Early Prediction For Chronic Kidney Disease Detection: A Progressive Approach To Health Management

Project Submitted by:

Team Leader:

K.M.Nivethini (Register number:1420128010)

Team members:

1. M.Velmurugan (Register number: 1420128028)
2. M.M.Ananthakrishnan (Register number: 1420128019)
3. G.R.Balaji (Register number:1420128020)

ABSTRACT:

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 todays 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 pretrained model to predict the Chronic Kidney Disease which can help in treatments of peoples who are suffer from this disease.

Introduction:

The present era, especially the last two decades, can be named the era of big data where digital data is turning out to be very crucial more and more in various fields such as science, healthcare, technology, and society. Huge data volumes have been produced and generated from multiple sensor networks and mobile applications in almost all fields, including healthcare in specific, and this multitude of data volumes is what we call big data [1]. Wide variety of data sources such as streaming machines, high-end output instruments, visualizing, and knowledge extraction across these vast and diverse types of data pose a significant challenge when sufficient cutting-edge technologies and tools are not used. One of the most eminent technological challenges facing big data analytics lays in exploring ways that are adequate to obtain useful and relevant information for different user categories in an effective manner.

Nowadays, the different forms and types of data sources in healthcare are being gathered in both clinical and nonclinical environments, where the most crucial data in healthcare analytics is the digital copy of a patient's medical history. On that account, the process of designing and making up a distributed data system to handle big data is challenged by three main issues. The first challenge is that it is difficult to collect data from distributed locations because of the diverse and large data volume. The second challenge is that storage is the chief issue for heterogeneous and enormous datasets as big data system requires to store while allowing performance guarantee. The third challenge is more connected to big data analytics, specifically to enormous mining datasets in real time, and this includes visualization, prediction, and optimization [2].

Considering the difficulty imposed by these challenges, they require an up-to-date and advanced processing paradigm provided that the present data

management systems do not provide adequate efficiency in handling the heterogeneous nature of data or the real-time aspect. Traditional database management systems cannot support the continuous increase in huge data size. To address these issues related to enormous and heterogeneous data storage, the research community has proposed a number of research works, such as Apache Spark, Apache Hadoop [3], Apache Kafka [4], and Apache Storm [5], to solve healthcare problems [6–8].

Chronic kidney disease (CKD) has received a lot of interest due to its high death rate. Chronic diseases have become a major hazard to emerging countries, according to the World Health Organization (WHO) [9]. CKD is a kidney illness that can be treated in its early stages, but it eventually leads to renal failure if not treated early. In 2016, chronic kidney disease claimed the lives of 753 million individuals globally, accounting for 336 million male deaths and 417 million female deaths [10]. Chronic renal disease can be prevented from progressing to kidney failure if diagnosed and treated early. Diagnosing chronic kidney disease early is the best method to treat it, while delaying treatment until it is too late may lead to renal failure, which necessitates dialysis or kidney transplantation to live normally. Therefore, global strategies for early detection and treatment of people with CKD are required. To mine hidden patterns from data for effective decision-making and to help doctors in making more accurate diagnoses, a computer-aided diagnosis system based on artificial intelligence strategies is needed for clinical information. Artificial intelligence techniques (machine learning and deep learning) have been used in the health field, namely, in disease prediction and diagnosis.

Chronic kidney disease (CKD) is a condition that affects the kidney's ability to function. In general, CKD is separated into phases, with renal failures occurring

when the kidneys are no longer able to complete their roles of blood purification and mineral balance in the body [11]. According to the current estimates, CKD is more common in adults over 65 years old (38%) than in people aged 45–64 years (12%) and people aged 18–44 years (6%). Women have a rather higher rate of CKD (14%) than males [12].

Machine learning is an exciting field that focuses on studying huge amounts of data with multiple variables. Machine learning has basically developed from studying the theory of pattern recognition and computational learning in artificial intelligence; it presupposes computational methods, algorithms, and analysis techniques. From the perspective of Medical Sciences, machine learning undertakes to aid health specialists and doctors in carrying out scintillate and flawless diagnoses, choosing the best-fit medicines for patients, determining patients at high risk, and, most importantly, improving patients' physical condition with minimal cost.

Machine learning (ML) has demonstrated remarkable performance across a range of applications, such as speech recognition [13], computer vision [14], medical diagnostics [15], and engineering [16].

Being a constituent of the ML process, feature selection (FS) is a crucial preprocessing step that determines the most relevant attributes within a dataset. Removing unimportant and unnecessary attributes can result in less complicated and more accurate models. In this paper, two feature selection methods based on Apache Spark are used, namely, Relief-F [17] and chi-squared [18] feature selection method. Some of the research works have used ML techniques to predict CKD. For example, Charleonnan [19] et al. used four ML algorithms, K-nearest neighbors (KNN), support vector machine (SVM), logistic regression (LR), and decision tree (DT), to predict CKD. Other research

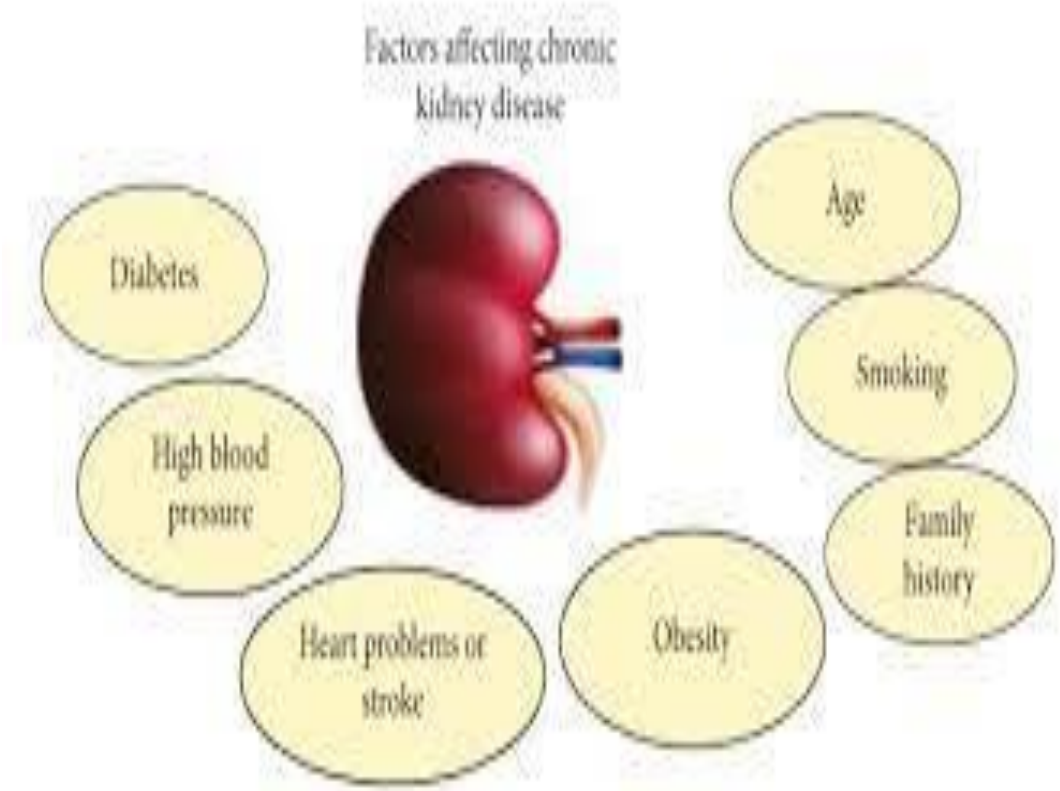
works used hybrid ML algorithms that are integrated between feature selection methods and ML to predict CKD. Feature selection methods have been used to reduce the number of features and select the optimal subsets of features from the dataset. For example [20], authors used chi-square, correlation-based feature selection (CFS), and Lasso feature selection to select the essential features from the database. They applied artificial neural network (ANN), C5.0, LR, SVM, KNN, and RF to both full features and the selected features.

Recently, researchers have been using big data platforms such as Apache Spark [21] which is a large-scale data processing engine with a unified analytics engine. Spark is 100 times quicker than Hadoop in running workloads on large-scale clusters. It includes Java, Scala, *Python*, and *R* high-level APIs, as well as an efficient engine that supports broad execution graphs. It also includes a number of higher-level tools such as Spark SQL for SQL and structured data processing, MLlib, GraphX, and Structured Streaming.

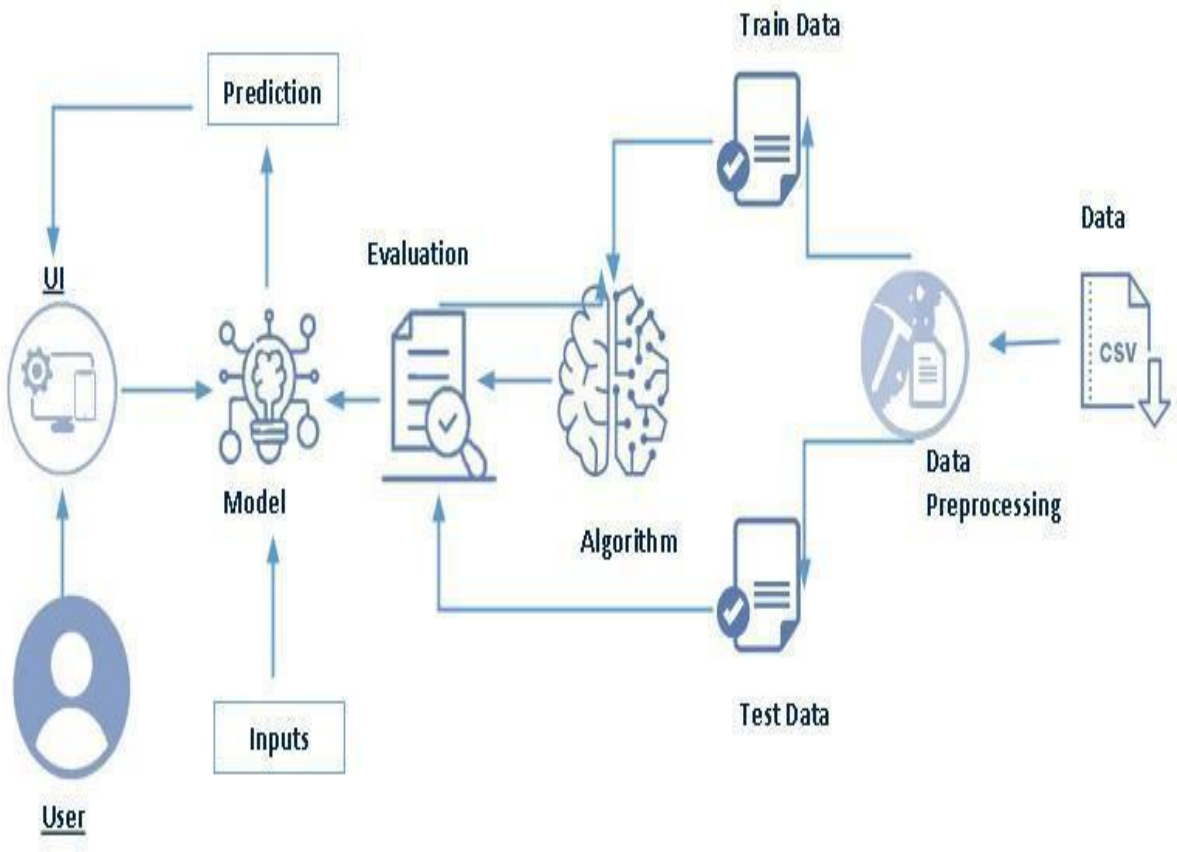
Spark's machine learning (ML) [21] library is called MLlib. Its purpose is to make scalable and simple machine learning a reality. It provides, at a high level, tools such as classification, regression, clustering, and collaborative filtering as examples of machine learning algorithms. It also provides feature extraction, transformation, dimensionality reduction, and selection as examples of featurization.

The previous studies of CKD prediction have not used big data platforms to solve this problem. The goal of this work is to predict CKD using hybrid ML techniques based on Apache Spark to predict CKD. Our contribution can be summarized as follows: Developing hybrid ML techniques based on Apache Spark to predict CKD Applying feature selection algorithms to select the

important features from the dataset Applying optimization techniques, including grid search with cross-validation to optimize ML algorithms to enhance performance Applying different ML classification algorithms to both full features and the selected features Applying ensemble learning such as Gradient-Boosted Trees based on Apache Spark to predict CKD.



Technical architecture:



Project flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
  - Specify the business problem
  - Business requirements
  - Literature Survey
  - Social or Business Impact.
- Data Collection & Preparation
  - Collect the dataset
  - Data Preparation
- Exploratory Data Analysis
  - Descriptive statistical
  - Visual Analysis
- Model Building
  - Training the model in multiple algorithms
  - Testing the model
- Performance Testing & Evaluate the results
  - Testing model with multiple evaluation metrics
  - Evaluate the results
- Model Deployment
  - Save the best model
  - Integrate with Web Framework
- Project Demonstration & Documentation
  - Record explanation Video for project end to end solution
  - Project Documentation-Step by step project development procedure

Test for CKD

- Chronic kidney disease is when a disease or condition makes it hard for the kidneys to work, causing the damage to the kidneys to get worse over time. This can occur when the kidneys are affected by another disease or condition.

- Studies[6,8] show that the number of people with CKD who are admitted to hospitals is going up by 6.23 percent every year, even though the global death rate has stayed the same.[8] There are just a few diagnostic tests available to check the status of CKD, including: (i) estimated glomerular filtration rate (eGFR) (ii) a urine test; (iii) a blood pressure reading; (iv) tests for CKD.

- The eGFR value provides information on how well your kidneys cleanse the blood. If your eGFR number is higher than 90, it means that your kidneys are working well. If the value of your eGFR is less than 60, this indicates that you have CKD.[8]

- In order to evaluate kidney function, the physician also requests a urine sample. Urine is produced by the kidneys. If your urine contains blood and protein,[24] it is an indication that one or both of your kidneys are not functioning normally.

- The doctor takes your blood pressure because the range of your blood pressure reveals how well your heart is pumping blood. If the patient's eGFR value falls below 15, this means they have reached the end stage of kidney disease. There are just 2two treatments that are now available for renal failure: (i) dialysis and (ii) kidney transplantation. The patient's life expectancy after dialysis is contingent on a number of characteristics, including age, gender, the frequency and length of dialysis treatments, the patient's level of physical mobility, and their mental state.[10] Kidney transplantation is the only option left for the doctor to consider if dialysis cannot be performed successfully. Nevertheless, the price is exorbitantly high.[15]

- When determining the extent of the damage to your kidneys, it is not uncommon for additional tests to be performed. These may include an ultrasound scan, a magnetic resonance imaging scan, or a computed tomography scan. Their purpose is to look at the kidneys and see if there are any blockages. A needle is used to take a small piece of kidney tissue, and the cells are looked at under a microscope to look for signs of kidney disease. This is done in order to diagnose kidney conditions.

- The field of medicine is an extremely important area for the application of intellectually sophisticated systems.[21] Then, data mining could be a big part of finding hidden information in the huge amount of patient medical and treatment data. This is information that doctors often get from their patients to learn more about their symptoms and make more accurate treatment plans.

Milestone – 1:

Define Problem / Problem Understanding:

In This milestone, we will see the problem understanding.

Specify The Business Problem

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 todays 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 pretrained model to predict the Chronic Kidney Disease which can help in treatments of peoples who are suffer from this disease.

Business Requirements

The business requirements for a machine learning model to predict chronic kidney disease include the ability to accurately predict the ckd based on given information, Minimise the number of false positives (predicting diseased) and false negatives (not diseased). Provide an explanation for the model's decision, to comply with regulations and improve transparency.

Literature Survey

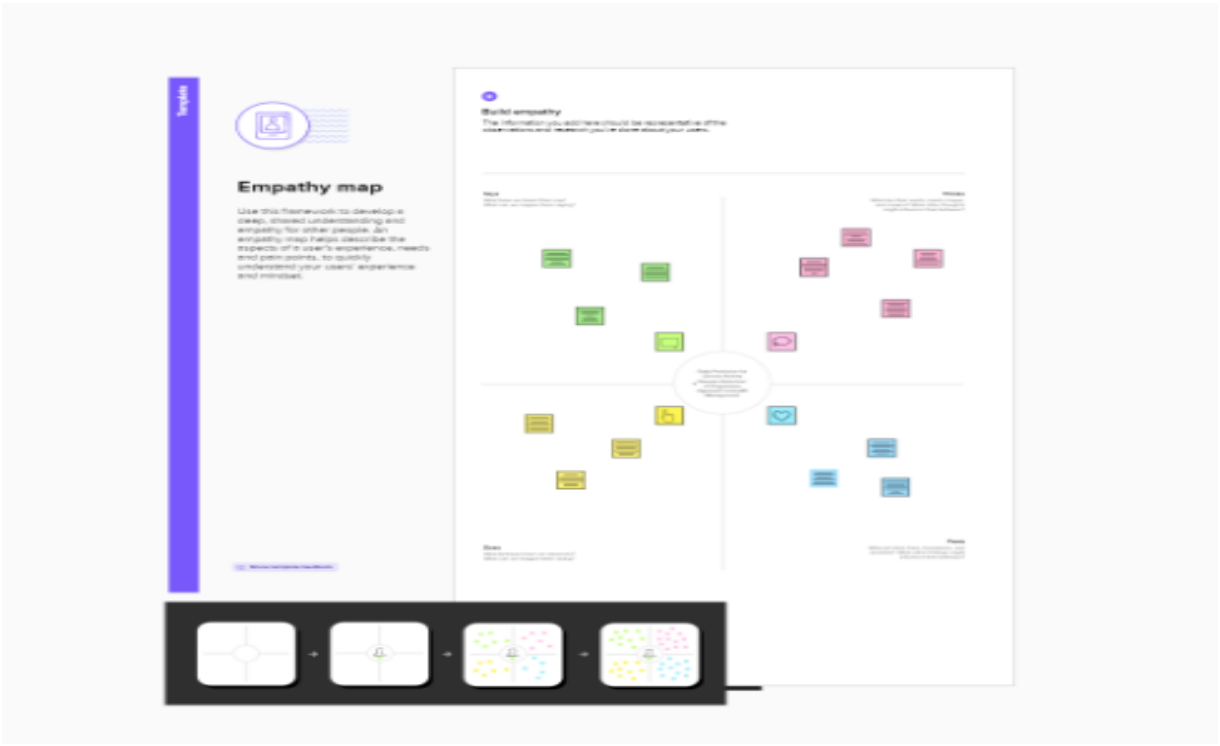
Chronic kidney disease (CKD) is a significant public health issue, affecting an estimated 14% of the global population. The disease is characterized by a gradual loss of kidney function over time, leading to a range of serious health complications, including end-stage renal disease (ESRD) requiring dialysis or kidney transplant. Early detection and management of CKD is crucial to prevent progression to ESRD and improve patient outcomes.

There have been numerous studies in recent years aimed at developing accurate and efficient methods for predicting CKD progression. These studies have employed a variety of techniques, including machine learning, deep learning, and artificial neural networks.

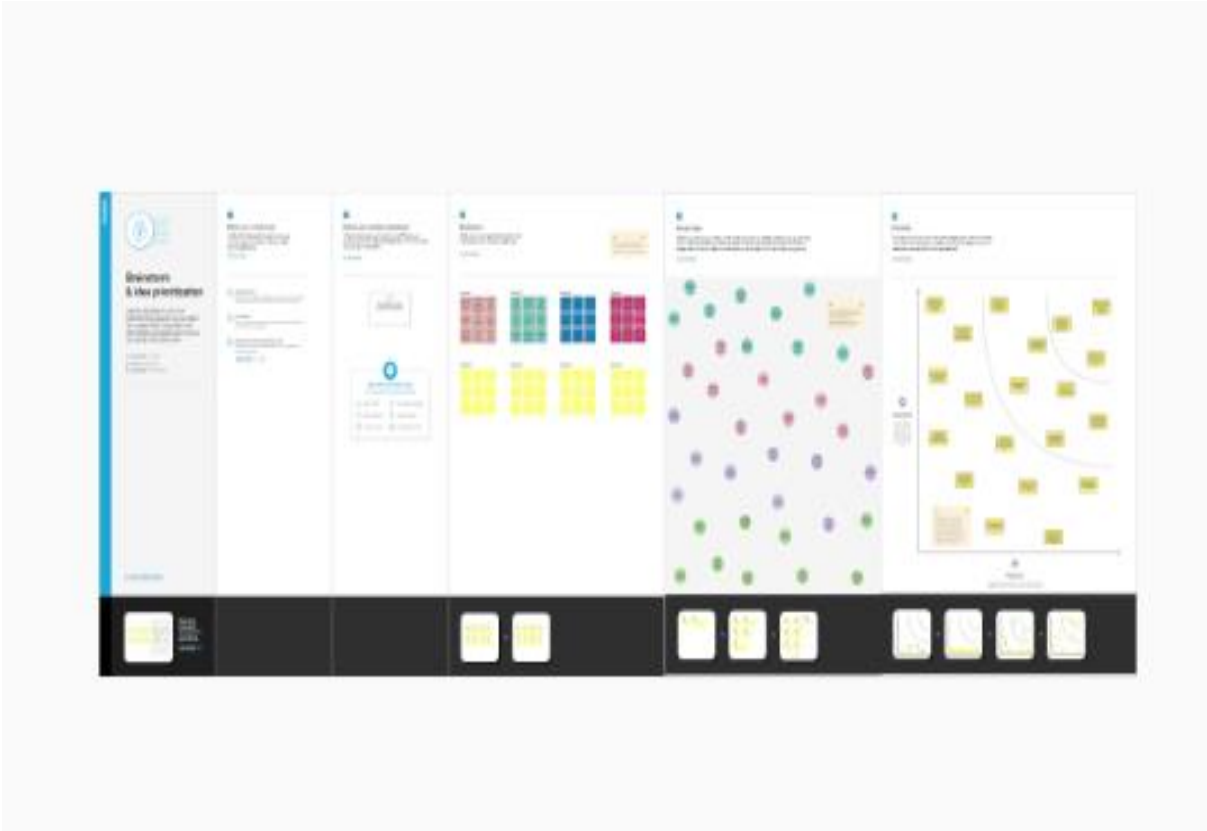
Social Or Business Impact

On a social level, early detection and prediction of CKD can lead to improved patient outcomes and quality of life. By identifying individuals at risk for CKD, healthcare providers can intervene early and slow the progression of the disease through lifestyle changes, medication management, and other treatments. This can help prevent the need for dialysis or kidney transplantation, which can be costly and life-altering for patients. Additionally, early prediction can also help reduce the overall burden of CKD on the healthcare system by reducing the number of hospitalizations and emergency room visits.

Empathy map:



Brainstorm map:



Results:

Cohort characteristics

The dataset contained a total of 748 subjects with the follow-up duration of 6.3 ± 2.3 years. The baseline characteristics are summarized in Table 1. Most patients were in stage 2 (24.5%) or 3 (47.1%) CKD at baseline. ESKD was observed in 70 patients (9.4%), all of whom subsequently received KRT, including hemodialysis in 49 patients, peritoneal dialysis in 17 and kidney transplantation in 4.

Model performance

Details of the five imputed sets are provided in the supplemental parameter. There was no significant difference between the imputed sets and the original dataset in each variable where missing data were replaced by imputed values. The hyper parameter settings for each classifier are displayed in Table 2. The best overall performance, as measured by the AUC score, was achieved by the random forest algorithm (0.81, see Table 3). Nonetheless, this score and its 95% confidence interval had overlap with those of the other three models, including the logistic regression, naïve Bayes, and the KFRE (Fig. 1). Interestingly, the KFRE model that was based on 3 simple variables, demonstrated not only a comparable AUC score but also the highest accuracy, specificity, and precision. At the default threshold, however, the KFRE was one of the least sensitive models (47%).

Advantages of CKD prediction:

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.

Disease Prediction using Machine Learning is the system that is **used to predict the diseases from the symptoms which are given by the patients or any user**. The system processes the symptoms provided by the user as input and gives the output as the probability of the disease.

Applications of CKD prediction:

We validate the results generated by the algorithm “bayesian classifier” and “KNN algorithm”.

TABLE I: Attributes and Description of CKD dataset.

Attribute	Description
age	Age
bp	Blood Pressure
sg	Specific Gravity
al	Albumin

rbc	Red Blood Cells
pc	Pus Cell
pcc	Pus Cell Clumps
ba	Bacteria
bgr	Blood Glucose Random
bu	Blood Urea
sc	Serum Creatinine
sod	Sodium
pot	Potassium
hemo	Hemoglobin
pcv	Packed Cell Volume
wc	White Blood Cell Count
rc	Red Blood Cell Count
htn	Hypertension
dm	Diabetes Mellitus
cad	Coronary Artery Disease
appet	Appetite
pe	Pedal Edema

ane	Anemia
class	Class
su	Sugar

TABLEII: Value range and Measurements of Attributes

Attribute	Measurement and Value Range
age	Age in Years
bp	bp in mm/Hg
sg	1.005,1.010,1.015,1.020,1.025
al	0,1,2,3,4,5
rbc	normal, abnormal
pc	normal, abnormal
pcc	Present, notpresent
ba	Present, notpresent
bgr	mgs/dl
bu	mgs/dl
sc	mgs/dl

sod	mEq/L
pot	mEq/L
hemo	gms
pcv	numerical values
wc	cells/cumm
rc	millions/cmm
htn	yes, no
dm	yes, no
cad	yes, no
appet	good, poor
pe	yes, no
ane	yes, no
class	ckd, notckd
su	0,1,2,3,4,5

Chronic Kidney Disease (CKD) data set for analyzing experiment is obtained from UCI machine learning repository and clinical data set, Among 700 data sets 420 are diagnosed as having CDK and 280 as NCKD(no CKD),data set includes 25 attributes and 2 outcomes i.e., 'CKD' and ' NCKD'.

TABLE III: Confusion matrix terminology

--	Class/Positive 1	Class/Negative0
Class1/Positive(1)	True Positive	False Positive
Class2/Negative(0)	False Negative	True Negative

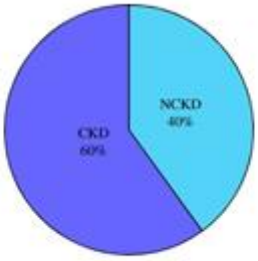


Fig. 2: Percentage of data belongs to CKD and NCKDclassification in a dataset

Milestone – 2:

Data collection and preparation:

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Collect The Dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset. We should download the dataset from the kaggle.
As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Importing the libraries

```
#importing required lib
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')


#checking for available styles

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']

#Applying styles to notebook

plt.style.use('fivethirtyeight')
```

Reading the datasets:

```
#Reading csv data

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

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

5 rows × 26 columns

Data preparation:

As we have understood how the data is, let's pre-process the collected data.
The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Rename the columns
- Handling missing values
- Handling categorical data
- Handling Numerical data

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Rename the columns

```
data.columns #return all the column names
```

```
Index(['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')
```

```
data.columns=['age','blood_pressure','specific_gravity','albumin',
              'sugar','red_blood_cells','pus_cell','pus_cell_clumps','bacteria',
              'blood glucose random','blood_urea','serum_creatinine','sodium','potassium',
              'hemoglobin','packed_cell_volume','white_blood_cell_count','red_blood_cell_count',
              'hypertension','diabetesmellitus','coronary_artery_disease','appetite',
              'pedal_edema','anemia','class'] # manually giving the name  of the columns
data.columns
```

```
Index(['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar',
       'red_blood_cells', 'pus_cell', 'pus_cell_clumps', 'bacteria',
       'blood glucose random', 'blood_urea', 'serum_creatinine', 'sodium',
       'potassium', '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() #info will give you a summary of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   age                     391 non-null    float64
 1   blood_pressure          388 non-null    float64
 2   specific_gravity        353 non-null    float64
 3   albumin                 354 non-null    float64
 4   sugar                   351 non-null    float64
 5   red_blood_cells         248 non-null    object
 6   pus_cell                335 non-null    object
 7   pus_cell_clumps         396 non-null    object
 8   bacteria                396 non-null    object
 9   blood glucose random    356 non-null    float64
 10  blood_urea              381 non-null    float64
 11  serum_creatinine        383 non-null    float64
 12  sodium                  313 non-null    float64
 13  potassium               312 non-null    float64
 14  hemoglobin              348 non-null    float64
 15  packed_cell_volume      330 non-null    object
 16  white_blood_cell_count  295 non-null    object
 17  red_blood_cell_count    270 non-null    object
 18  hypertension            398 non-null    object
 19  diabetesmellitus        398 non-null    object
 20  coronary_artery_disease 398 non-null    object
 21  appetite                399 non-null    object
 22  pedal_edema             399 non-null    object
 23  anemia                  399 non-null    object
 24  class                   400 non-null    object
dtypes: float64(11), object(14)
memory usage: 78.2+ KB
```

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

```
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
```

```
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['packed_cell_volume'].fillna(data['packed_cell_volume'].mean(),inplace=True)
data['potassium'].fillna(data['potassium'].mean(),inplace=True)
data['red_blood_cell_count'].fillna(data['red_blood_cell_count'].mean(),inplace=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'].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['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],inplace=True)
data['pedal_edema'].fillna(data['pedal_edema'].mode()[0],inplace=True)
data[ 'specific_gravity'].fillna(data[ 'specific_gravity'].mode()[0],inplace=True)
```

Handling categorical data

```
catcols=set(data.dtypes[data.dtypes=='0'].index.values) # only fetch the object type columns
print(catcols)
```

```
{'hypertension', 'packed_cell_volume', 'class', 'coronary_artery_disease', 'anemia', 'red_blood_cell_count', 'red_blood_cells', 'bacteri
a', 'pedal_edema', 'appetite', 'pus_cell', 'diabetesmellitus', 'pus_cell_clumps', 'white_blood_cell_count'}
```

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

```
Columns : hypertension
Counter({'no': 251, 'yes': 147, nan: 2})
************************************************************************************************************************

Columns : packed_cell_volume
Counter({nan: 70, '52': 21, '41': 21, '44': 19, '48': 19, '40': 16, '43': 14, '45': 13, '42': 13, '32': 12, '36': 12, '33': 12, '28': 12,
'50': 12, '37': 11, '34': 11, '35': 9, '29': 9, '30': 9, '46': 9, '31': 8, '39': 7, '24': 7, '26': 6, '38': 5, '47': 4, '49': 4, '53': 4,
'51': 4, '54': 4, '27': 3, '22': 3, '25': 3, '23': 2, '19': 2, '16': 1, '\t?': 1, '14': 1, '18': 1, '17': 1, '15': 1, '21': 1, '20': 1,
'\t43': 1, '9': 1})
************************************************************************************************************************

Columns : class
Counter({'ckd': 250, 'notckd': 150})
************************************************************************************************************************

Columns : coronary_artery_disease
Counter({'no': 362, 'yes': 34, '\tno': 2, nan: 2})
************************************************************************************************************************

Columns : anemia
Counter({'no': 339, 'yes': 60, nan: 1})
************************************************************************************************************************

Columns : red_blood_cell_count
Counter({nan: 130, '5.2': 18, '4.5': 16, '4.9': 14, '4.7': 11, '3.9': 10, '4.8': 10, '4.6': 9, '3.4': 9, '3.7': 8, '5.0': 8, '6.1': 8, '5.
5': 8, '5.9': 8, '3.8': 7, '5.4': 7, '5.8': 7, '5.3': 7, '4.3': 6, '4.2': 6, '5.6': 6, '4.4': 5, '3.2': 5, '4.1': 5, '6.2': 5, '5.1': 5,
'6.4': 5, '5.7': 5, '6.5': 5, '3.6': 4, '6.0': 4, '6.3': 4, '4.0': 3, '4': 3, '3.5': 3, '3.3': 3, '5': 2, '2.6': 2, '2.8': 2, '2.5': 2,
'3.1': 2, '2.1': 2, '2.9': 2, '2.7': 2, '3.0': 2, '2.3': 1, '8.0': 1, '3': 1, '2.4': 1, '\t?': 1})
************************************************************************************************************************
```

```
Columns : red_blood_cells
Counter({'normal': 201, nan: 152, 'abnormal': 47})
**********************************************************************************************************************

Columns : bacteria
Counter({'notpresent': 374, 'present': 22, nan: 4})
**********************************************************************************************************************

Columns : pedal_edema
Counter({'no': 323, 'yes': 76, nan: 1})
**********************************************************************************************************************

Columns : appetite
Counter({'good': 317, 'poor': 82, nan: 1})
**********************************************************************************************************************

Columns : pus_cell
Counter({'normal': 259, 'abnormal': 76, nan: 65})
**********************************************************************************************************************

Columns : diabetesmellitus
Counter({'no': 258, 'yes': 134, '\tno': 3, '\tyes': 2, nan: 2, ' yes': 1})
**********************************************************************************************************************

Columns : pus_cell_clumps
Counter({'notpresent': 354, 'present': 42, nan: 4})
**********************************************************************************************************************

Columns : white_blood_cell_count
Counter({nan: 105, '9800': 11, '6700': 10, '9600': 9, '9200': 9, '7200': 9, '6900': 8, '11000': 8, '5800': 8, '7800': 7, '9100': 7, '940
0': 7, '7000': 7, '4300': 6, '6300': 6, '10700': 6, '10500': 6, '7500': 5, '8300': 5, '7900': 5, '8600': 5, '5600': 5, '10200': 5, '5000':
5, '8100': 5, '9500': 5, '6000': 4, '6200': 4, '10300': 4, '7700': 4, '5500': 4, '10400': 4, '6800': 4, '6500': 4, '4700': 4, '7300': 3,
'4500': 3, '8400': 3, '6400': 3, '4200': 3, '7400': 3, '8000': 3, '5400': 3, '3800': 2, '11400': 2, '5300': 2, '8500': 2, '14600': 2, '710
0': 2, '13200': 2, '9000': 2, '8200': 2, '15200': 2, '12400': 2, '12800': 2, '8800': 2, '5700': 2, '9300': 2, '6600': 2, '12100': 1, '1220
0': 1, '18900': 1, '21600': 1, '11300': 1, '\t6200': 1, '11800': 1, '12500': 1, '11900': 1, '12700': 1, '13600': 1, '14900': 1, '16300':
1, '\t8400': 1, '10900': 1, '2200': 1, '11200': 1, '19100': 1, '\t?': 1, '12300': 1, '16700': 1, '2600': 1, '26400': 1, '4900': 1, '1200
0': 1, '15700': 1, '4100': 1, '11500': 1, '10800': 1, '9900': 1, '5200': 1, '5900': 1, '9700': 1, '5100': 1})
**********************************************************************************************************************
```

```
1 catcols.remove('red_blood_cell_count') # remove is used for removing a particular column
2 catcols.remove('packed_cell_volume')
3 catcols.remove('white_blood_cell_count')
4 print(catcols)
```

```
{'hypertension', 'class', 'coronary_artery_disease', 'anemia', 'red_blood_cells', 'bacteria', 'pedal_edema', 'appetite', 'pus_cell', 'diab
etesmellitus', 'pus_cell_clumps'}
```

Label Encoding For Categorical Columns

Typically, any structured dataset includes multiple columns with combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithm too. We need to convert each text category to numbers in order for the machine to process those using mathematical equations.

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

Labeling Encoding of Categorical Column

```
#'specific_gravity','albumin', 'sugar'(as these columns are  numerical it is removed)
catcols=['anemia','pedal_edema','appetite','bacteria','class','coronary_artery_disease','diabetesmellit
 'hypertension','pus_cell','pus_cell_clumps','red_blood_cells'] #only considered the text class columns
```

```
from sklearn.preprocessing import LabelEncoder #imorting the LabelEncoding from sklearn
for i in catcols: #looping through all the categorical columns
    print("LABEL ENCODING OF:",i)
    LEi = LabelEncoder() # creating an object of LabelEncoder
    print(c(data[i])) #getting the classes values before transformation
    data[i] = LEi.fit_transform(data[i])# trannsforming our text classes to numerical values
    print(c(data[i])) #getting the classes values after transformation
    print("*"*100)
```

```
LABEL ENCODING OF: anemia
Counter({'no': 340, 'yes': 60})
Counter({0: 340, 1: 60})
**********************************************************************************************
LABEL ENCODING OF: pedal_edema
Counter({'no': 324, 'yes': 76})
Counter({0: 324, 1: 76})
**********************************************************************************************
LABEL ENCODING OF: appetite
Counter({'good': 318, 'poor': 82})
Counter({0: 318, 1: 82})
**********************************************************************************************
LABEL ENCODING OF: bacteria
Counter({'notpresent': 378, 'present': 22})
Counter({0: 378, 1: 22})
**********************************************************************************************
LABEL ENCODING OF: class
Counter({'ckd': 250, 'notckd': 150})
Counter({0: 250, 1: 150})
**********************************************************************************************
LABEL ENCODING OF: coronary_artery_disease
Counter({'no': 366, 'yes': 34})
Counter({0: 366, 1: 34})
**********************************************************************************************
LABEL ENCODING OF: diabetesmellitus
Counter({'no': 263, 'yes': 137})
Counter({0: 263, 1: 137})
**********************************************************************************************
LABEL ENCODING OF: hypertension
Counter({'no': 253, 'yes': 147})
Counter({0: 253, 1: 147})
**********************************************************************************************
LABEL ENCODING OF: pus_cell
Counter({'normal': 324, 'abnormal': 76})
Counter({1: 324, 0: 76})
**********************************************************************************************
LABEL ENCODING OF: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
Counter({0: 358, 1: 42})
**********************************************************************************************
LABEL ENCODING OF: red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
```

Handling Numerical columns

```
contcols=set(data.dtypes[data.dtypes!='0'].index.values)# only fetech the float and int type columns
#contcols=pd.DataFrame(data,columns=contcols)
print(contcols)
```

```
{'blood_urea', 'serum_creatinine', 'albumin', 'blood_pressure', 'blood glucose random', 'sugar', 'sodium', 'hemoglobin', 'specific_gravit
y', 'age', 'potassium'}
```

```
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') # using add we can add the column
contcols.add('packed_cell_volume')
contcols.add('white_blood_cell_count')
print(contcols)
```

```
{'blood_urea', 'serum_creatinine', 'packed_cell_volume', 'blood_pressure', 'blood glucose random', 'sodium', 'hemoglobin', 'red_blood_cell
_count', 'age', 'potassium', 'white_blood_cell_count'}
```

```
catcols.add('specific_gravity')
catcols.add('albumin')
catcols.add('sugar')
print(catcols)
```

```
{'hypertension', 'class', 'albumin', 'coronary_artery_disease', 'anemia', 'sugar', 'red_blood_cells', 'specific_gravity', 'bacteria', 'ped
al_edema', 'appetite', 'pus_cell', 'diabetesmellitus', 'pus_cell_clumps'}
```

```
data['coronary_artery_disease'] = data.coronary_artery_disease.replace('\tno','no') # replacing \tno wi
c(data['coronary_artery_disease'])
```

```
Counter({'no': 364, 'yes': 34, nan: 2})
```

```
data['diabetesmellitus'] = data.diabetesmellitus.replace(to_replace={'\tno':'no','\tyes':'yes',' yes':'
c(data['diabetesmellitus'])
```

```
Counter({'yes': 137, 'no': 261, nan: 2})
```

Milestone – 3:

Exploratory Data Analysis:

Descriptive statistical analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
#descriptive analysis
df.describe(include='all')
```

	id	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo
count	400.000000	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000
mean	199.500000	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437
std	115.614301	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587
min	0.000000	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000
25%	99.750000	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000
50%	199.500000	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000
75%	299.250000	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000
max	399.000000	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000

Visual Analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

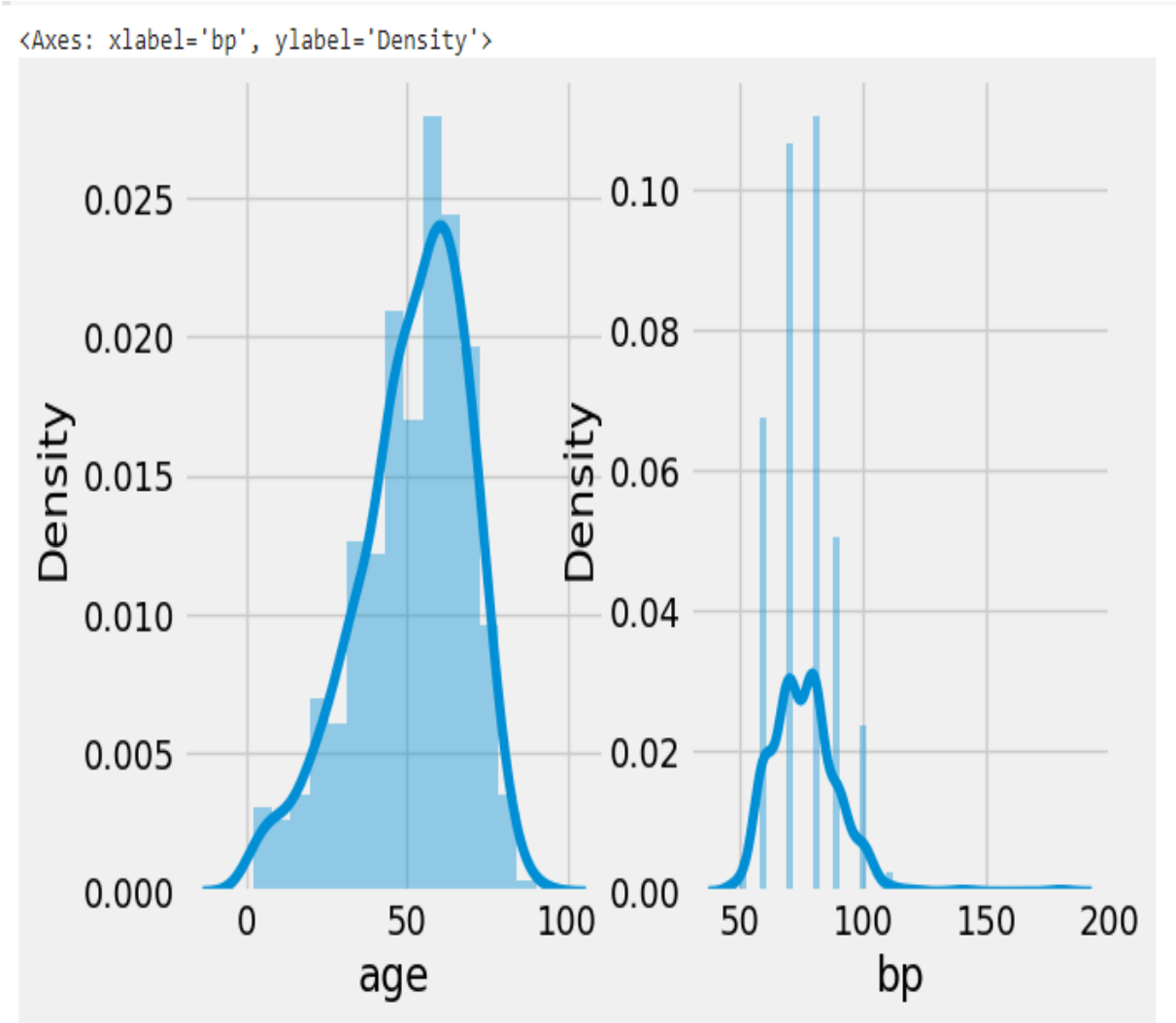
Univariate Analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.

The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature.

```
#Univariate analysis -
 Extracting info from a single column
#Checking data distribution

plt.subplot(121)
sns.distplot(df['age'])
plt.subplot(122)
sns.distplot(df['bp'])
```

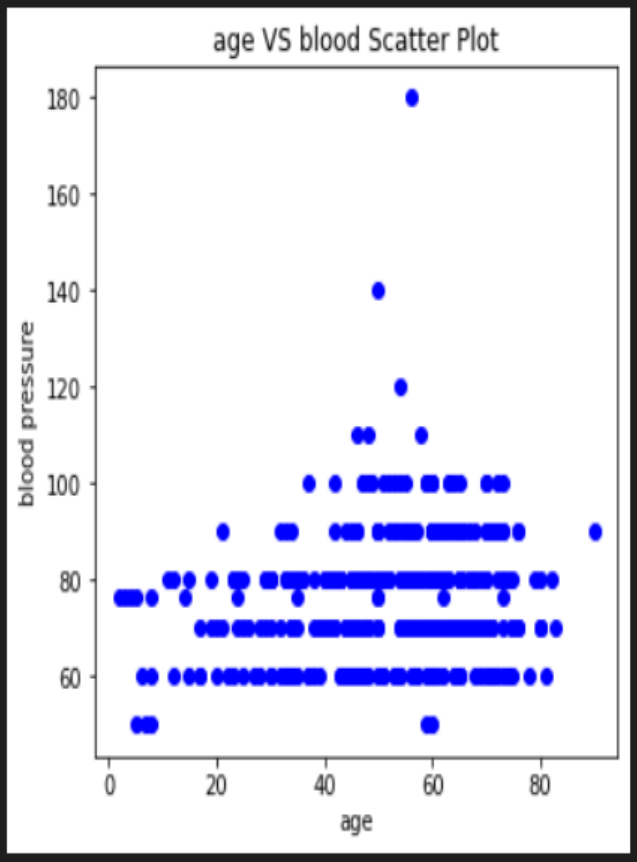


Bivariate analysis

Age vs Blood Pressure

```
import matplotlib.pyplot as plt # import the matplotlib libaray
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 label for 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')
```

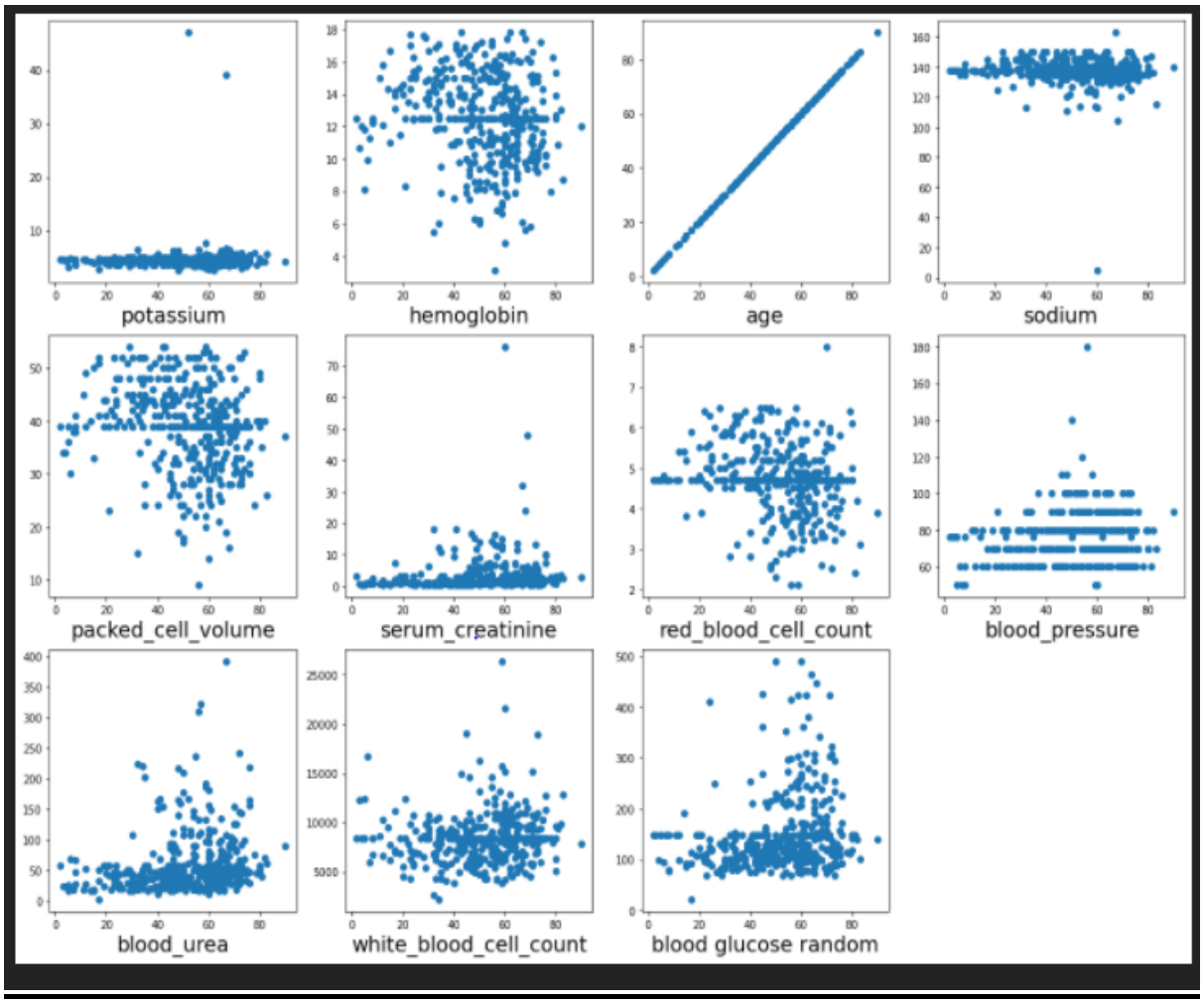


Multivariate analysis

Age vs all continous columns ¶

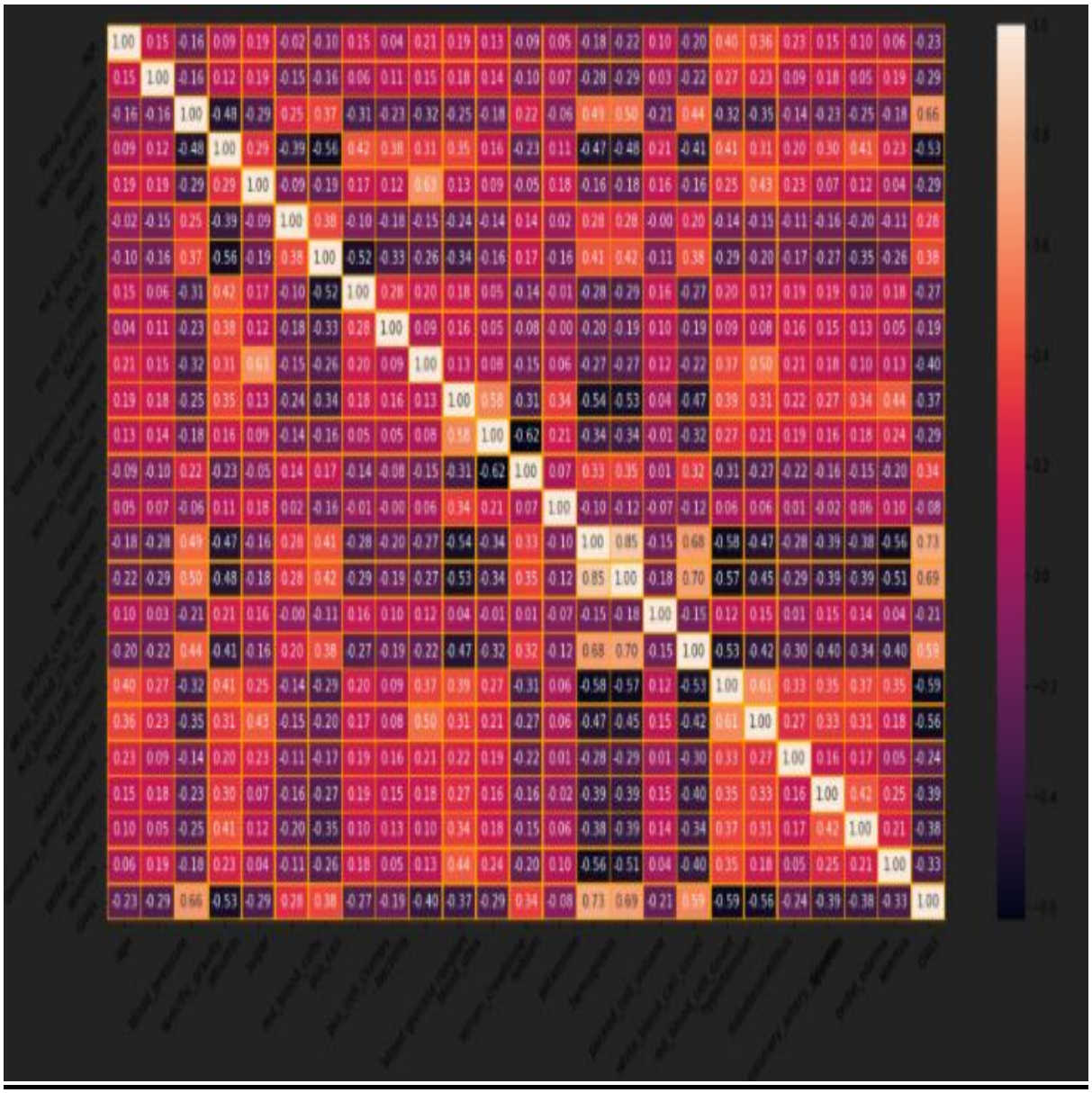
```
plt.figure(figsize=(20,15), facecolor='white')
plotnumber = 1

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)
        #plt.ylabel('Salary',fontsize=20)
    plotnumber+=1
plt.show()
```



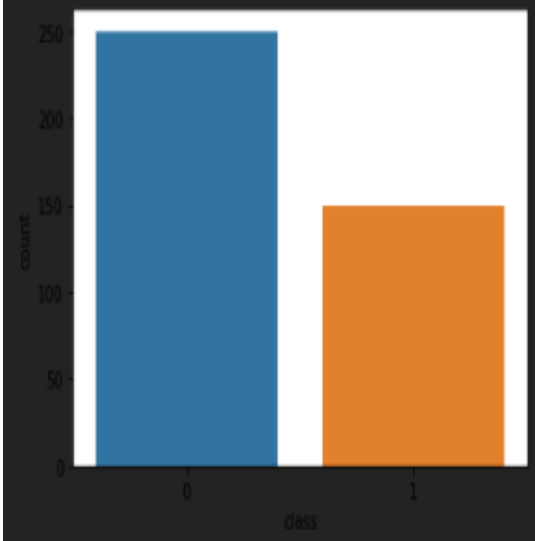
Finding correlation between the independent Columns

```
#HEAT MAP #correlation of parameters
f,ax=plt.subplots(figsize=(18,10))
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(data['class'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20c1d390d30>
```




```
# perfroming feature Scaling op[eration using standard scaller on X part of the dataset because
# there different type of values in the columns
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_bal=sc.fit_transform(x)
```

Creating Independent and Dependent

```
selcols=['red_blood_cells','pus_cell', 'blood glucose random','blood_urea',
            '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)
```

```
(400, 8)
(400, 1)
```

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=2)#train test split
```

Milestone – 4:

Data modelling:

Training the Model In Multiple Algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

ANN Model

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs

```python
# Importing the Keras libraries and packages
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```python
# 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='sigmoid'))
```

```python
# Compiling the ANN model

classification.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```python
# Training the model

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

```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/100
26/26 [==============================] - 0s 6ms/step - loss: 0.1151 - accuracy: 0.9531 - val_loss: 0.2476 - val_accuracy: 0.9062
Epoch 2/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1171 - accuracy: 0.9570 - val_loss: 0.2498 - val_accuracy: 0.9062
Epoch 3/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1146 - accuracy: 0.9531 - val_loss: 0.2317 - val_accuracy: 0.9219
Epoch 4/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1305 - accuracy: 0.9531 - val_loss: 0.2855 - val_accuracy: 0.8906
Epoch 5/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1387 - accuracy: 0.9492 - val_loss: 0.2068 - val_accuracy: 0.9219
Epoch 6/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1230 - accuracy: 0.9492 - val_loss: 0.2576 - val_accuracy: 0.9062
Epoch 7/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1241 - accuracy: 0.9531 - val_loss: 0.2688 - val_accuracy: 0.8906
Epoch 8/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1128 - accuracy: 0.9570 - val_loss: 0.2334 - val_accuracy: 0.9219
Epoch 9/100
26/26 [==============================] - 0s 4ms/step - loss: 0.1180 - accuracy: 0.9531 - val_loss: 0.2435 - val_accuracy: 0.9062
Epoch 10/100
```

```
[==============================] - 0s 4ms/step - loss: 0.1139 - accuracy: 0.9531 - val_loss: 0.2799 - val_accuracy: 0.8906 Ep
...
Epoch 99/100 26/26 [==============================] - 0s 3ms/step - loss: 0.1074 - accuracy: 0.9570 - val_loss: 0.2439 - va
[==============================] - 0s 4ms/step - loss: 0.1062 - accuracy: 0.9570 - val_loss: 0.2572 - val_accuracy: 0.9062

<tensorflow.python.keras.callbacks.History at 0x1fdf3ca7b20>
```

Random Forest Model

A function named random Forest is created and train and test data are passed as the parameters. Inside the function, Random Forest Classifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with. predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
[54]
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=10,criterion='entropy')

[55]
rfc.fit(x_train,y_train)

<ipython-input-255-b87bb2ba9825>:1: DataConversionWarning: A column-vector y wa
(n_samples,), for example using ravel().
  rfc.fit(x_train,y_train)

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

[56]
y_predict = rfc.predict(x_test)

+ Code

[57]
y_predict_train = rfc.predict(x_train)
```

Decision Tree Model

A function named decision Tree is created and train and test data are passed as the parameters. Inside the function, Decision Tree Classifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with. predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
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)
```

```
y_predict= dtc.predict(x_test)
y_predict
```

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

```
y_predict_train = dtc.predict(x_train)
```

Logistic regression

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

```
C:\Users\Saumya\Anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWar
Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

LogisticRegression()
```

Predicting our output with the model which we build

```
from sklearn.metrics import accuracy_score,classification_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)
(y_pred)
```

```
[0]

array([0])
```

```
# DecisionTree classifier

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

print(y_pred)
(y_pred)
```

```
[0]

array([0])
```

```
# Random Forest Classifier
y_pred = rfc.predict([[1,1,121.000000,36.0,0,0,1,0]])

print(y_pred)
(y_pred)
```

```
[0]

array([0])
```

```
classification.save("ckd.h5")
```

```
# Testing the model

y_pred = classification.predict(x_test)
```

```
y_pred
```

```
Output exceeds the size limit. Open the full output data in a text editor
array([[2.07892948e-12],
       [7.16007332e-13],
       [0.00000000e+00],
       [6.47086192e-23],
       [9.99349952e-01],
       [1.47531908e-22],
```

```
y_pred = (y_pred > 0.5)
y_pred
```

```
Output exceeds the size limit. Open the full output data in a te
array([[False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
```

```
def predict_exit(sample_value):

  # Convert list to numpy array
    sample_value = np.array(sample_value)

  # Reshape because sample_value contains 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.')
```

```
Prediction: Low chance of CKD.
```

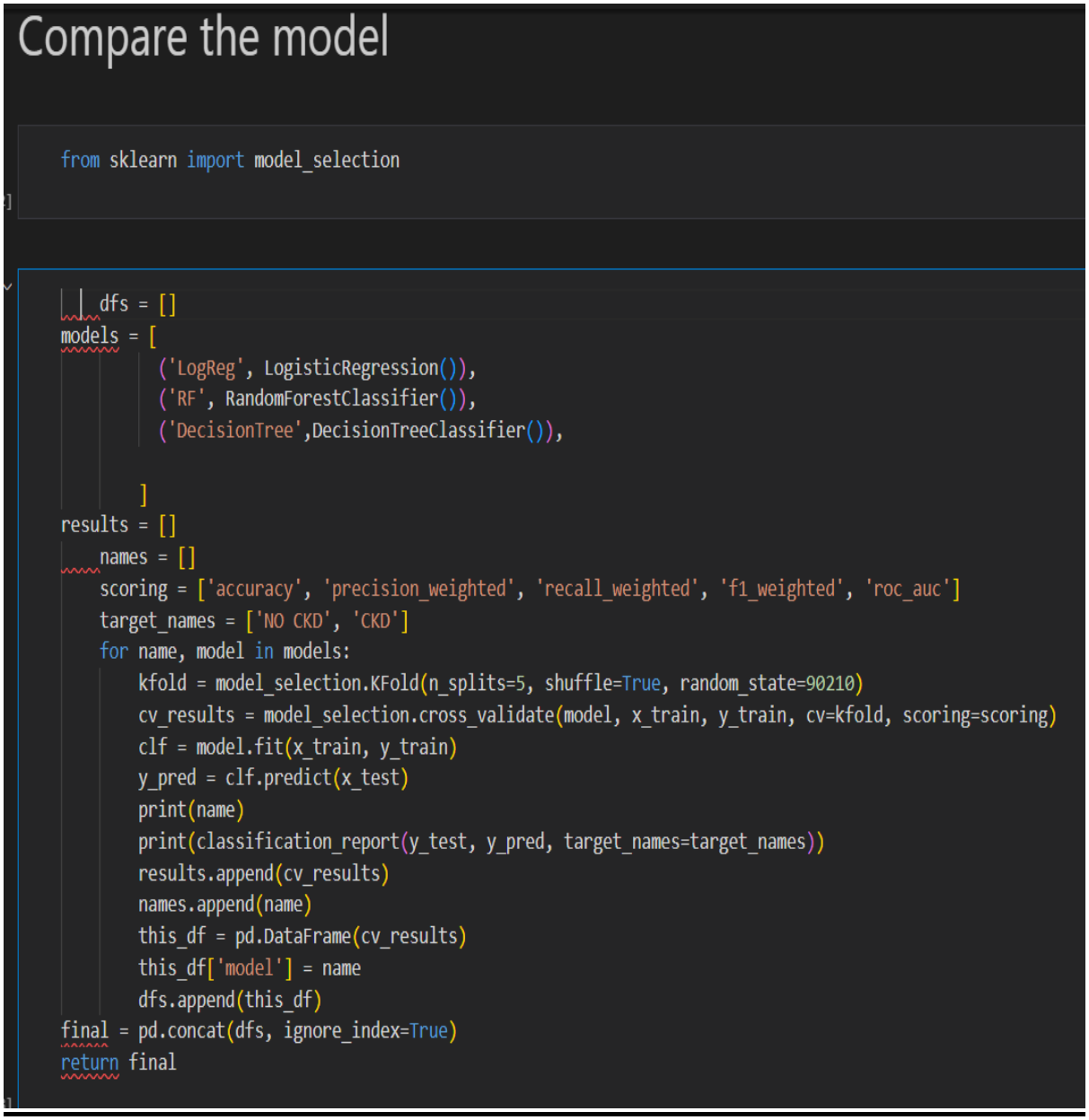
Milestone – 5:

Performance testing and evaluating the results:

In this milestone, we will see the performance testing

Testing Model With Multiple Evaluation Metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.



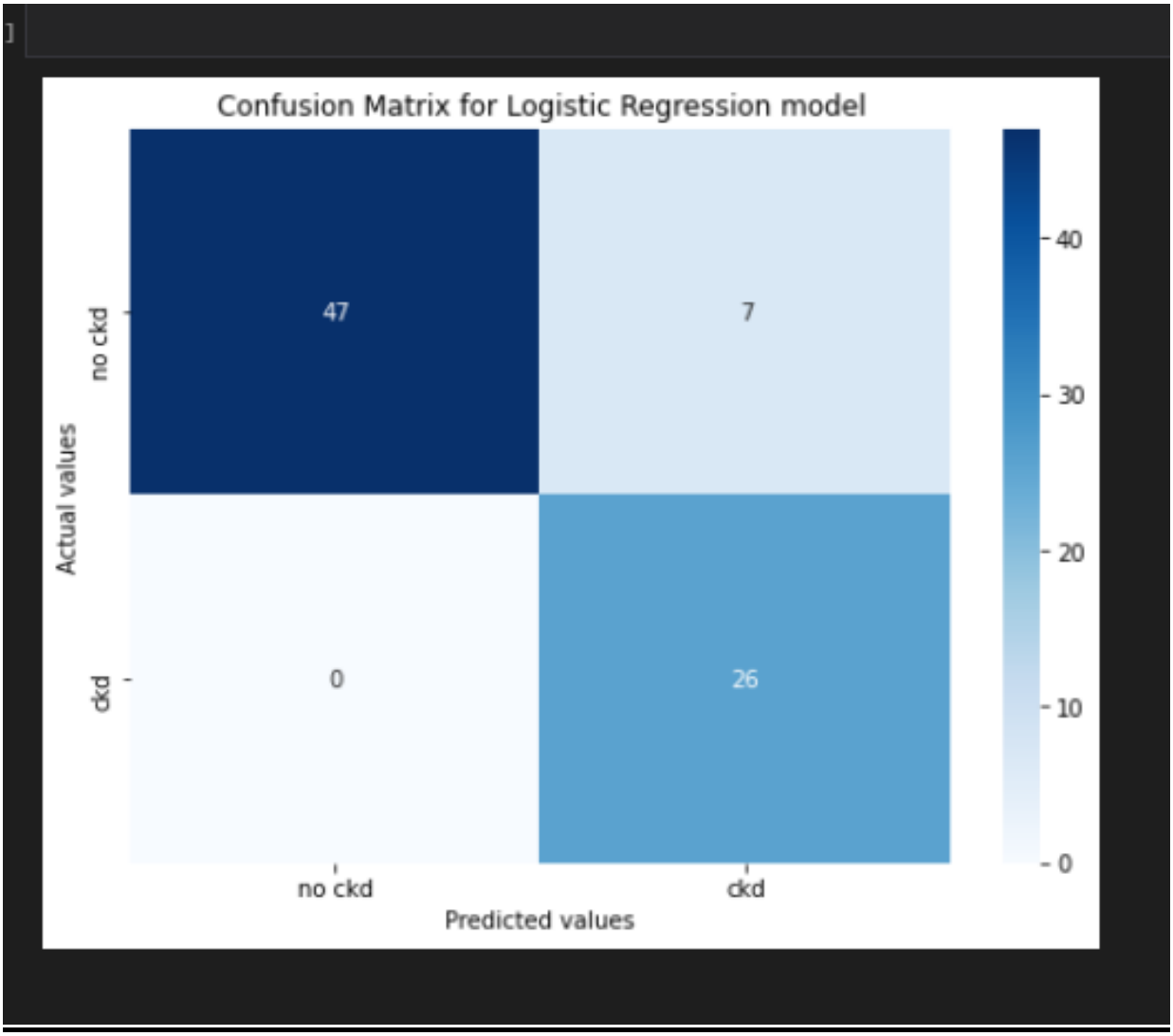
LogReg

	precision	recall	f1-score	support
NO CKD	1.00	0.87	0.93	54
CKD	0.79	1.00	0.88	26
accuracy			0.91	80
macro avg	0.89	0.94	0.91	80
weighted avg	0.93	0.91	0.91	80

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[47,  7],
       [ 0, 26]], dtype=int64)
```

```
# Plotting 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 Regression model')
plt.show()
```



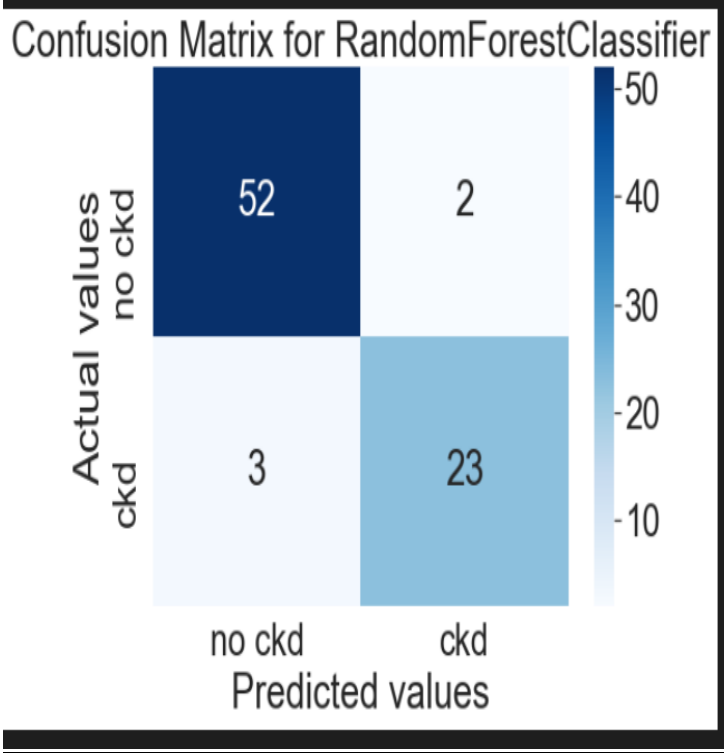
RF

	precision	recall	f1-score	support
NO CKD	0.96	0.96	0.96	54
CKD	0.92	0.92	0.92	26
accuracy			0.95	80
macro avg	0.94	0.94	0.94	80
weighted avg	0.95	0.95	0.95	80

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[52,  2],
       [ 3, 23]], dtype=int64)
```

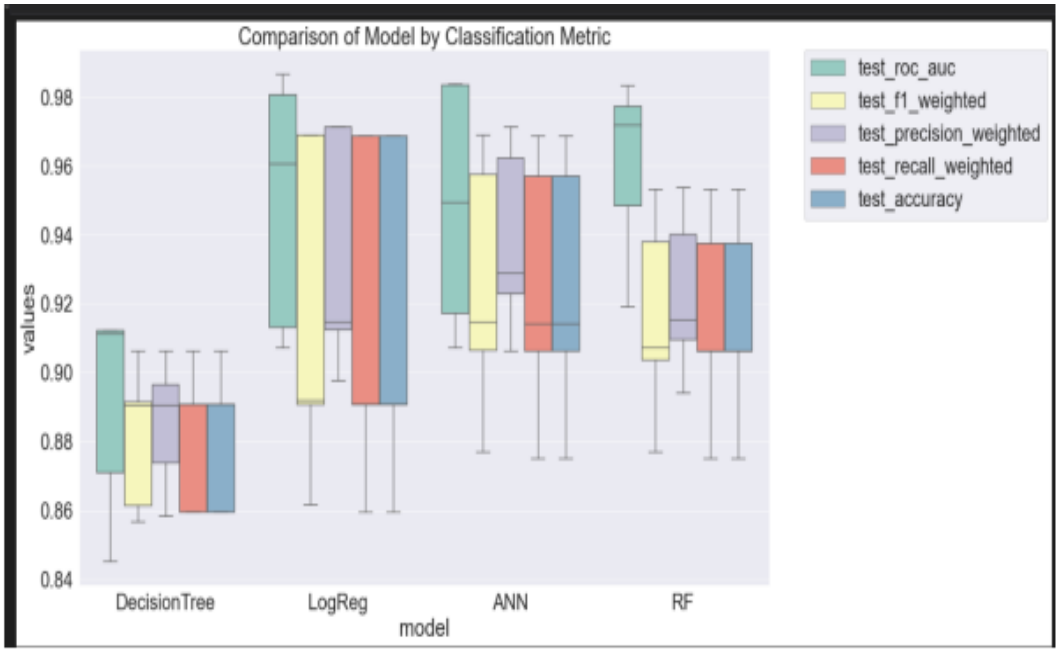
```
# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'], yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for RandomForestClassifier')
plt.show()
```



Evaluating the results

```
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_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
results_long_nofit = results_long.loc[~results_long['metrics'].isin(time_metrics)] # get df without fit data
results_long_nofit = results_long_nofit.sort_values(by='values')
## TIME METRICS
results_long_fit = results_long.loc[results_long['metrics'].isin(time_metrics)] # df with fit data
results_long_fit = results_long_fit.sort_values(by='values')
```



Milestone – 6:

Model deployment:

In this milestone, we will see the model deployment.

Save The Best Model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and to be able to use it in the future.

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

Integrate With Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

Build python code

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

```
app = Flask(__name__) # initializing a flask app
model = pickle.load(open('CKD.pkl', 'rb')) #loading the model
```

```
@app.route('/')# route to display the home page
def home():
    return render_template('home.html') #rendering the home page
```

```
@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')

@app.route('/predict',methods=['POST'])# route to show the predictions in a web UI
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)]

    features_name = ['blood_urea', 'blood glucose random', 'anemia',
       'coronary_artery_disease', 'pus_cell', 'red_blood_cells',
       'diabetesmellitus', 'pedal_edema']

    df = pd.DataFrame(features_value, columns=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)
```

Run the web application

```
(base) D:\SmartBridge\Chronic Kidney Disease>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Source code

Importing necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV,
train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
import scipy.stats as stats
import seaborn as sns
%matplotlib inline
```

Now, let's read our dataset

```
df = pd.read_csv('kidney.csv')
data = df
data.head()
```

Output:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	44.0	7800.0	5.2	yes	yes	no	good	no	no	ckd
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	38.0	6000.0	NaN	no	no	no	good	no	no	ckd
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	31.0	7500.0	NaN	no	yes	no	poor	no	yes	ckd
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	32.0	6700.0	3.9	yes	no	no	poor	yes	yes	ckd
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	35.0	7300.0	4.6	no	no	no	good	no	no	ckd

```
data.shape
```

```
(400, 25)
```

Data Pre-processing

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   age     391 non-null    float64
 1   bp      388 non-null    float64
 2   sg      353 non-null    float64
 3   al      354 non-null    float64
 4   su      351 non-null    float64
 5   rbc     248 non-null    object 
 6   pc      335 non-null    object 
 7   pcc     396 non-null    object 
 8   ba      396 non-null    object 
 9   bgr     356 non-null    float64
 10  bu      381 non-null    float64
 11  sc      383 non-null    float64
 12  sod     313 non-null    float64
 13  pot     312 non-null    float64
 14  hemo    348 non-null    float64
 15  pcv     329 non-null    float64
 16  wbcc    294 non-null    float64
 17  rbcc    269 non-null    float64
 18  htn     398 non-null    object 
 19  dm      398 non-null    object 
 20  cad     398 non-null    object 
 21  appet   399 non-null    object 
 22  pe      399 non-null    object 
 23  ane     399 non-null    object 
 24  class   400 non-null    object 
dtypes: float64(14), object(11)
memory usage: 78.2+ KB
```

Now, let's see our data statistically

```
df.describe()
```

Output:

	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000	329.000000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437	38.884498
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587	8.990105
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000	9.000000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000	32.000000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000	40.000000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000	45.000000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000	54.000000

Now, let's see the total count of null values that every feature holds

```
df.isna().sum()
```

Output:

```
age          9
bp          12
sg          47
al          46
su          49
rbc        152
pc          65
pcc          4
ba           4
bgr         44
bu          19
sc          17
sod         87
pot         88
hemo        52
pcv         71
wbcc       106
rbcc       131
htn          2
dm           2
cad          2
appet        1
pe           1
ane          1
class        0
dtype: int64
```

Correlation matrix & Matrix Visualisation

```
df.corr()
```

Output:

	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wbcc	rbcc
age	1.000000	0.159480	-0.191096	0.122091	0.220866	0.244992	0.196985	0.132531	-0.100046	0.058377	-0.192928	-0.242119	0.118339	-0.268896
bp	0.159480	1.000000	-0.218836	0.160689	0.222576	0.160193	0.188517	0.146222	-0.116422	0.075151	-0.306540	-0.326319	0.029753	-0.261936
sg	-0.191096	-0.218836	1.000000	-0.469760	-0.296234	-0.374710	-0.314295	-0.361473	0.412190	-0.072787	0.602582	0.603560	-0.236215	0.579476
al	0.122091	0.160689	-0.469760	1.000000	0.269305	0.379464	0.453528	0.399198	-0.459896	0.129038	-0.634632	-0.611891	0.231989	-0.566437
su	0.220866	0.222576	-0.296234	0.269305	1.000000	0.717827	0.168583	0.223244	-0.131776	0.219450	-0.224775	-0.239189	0.184893	-0.237448
bgr	0.244992	0.160193	-0.374710	0.379464	0.717827	1.000000	0.143322	0.114875	-0.267848	0.066966	-0.306189	-0.301385	0.150015	-0.281541
bu	0.196985	0.188517	-0.314295	0.453528	0.168583	0.143322	1.000000	0.586368	-0.323054	0.357049	-0.610360	-0.607621	0.050462	-0.579087
sc	0.132531	0.146222	-0.361473	0.399198	0.223244	0.114875	0.586368	1.000000	-0.690158	0.326107	-0.401670	-0.404193	-0.006390	-0.400852
sod	-0.100046	-0.116422	0.412190	-0.459896	-0.131776	-0.267848	-0.323054	-0.690158	1.000000	0.097887	0.365183	0.376914	0.007277	0.344873
pot	0.058377	0.075151	-0.072787	0.129038	0.219450	0.066966	0.357049	0.326107	0.097887	1.000000	-0.133746	-0.163182	-0.105576	-0.158309
hemo	-0.192928	-0.306540	0.602582	-0.634632	-0.224775	-0.306189	-0.610360	-0.401670	0.365183	-0.133746	1.000000	0.895382	-0.169413	0.798880
pcv	-0.242119	-0.326319	0.603560	-0.611891	-0.239189	-0.301385	-0.607621	-0.404193	0.376914	-0.163182	0.895382	1.000000	-0.197022	0.791625
wbcc	0.118339	0.029753	-0.236215	0.231989	0.184893	0.150015	0.050462	-0.006390	0.007277	-0.105576	-0.169413	-0.197022	1.000000	-0.158163
rbcc	-0.268896	-0.261936	0.579476	-0.566437	-0.237448	-0.281541	-0.579087	-0.400852	0.344873	-0.158309	0.798880	0.791625	-0.158163	1.000000

Let's find out how many of each class are:

```
df['class'].value_counts()
```

Output:

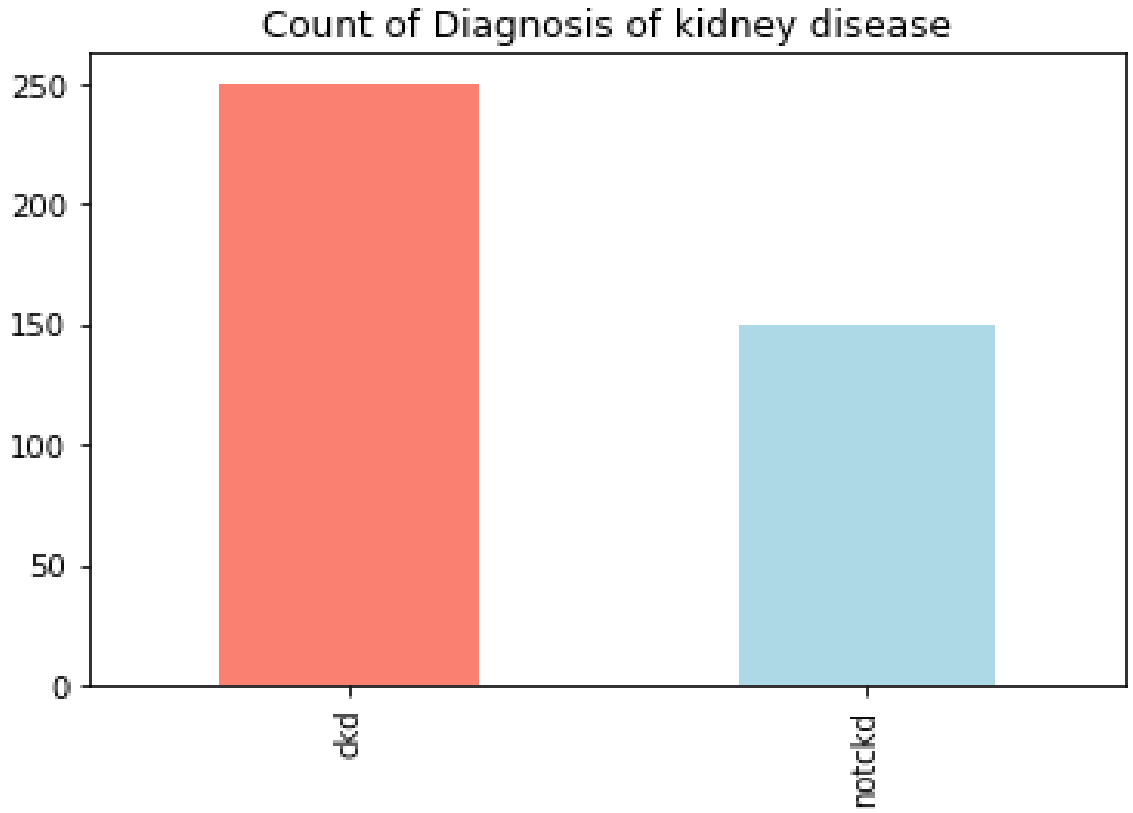
```
ckd       250
notckd    150
Name: class, dtype: int64
```

Representation of Target variable in Percentage

```
countNoDisease = len(df[df['class'] == 0])
countHaveDisease = len(df[df['class'] == 1])
print("Percentage of Patients Haven't Heart Disease:
{:.2f}%".format((countNoDisease /
(len(df['class']))*100)))
print("Percentage of Patients Have Heart Disease:
{:.2f}%".format((countHaveDisease /
(len(df['class']))*100)))
```

Understanding the balancing of the data visually

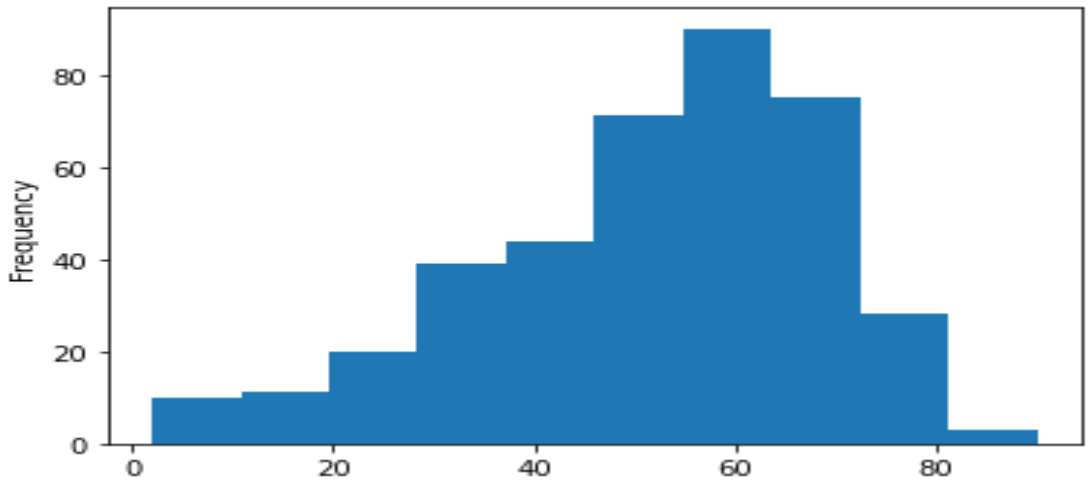
```
df['class'].value_counts().plot(kind='bar',color=['salmon'
,'lightblue'],title="Count of Diagnosis of kidney
disease")
```



Here we will be checking the distribution of the age column

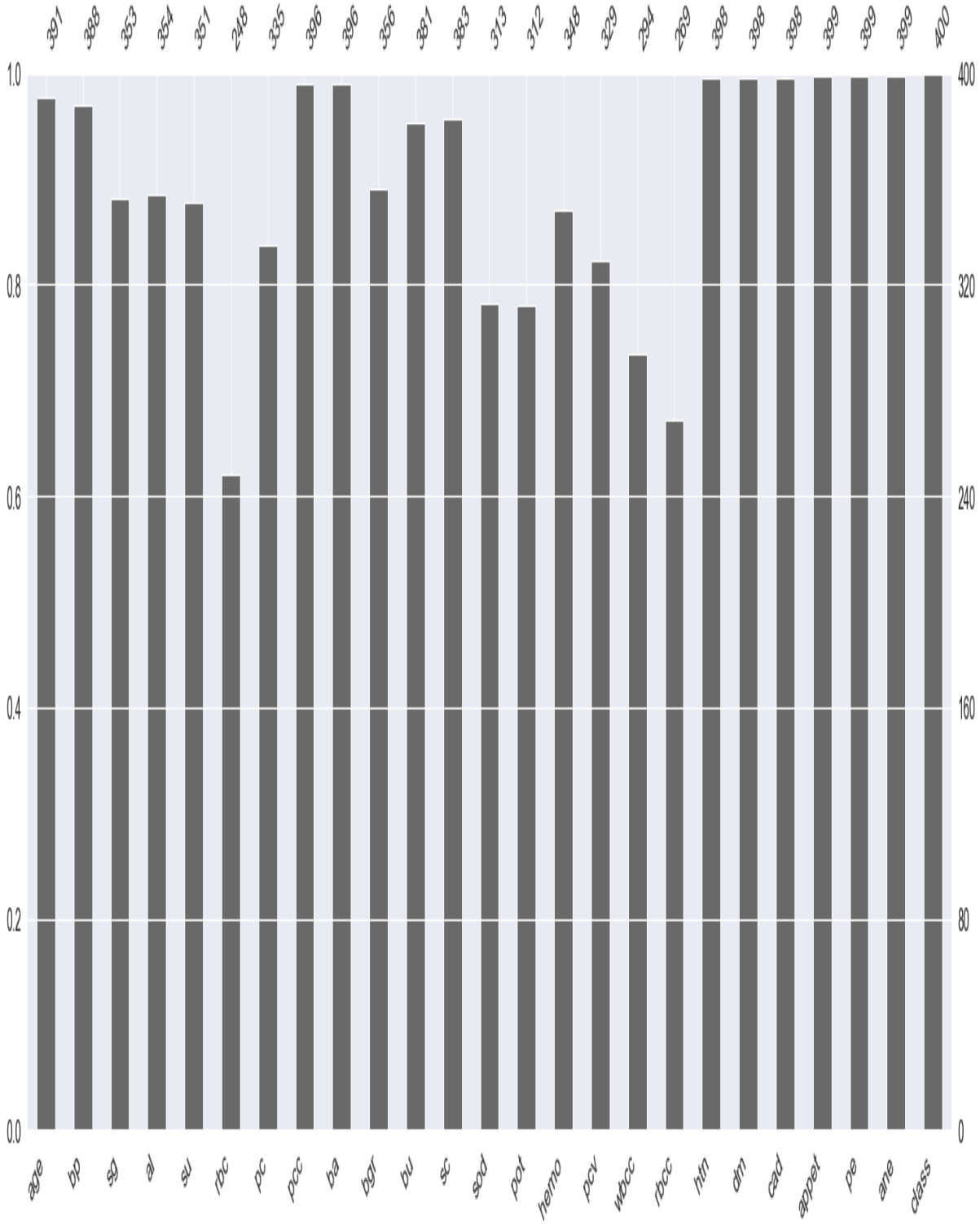
```
df['age'].plot(kind='hist')
```

Output:

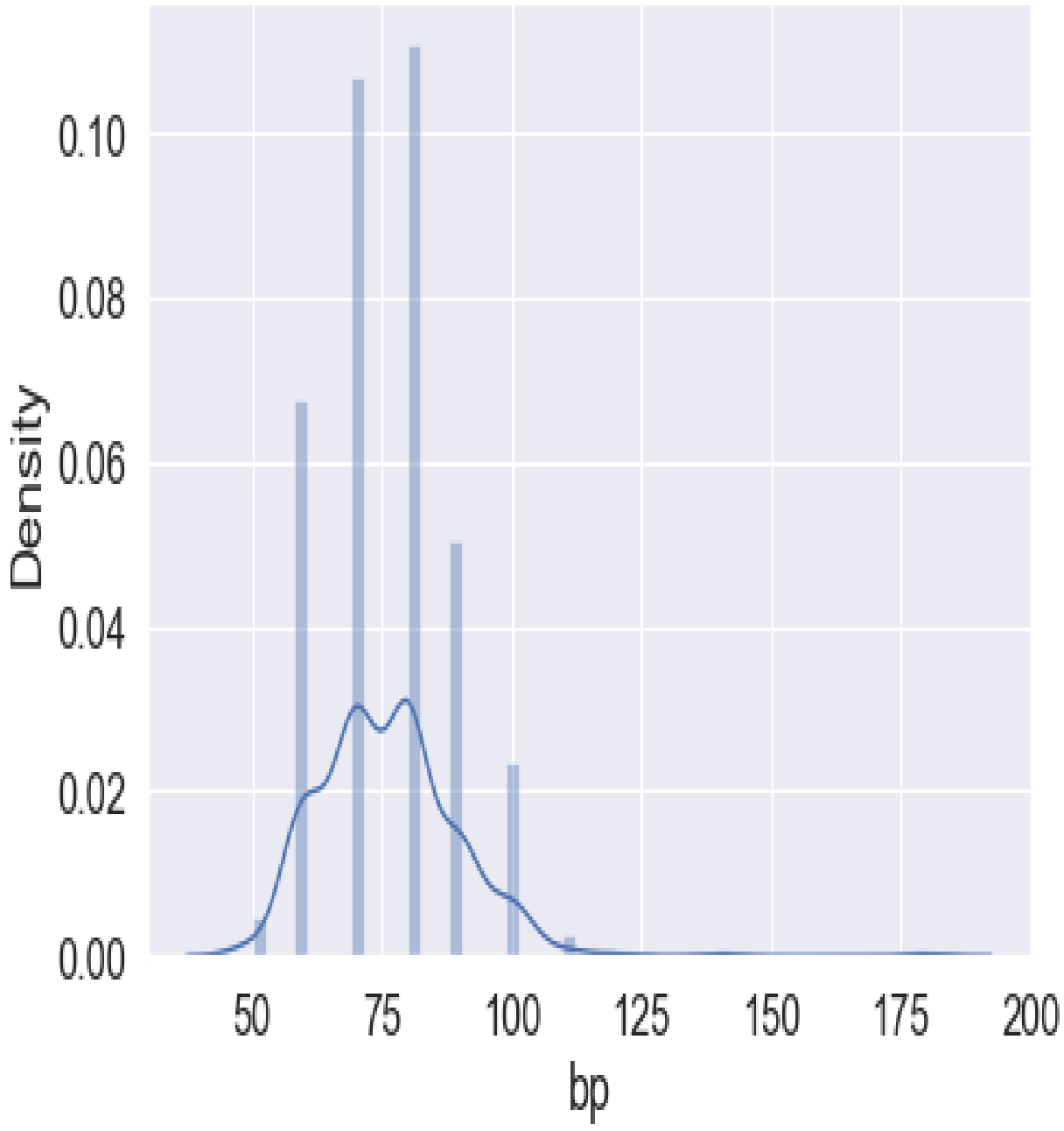


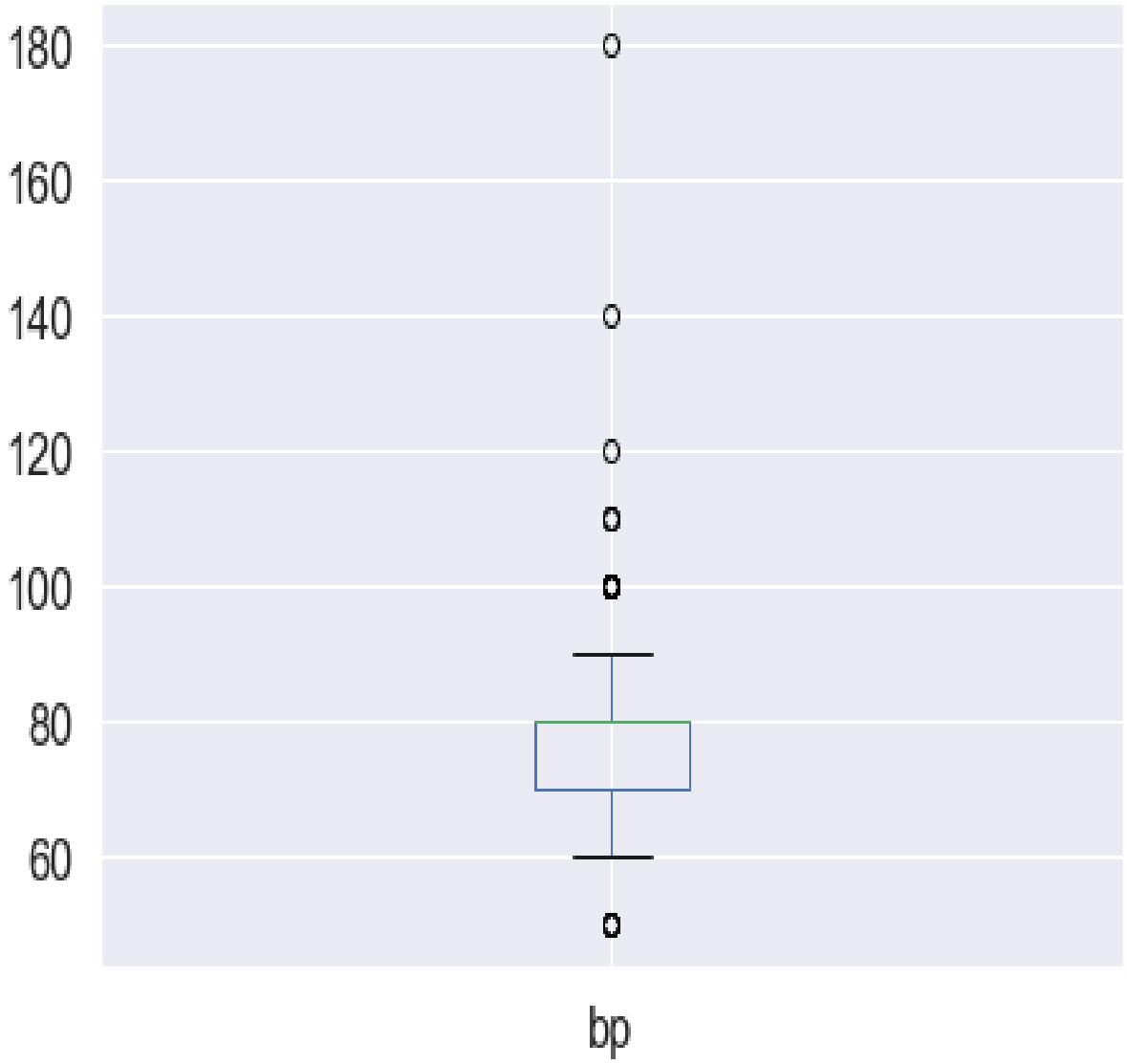
Here we are plotting the graph to see the null values in the dataset.

```
p = msno.bar(data)
```



```
plt.subplot(121), sns.distplot(data['bp'])
plt.subplot(122), data['bp'].plot.box(figsize=(16,5))
plt.show()
```





Now we will convert the categorical values(object) to categorical values(int)

```
data['class'] = data['class'].map({'ckd':1,'notckd':0})
data['htn'] = data['htn'].map({'yes':1,'no':0})
data['dm'] = data['dm'].map({'yes':1,'no':0})
data['cad'] = data['cad'].map({'yes':1,'no':0})
data['appet'] = data['appet'].map({'good':1,'poor':0})
data['ane'] = data['ane'].map({'yes':1,'no':0})
```

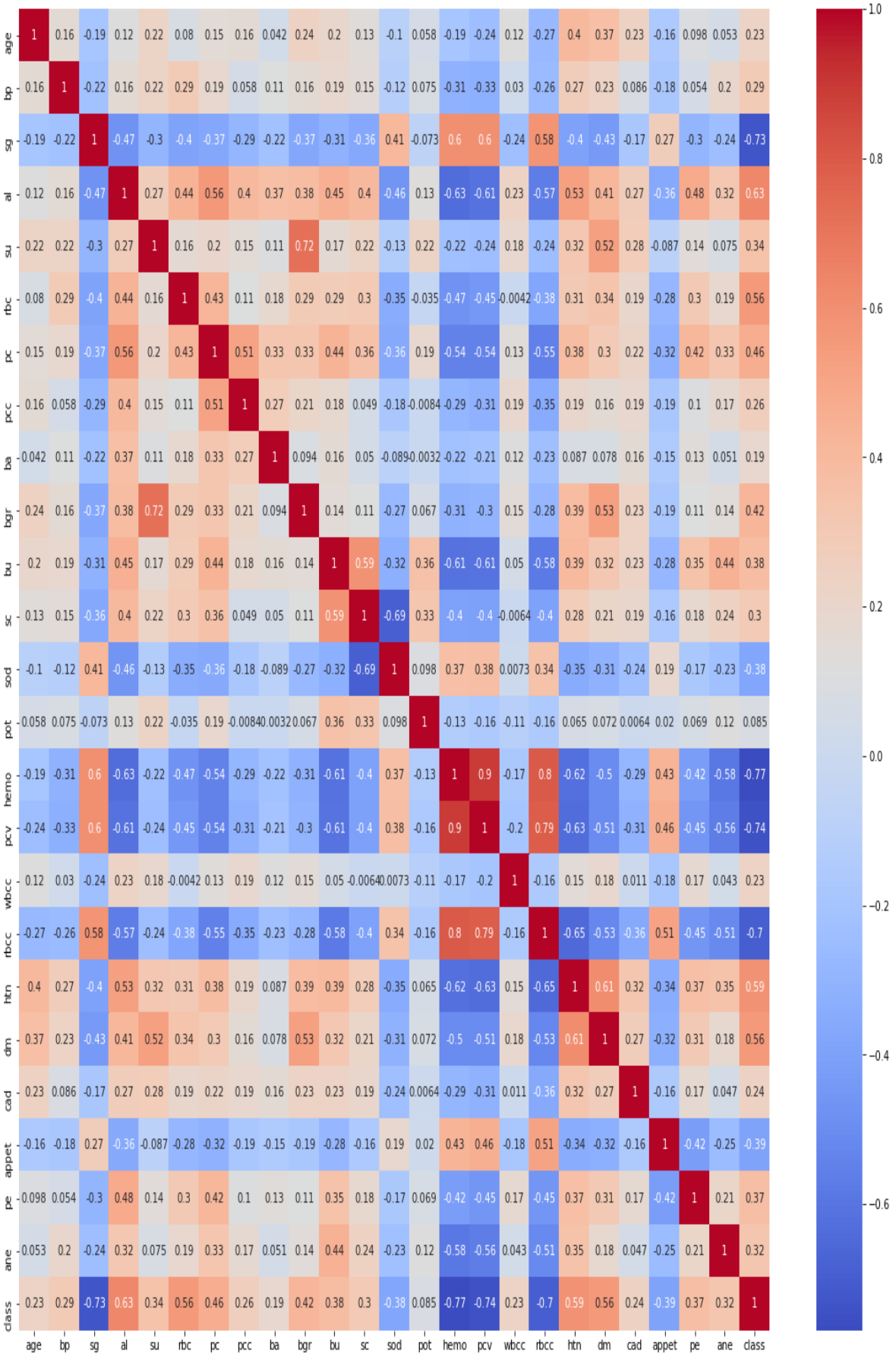
```
data['pe'] = data['pe'].map({'yes':1,'no':0})
data['ba'] = data['ba'].map({'present':1,'notpresent':0})
data['pcc'] =
data['pcc'].map({'present':1,'notpresent':0})
data['pc'] = data['pc'].map({'abnormal':1,'normal':0})
data['rbc'] = data['rbc'].map({'abnormal':1,'normal':0})
data['class'].value_counts()
```

Output:

```
1    250
0    150
Name: class, dtype: int64
```

Finding the Correlation between the plots

```
plt.figure(figsize = (19,19))
sns.heatmap(data.corr(), annot = True, cmap = 'coolwarm')
# looking for strong correlations with "class" row
```


age 1 0.16 -0.19 0.12 0.22 0.08 0.15 0.16 0.042 0.24 0.2 0.13 -0.1 0.058 -0.19 -0.24 0.12 -0.27 0.4 0.37 0.23 -0.16 0.098 0.053 0.23
bp 0.16 1 -0.22 0.16 0.22 0.29 0.19 0.058 0.11 0.16 0.19 0.15 -0.12 0.075 -0.31 -0.33 0.03 -0.26 0.27 0.23 0.086 -0.18 0.054 0.2 0.29
sg -0.19 -0.22 1 -0.47 -0.3 -0.4 -0.37 -0.29 -0.22 -0.37 -0.31 -0.36 0.41 -0.073 0.6 0.6 -0.24 0.58 -0.4 -0.43 -0.17 0.27 -0.3 -0.24 -0.73
al 0.12 0.16 -0.47 1 0.27 0.44 0.56 0.4 0.37 0.38 0.45 0.4 -0.46 0.13 -0.63 -0.61 0.23 -0.57 0.53 0.41 0.27 -0.36 0.48 0.32 0.63
su 0.22 0.22 -0.3 0.27 1 0.16 0.2 0.15 0.11 0.72 0.17 0.22 -0.13 0.22 -0.22 -0.24 0.18 -0.24 0.32 0.52 0.28 -0.087 0.14 0.075 0.34
rbc 0.08 0.29 -0.4 0.44 0.16 1 0.43 0.11 0.18 0.29 0.29 0.3 -0.35 -0.035 -0.47 -0.45 -0.0042 -0.38 0.31 0.34 0.19 -0.28 0.3 0.19 0.56
pc 0.15 0.19 -0.37 0.56 0.2 0.43 1 0.51 0.33 0.33 0.44 0.36 -0.36 0.19 -0.54 -0.54 0.13 -0.55 0.38 0.3 0.22 -0.32 0.42 0.33 0.46
pcc 0.16 0.058 -0.29 0.4 0.15 0.11 0.51 1 0.27 0.21 0.18 0.049 -0.18 -0.0084 -0.29 -0.31 0.19 -0.35 0.19 0.16 0.19 -0.19 0.1 0.17 0.26
ba 0.042 0.11 -0.22 0.37 0.11 0.18 0.33 0.27 1 0.094 0.16 0.05 -0.089 -0.0032 -0.22 -0.21 0.12 -0.23 0.087 0.078 0.16 -0.15 0.13 0.051 0.19
bgr 0.24 0.16 -0.37 0.38 0.72 0.29 0.33 0.21 0.094 1 0.14 0.11 -0.27 0.067 -0.31 -0.3 0.15 -0.28 0.39 0.53 0.23 -0.19 0.11 0.14 0.42
bu 0.2 0.19 -0.31 0.45 0.17 0.29 0.44 0.18 0.16 0.14 1 0.59 -0.32 0.36 -0.61 -0.61 0.05 -0.58 0.39 0.32 0.23 -0.28 0.35 0.44 0.38
sc 0.13 0.15 -0.36 0.4 0.22 0.3 0.36 0.049 0.05 0.11 0.59 1 -0.69 0.33 -0.4 -0.4 -0.0064 -0.4 0.28 0.21 0.19 -0.16 0.18 0.24 0.3
sod -0.1 -0.12 0.41 -0.46 -0.13 -0.35 -0.36 -0.18 -0.089 -0.27 -0.32 -0.69 1 0.098 0.37 0.38 0.0073 0.34 -0.35 -0.31 -0.24 0.19 -0.17 -0.23 -0.38
pot 0.058 0.075 -0.073 0.13 0.22 -0.035 0.19 -0.0084 -0.0032 0.067 0.36 0.33 0.098 1 -0.13 -0.16 -0.11 -0.16 0.065 0.072 0.0064 0.02 0.069 0.12 0.085
hemo -0.19 -0.31 0.6 -0.63 -0.22 -0.47 -0.54 -0.29 -0.22 -0.31 -0.61 -0.4 0.37 -0.13 1 0.9 -0.17 0.8 -0.62 -0.5 -0.29 0.43 -0.42 -0.58 -0.77
pcv -0.24 -0.33 0.6 -0.61 -0.24 -0.45 -0.54 -0.31 -0.21 -0.3 -0.61 -0.4 0.38 -0.16 0.9 1 -0.2 0.79 -0.63 -0.51 -0.31 0.46 -0.45 -0.56 -0.74
wbcc 0.12 0.03 -0.24 0.23 0.18 -0.0042 0.13 0.19 0.12 0.15 0.05 -0.0064 0.0073 -0.11 -0.17 -0.2 1 -0.16 0.15 0.18 0.011 -0.18 0.17 0.043 0.23
rbcc -0.27 -0.26 0.58 -0.57 -0.24 -0.38 -0.55 -0.35 -0.23 -0.28 -0.58 -0.4 0.34 -0.16 0.8 0.79 -0.16 1 -0.65 -0.53 -0.36 0.51 -0.45 -0.51 -0.7
htn 0.4 0.27 -0.4 0.53 0.32 0.31 0.38 0.19 0.087 0.39 0.39 0.28 -0.35 0.065 -0.62 -0.63 0.15 -0.65 1 0.61 0.32 -0.34 0.37 0.35 0.59
dm 0.37 0.23 -0.43 0.41 0.52 0.34 0.3 0.16 0.078 0.53 0.32 0.21 -0.31 0.072 -0.5 -0.51 0.18 -0.53 0.61 1 0.27 -0.32 0.31 0.18 0.56
cad 0.23 0.086 -0.17 0.27 0.28 0.19 0.22 0.19 0.16 0.23 0.23 0.19 -0.24 0.0064 -0.29 -0.31 0.011 -0.36 0.32 0.27 1 -0.16 0.17 0.047 0.24
appet -0.16 -0.18 0.27 -0.36 -0.087 -0.28 -0.32 -0.19 -0.15 -0.19 -0.28 -0.16 0.19 0.02 0.43 0.46 -0.18 0.51 -0.34 -0.32 -0.16 1 -0.42 -0.25 -0.39
pe 0.098 0.054 -0.3 0.48 0.14 0.3 0.42 0.1 0.13 0.11 0.35 0.18 -0.17 0.069 -0.42 -0.45 0.17 -0.45 0.37 0.31 0.17 -0.42 1 0.21 0.37
ane 0.053 0.2 -0.24 0.32 0.075 0.19 0.33 0.17 0.051 0.14 0.44 0.24 -0.23 0.12 -0.58 -0.56 0.043 -0.51 0.35 0.18 0.047 -0.25 0.21 1 0.32
class 0.23 0.29 -0.73 0.63 0.34 0.56 0.46 0.26 0.19 0.42 0.38 0.3 -0.38 0.085 -0.77 -0.74 0.23 -0.7 0.59 0.56 0.24 -0.39 0.37 0.32 1
age bp sg al su rbc pc pcc ba bgr bu sc sod pot hemo pcv wbcc rbcc htn dm cad appet pe ane class
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Exploratory data analysis (EDA)

Let's see the shape of the dataset again after analysis

```
data.shape
```

```
(400, 25)
```

Now let's see the final columns present in our dataset.

```
data.columns
```

Output:

```
Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc',
'ba', 'bgr', 'bu',
       'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc',
'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'class'],
      dtype='object')
```

Now dropping the null values.

```
data.shape[0], data.dropna().shape[0]
```

Results:

Here from the above output, **we can see that there are 158 null values in the dataset.** Now here we are left with two choices that we could either drop all the null values or keep them when we will drop that NA values so we should understand that our dataset is not that large and if we drop those null values then **it would be even smaller in that case if we provide very fewer data to our machine learning model then the performance would be very less also we yet**

don't know that these null values are related to some other features in the dataset.

So for this time I'll keep these values and see how the model will perform in this dataset.

Also when we are working on some healthcare project where we will be predicting whether the person is suffering from that disease or not then one thing we should **keep in my mind is that the model evaluation should have the least false positive errors.**

```
data.dropna(inplace=True)
data.shape
```

Output:

```
(158, 25)
```

Model Building

1. **Logistic regression**

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

X = data.iloc[:,:-1]
y = data['class']

X_train, X_test, y_train, y_test = train_test_split(X,y,
stratify = y, shuffle = True)
```

```
logreg.fit(X_train,y_train)
```

Output:

```
LogisticRegression()
```

Training score

```
logreg.score(X_train,y_train)
```

Output:

```
1.0
```

Testing accuracy

```
logreg.score(X_test,y_test)
```

Output:

```
0.975
Train Accuracy:  1.0
Test Accuracy:  0.975
```

The cell below shows the coefficients for each variable.

(example on reading the coefficients from a Logistic Regression: a one-unit increase in age makes an individual about e^0.14 time as likely to have CKD, while a one-unit increase in blood pressure makes an individual about e^-0.07 times as likely to have CKD.

```
pd.DataFrame(logreg.coef_, columns=X.columns)
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc
0	0.28582	-0.132118	0.002671	0.309953	0.010789	0.019856	0.091069	0.003106	0.006829	0.414045	...	-0.282868	-0.62111	0.001157	-0.141783

1 rows × 24 columns

Confusion Matrix

```
sns.set(font_scale=1.5)
def plot_conf_mat(y_test,y_preds):
    """
    This function will be heloing in plotting the
confusion matrix by using seaborn
    """

    fig,ax=plt.subplots(figsize=(3,3))

ax=sns.heatmap(confusion_matrix(y_test,y_preds),annot=True
,cbar=False)
    plt.xlabel("True Label")
    plt.ylabel("Predicted Label")

log_pred = logreg.predict(X_test)
plot_conf_mat(y_test, log_pred)
```

Output:

```
tn, fp, fn, tp = confusion_matrix(y_test,
test_pred).ravel()

print(f'True Neg: {tn}')
print(f'False Pos: {fp}')
print(f'False Neg: {fn}')
print(f'True Pos: {tp}')
```

Output:

```
True Neg: 28
False Pos: 1
False Neg: 0
True Pos: 11
```

K-Nearest Neighbors Classifier

It is a good practice to first balance the class well before using the KNN, as we know that in the case of unbalanced classes KNN doesn't perform well.

```
df["class"].value_counts()
```

Output:

```
0    115
1     43
Name: class, dtype: int64
```

Now let's CONCATENATE our class variables together.

```
balanced_df = pd.concat([df[df["class"] == 0],
df[df["class"] == 1].sample(n = 115, replace = True)],
axis = 0)
balanced_df.reset_index(drop=True, inplace=True)
balanced_df["class"].value_counts()
```

Output:

```
1    115
0    115
Name: class, dtype: int64
```

Now let's scale down the data

```
ss = StandardScaler()
ss.fit(X_train)
X_train = ss.transform(X_train)
X_test = ss.transform(X_test)
```

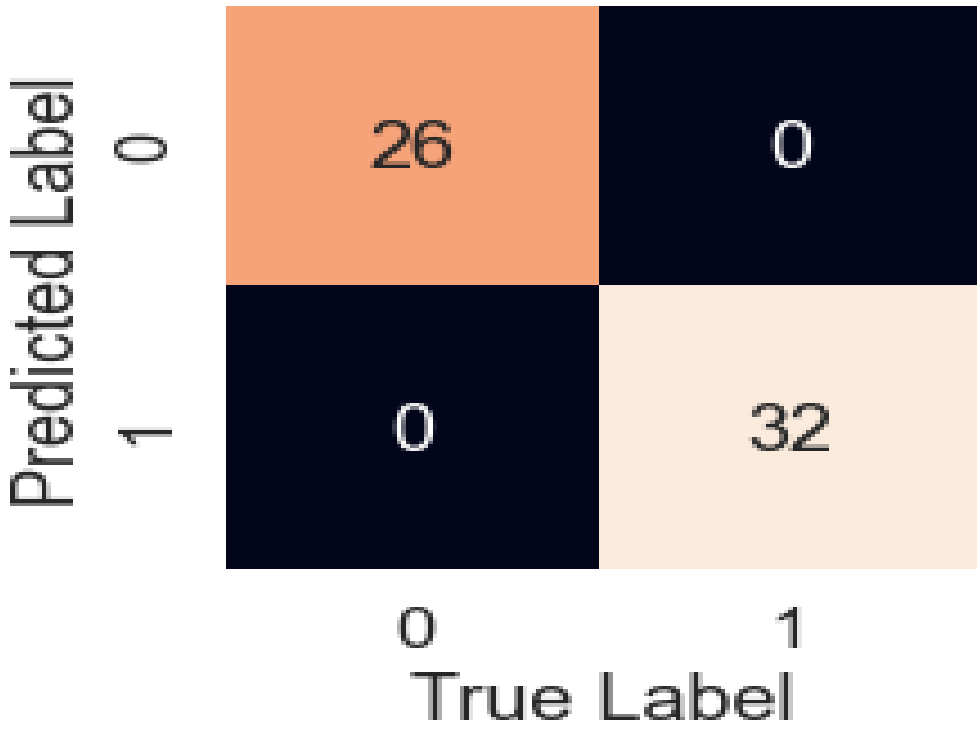
Output:

```
1.0
```

Confusion matrix for KNN model

```
knn_pred = model.predict(X_test)
plot_conf_mat(y_test, knn_pred)
```

Output:




```
tn, fp, fn, tp = confusion_matrix(y_test, preds).ravel()
print(f'True Neg: {tn}')
print(f'False Pos: {fp}')
print(f'False Neg: {fn}')
print(f'True Pos: {tp}')
```

Output:

```
True Neg: 26
False Pos: 0
False Neg: 0
True Pos: 32
```

Feature Importance

```
feature_dict=dict(zip(df.columns,list(logreg.coef_[0])))
feature_dict
```

Here we will get the coefficient from the features which will tell the weightage of each feature.

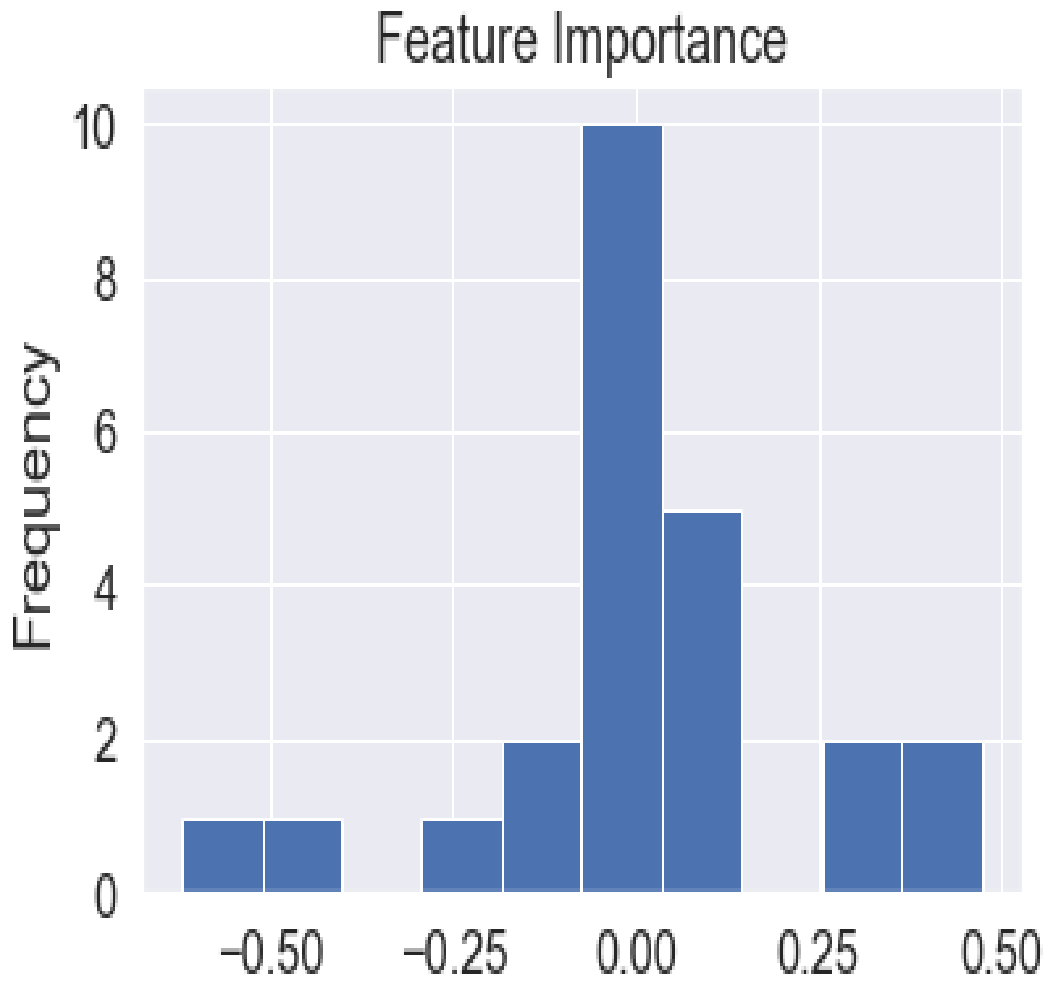
Output:

```
{'age': 0.2858203378727209,
 'bp': -0.13211767022170745,
 'sg': 0.0026714534270341444,
 'al': 0.3099528545093234,
 'su': 0.010788785962584712,
 'rbc': 0.01985635073828642,
 'pc': 0.09106895589320387,
 'pcc': 0.003106393240262293,
 'ba': 0.006828878616469952,
 'bgr': 0.4140451203997997,
 'bu': 0.47262371944289844,
 'sc': 0.12893875993072498,
 'sod': -0.4419699201228987,
 'pot': 0.05909714695858163,
 'hemo': -0.28286805186344094,
 'pcv': -0.6211104727832718,
 'wbcc': 0.001157338688486265,
 'rbcc': -0.1417833283935927,
 'htn': 0.08881269207443204,
 'dm': 0.08689401433102413,
 'cad': 0.0018059681932075433,
 'appet': -0.004481530609769657,
 'pe': 0.0051126943850270425,
 'ane': 0.00349950552414094}
```

Visualize feature importance

```
feature_df=pd.DataFrame(feature_dict,index=[0])
feature_df.T.plot(kind="hist",legend=False,title="Feature
Importance")
```

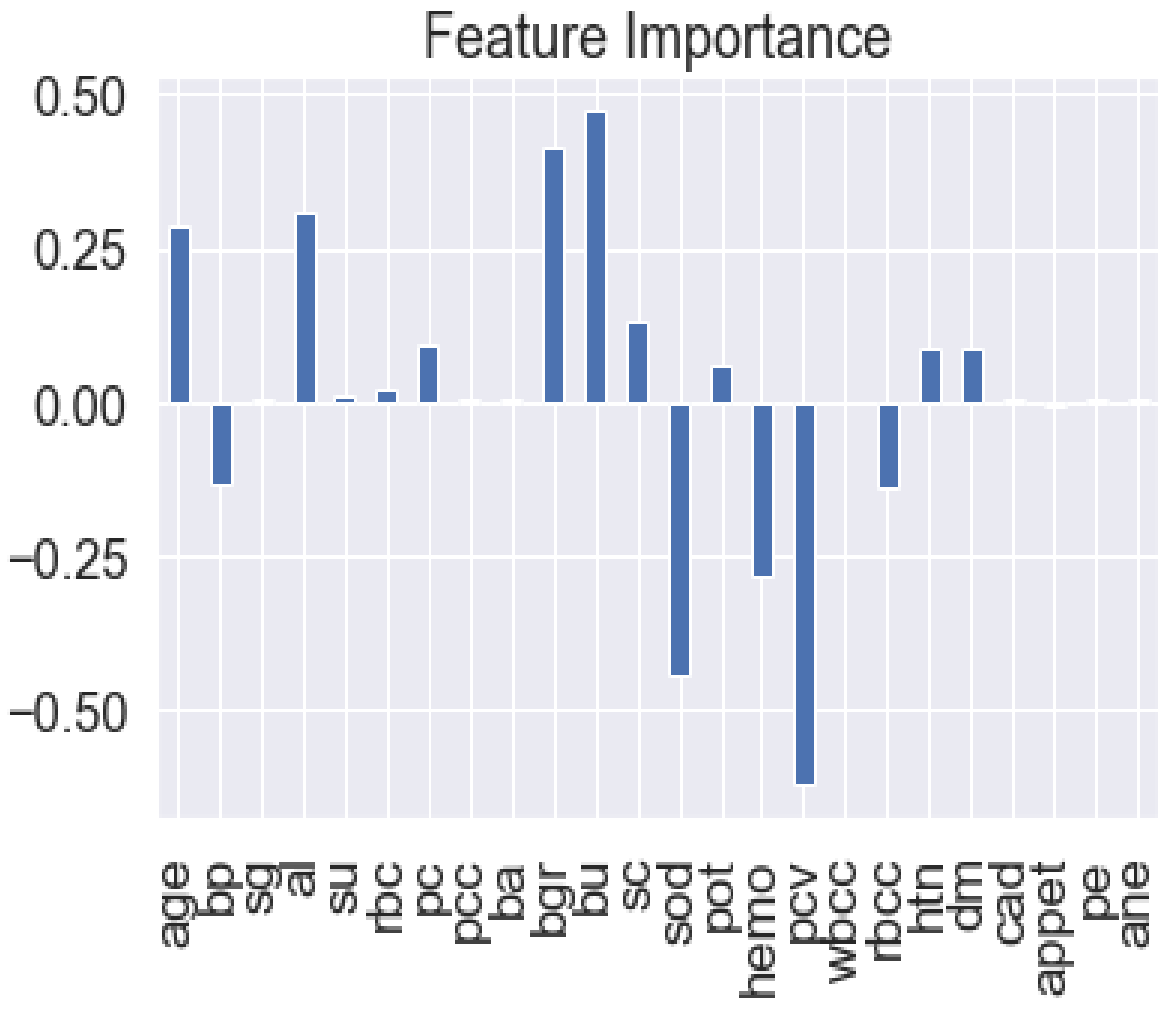
Output:



Visualize feature importance – Transpose

```
feature_df=pd.DataFrame(feature_dict,index=[0])
feature_df.T.plot(kind="bar",legend=False,title="Feature Importance")
```

Output:



Saving the model

```
import pickle

# Now with the help of pickle model we will be saving the trained model
saved_model = pickle.dumps(logreg)

# Load the pickled model
logreg_from_pickle = pickle.loads(saved_model)

# Now here we will load the model
logreg_from_pickle.predict(X_test)
```

Output:

```
array([1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1], dtype=int64)
```

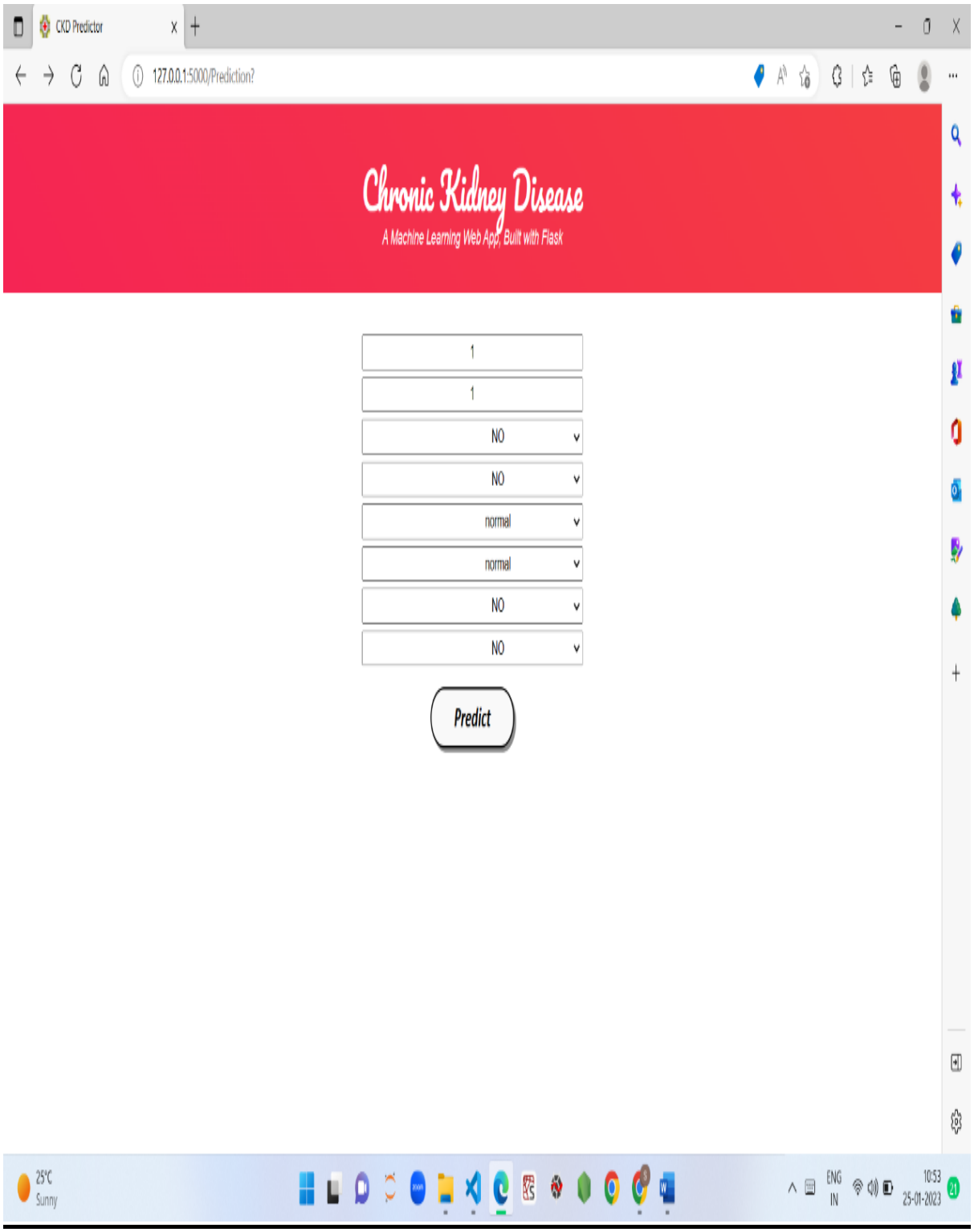
Screen layouts


Chronic Kidney Disease
A Machine Learning Web App, Built with Flask
Enter your blood_urea
Enter your blood glucose random
Select anemia or not
Select coronary artery disease or not
Select pus_cell or not
Select red_blood_cell level
Select diabetesmellitus or not
Select pedal_edema or not
Predict

Chronic Kidney Disease

A Machine Learning Web App, Built with Flask

Prediction: Oops! You have Chronic Kidney Disease.


CKD Predictor
127.0.0.1:5000/Prediction?
Chronic Kidney Disease
A Machine Learning Web App, Built with Flask
1
1
NO
NO
normal
normal
NO
NO
Predict
25°C
Sunny
ENG
IN
10:53
25-01-2023


Chronic Kidney Disease
127.0.0.1:5000/predict
Chronic Kidney Disease
A Machine Learning Web App, Built with Flask
Prediction: Great! You DON'T have Chronic Kidney Disease
IT'S TIME TO PARTYYYYYY !
25°C
Sunny
ENG
IN
10:52
25-01-2023

Conclusions:

This study explored how a learning model can be used to classify the possibility of a CKD diagnosis.

Consequently, the indirect agreement with the objectives of the project, the CRISP-DM methodology was adapted to the context of the problem so that the different logically organized stages were taken; data collection, pre-processing, learning, evaluation, and selection, which allowed the construction of a model capable of classifying the possibility of a diagnosis of CKD with an accuracy of 93%. As evidenced in the results, the decision forests algorithm has obtained quite optimal results, where predictions of 93% have been obtained. Data preparation is a fundamental step in the process and the absolute precision of the model is directly dependent on this phase. Thanks to the models, we can see how changing the characteristics affects the search for the target value with a simple change of column selection or improvements in the data. The innovation of this work results from the design adjusted to the environment of the health system in Iraq and the pathology of the CKD in our country, with a methodology adapted to the case study and a production architecture proposal for the model with Microsoft Azure tools of form that allows satisfying in the future the scalability of the solution. Furthermore, this methodology could apply to clinical data of other diseases and pathologies inaccurate medical diagnoses. The development of this project allowed the author to acquire more excellent knowledge through both practical and theoretical work about current techniques for the development of machine learning.

This study has limitations, so there is a room for future research. The study did not have a significant data sample due to the restrictions of medical data and its legal effects in Iraq. Continuing with the expansion of the database (increasing the number of examples for each variable) would reduce the limited generalization error for the model and, at the same time, allow the severity of the disease to be detected. This model can be refined with increasing data size and quality. It also opens the space for a variety of studies from other disciplines, such as economic studies around the impact of obtaining a diagnosis in less time to treat the disease in early stages, reducing costs in the health system. In addition, a variety of sociological and clinical studies on the consequences of early CKD management brings about the quality of life of patients and their families. Although the validity of this research is internal, since the document corpus is private and cannot be published for other works, it will help interested professionals with machine learning to carry out their studies in the classification area.

This study presented a number of different machine learning algorithms with the intention of making a CKD diagnosis at an earlier stage. The models that are constructed using CKD patients are then trained and validated using the input parameters that were discussed earlier. Studies have been done on the associations between different factors so that the number of features can be cut down and redundant information eliminated. When applying a filter feature selection approach to the remaining attributes, it was discovered that hemoglobin, albumin, and specific gravity had the biggest impact when it comes to predicting CKD. This was the case after the method was used. This work presented a number of different machine learning algorithms with the intention of making a CKD diagnosis at an earlier stage. The original CKD

dataset has been preprocessed first to validate the machine learning-based detection models. After that, the PCA has been performed to identify the most dominant features, thereby detecting CKD. The models that are constructed using CKD patients are then trained and validated using the input parameters that were discussed earlier. The accuracy of such algorithms was the primary criterion that was utilized in evaluating their overall performance.