Chronic Kidney Disease

1.1 Introduction

Chronic kidney disease (CKD), also known as chronic renal disease, is a condition that occurs when your kidneys don't work as well as they should to filter waste, toxins and excess fluid from your body. Kidney disease progresses in stages—defined by your eGFR—and may eventually lead to kidney failure. The goal of treating CKD is to best manage your health at every stage, which can help slow progression and keep your kidneys functioning as long as possible. Treatment options for kidney failure include dialysis or a kidney transplant.

One of the trickier aspects of chronic kidney disease (CKD) detection is that the signs and symptoms of kidney disease occurs late, after the condition has progressed. In fact, CKD is sometimes known as a "silent" condition because it's hard to detect—and most people with early stage CKD are completely unaware of it.

1.2 Objectives Of Research

To quickly predict the severity of CKD using more easily available demographic and blood biochemical features during follow-up, we can develop a predictive model using statistical, machine learning and neural network approaches.

1.3 Problem Statement

Development of machine learning tools in the prediction of chronic kidney disease, This problem requires a Supervised Learning Model and is a Classification problem that requires a Logistic Regression approach to solve it.

1.4 Industry Profile

Urinary protein quantification is critical for assessing the severity of chronic kidney disease (CKD). However, the current procedure for determining the severity of CKD is completed through evaluating 24-h urinary protein, which is inconvenient during follow-up.

Chronic kidney disease (CKD) is associated with an increased risk for adverse clinical events, which makes it a major public health problem worldwide. Although it is well recognized that CKD is independently associated with increased risks for end stage renal disease, cardiovascular events, and all-cause mortality, the prognosis for individual patients still lacks sufficient information. Clinically usable strategies for the risk stratification of each outcome are important for
making treatment decisions.

2.Review Of Literature

In this project our main intention is to successfully predict whether a patient has a Chronic Kidney Disease based on their basic medical report. This is done by training a Logistic Regression model using training set of a given data of patients who are either diagnosed with the disease or not.

Once we train our model, we test it using a Test set from the dataset which wasn't used to train the model. We then plot a graph (True Positive Vs False Positive) to see the AUC(Area Under Curve) which shows how accurate our model is.

3.Data Collection

The data was taken over a 2-month period in India with 25 features (eg, red blood cell count, white blood cell count, etc). The target is the 'classification', which is either 'ckd' or 'not ckd' - ckd=chronic kidney disease. There are 400 rows and 26 columns. The data needs cleaning: in that it has NaNs and the numeric features need to be forced to floats.

This data was collected from Kaggle.

4.Methodology

4.1 Exploratory Data Analysis

The Data has 26 Columns and 400 Rows, Each row is the data of one patient and these are the following attributes in the data:

Ind	**-**	**Index**	**sod**	**-**	**sodium**
age	**-**	**age**	**pot**	**-**	**potassium**
bp	**-**	**blood pressure**	**hemo**	**-**	**hemoglobin**
sg	**-**	**specific gravity**	**pcv**	**-**	**packed cell volume**
al	**-**	**albumin**	**wc**	**-**	**white blood cell count**
su	**-**	**sugar**	**rc**	**-**	**red blood cell count**
rbc	**-**	**red blood cells**	**htn**	**-**	**hypertension**
pc	**-**	**pus cell**	**dm**	**-**	**diabetes mellitus**
pcc	**-**	**pus cell clumps**	**cad**	**-**	**coronary artery disease**
ba	**-**	**bacteria**	**appet**	**-**	**appetite**
bgr	**-**	**blood glucose random**	**pe**	**-**	**pedal edema**
bu	**-**	**blood urea**	**ane**	**-**	**anemia**
sc	**-**	**serum creatinine**	**class**	**-**	**classification**

	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	...	44.0	7800.0	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	...	38.0	6000.0	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31.0	7500.0	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32.0	6700.0	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	...	35.0	7300.0	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39.0	7800.0	4.4	yes	yes	no	good	yes	no	ckd
6	6	68.0	70.0	1.010	0.0	0.0	NaN	normal	notpresent	notpresent	...	36.0	NaN	NaN	no	no	no	good	no	no	ckd
7	7	24.0	NaN	1.015	2.0	4.0	normal	abnormal	notpresent	notpresent	...	44.0	6900.0	5.0	no	yes	no	good	yes	no	ckd
8	8	52.0	100.0	1.015	3.0	0.0	normal	abnormal	present	notpresent	...	33.0	9600.0	4.0	yes	yes	no	good	no	yes	ckd
9	9	53.0	90.0	1.020	2.0	0.0	abnormal	abnormal	present	notpresent	...	29.0	12100.0	3.7	yes	yes	no	poor	no	yes	ckd
10	10	50.0	60.0	1.010	2.0	4.0	NaN	abnormal	present	notpresent	...	28.0	NaN	NaN	yes	yes	no	good	no	yes	ckd
11	11	63.0	70.0	1.010	3.0	0.0	abnormal	abnormal	present	notpresent	...	32.0	4500.0	3.8	yes	yes	no	poor	yes	no	ckd
12	12	68.0	70.0	1.015	3.0	1.0	NaN	normal	present	notpresent	...	28.0	12200.0	3.4	yes	yes	yes	poor	yes	no	ckd
13	13	68.0	70.0	NaN	NaN	NaN	NaN	NaN	notpresent	notpresent	...	NaN	NaN	NaN	yes	yes	yes	poor	yes	no	ckd
14	14	68.0	80.0	1.010	3.0	2.0	normal	abnormal	present	present	...	16.0	11000.0	2.6	yes	yes	yes	poor	yes	no	ckd
15	15	40.0	80.0	1.015	3.0	0.0	NaN	normal	notpresent	notpresent	...	24.0	3800.0	2.8	yes	no	no	good	no	yes	ckd
16	16	47.0	70.0	1.015	2.0	0.0	NaN	normal	notpresent	notpresent	...	NaN	NaN	NaN	no	no	no	good	no	no	ckd
17	17	47.0	80.0	NaN	NaN	NaN	NaN	NaN	notpresent	notpresent	...	NaN	NaN	NaN	yes	no	no	poor	no	no	ckd
18	18	60.0	100.0	1.025	0.0	3.0	NaN	normal	notpresent	notpresent	...	37.0	11400.0	4.3	yes	yes	yes	good	no	no	ckd

This Dataset had a lot of records with empty values(NAN), The Records with empty values for **age, rbc, pc, pcc, bcc, cad, appet** were discarded using :

dataset=dataset[pd.notnull(dataset['age'])]

dataset=dataset[pd.notnull(dataset['pcc'])]

dataset=dataset[pd.notnull(dataset['appet'])]

dataset=dataset[pd.notnull(dataset['cad'])]

The records having Null values for attributes **rbc** & **pc** were substituted by “Normal” which is the mode of the attributes.

dataset['rbc'].fillna('normal',inplace=True)

dataset['pc'].fillna('normal',inplace=True)

The other NAN values of the dataset were replaced by their means.

dataset.fillna(dataset.mean(),inplace=True)

Some entries in the classification column that had a tab pressed unintentionally while entering data had to be removed.

dataset=dataset[~dataset.classification.str.contains('ckd\t')]

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	app
0	0	48.0	80.000000	1.020000	1.000000	0.000000	normal	normal	notpresent	notpresent	...	44.000000	7800.000000	5.200000	yes	yes	no	go
1	1	7.0	50.000000	1.020000	4.000000	0.000000	normal	normal	notpresent	notpresent	...	38.000000	6000.000000	4.687645	no	no	no	go
2	2	62.0	80.000000	1.010000	2.000000	3.000000	normal	normal	notpresent	notpresent	...	31.000000	7500.000000	4.687645	no	yes	no	pc
3	3	48.0	70.000000	1.005000	4.000000	0.000000	normal	abnormal	present	notpresent	...	32.000000	6700.000000	3.900000	yes	no	no	pc
4	4	51.0	80.000000	1.010000	2.000000	0.000000	normal	normal	notpresent	notpresent	...	35.000000	7300.000000	4.600000	no	no	no	go
5	5	60.0	90.000000	1.015000	3.000000	0.000000	normal	normal	notpresent	notpresent	...	39.000000	7800.000000	4.400000	yes	yes	no	go
6	6	68.0	70.000000	1.010000	0.000000	0.000000	normal	normal	notpresent	notpresent	...	36.000000	8407.773852	4.687645	no	no	no	go
7	7	24.0	76.675603	1.015000	2.000000	4.000000	normal	abnormal	notpresent	notpresent	...	44.000000	6900.000000	5.000000	no	yes	no	go
8	8	52.0	100.000000	1.015000	3.000000	0.000000	normal	abnormal	present	notpresent	...	33.000000	9600.000000	4.000000	yes	yes	no	go
9	9	53.0	90.000000	1.020000	2.000000	0.000000	abnormal	abnormal	present	notpresent	...	29.000000	12100.000000	3.700000	yes	yes	no	pc
10	10	50.0	60.000000	1.010000	2.000000	4.000000	normal	abnormal	present	notpresent	...	28.000000	8407.773852	4.687645	yes	yes	no	go
11	11	63.0	70.000000	1.010000	3.000000	0.000000	abnormal	abnormal	present	notpresent	...	32.000000	4500.000000	3.800000	yes	yes	no	pc
12	12	68.0	70.000000	1.015000	3.000000	1.000000	normal	normal	present	notpresent	...	28.000000	12200.000000	3.400000	yes	yes	yes	pc
13	13	68.0	70.000000	1.017382	1.020528	0.461538	normal	normal	notpresent	notpresent	...	38.860759	8407.773852	4.687645	yes	yes	yes	pc
14	14	68.0	80.000000	1.010000	3.000000	2.000000	normal	abnormal	present	present	...	16.000000	11000.000000	2.600000	yes	yes	yes	pc
15	15	40.0	80.000000	1.015000	3.000000	0.000000	normal	normal	notpresent	notpresent	...	24.000000	3800.000000	2.800000	yes	no	no	go
16	16	47.0	70.000000	1.015000	2.000000	0.000000	normal	normal	notpresent	notpresent	...	38.860759	8407.773852	4.687645	no	no	no	go
17	17	47.0	80.000000	1.017382	1.020528	0.461538	normal	normal	notpresent	notpresent	...	38.860759	8407.773852	4.687645	yes	no	no	pc
18	18	60.0	100.000000	1.025000	0.000000	3.000000	normal	normal	notpresent	notpresent	...	37.000000	11400.000000	4.300000	yes	yes	yes	go
19	19	62.0	60.000000	1.015000	1.000000	0.000000	normal	abnormal	present	notpresent	...	30.000000	5300.000000	3.700000	yes	no	yes	go

Final Dataset After All Changes

4.2 Statistical Techniques and Data Visualization

Splitting of data into X - variables & attributes and y - output (whether chronic kidney disease or not)

X = dataset.iloc[:, 1:24].values

y = dataset.iloc[:, 25].values

Next the categorical data was to be encoded from string to float format.

This was done using the fit_transform method.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
lb =LabelEncoder()
lb_y =LabelEncoder()
y=lb_y.fit_transform(y)
X[:,5]=lb.fit_transform(X[:,5])
X[:,6]=lb.fit_transform(X[:,6])
X[:,7]=lb.fit_transform(X[:,7])
X[:,8]=lb.fit_transform(X[:,8])
X[:,18]=lb.fit_transform(X[:,18])
X[:,19]=lb.fit_transform(X[:,19])
X[:,20]=lb.fit_transform(X[:,20])
X[:,21]=lb.fit_transform(X[:,21])
X[:,22]=lb.fit_transform(X[:,22])
X[:,23]=lb.fit_transform(X[:,23])
```

X before transforming:

```
X
```

```
array([[48.0, 80.0, 1.02, 1.0, 0.0, 'normal', 'normal', 'notpresent',
        'notpresent', 121.0, 36.0, 1.2, 137.48166666666665,
        4.632775919732443, 15.4, 44.0, 7800.0, 5.2, 'yes', 'yes', 'no',
        'good', 'no'],
       [7.0, 50.0, 1.02, 4.0, 0.0, 'normal', 'normal', 'notpresent',
        'notpresent', 148.9824046920821, 18.0, 0.8, 137.48166666666665,
        4.632775919732443, 11.3, 38.0, 6000.0, 4.687644787644789, 'no',
        'no', 'no', 'good', 'no'],
       [62.0, 80.0, 1.01, 2.0, 3.0, 'normal', 'normal', 'notpresent',
        'notpresent', 423.0, 53.0, 1.8, 137.48166666666665,
        4.632775919732443, 9.6, 31.0, 7500.0, 4.687644787644789, 'no',
        'yes', 'no', 'poor', 'no'],
       [48.0, 70.0, 1.005, 4.0, 0.0, 'normal', 'abnormal', 'present',
        'notpresent', 117.0, 56.0, 3.8, 111.0, 2.5, 11.2, 32.0, 6700.0,
        3.9, 'yes', 'no', 'no', 'poor', 'yes'],
       [51.0, 80.0, 1.01, 2.0, 0.0, 'normal', 'normal', 'notpresent',
        'notpresent', 106.0, 26.0, 1.4, 137.48166666666665,
        4.632775919732443, 11.6, 35.0, 7300.0, 4.6, 'no', 'no', 'no',
        'good', 'no'],
```

X after transforming:

```
: X
```

```
: array([[48.0, 80.0, 1.02, 1.0, 0.0, 1, 1, 0, 0, 121.0, 36.0, 1.2,
         137.48166666666665, 4.632775919732443, 15.4, 44.0, 7800.0, 5.2,
         1, 3, 1, 0, 0],
        [7.0, 50.0, 1.02, 4.0, 0.0, 1, 1, 0, 0, 148.9824046920821, 18.0,
         0.8, 137.48166666666665, 4.632775919732443, 11.3, 38.0, 6000.0,
         4.687644787644789, 0, 2, 1, 0, 0],
        [62.0, 80.0, 1.01, 2.0, 3.0, 1, 1, 0, 0, 423.0, 53.0, 1.8,
         137.48166666666665, 4.632775919732443, 9.6, 31.0, 7500.0,
         4.687644787644789, 0, 3, 1, 1, 0],
        [48.0, 70.0, 1.005, 4.0, 0.0, 1, 0, 1, 0, 117.0, 56.0, 3.8, 111.0,
         2.5, 11.2, 32.0, 6700.0, 3.9, 1, 2, 1, 1, 1],
        [51.0, 80.0, 1.01, 2.0, 0.0, 1, 1, 0, 0, 106.0, 26.0, 1.4,
         137.48166666666665, 4.632775919732443, 11.6, 35.0, 7300.0, 4.6,
         0, 2, 1, 0, 0],
        [60.0, 90.0, 1.015, 3.0, 0.0, 1, 1, 0, 0, 74.0, 25.0, 1.1, 142.0,
         3.2, 12.2, 39.0, 7800.0, 4.4, 1, 3, 1, 0, 1],
        [68.0, 70.0, 1.01, 0.0, 0.0, 1, 1, 0, 0, 100.0, 54.0, 24.0, 104.0,
         4.0, 12.4, 36.0, 8407.773851590106, 4.687644787644789, 0, 2, 1,
         0, 0],
```

y before transforming:

```
array(['ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd',
       'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'ckd', 'notckd', 'notckd',
       'notckd', 'notckd', 'notckd', 'notckd', 'notckd', 'notckd',
       'notckd', 'notckd', 'notckd', 'notckd', 'notckd', 'notckd',
       'notckd', 'notckd', 'notckd', 'notckd', 'notckd', 'notckd',
       'notckd', 'notckd', 'notckd', 'notckd', 'notckd', 'notckd',
```

Y after transforming:

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

4.3 Data Modeling using Supervised ML techniques

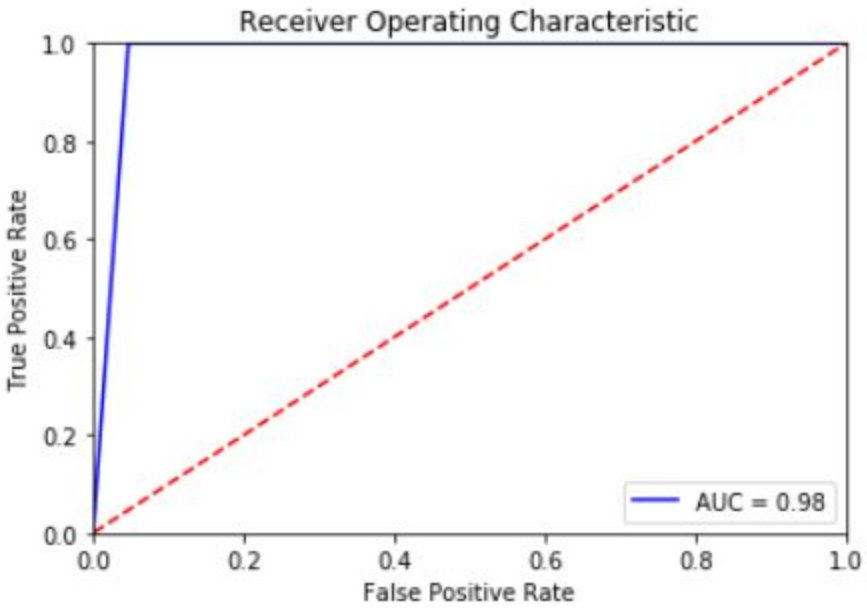
The Dataset is now split into the Training set and Test set using the sklearn library function train_test_split.

Now Feature Scaling is done to round off the data in a feasible format so it is easier to work on using the function StandardScaler from the skLearn Library.

Next Logistic Regression is fitted into the Training set, thereby training our model to the values of the training set.

After the training, a prediction set is made from the test set and the values of the test set and predict set is compared to see the accuracy of the model.

To do this a Confusion Matrix is made and an AUC graph is plotted (True Positive vs False Positive).



5.Findings and Suggestions

AUC=0.98

More Data could be used to make a better Model.

6.Conclusion

Blood-derived tests could be applied as non-urinary predictors during outpatient follow-up. Features in routine blood tests, including ALB, Scr, TG, LDL and EGFR levels, showed predictive ability for CKD severity. The developed online tool can facilitate the prediction of Chronic Kidney Disease during follow-up in clinical practice.

Hence, Using Logistic Regression (Supervised Learning) we were successfully able to build a model with a 98% accuracy to predict whether a patient has a Chronic Kidney Disease or not with the help of their clinical data/report.

7.References

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6458616/

https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease

https://www.kaggle.com/mansoordaku/ckdisease

https://www.freseniuskidneycare.com/about-chronic-kidney-disease/understanding-ckd/diagnosis-and-testing