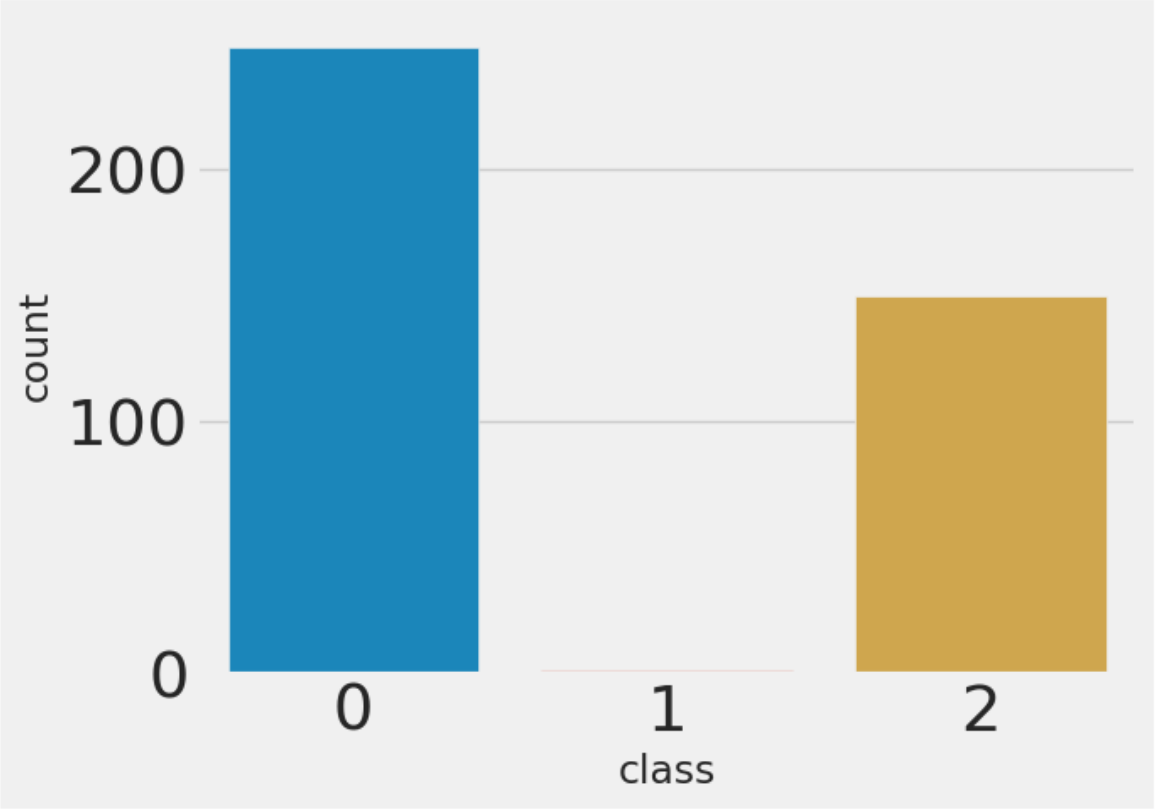


Finding correlation between the independent Columns

```
<ipython-input-563-3789efa46b1b>:3: FutureWarning: The defa
  sns.heatmap(data.corr(),annot=True,fmt=".2f",ax=ax,linewi
```



```
<Axes: xlabel='class', ylabel='count'>
```




```
0      0
1      0
2      0
3      0
4      0
      ..
395    2
396    2
397    2
398    2
399    2
Name: class, Length: 400, dtype: int64
```

	red_blood_cells	pus_cell	blood glucose random	blood_urea	pedal_edema	anemia	diabetesmellitus	coronary_artery_disease
205	1	1	100.000000	28.0	0	0	4	1
354	1	1	102.000000	17.0	0	0	3	1
3	1	0	117.000000	56.0	1	1	3	1
264	1	1	132.000000	24.0	0	0	3	1
194	1	0	148.036517	49.0	0	0	1	1
...	...	...	...	...	...	...	...	...
299	1	1	127.000000	48.0	0	0	3	1
22	1	0	95.000000	163.0	0	1	3	1
72	1	0	148.036517	35.0	1	0	4	1
15	1	1	76.000000	162.0	0	1	3	1
168	1	1	307.000000	28.0	0	0	4	1

320 rows × 8 columns

	class
205	0
354	2
3	0
264	2
194	0
...	...
299	2
22	0
72	0
15	0
168	0

320 rows × 1 columns

Model Building

ANN Model

```
Epoch 1/100
26/26 [==============================] - 1s 12ms/step - loss: 2.2482 - accuracy: 0.2422 - val_loss: 1.2385 - val_accuracy: 0.2031
Epoch 2/100
26/26 [==============================] - 0s 4ms/step - loss: 0.6130 - accuracy: 0.2852 - val_loss: 0.4760 - val_accuracy: 0.2812
Epoch 3/100
26/26 [==============================] - 0s 4ms/step - loss: 0.5477 - accuracy: 0.2461 - val_loss: 1.1688 - val_accuracy: 0.6562
Epoch 4/100
26/26 [==============================] - 0s 5ms/step - loss: 0.7138 - accuracy: 0.2539 - val_loss: 0.5490 - val_accuracy: 0.4531
Epoch 5/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4611 - accuracy: 0.2734 - val_loss: 0.5616 - val_accuracy: 0.4375
Epoch 6/100
26/26 [==============================] - 0s 5ms/step - loss: 0.4779 - accuracy: 0.2734 - val_loss: 1.0705 - val_accuracy: 0.1875
Epoch 7/100
26/26 [==============================] - 0s 4ms/step - loss: 0.6325 - accuracy: 0.2500 - val_loss: 0.3806 - val_accuracy: 0.2344
Epoch 8/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3746 - accuracy: 0.2383 - val_loss: 0.6245 - val_accuracy: 0.1875
Epoch 9/100
26/26 [==============================] - 0s 4ms/step - loss: 0.4739 - accuracy: 0.2461 - val_loss: 0.5668 - val_accuracy: 0.1875
Epoch 10/100
26/26 [==============================] - 0s 4ms/step - loss: 0.8838 - accuracy: 0.2812 - val_loss: 0.2906 - val_accuracy: 0.2969
Epoch 11/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3571 - accuracy: 0.2734 - val_loss: 0.3126 - val_accuracy: 0.2812
Epoch 12/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3477 - accuracy: 0.2422 - val_loss: 0.6372 - val_accuracy: 0.1875
Epoch 13/100
26/26 [==============================] - 0s 4ms/step - loss: 0.3018 - accuracy: 0.2344 - val_loss: 0.2780 - val_accuracy: 0.3281
Epoch 14/100
26/26 [==============================] - 0s 5ms/step - loss: 0.1611 - accuracy: 0.2500 - val_loss: 0.4496 - val_accuracy: 0.2031
```

```
Epoch 29/100
26/26 [==============================] - 0s 7ms/step - loss: -1.2488 - accuracy: 0.3125 - val_loss: -0.0597 - val_accuracy: 0.2500
Epoch 30/100
26/26 [==============================] - 0s 6ms/step - loss: -1.2488 - accuracy: 0.3164 - val_loss: -0.8083 - val_accuracy: 0.3125
Epoch 31/100
26/26 [==============================] - 0s 6ms/step - loss: -1.4867 - accuracy: 0.2930 - val_loss: -0.1097 - val_accuracy: 0.5469
Epoch 32/100
26/26 [==============================] - 0s 6ms/step - loss: -2.5510 - accuracy: 0.3164 - val_loss: -1.8023 - val_accuracy: 0.3281
Epoch 33/100
26/26 [==============================] - 0s 5ms/step - loss: -3.3051 - accuracy: 0.3398 - val_loss: -1.7565 - val_accuracy: 0.3125
Epoch 34/100
26/26 [==============================] - 0s 5ms/step - loss: -5.9388 - accuracy: 0.3359 - val_loss: -2.5469 - val_accuracy: 0.2656
Epoch 35/100
26/26 [==============================] - 0s 4ms/step - loss: -8.5665 - accuracy: 0.2969 - val_loss: -0.1802 - val_accuracy: 0.2344
Epoch 36/100
26/26 [==============================] - 0s 4ms/step - loss: -10.2331 - accuracy: 0.3047 - val_loss: -5.1283 - val_accuracy: 0.5156
Epoch 37/100
26/26 [==============================] - 0s 4ms/step - loss: -16.6568 - accuracy: 0.3555 - val_loss: -13.7441 - val_accuracy: 0.4531
Epoch 38/100
26/26 [==============================] - 0s 4ms/step - loss: -31.4417 - accuracy: 0.2891 - val_loss: -3.3614 - val_accuracy: 0.5469
Epoch 39/100
26/26 [==============================] - 0s 5ms/step - loss: -40.4390 - accuracy: 0.3242 - val_loss: -22.7793 - val_accuracy: 0.3125
Epoch 40/100
26/26 [==============================] - 0s 5ms/step - loss: -59.5310 - accuracy: 0.3281 - val_loss: -25.3112 - val_accuracy: 0.2344
Epoch 41/100
26/26 [==============================] - 0s 4ms/step - loss: -106.9210 - accuracy: 0.3242 - val_loss: -55.0516 - val_accuracy: 0.2656
Epoch 42/100
26/26 [==============================] - 0s 4ms/step - loss: -153.0772 - accuracy: 0.3203 - val_loss: -98.7748 - val_accuracy: 0.3281
```

```
Epoch 15/100
26/26 [==============================] - 0s 5ms/step - loss: 0.2236 - accuracy: 0.2578 - val_loss: 0.5958 - val_accuracy: 0.1875
Epoch 16/100
26/26 [==============================] - 0s 6ms/step - loss: 0.2300 - accuracy: 0.2891 - val_loss: 0.2058 - val_accuracy: 0.3438
Epoch 17/100
26/26 [==============================] - 0s 6ms/step - loss: 0.1782 - accuracy: 0.2539 - val_loss: 0.3520 - val_accuracy: 0.2031
Epoch 18/100
26/26 [==============================] - 0s 6ms/step - loss: 0.0676 - accuracy: 0.2695 - val_loss: 0.3515 - val_accuracy: 0.4375
Epoch 19/100
26/26 [==============================] - 0s 6ms/step - loss: 0.0707 - accuracy: 0.2930 - val_loss: 0.3547 - val_accuracy: 0.2188
Epoch 20/100
26/26 [==============================] - 0s 7ms/step - loss: 0.2881 - accuracy: 0.2656 - val_loss: 0.3327 - val_accuracy: 0.4844
Epoch 21/100
26/26 [==============================] - 0s 5ms/step - loss: 0.0803 - accuracy: 0.2969 - val_loss: 0.3763 - val_accuracy: 0.2656
Epoch 22/100
26/26 [==============================] - 0s 6ms/step - loss: 0.0457 - accuracy: 0.2812 - val_loss: 0.5639 - val_accuracy: 0.1719
Epoch 23/100
26/26 [==============================] - 0s 7ms/step - loss: -0.0087 - accuracy: 0.2383 - val_loss: 0.2283 - val_accuracy: 0.2656
Epoch 24/100
26/26 [==============================] - 0s 5ms/step - loss: -0.0372 - accuracy: 0.3125 - val_loss: 0.2404 - val_accuracy: 0.5625
Epoch 25/100
26/26 [==============================] - 0s 6ms/step - loss: -0.3087 - accuracy: 0.2969 - val_loss: 0.2049 - val_accuracy: 0.2344
Epoch 26/100
26/26 [==============================] - 0s 6ms/step - loss: -0.2205 - accuracy: 0.2617 - val_loss: -0.1962 - val_accuracy: 0.4062
Epoch 27/100
26/26 [==============================] - 0s 6ms/step - loss: -0.5698 - accuracy: 0.3203 - val_loss: -0.0593 - val_accuracy: 0.5312
Epoch 28/100
26/26 [==============================] - 0s 6ms/step - loss: -0.5763 - accuracy: 0.3242 - val_loss: -0.5026 - val_accuracy: 0.2969
```

```
Epoch 43/100
26/26 [==============================] - 0s 4ms/step - loss: -219.6395 - accuracy: 0.3086 - val_loss: -172.5312 - val_accuracy: 0.312
Epoch 44/100
26/26 [==============================] - 0s 4ms/step - loss: -316.4592 - accuracy: 0.3047 - val_loss: -148.0850 - val_accuracy: 0.4062
Epoch 45/100
26/26 [==============================] - 0s 4ms/step - loss: -454.2701 - accuracy: 0.3555 - val_loss: -268.1475 - val_accuracy: 0.2500
Epoch 46/100
26/26 [==============================] - 0s 5ms/step - loss: -501.3105 - accuracy: 0.3242 - val_loss: -413.6888 - val_accuracy: 0.4688
Epoch 47/100
26/26 [==============================] - 0s 4ms/step - loss: -400.8161 - accuracy: 0.3398 - val_loss: -415.1920 - val_accuracy: 0.2500
Epoch 48/100
26/26 [==============================] - 0s 5ms/step - loss: -1004.7648 - accuracy: 0.3125 - val_loss: -510.1062 - val_accuracy: 0.2812
Epoch 49/100
26/26 [==============================] - 0s 4ms/step - loss: -1306.7628 - accuracy: 0.2969 - val_loss: -677.8797 - val_accuracy: 0.3281
Epoch 50/100
26/26 [==============================] - 0s 4ms/step - loss: -1645.2295 - accuracy: 0.3555 - val_loss: -973.1434 - val_accuracy: 0.3281
Epoch 51/100
26/26 [==============================] - 0s 4ms/step - loss: -1785.2308 - accuracy: 0.3359 - val_loss: -1293.5042 - val_accuracy: 0.3125
Epoch 52/100
26/26 [==============================] - 0s 4ms/step - loss: -2392.9475 - accuracy: 0.3281 - val_loss: -1083.3379 - val_accuracy: 0.3125
Epoch 53/100
26/26 [==============================] - 0s 4ms/step - loss: -2848.7407 - accuracy: 0.3086 - val_loss: -1719.9709 - val_accuracy: 0.2812
Epoch 54/100
26/26 [==============================] - 0s 4ms/step - loss: -3587.9424 - accuracy: 0.3125 - val_loss: -2249.9258 - val_accuracy: 0.3125
Epoch 55/100
26/26 [==============================] - 0s 5ms/step - loss: -4841.3765 - accuracy: 0.3320 - val_loss: -3019.6611 - val_accuracy: 0.3438
Epoch 56/100
26/26 [==============================] - 0s 4ms/step - loss: -6083.5044 - accuracy: 0.3164 - val_loss: -3268.5559 - val_accuracy: 0.3594
Epoch 57/100
26/26 [==============================] - 0s 5ms/step - loss: -6245.4102 - accuracy: 0.3203 - val_loss: -1769.3025 - val_accuracy: 0.4844
```

```
Epoch 58/100
26/26 [==============================] - 0s 4ms/step - loss: -6651.9727 - accuracy: 0.3555 - val_loss: -4692.4668 - val_accuracy: 0.4
Epoch 59/100
26/26 [==============================] - 0s 4ms/step - loss: -8870.4355 - accuracy: 0.3086 - val_loss: -5611.0488 - val_accuracy: 0.3594
Epoch 60/100
26/26 [==============================] - 0s 4ms/step - loss: -11601.1172 - accuracy: 0.3125 - val_loss: -5759.3013 - val_accuracy: 0.3594
Epoch 61/100
26/26 [==============================] - 0s 4ms/step - loss: -13074.0078 - accuracy: 0.3203 - val_loss: -7920.7905 - val_accuracy: 0.3594
Epoch 62/100
26/26 [==============================] - 0s 4ms/step - loss: -16000.1396 - accuracy: 0.3320 - val_loss: -8545.4297 - val_accuracy: 0.3594
Epoch 63/100
26/26 [==============================] - 0s 5ms/step - loss: -18225.4531 - accuracy: 0.3242 - val_loss: -9022.6035 - val_accuracy: 0.2656
Epoch 64/100
26/26 [==============================] - 0s 4ms/step - loss: -17600.5039 - accuracy: 0.3164 - val_loss: -13098.4902 - val_accuracy: 0.3438
Epoch 65/100
26/26 [==============================] - 0s 4ms/step - loss: -21776.1719 - accuracy: 0.3359 - val_loss: -14211.8564 - val_accuracy: 0.2969
Epoch 66/100
26/26 [==============================] - 0s 4ms/step - loss: -26435.8203 - accuracy: 0.3164 - val_loss: -17618.7910 - val_accuracy: 0.3125
Epoch 67/100
26/26 [==============================] - 0s 5ms/step - loss: -30004.2539 - accuracy: 0.3203 - val_loss: -17396.3125 - val_accuracy: 0.3281
Epoch 68/100
26/26 [==============================] - 0s 4ms/step - loss: -34496.7617 - accuracy: 0.3203 - val_loss: -20633.0684 - val_accuracy: 0.3594
Epoch 69/100
26/26 [==============================] - 0s 4ms/step - loss: -42130.0039 - accuracy: 0.3477 - val_loss: -22152.7070 - val_accuracy: 0.3281
Epoch 70/100
26/26 [==============================] - 0s 4ms/step - loss: -43348.9258 - accuracy: 0.3281 - val_loss: -22096.2734 - val_accuracy: 0.2656
Epoch 71/100
26/26 [==============================] - 0s 4ms/step - loss: -53258.8477 - accuracy: 0.3281 - val_loss: -26602.9258 - val_accuracy: 0.2656
Epoch 72/100
26/26 [==============================] - 0s 4ms/step - loss: -58444.5586 - accuracy: 0.3242 - val_loss: -34258.4023 - val_accuracy: 0.3281
```

```
Epoch 73/100
26/26 [==============================] - 0s 4ms/step - loss: -67377.5312 - accuracy: 0.3164 - val_loss: -39737.2500 - val_accuracy:
Epoch 74/100
26/26 [==============================] - 0s 4ms/step - loss: -72065.3828 - accuracy: 0.3359 - val_loss: -44814.0156 - val_accuracy: 0.4219
Epoch 75/100
26/26 [==============================] - 0s 4ms/step - loss: -83202.5156 - accuracy: 0.3086 - val_loss: -52438.6602 - val_accuracy: 0.2969
Epoch 76/100
26/26 [==============================] - 0s 4ms/step - loss: -97375.8047 - accuracy: 0.3477 - val_loss: -49858.5703 - val_accuracy: 0.2656
Epoch 77/100
26/26 [==============================] - 0s 4ms/step - loss: -101303.4531 - accuracy: 0.3086 - val_loss: -58465.9180 - val_accuracy: 0.3594
Epoch 78/100
26/26 [==============================] - 0s 4ms/step - loss: -120416.6094 - accuracy: 0.3164 - val_loss: -63674.5430 - val_accuracy: 0.4062
Epoch 79/100
26/26 [==============================] - 0s 4ms/step - loss: -133291.5156 - accuracy: 0.3359 - val_loss: -74883.8125 - val_accuracy: 0.3594
Epoch 80/100
26/26 [==============================] - 0s 4ms/step - loss: -146242.5312 - accuracy: 0.3125 - val_loss: -89559.5234 - val_accuracy: 0.3438
Epoch 81/100
26/26 [==============================] - 0s 4ms/step - loss: -160524.6875 - accuracy: 0.3594 - val_loss: -92203.9141 - val_accuracy: 0.3594
Epoch 82/100
26/26 [==============================] - 0s 4ms/step - loss: -167900.7969 - accuracy: 0.3242 - val_loss: -104326.5938 - val_accuracy: 0.3594
Epoch 83/100
26/26 [==============================] - 0s 4ms/step - loss: -184419.9219 - accuracy: 0.3125 - val_loss: -108856.8047 - val_accuracy: 0.3594
Epoch 84/100
26/26 [==============================] - 0s 4ms/step - loss: -201372.6406 - accuracy: 0.3047 - val_loss: -121557.8359 - val_accuracy: 0.3594
Epoch 85/100
26/26 [==============================] - 0s 4ms/step - loss: -238435.8750 - accuracy: 0.3242 - val_loss: -124998.5625 - val_accuracy: 0.4062
Epoch 86/100
26/26 [==============================] - 0s 3ms/step - loss: -238217.1094 - accuracy: 0.3203 - val_loss: -143415.3594 - val_accuracy: 0.3125
Epoch 87/100
26/26 [==============================] - 0s 5ms/step - loss: -277840.3438 - accuracy: 0.3398 - val_loss: -148227.5312 - val_accuracy: 0.3594
```

```
26/26 [==============================] - 0s 5ms/step - loss: -277840.3438 - accuracy: 0.3398 - val_loss: -148227.5312 - val_accuracy: 0.3594
Epoch 88/100
26/26 [==============================] - 0s 4ms/step - loss: -281629.3125 - accuracy: 0.3164 - val_loss: -184511.0000 - val_accuracy: 0.3594
Epoch 89/100
26/26 [==============================] - 0s 4ms/step - loss: -326110.4375 - accuracy: 0.3398 - val_loss: -189703.6875 - val_accuracy: 0.3594
Epoch 90/100
26/26 [==============================] - 0s 4ms/step - loss: -354661.7188 - accuracy: 0.3242 - val_loss: -207185.8750 - val_accuracy: 0.3594
Epoch 91/100
26/26 [==============================] - 0s 4ms/step - loss: -384787.0312 - accuracy: 0.3164 - val_loss: -216284.1562 - val_accuracy: 0.3594
Epoch 92/100
26/26 [==============================] - 0s 4ms/step - loss: -410081.7812 - accuracy: 0.3281 - val_loss: -240569.1094 - val_accuracy: 0.3125
Epoch 93/100
26/26 [==============================] - 0s 4ms/step - loss: -422704.4062 - accuracy: 0.3125 - val_loss: -239625.1406 - val_accuracy: 0.3594
Epoch 94/100
26/26 [==============================] - 0s 4ms/step - loss: -484892.9062 - accuracy: 0.3711 - val_loss: -249042.3750 - val_accuracy: 0.2812
Epoch 95/100
26/26 [==============================] - 0s 4ms/step - loss: -448875.3750 - accuracy: 0.3438 - val_loss: -256833.6250 - val_accuracy: 0.2656
Epoch 96/100
26/26 [==============================] - 0s 4ms/step - loss: -492224.5312 - accuracy: 0.2930 - val_loss: -310311.0625 - val_accuracy: 0.3594
Epoch 97/100
26/26 [==============================] - 0s 4ms/step - loss: -583043.5000 - accuracy: 0.3086 - val_loss: -320750.0312 - val_accuracy: 0.3594
Epoch 98/100
26/26 [==============================] - 0s 4ms/step - loss: -628678.8125 - accuracy: 0.3477 - val_loss: -336677.6875 - val_accuracy: 0.2969
Epoch 99/100
26/26 [==============================] - 0s 4ms/step - loss: -656545.8125 - accuracy: 0.3281 - val_loss: -358740.8750 - val_accuracy: 0.3281
Epoch 100/100
26/26 [==============================] - 0s 4ms/step - loss: -731200.8750 - accuracy: 0.3203 - val_loss: -406546.7500 - val_accuracy: 0.3594
<keras.callbacks.History at 0x7f34e1fa2b80>
```

	red_blood_cells	pus_cell	blood glucose random	blood_urea	pedal_edema	anemia	diabetesmellitus	coronary_artery_disease
0	1	1	121.000000	36.0	0	0	4	1
1	1	1	148.036517	18.0	0	0	3	1
2	1	1	423.000000	53.0	0	1	4	1
3	1	0	117.000000	56.0	1	1	3	1
4	1	1	106.000000	26.0	0	0	3	1
...	...	...	...	...	...	...	...	...
395	1	1	140.000000	49.0	0	0	3	1
396	1	1	75.000000	31.0	0	0	3	1
397	1	1	100.000000	26.0	0	0	3	1
398	1	1	114.000000	50.0	0	0	3	1
399	1	1	131.000000	18.0	0	0	3	1

400 rows × 8 columns

	class
0	0
1	0
2	0
3	0
4	0
...	...
395	2
396	2
397	2
398	2
399	2

400 rows × 1 columns

```
shape of independent training data is{} (320, 8)
shape of independent testing data is{} (80, 8)
shape of dependent training data is{} (320, 8)
shape of dependent testing data is{} (80, 8)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

```
▾ LogisticRegression
LogisticRegression()
```

```
#RandomForestmodel
```

```
<ipython-input-594-b87bb2ba9825>:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,)
  rfc.fit(x_train,y_train)
```

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

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

```
array([0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2, 2,
       0, 2, 0, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0,
       0, 2, 0, 2, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 2, 0, 0, 2, 2, 0, 0, 0,
       0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 0, 0, 2, 0])
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

```
LogisticRegression
LogisticRegression()
```

```
[2]
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
array([2])
```

```
[2]
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature r
  warnings.warn(
array([2])
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature r
  warnings.warn(
array([2])
```

```
3/3 [==============================] - 0s 5ms/step
```

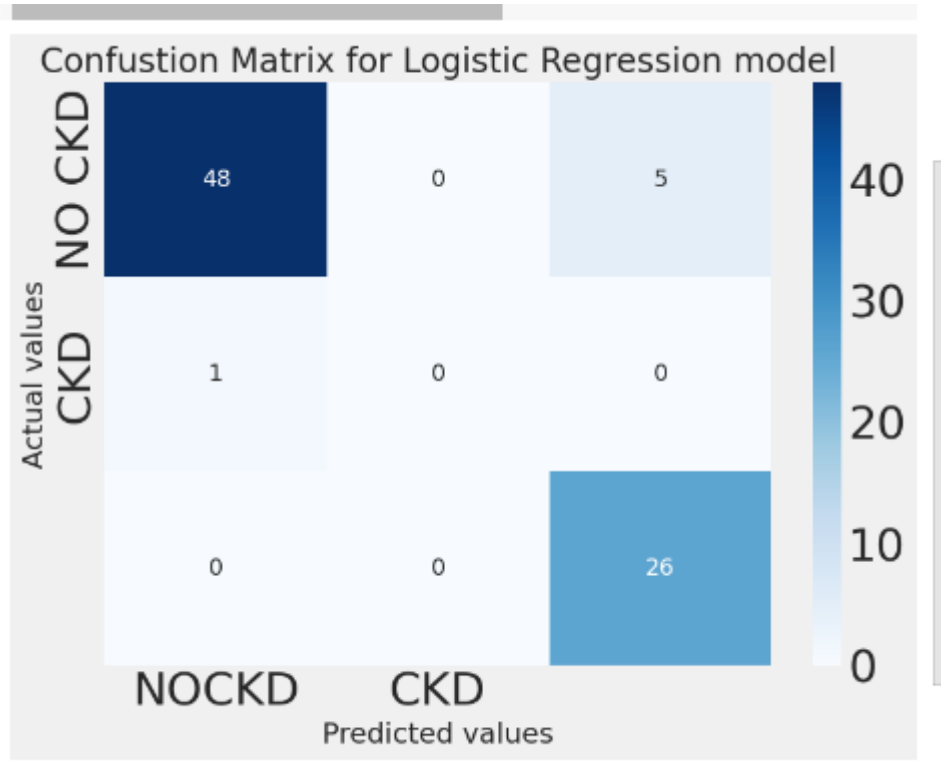
```
array([[1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.]], dtype=float32)
```

```
array([[ True],
       [ True],
       [False],
       [ True],
       [ True],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [ True],
       [ True],
       [False],
       [False],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [ True]])
```

```
1/1 [==============================] - 0s 92ms/step
prediction:High chance of CKD!
```

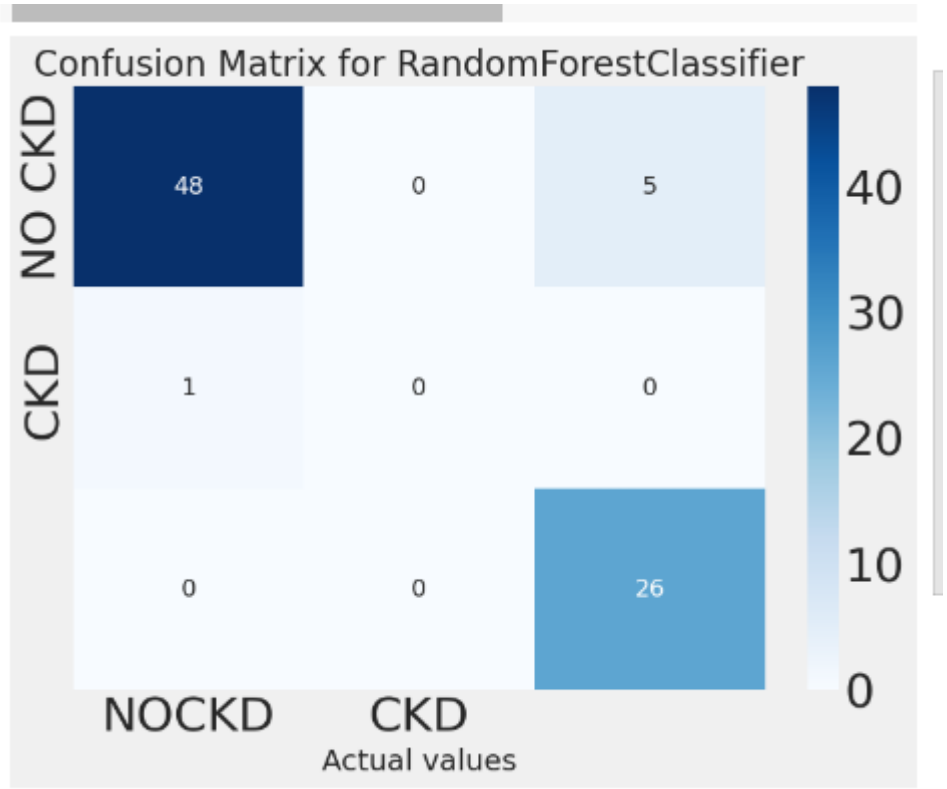
Logistic Regressign Model

```
array([[48,  0,  5],
       [ 1,  0,  0],
       [ 0,  0, 26]])
```



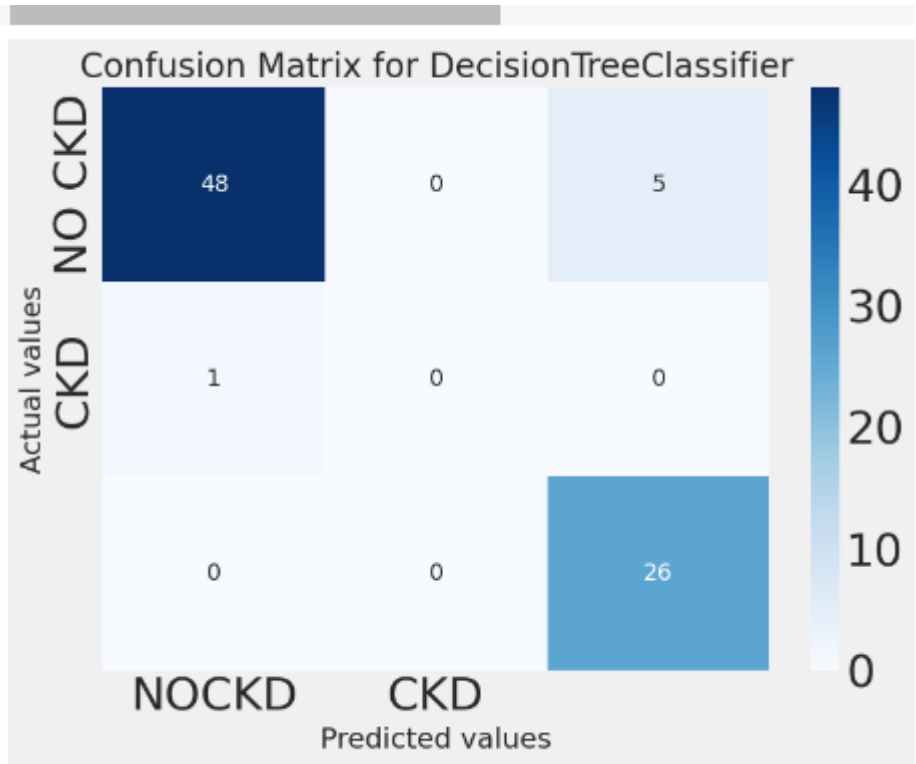
RandomForestCassifier

```
array([[48,  0,  5],
       [ 1,  0,  0],
       [ 0,  0, 26]])
```



DecisionTreeClassifier

```
array([[48,  0,  5],
       [ 1,  0,  0],
       [ 0,  0, 26]])
```

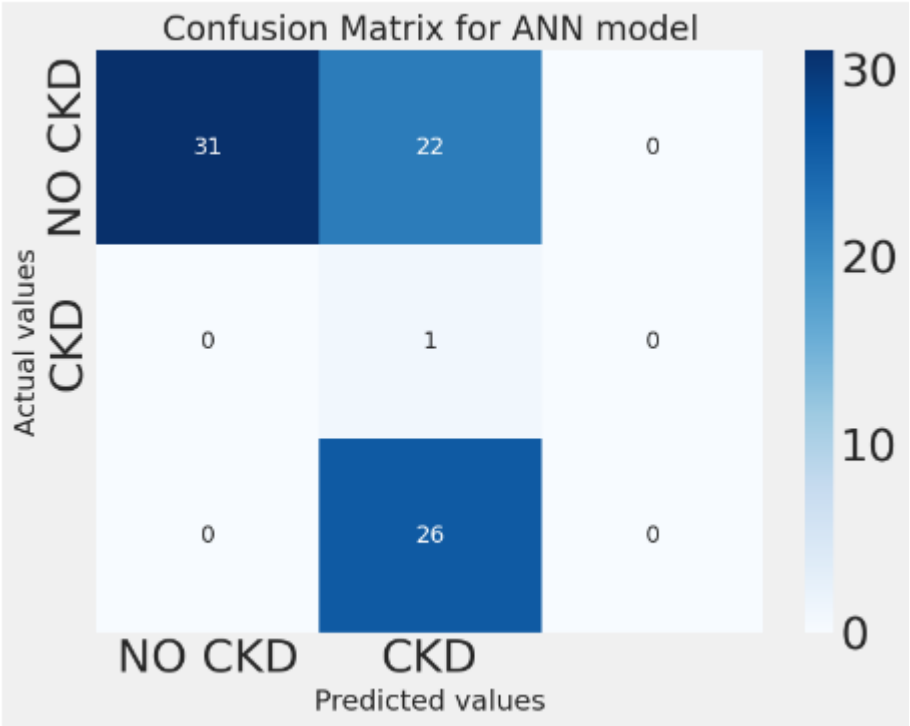


```
array([[48,  0,  5],
       [ 1,  0,  0],
       [ 0,  0, 26]])
```

```
              precision    recall  f1-score   support

           0       1.00      0.58      0.74        53
           1       0.02      1.00      0.04         1
           2       0.00      0.00      0.00        26

    accuracy                           0.40        80
   macro avg       0.34      0.53      0.26        80
weighted avg       0.66      0.40      0.49        80

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.6
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.6
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.6
  _warn_prf(average, modifier, msg_start, len(result))
```

ANN Model

```
array([[31, 22,  0],
       [ 0,  1,  0],
       [ 0, 26,  0]])
```



4 ADVANTAGES

- The early detection of CKD allows patients to receive timely treatment, showing the disease's progression. Due to its rapid recognition performance and accuracy, machine learning models can effectively assist physicians in achieving this goal.
- We purpose a machine learning methodology for the CKD diagnosis in this paper.
- Various researches have been carried out using machine learning techniques on the detection of CKD at the premature stage. Their focus was not mainly on the specific stages prediction.
- Diagnostic test results that would help to predict the likelihood of diagnoses or predict treatment's effect.
- Handling Categorical Data: In this step, data has been transformed into the required format.

- After preprocessing the data then the resultant CSV file comprises all the integer and float values for different CKD related features.
- Handling Missing Values: data is not always available (or missed) due to equipment malfunction, inconsistent with other recorded data and thus deleted, not entered into the database due to misunderstanding, some data may not be considered important at the time of entry.
- There are several ways of handling missing values including dropping missing values and filling missing values.
- Because the missing features are numeric and mean imputation is better for numerical missing values.
- After preprocessing the data then the resultant CSV file comprises all the integer and float values for different CKD related features.

DISADVANDAGES

- Having CKD increases the chances of having heart disease and stroke.
- Managing high blood pressure, blood sugar, and cholesterol levels—all factors that increase the risk for heart disease and stroke—is very important for people with CKD.
- 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
- Kidney diseases are a **leading cause of death** in the United States.
- CKD has varying levels of seriousness. It usually gets worse over time though treatment has been shown to slow progression.

5 APPLICATION

• Web-based applications

The internet has become one of the most important sources of health information for patients and their families. Recent studies suggest that most adults seek health information online.Many digital educational materials have been available on-line for patients with CKD by professional societies and patient advocacy groups, satisfying the knowledge component of Kolb's learning cycle. Systematic reviews of these educational materials suggest that most are adequate for use as determined by validated instruments, though relatively few are outstanding and many are written at a literacy level too high to be appreciated by most patients with CKD.

• Virtual support groups

The Internet has also become a resource for the development of social support systems for those affected by chronic diseases, including kidney disease.Internet support groups with videoconferencing and virtual group education classes, facilitated by health educators or peer leaders, deliver chronic disease education and promote collaborative problem solving, self-reflection and conceptualization.

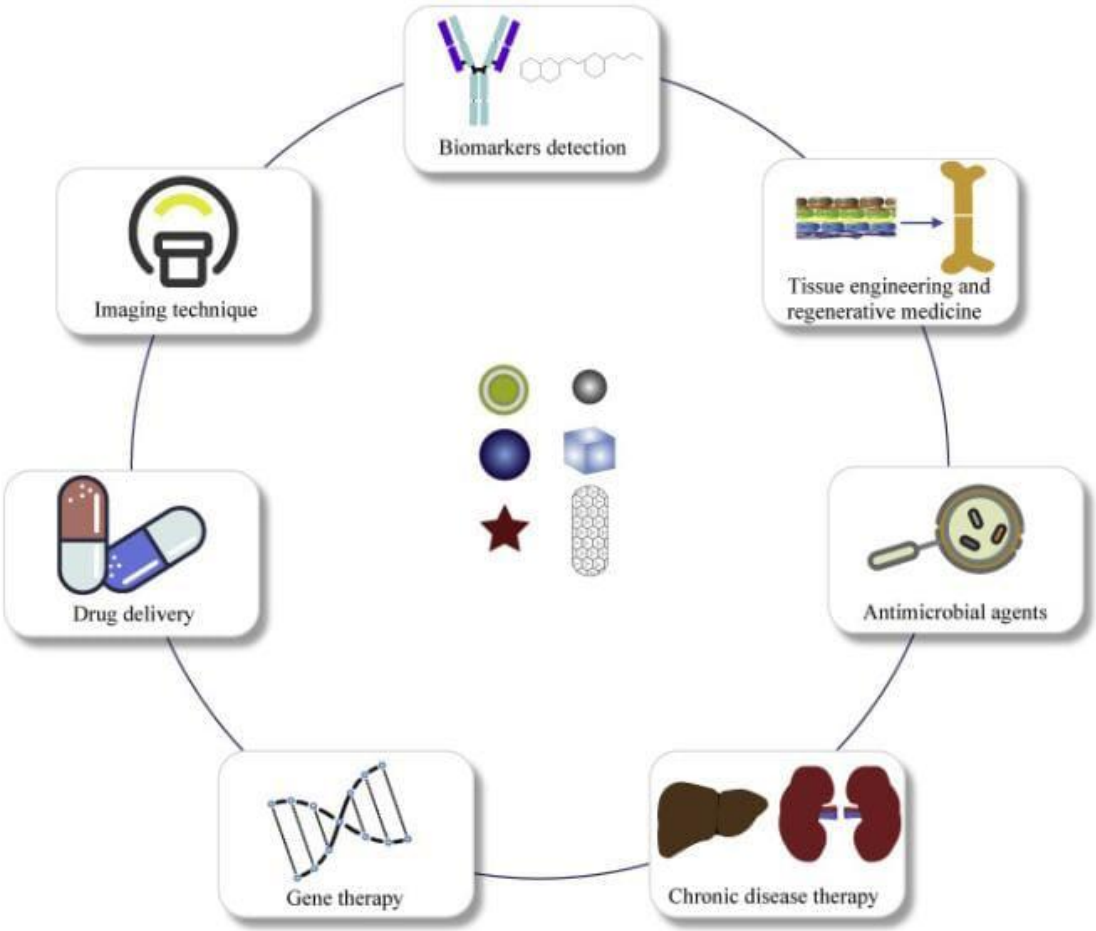
- ## Mobile Health Applications

Chronic disease education programs are increasingly adopting mobile health applications to support self-management practices, reinforcing both the knowledge and understanding components of Kolb's educational cycle. Many of these programs rely on wireless communication among peripheral objects (i.e., scales, blood pressure cuffs, glucometers) and smartphones, allowing patients to view their home-recorded data (i.e., blood pressure, weight, eating habits) and potentially submit them to a health care provider for clinical care.

- ## Interactive voice response (IVR)

IVR-based applications have great potential for delivery of CKD education, as they rely on simple-to-use telephone technology, are multilingual,

and require minimal literacy or numeracy skills



This point-of-care platform could perform tests anywhere from the home to the bedside and the data is transferred wirelessly to electronic equipment or the cloud for early detection and monitoring of CKD .

6. CONCLUTION

In this study, we developed and evaluated a series of artificial intelligence-based models considering minimum variables such as sex, age, comorbidities, and medications. These models predict patients' risk of developing chronic kidney disease after a period of 6 or 12 months. Among various models tested, convolutional neural networks (CNN) performed best, with an AUROC metric of 0.957 and 0.954 for 6 and 12 months, respectively. To see which features are the most prominent for prediction, we looked at the tree-based LightGBM model. The most prominent features included diabetes mellitus, age, gout, and use of sulfonamides and angiotensins, which are all reasonable in view of CKD. From a policymaker's point of view, these ML-based models could be

efficiently used in resource management and initiating public health initiatives such as closely monitoring and early detection of CKD.

7 FUTURE SCOPE

This study used a supervised machine-learning algorithm, feature selection methods to select the best subset features to develop the models. It is better to see the difference in performance results using unsupervised or deep learning algorithms models. To proposed model supports the experts to give the fast decision, it is better to make it a mobile-based system that enables the experts to follow the status of the patients and help the patients to use the system to know their status.

Chronic kidney disease (CKD) is a major emerging global public health problem that affects >850 million people and is currently

one of the most common diseases worldwide . The large number of comorbidities that accompany CKD, large number of prescribed medications, poor mental health condition, high hospitalization rate and high mortality rate illustrate that the patient complexity of CKD is enormous, and surpassed all other medical specialities when nine markers of complexity were assessed . Although significant progress has been made in the understanding of the causes of kidney disease and factors that drive progression to end-stage kidney disease, clinical decision support systems tailoring individual patient therapy may reduce the risk of progression and be used to monitor and change the therapy in individual CKD patients .

8 APPENDIX

A. **Source code**

Task 1:Problem Understanding

1)Specify the business problem,

2)Business requirements

3)Literature Survey

4)Social/Business impact

Task 2:Data Understanding

1)Data collection

2)Loading data

Task 3:EDA

1)Discriptive statistical

2)Visual Analysis

Task 4:Model Building

Task 5:Testing and model

Task 6:deplyment

Task 7:Doc

```
import pandas as pd
import numpy as np
from collections import Counter as c
import seaborn as sns
```

```
import missingno as msno
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pickle
plt.style.available
plt.style.use("fivethirtyeight")
data=pd.read_csv("/content/kidney_disease.csv")
data.head()
data.columns
data.columns=['id','age','blood_pressure','specific_gravity','albumin',
'sugar','red_blood_cells','pus_cell','pus_cell_clumps','bacteria','bloo
d_glucose_random','blood_urea','serum_creatinine','sodium','potassium',
'hemoglobin','packed_cell_volume','white_blood_cell_count','red_blood_c
ell_count','hypertenstion','diabetesmellitus','coronary_artery_disease'
,'appetite','pedal_edema','anemia','class']
data.columns
data.info()
data.isnull().any()
data['blood_glucose_random'].fillna(data['blood_glucose_random'].mean()
,inplace=True)
data['blood_pressure'].fillna(data['blood_pressure'].mean(),inplace=Tru
e)
data['blood_urea'].fillna(data['blood_urea'].mean(),inplace=True)
data['hemoglobin'].fillna(data['hemoglobin'].mean(),inplace=True)
data['packed_cell_volume'].fillna(data['packed_cell_volume'].mode()[0],
inplace=True)
data['potassium'].fillna(data['potassium'].mean(),inplace=True)
data['red_blood_cell_count'].fillna(data['red_blood_cell_count'].mean,i
nplace=True)
data['serum_creatinine'].fillna(data['serum_creatinine'].mean(),inplace
=True)
data['sodium'].fillna(data['sodium'].mean(),inplace=True)
data['white_blood_cell_count'].fillna(data['white_blood_cell_count'].mo
de()[0],inplace=True)
data['age'].fillna(data['age'].mode()[0],inplace=True)
data['hypertenstion'].fillna(data['hypertenstion'].mode()[0],inplace=Tr
ue)
data['pus_cell_clumps'].fillna(data['pus_cell_clumps'].mode()[0],inplac
e=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],inplac
e=True)
```

```
data['coronary_artery_disease'].fillna(data['coronary_artery_disease'].
mode()[0],inplace=True)
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],inplace=True)
data['diabetesmellitus'].fillna(data['diabetesmellitus'].mode()[0],inpl
ace=True)
data['pedal_edema'].fillna(data['pedal_edema'].mode()[0],inplace=True)
data['specific_gravity'].fillna(data['specific_gravity'].mode()[0],inpl
ace=True)
catcols=set(data.dtypes[data.dtypes=='O'].index.values)
print(catcols)
for i in catcols:
    print("columns:",i)
    print(c([i]))
    print('*'*120 +' \n')
catcols.remove('red_blood_cell_count')
catcols.remove('packed_cell_volume')
catcols.remove('white_blood_cell_count')
print(catcols)
catcols=['anemia','pedal_edema','appetite','bacteria','class','coronary
_artery_disease','diabetesmellitus','hypertenstion','pus_cell','pus_cel
l_clumps','red_blood_cells']
from sklearn.preprocessing import LabelEncoder
for i in catcols:
  print("LABLE ENCODING:",i)
  LEi=LabelEncoder()
  print(c(data[i]))
  data[i]=LEi.fit_transform(data[i])
  print(c(data[i]))
  print("*"*100)
contcols=set(data.dtypes[data.dtypes!='O'].index.values)
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)
data['coronary_artery_disease']=data.coronary_artery_disease.replace('\
tno','no')
data['coronary_artery_disease']
```

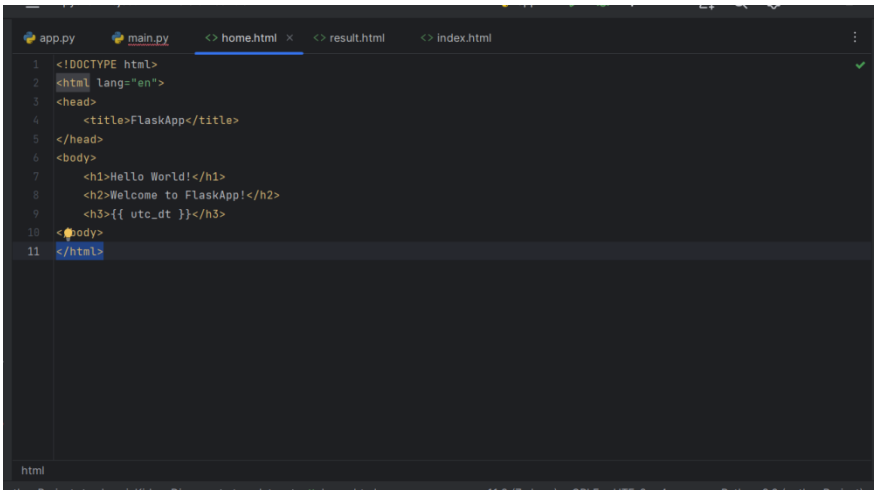
```
from matplotlib.font_manager import dataclasses
data['diabetesmellitus']=data.diabetesmellitus.replace(to_replace={'\tn
o':'no','\types':'yes'})
data['diabetesmellitus']
data.describe()
sns.distplot(data.age)
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(6,6))
plt.scatter(data['age'],data['blood_pressure'],color='blue')
plt.xlabel('age')
plt.ylabel('blood pressure')
plt.title("age vs blood pressure")
plt.figure(figsize=(20,15),facecolor='white')
plotnumber=1

for column in contcols:
  if plotnumber<=11 :
    ax=plt.subplot(3,4,plotnumber)
    plt.scatter(data['age'],data[column])
    plt.xlabel(column,fontsize=20)
  plotnumber+=1
plt.show()
f,ax=plt.subplots(figsize=(18,10))
sns.heatmap(data.corr(),annot=True,fmt=".2f",ax=ax,linewidths=0.5,linec
olor="orange")
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.show()
sns.countplot(x=data['class'])
data['class'].replace({"CKD":1,"CKD\t":2,"NO CKD":0})
X=data.drop('class',axis=1)
Y=data['class']
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
selcols=['red_blood_cells','pus_cell','blood glucose random','blood_ure
a','pedal_edema','anemia','diabetesmellitus','coronary_artery_disease']
X=pd.DataFrame(data,columns=selcols)
Y=pd.DataFrame(data,columns=['class'])
print(X.shape)
print(Y.shape)
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=2)
x_train
y_train
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
classification=Sequential()
classification.add(Dense(26,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='sigmoid'))
classification.compile(optimizer='adam',loss='binary_crossentropy',metr
ics=['accuracy'])
classification.fit(x_train,y_train,batch_size=10,validation_split=0.2,e
pochs=100)
print("shape of independent training data is{}",format(x_train.shape))
print("shape of independent testing data is{}",format(x_test.shape))
print("shape of dependent training data is{}",format(x_train.shape))
print("shape of dependent testing data is{}",format(x_test.shape))
log_r = LogisticRegression()
log_r.fit(x_train,y_train)
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=10,criterion='entropy')
y_predict = rfc.predict(x_test)
y_predict_train = rfc.predict(x_train)
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(max_depth=4,splitter='best',criterion='ent
ropy')
dtc.fit(x_train,y_train)
y_predict= dtc.predict(x_test)
y_predict
y_predict_train = dtc.predict(x_train)
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()
lgr.fit(x_train,y_train)
from sklearn.metrics import accuracy_score,classification_report
y_predict=lgr.predict(x_test)
y_pred = lgr.predict([[1,1,121.000000,36.0,0,0,1,0]])
print(y_pred)
(y_pred)
y_pred=dtc.predict([[1,1,121.000000,36.0,0,0,1,0]])
print(y_pred)
(y_pred)
y_pred=rfc.predict([[1,1,121.000000,36.0,0,0,1,0]])
(y_pred)
classification.save("ckd.h5")
y_pred=classification.predict(x_test)
y_pred
y_pred=(y_pred>0.5)
y_pred
def predict_exit(sample_value):
  sample_value=np.array(sample_value)#Convert list to numpy array
  #Reshape because sample_value contain only 1 record
```

```
  sample_value=sample_value.reshape(1,-1)
  #Feature scaling
  sample_value=sc.transform(sample_value)
  return classifier.predict(sample_value)
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')
from sklearn import model_selection
from pandas import DataFrame
from sklearn import model_selection

dfs = []
models = [('LogReg',LogisticRegression()),
    ('RF',RandomForestClassifier()),
    ('DecisionTree',DecisionTreeClassifier())]
results = []
names = []
scoring = ['accuracy','precision_weighted','recall_weighted','f1_weight
ed','roc_auc']
target_names = ['NO CKD',' CKD']
for name, model in models:
      kfold = model_selection.KFold(n_splits=5, shuffle=True, random_st
ate=90210)
      cv_results= model_selection.cross_validate(model, x_train, y_trai
n, 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,target_names=target_nam
es))
      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)
return final
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_predict)
cm
plt.figure(figsize=(8,6))
sns.heatmap(cm,cmap='Blues',annot=True,xticklabels=['NOCKD','CKD'],ytic
klabels=['NO CKD','CKD'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confustion Matrix for Logistic Regression model')
```

```
plt.show()
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
plt.figure(figsize=(8,6))
sns.heatmap(cm,cmap='Blues',annot=True,xticklabels=['NOCKD','CKD'],ytic
klabels=['NO CKD','CKD'])
plt.xlabel('Predicted values')
plt.xlabel('Actual values')
plt.title('Confusion Matrix for RandomForestClassifier')
plt.show()
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
plt.figure(figsize=(8,6))
sns.heatmap(cm,cmap='Blues',annot=True,xticklabels=['NOCKD','CKD'],ytic
klabels=['NO CKD','CKD'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for DecisionTreeClassifier')
plt.show()
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
print(classification_report(y_test,y_pred))
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
#plotting confusion matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
plt.figure(figsize=(8,6))
sns.heatmap(cm,cmap='Blues',annot=True,xticklabels=['NO CKD','CKD'],yti
cklabels=['NO CKD','CKD'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for ANN model')
plt.show()
bootstraps=[]
for model in list(set(final.model.values)):
  model_df=final.loc[final==model]
  bootstrap=model_df.sample(n=30,replace=True)
  bootstraps.append(bootstrap)

bootstrap_df=pd.concat(bootstraps,ignore_index=True)
results_long=pd.melt(bootstrap_df,id_vars=['model'],var_name='metrics',
value_name='values')
time_metrics=['fit_time','score_time']#fit time metrics
##PERFORMANCE METRICS
```

```
result_long_nofit=result_long.loc[~results_long['metrics'].isin(time_me
trics)]#get df without fit data
results_long_nofit=result_long_nofit.sort_values(by='values')
##TIME METRICS
result_long_fit=result_long_loc[results_long['metrics'].isin(time_metri
cs)]#df with fit data
results_long_fit=result_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)
g=sns.boxplot(x="model",y="values",hue="metrics",data=results_long_nofi
t,palette="set3")
plt.legend(bbox_to_anchor=(1.05,1),loc=2,borderaxespad=0.)
plt.title('Comparison of model by Classification Metric')
plt.savefig('./benchmark_model_performance.png',dpi=300)
```



app.py main.py home.html result.html index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>FlaskBlog</title>
</head>
<body>
    <h1>Welcome to FlaskBlog</h1>
</body>

```

html

app.py main.py home.html result.html index.html

```html
            font-size:3em;
            margin-left:50px;
            text-decoration:none;
        }
    </style>
</haad>
<body>
    </nav>
        <a href="%">FlaskApp</a>
        <a href="#">About</a>
    </nav>
    <hr>
    <div class="content">
        {% block content %} {% endblock %}
    </div>
</body>
</html>
```

html › body › div.content

app.py main.py home.html result.html index.html

```python
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.


1 usage
def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
# print(f'Hi, {name}')  # Press Ctrl+F8 to toggle the breakpoint.

# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
S
```

app.py × main.py home.html result.html index.html

```python
import numpy as np
import pickle
app=flask(__name__)
model=pickle.load(open('CKD.pkl','rb'))
@app.route('/')
def home():
    return render_template('home.html')
@app.route('/prediction',methods=['POST','GET'])

def prediction():
    return render_template('indexnew.html')
@app.route('\Home',methods=['POST','GET'])
def my_home():
    return render_template('home.html')

1 usage (1 dynamic)
@app.route('/predict',methods=['POST'])
def predict():
    #reading the inputs given by the user
    input_features=[float(x)for x in request.form.values()]
    features_value=[np.array(input_features)]
```

my_home()

app.py × main.py home.html result.html index.html

```python
@app.route('/predict',methods=['POST'])
def predict():
    #reading the inputs given by the user
    input_features=[float(x)for x in request.form.values()]
    features_value=[np.array(input_features)]

    feature_name=['blood_urea','blood glucose random','anemia','coronary_artery_disease','pus_cell','red_blood_cells','d
    df=pd.DatFrame(feature_value,colums=features_name)

    #predictions using the loaded model file
    output=model.predict(df)
    #showing the prediction results in a UI# showing the prediction results in a UI
    return render_template('result.html',prediction_text=output)
if name_=='--main--':
# running the app
app.run(debug=True)
```