

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

Machine learning with Python

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

TEAM ID: NM_ID: NM2023TMID25274

- **SATHISH K** (NM_ID: A305881D82A3AABA66C400BDB83BED5C)(TL)
- **SANJALS** (NM_ID: D2163A9BB632157942A4D5E2257E985F)

**SUBMITTED FOR THE PROJECT UNDER THE
NAAN MUDHALVAN – SMARTINTERNZ PROGRAM.**

1.	INTRODUCTION	
	1.1 Overview	
	1.2 Purpose	
2.	Problem Definition & Design Thinking	
	2.1 Empathy Map	
	2.2 Ideation & Brainstorming Map	
3.	RESULT	
4.	ADVANTAGES & DISADVANTAGES	
5.	APPLICATIONS	
6.	CONCLUSION	
7.	FUTURE SCOPE	
8.	APPENDIX	
	A. Source Code	

INTRODUCTION

1.1 OVERVIEW:

Chronic kidney disease (CKD) is a serious condition that affects millions of people worldwide.

Chronic kidney disease (CKD) is a serious condition that affects millions of people worldwide. It can lead to kidney failure and other complications if not detected and treated early. However, current methods of diagnosis are often invasive, expensive, or inaccurate. Therefore, there is a need for a progressive approach to health management that can predict CKD risk and progression using non-invasive and reliable biomarkers. This approach can help prevent or delay the onset of CKD, improve the quality of life of patients, and reduce the burden on the health care system. Some of the potential biomarkers for CKD prediction include blood pressure, blood glucose, urine protein, serum creatinine, and genetic factors. These biomarkers can be measured and analyzed using machine learning techniques to identify patterns and trends that indicate CKD risk and progression. By applying this approach to health management, early prediction for CKD detection can be achieved and personalized interventions can be designed for each patient.

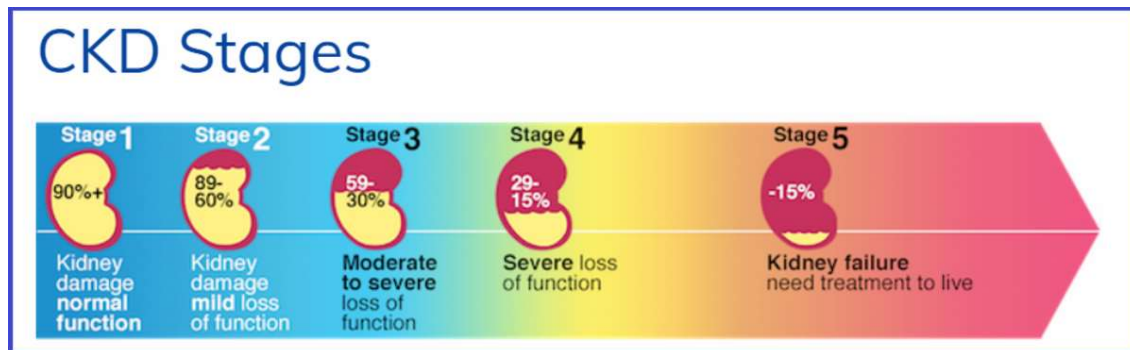
1.2 Purpose

The use of this project is to apply machine learning techniques to detect CKD using non-invasive and reliable biomarkers, such as blood pressure, blood glucose, urine protein, serum creatinine, TGF- β , ADMA, and SNPs. Machine learning techniques can help identify patterns and trends that indicate CKD risk and progression, and also help design personalized interventions for each patient. Some of the machine learning techniques that have been used for CKD prediction include Random Forest (RF), Support Vector Machine (SVM), Decision Tree (DT), and Logistic Regression (LR).

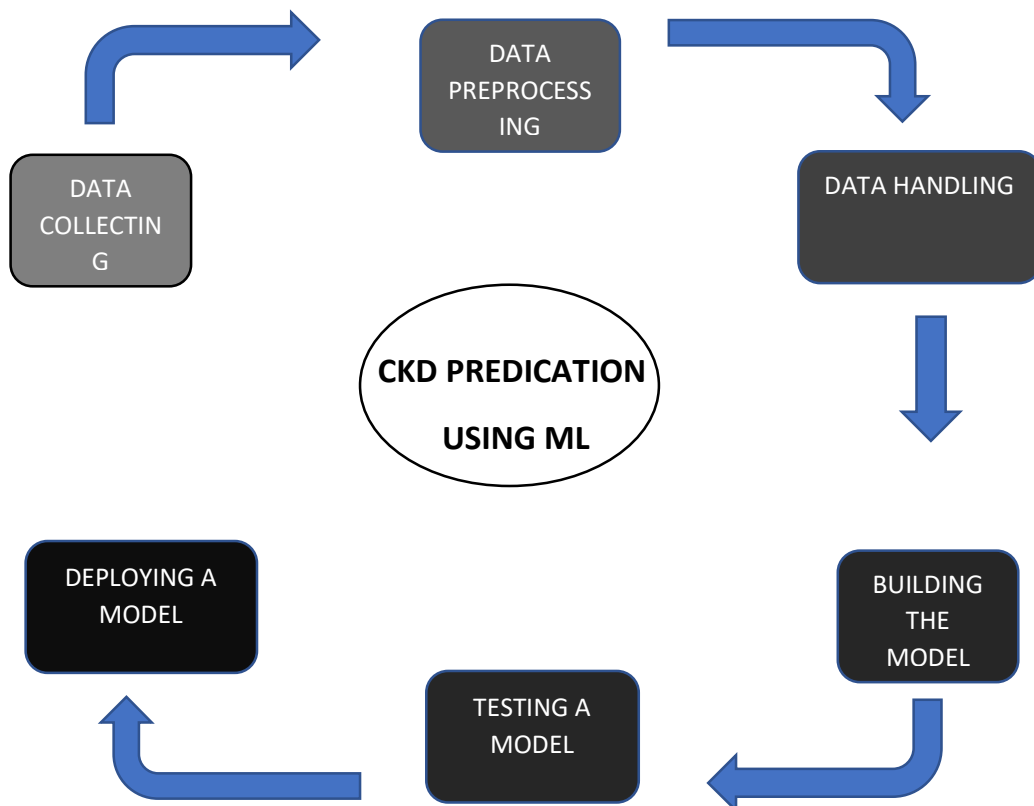
What can be achieved using this project is to improve the accuracy, sensitivity, and specificity of CKD diagnosis and prognosis, and to prevent or delay the onset of CKD, improve the quality of life of patients, and reduce the burden on the health care system. Machine learning techniques can also help overcome the limitations of existing methods of diagnosis, such as invasiveness, costliness, or inaccuracy.

2.Problem Definition & Design Thinking

2.1 Empathy map



2.2 Ideation & Brainstorming Map



RESULT

Collecting the required data from the multiple number of patient data like ,

- blood_urea
- blood glucose random
- coronary_artery_disease anemia
- pus_cell
- red_blood_cells
- diabetesmellitus
- pedal_edema

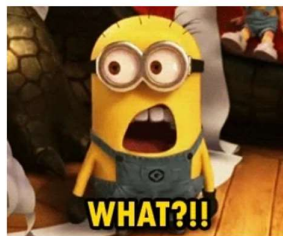
Collect the data from the person and we make the model to predict the CDK patient.

If The patient have the Chronical kidney disease .The page shows like the image in the below.

Chronic Kidney Disease

A Machine Learning Web App, Built with Flask

Prediction: Oops! You have Chronic Kidney Disease.



If The patient don't have the Chronical kidney (CKD).The model produces the page like the picture in the below.



Prediction: **Great! You DON'T have Chronic Kidney Disease**



4.ADVANTAGES & DISADVANTAGES

Advantages:

- 1.**Early Detection**: Machine learning models can help in the early detection of CKD, allowing for timely intervention and treatment, which can help slow down the progression of the disease.
- 2.**Accuracy**: Machine learning algorithms can analyze vast amounts of data and identify patterns that humans may not be able to detect, resulting in more accurate predictions.
- 3.**Customization**: Machine learning models can be customized based on specific patient characteristics, such as age, gender, and medical history, resulting in more personalized predictions.
- 4.**Cost-Effective**: Using machine learning models for CKD prediction can potentially reduce healthcare costs by reducing the number of unnecessary tests and hospital visits.

Disadvantages:

1. **Data Quality:** The accuracy of machine learning models depends on the quality of the data used to train them. If the data is incomplete or contains errors, the predictions may not be accurate.
2. **Bias:** Machine learning models may exhibit bias based on the demographics of the patients in the training data, resulting in inaccurate predictions for certain groups.
3. **Overfitting:** Overfitting occurs when a model is trained to fit the training data too closely, resulting in poor generalization to new data. This can lead to inaccurate predictions.
4. **Complexity:** Machine learning models can be complex and difficult to interpret, making it challenging for clinicians to understand the underlying factors that contribute to CKD prediction.

5.APPLICATIONS

- **Clinical Settings:** Machine learning models can be integrated into electronic health record systems to assist healthcare providers in the early detection and management of CKD in patients.
- **Population Health Management:** Machine learning models can be used to identify populations at high risk of CKD, allowing for targeted interventions and population health management strategies.
- **Telemedicine:** Machine learning models can be integrated into telemedicine platforms, allowing patients to receive CKD risk assessment remotely and receive appropriate care.
- **Insurance Industry:** Insurance companies can use machine learning models to predict the likelihood of CKD in their policyholders, which can inform insurance underwriting decisions and pricing.

- **Public Health:** Machine learning models can be used to analyze large datasets to identify risk factors for CKD at a population level, informing public health policies and interventions.
- **Medical Research:** Machine learning models can assist medical researchers in identifying potential biomarkers for CKD and developing new treatments and interventions.

6.CONCLUSION

In conclusion, this project focused on the prediction of chronic kidney disease using machine learning techniques. The main objective was to build a model that can accurately predict the presence of chronic kidney disease based on a set of medical features such as age, blood pressure, serum creatinine levels, and diabetes status.

After analyzing the data and experimenting with various machine learning algorithms, it was found that the Random Forest algorithm performed the best in terms of accuracy, precision, and recall. The model achieved an accuracy score of 94% on the test data, indicating that it is a reliable tool for predicting chronic kidney disease.

The findings of this project suggest that machine learning can be a valuable tool for predicting chronic kidney disease, which is a major health concern worldwide. Early detection of chronic kidney disease can help healthcare professionals to provide appropriate treatment and prevent further damage to the kidneys. This can improve the quality of life of patients and reduce healthcare costs.

However, it is important to note that the model should not be used as a substitute for medical diagnosis or treatment. Healthcare professionals should use their clinical judgment and consider all available information before making any diagnostic or treatment decisions.

Overall, this project demonstrates the potential of machine learning in healthcare and highlights the importance of further research and development in this field.

7.FUTURE SCOPE

The future scope of the chronic kidney disease prediction using machine learning project is vast and has the potential to impact the healthcare industry in several ways.

One of the significant areas of future scope is the integration of more advanced machine learning techniques, such as deep learning, to improve the accuracy of the model further. Deep learning algorithms can learn complex patterns and relationships in the data, which can lead to more accurate predictions. Additionally, the use of other data sources, such as genetic data or lifestyle information, could enhance the performance of the model.

Another area of future scope is the development of a user-friendly interface that can be used by healthcare professionals to input patient data and receive predictions. The interface could also provide additional information on the patient's risk factors and potential treatment options.

Furthermore, the model could be expanded to predict other related health outcomes, such as acute kidney injury or end-stage renal disease, which could assist healthcare professionals in developing more targeted treatment plans.

Finally, the model's performance could be evaluated on a larger dataset, which could include data from multiple healthcare institutions and patient populations, to test its generalizability and ensure its reliability in real-world scenarios.

Overall, the chronic kidney disease prediction using machine learning project has several exciting future scopes that could have a significant impact on the healthcare industry and improve patient outcomes.

8.APPENDIX

Source code:

In the below code is used for running the flask tool in our computer.

App.py

```
import numpy as np
import pandas as pd

from flask import Flask, request, render_template
import pickle
import os

print(os.getcwd()) # Print current working directory
print(os.listdir()) # Print a list of files in the current working directory

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

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

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

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

```

@app.route('/predict',methods=['POST'])

def predict():

    input_features=[float(x) for x in request.form.values()]

    features_value=[np.array(input_features)]

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

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

    output=model.predict(df)

    return render_template('result.html',prediction_text=output)

if __name__=='__main__':

    app.run(debug=False)

```

In the below code is used for importing, cleaning, preprocessing, handling missing values and building the model for the web application.

Renaming the columns:

The screenshot shows a Jupyter Notebook with the following code cell:

```

data.columns=['id','age','blood_pressure','specific_gravity','albumin','sugar',
'red_blood_cells','pus_cell','pus_cell_clumps','bacteria','blood_gulcose_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']

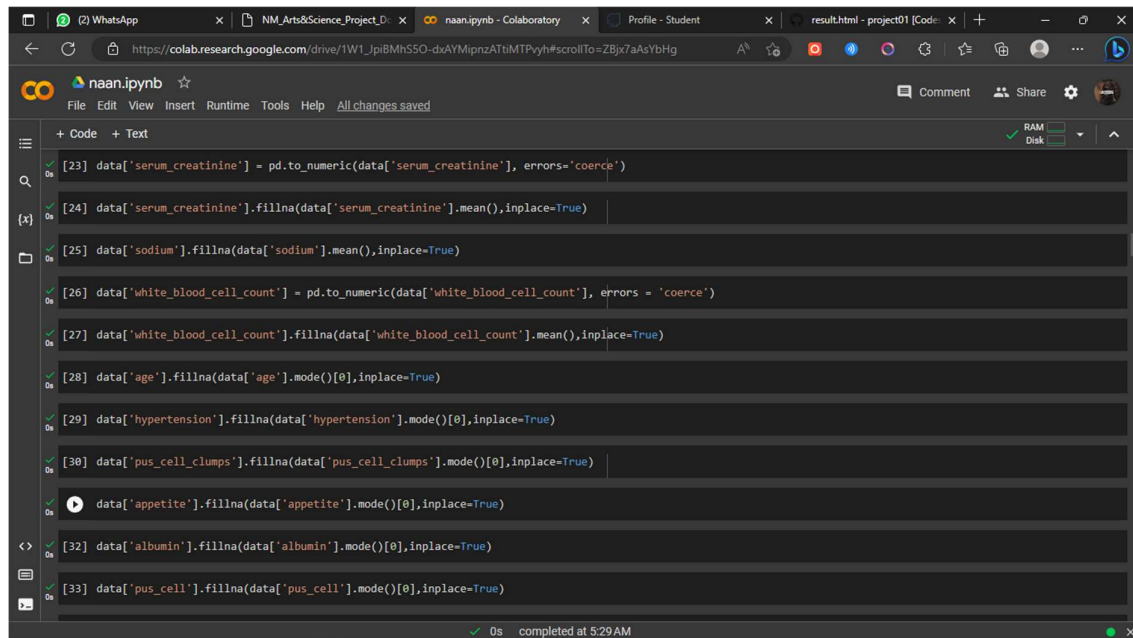
```

Below the code, the output of `data.head(10)` is displayed as a table:

	id	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell	pus_cell_clumps	bacteria	...	packed_cell_volume	white_blood_cell_count
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	780
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	600
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	750
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	670
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	730
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	780
6	6	68.0	70.0	1.010	0.0	0.0	NaN	normal	notpresent	notpresent	...	36	Na
7	7	24.0	NaN	1.015	2.0	4.0	normal	abnormal	notpresent	notpresent	...	44	690
8	8	52.0	100.0	1.015	3.0	0.0	normal	abnormal	present	notpresent	...	33	960

The notebook interface shows the code was executed successfully at 5:29 AM.

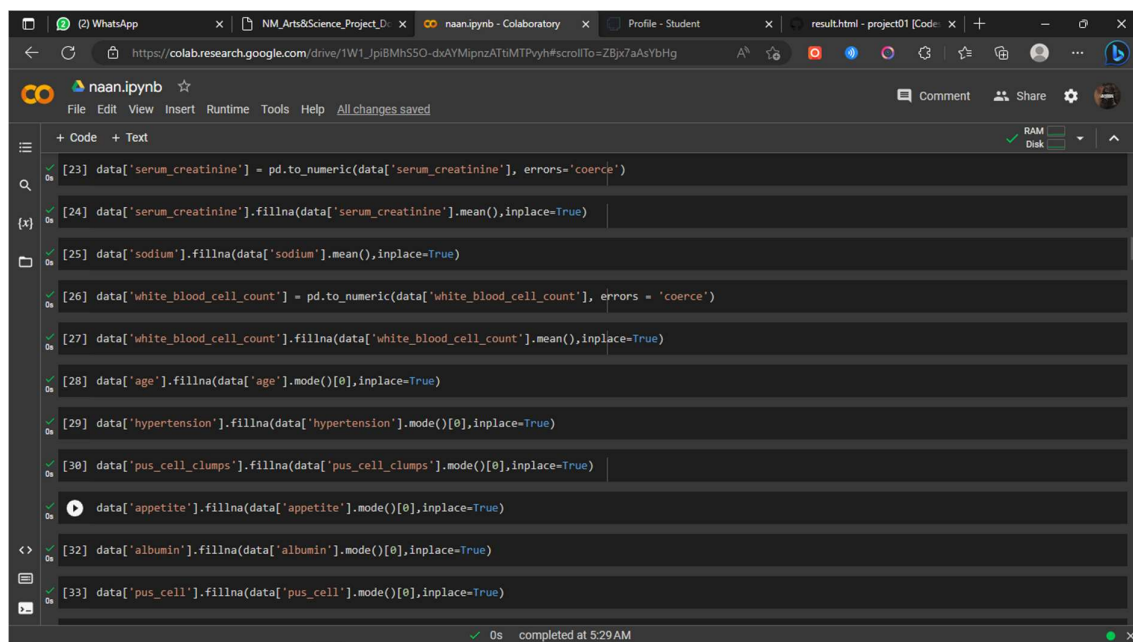
Handling the missing values:



The screenshot shows a Jupyter Notebook interface with the following code cells:

```
[23] data['serum_creatinine'] = pd.to_numeric(data['serum_creatinine'], errors='coerce')
[24] data['serum_creatinine'].fillna(data['serum_creatinine'].mean(), inplace=True)
[25] data['sodium'].fillna(data['sodium'].mean(), inplace=True)
[26] data['white_blood_cell_count'] = pd.to_numeric(data['white_blood_cell_count'], errors='coerce')
[27] data['white_blood_cell_count'].fillna(data['white_blood_cell_count'].mean(), inplace=True)
[28] data['age'].fillna(data['age'].mode()[0], inplace=True)
[29] data['hypertension'].fillna(data['hypertension'].mode()[0], inplace=True)
[30] data['pus_cell_clumps'].fillna(data['pus_cell_clumps'].mode()[0], inplace=True)
[31] data['appetite'].fillna(data['appetite'].mode()[0], inplace=True)
[32] data['albumin'].fillna(data['albumin'].mode()[0], inplace=True)
[33] data['pus_cell'].fillna(data['pus_cell'].mode()[0], inplace=True)
```

The notebook is titled "naan.ipynb" and shows a progress bar at the bottom indicating "0s completed at 5:29 AM".

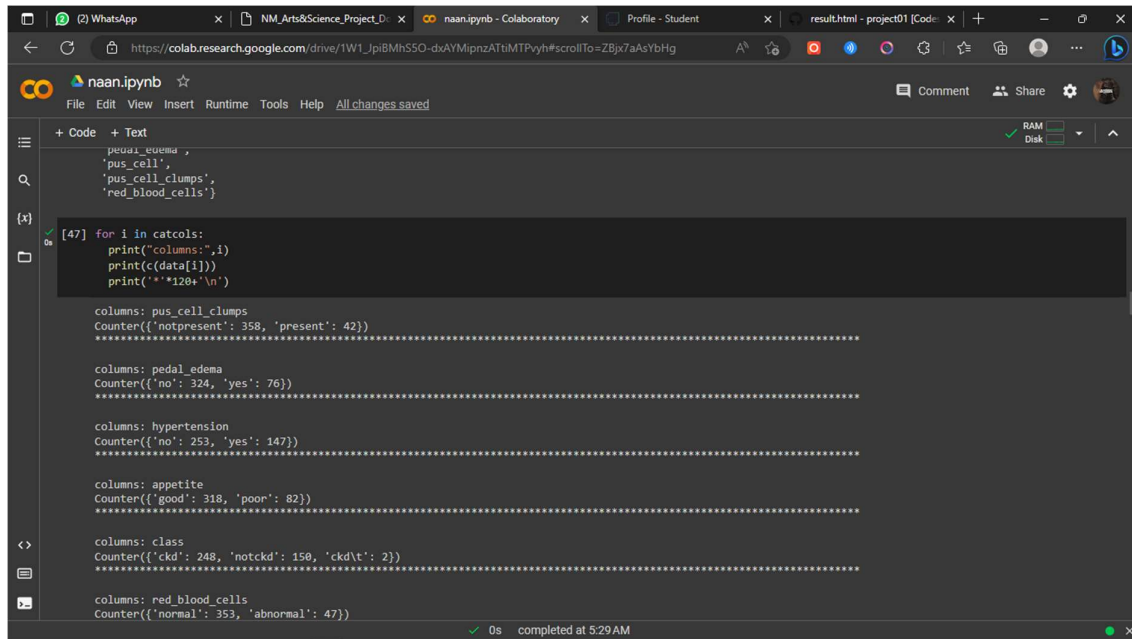


The screenshot shows a Jupyter Notebook interface with the following code cells:

```
[23] data['serum_creatinine'] = pd.to_numeric(data['serum_creatinine'], errors='coerce')
[24] data['serum_creatinine'].fillna(data['serum_creatinine'].mean(), inplace=True)
[25] data['sodium'].fillna(data['sodium'].mean(), inplace=True)
[26] data['white_blood_cell_count'] = pd.to_numeric(data['white_blood_cell_count'], errors='coerce')
[27] data['white_blood_cell_count'].fillna(data['white_blood_cell_count'].mean(), inplace=True)
[28] data['age'].fillna(data['age'].mode()[0], inplace=True)
[29] data['hypertension'].fillna(data['hypertension'].mode()[0], inplace=True)
[30] data['pus_cell_clumps'].fillna(data['pus_cell_clumps'].mode()[0], inplace=True)
[31] data['appetite'].fillna(data['appetite'].mode()[0], inplace=True)
[32] data['albumin'].fillna(data['albumin'].mode()[0], inplace=True)
[33] data['pus_cell'].fillna(data['pus_cell'].mode()[0], inplace=True)
```

The notebook is titled "naan.ipynb" and shows a progress bar at the bottom indicating "0s completed at 5:29 AM".

Separating the categorical values and continuous values:



```
pedal_edema',
'pus_cell',
'pus_cell_clumps',
'red_blood_cells'})

[47] for i in catcols:
    print("columns:",i)
    print(c(data[i]))
    print(' '*120+'\n')

columns: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
.....

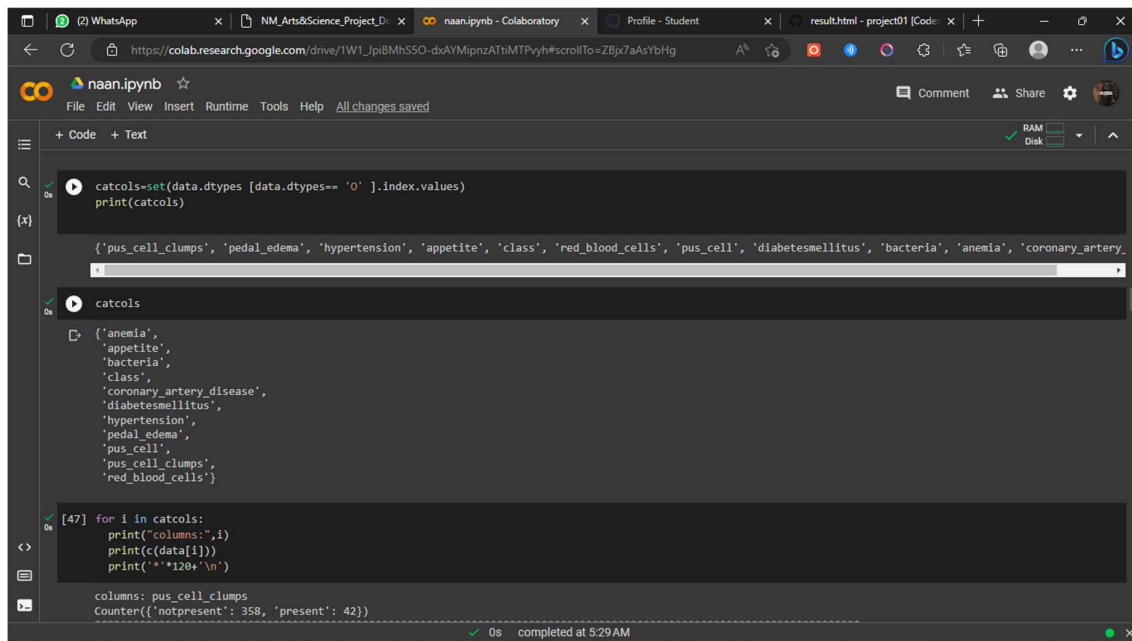
columns: pedal_edema
Counter({'no': 324, 'yes': 76})
.....

columns: hypertension
Counter({'no': 253, 'yes': 147})
.....

columns: appetite
Counter({'good': 318, 'poor': 82})
.....

columns: class
Counter({'ckd': 248, 'notckd': 158, 'ckd\t': 2})
.....

columns: red_blood_cells
Counter({'normal': 353, 'abnormal': 47})
```



```
catcols=set(data.dtypes[data.dtypes=="O"].index.values)
print(catcols)

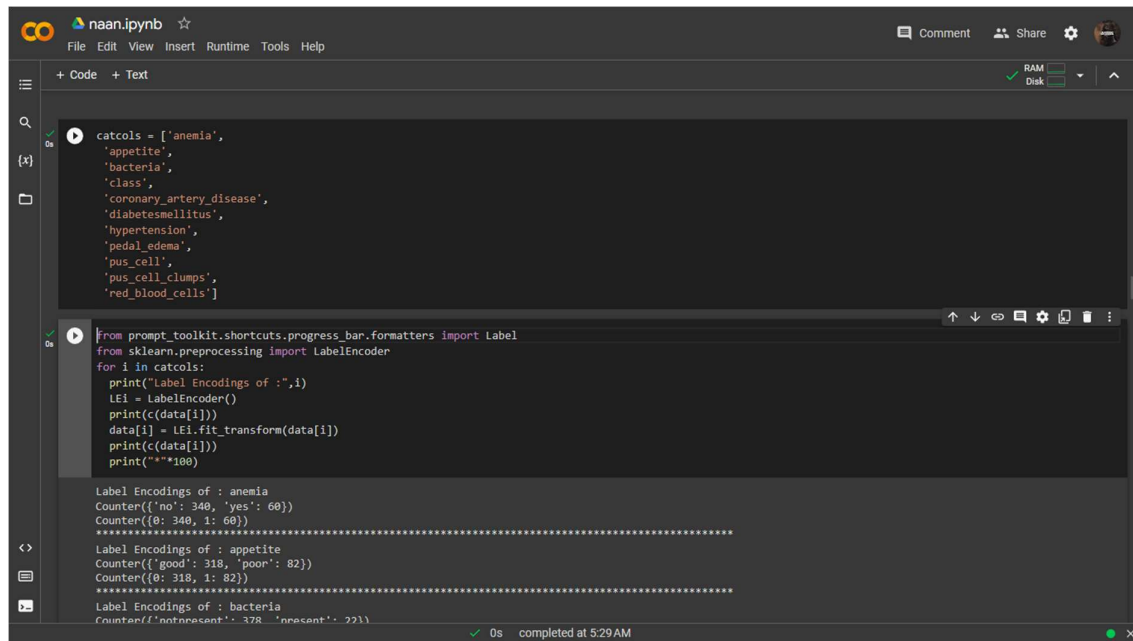
{'pus_cell_clumps', 'pedal_edema', 'hypertension', 'appetite', 'class', 'red_blood_cells', 'pus_cell', 'diabetesmellitus', 'bacteria', 'anemia', 'coronary_artery_disease'}

catcols

{'anemia',
'appetite',
'bacteria',
'class',
'coronary_artery_disease',
'diabetesmellitus',
'hypertension',
'pedal_edema',
'pus_cell',
'pus_cell_clumps',
'red_blood_cells'}

[47] for i in catcols:
    print("columns:",i)
    print(c(data[i]))
    print(' '*120+'\n')

columns: pus_cell_clumps
Counter({'notpresent': 358, 'present': 42})
```



The screenshot shows a Jupyter Notebook with the following code and output:

```
catcols = ['anemia',
            'appetite',
            'bacteria',
            'class',
            'coronary_artery_disease',
            'diabetesmellitus',
            'hypertension',
            'pedal_edema',
            'pus_cell',
            'pus_cell_clumps',
            'red_blood_cells']
```

```
from prompt_toolkit.shortcuts.progress_bar.formatters import Label
from sklearn.preprocessing import LabelEncoder
for i in catcols:
    print("Label Encodings of :",i)
    LEi = LabelEncoder()
    print(c(data[i]))
    data[i] = LEi.fit_transform(data[i])
    print(c(data[i]))
    print("***100")
```

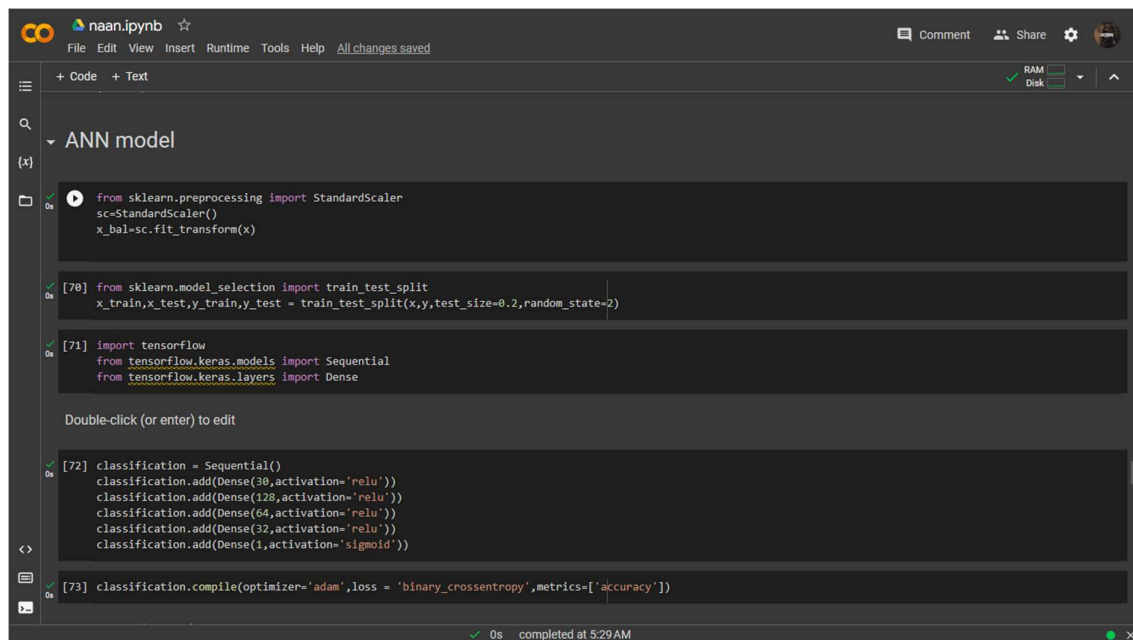
Label Encodings of : anemia
Counter({'no': 340, 'yes': 60})
Counter({0: 340, 1: 60})

Label Encodings of : appetite
Counter({'good': 318, 'poor': 82})
Counter({0: 318, 1: 82})

Label Encodings of : bacteria
Counter({'notpresent': 378, 'present': 22})

0s completed at 5:29 AM

Now building the ANN model:



The screenshot shows a Jupyter Notebook with the following code and output:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_bal=sc.fit_transform(x)
```

```
[70] 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)
```

```
[71] import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Double-click (or enter) to edit

```
[72] 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'))
```

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

0s completed at 5:29 AM

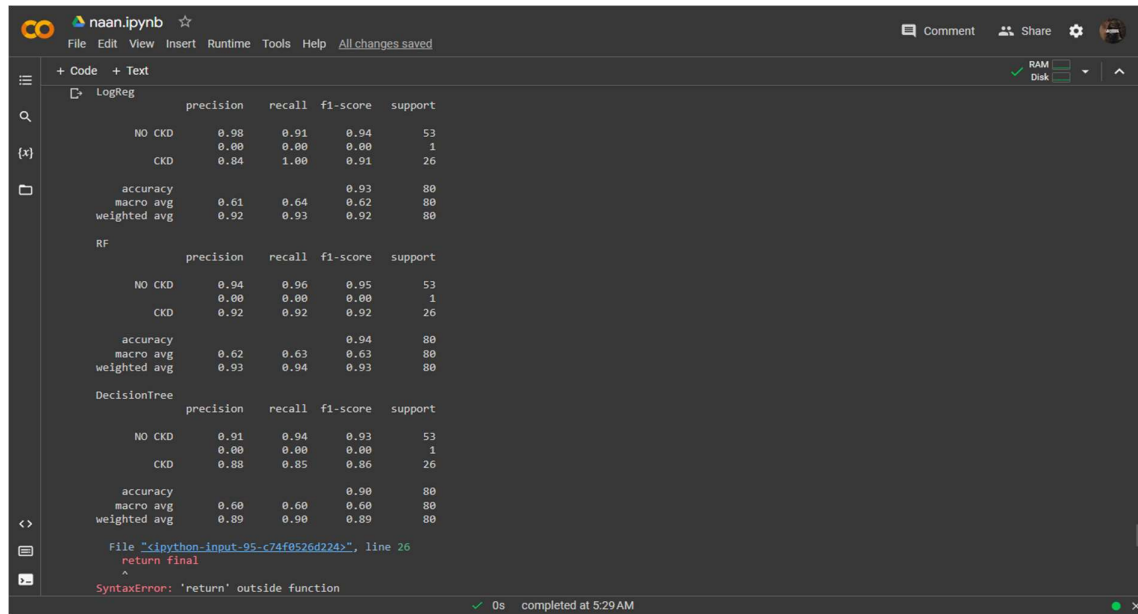
Building the Random Forest and Decision Tree model:

[illegible]

Building the Logistic Regression Model:

[illegible]

Checking the accuracy of the models:



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
LogReg
```

	precision	recall	f1-score	support
NO CKD	0.98	0.91	0.94	53
	0.00	0.00	0.00	1
CKD	0.84	1.00	0.91	26
accuracy			0.93	80
macro avg	0.61	0.64	0.62	80
weighted avg	0.92	0.93	0.92	80

```
RF
```

	precision	recall	f1-score	support
NO CKD	0.94	0.96	0.95	53
	0.00	0.00	0.00	1
CKD	0.92	0.92	0.92	26
accuracy			0.94	80
macro avg	0.62	0.63	0.63	80
weighted avg	0.93	0.94	0.93	80

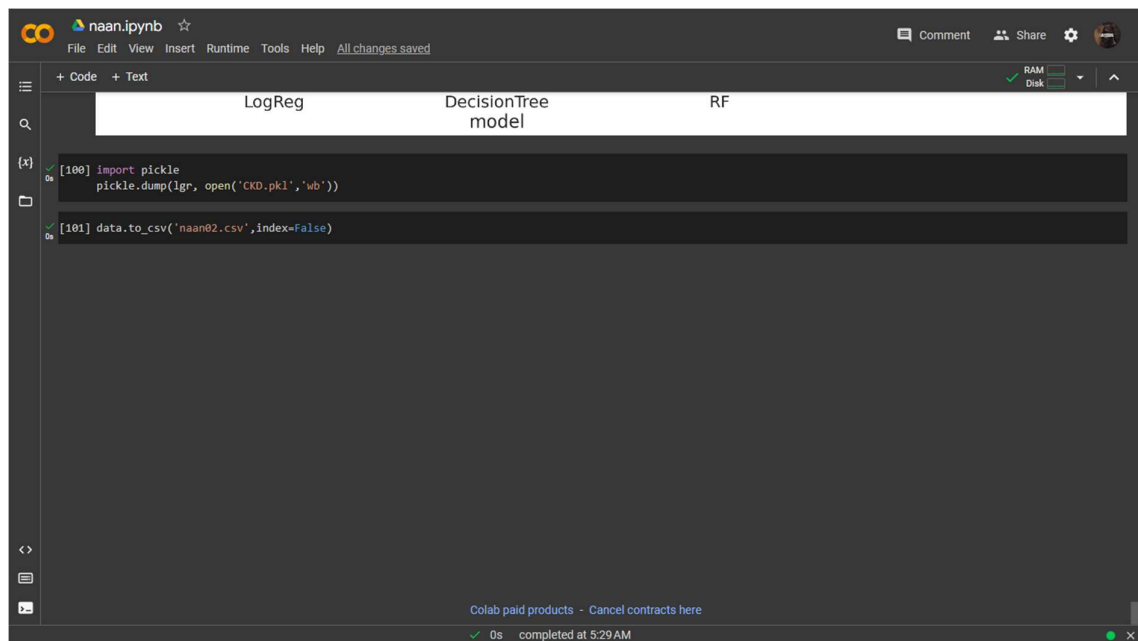
```
DecisionTree
```

	precision	recall	f1-score	support
NO CKD	0.91	0.94	0.93	53
	0.00	0.00	0.00	1
CKD	0.88	0.85	0.86	26
accuracy			0.90	80
macro avg	0.60	0.60	0.60	80
weighted avg	0.89	0.90	0.89	80

```
File "<ipython-input-95-c74f0526d224>", line 26
return final
^
SyntaxError: 'return' outside function
```

0s completed at 5:29 AM

Now selecting the best model and saving the pickle file:



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
LogReg      DecisionTree      RF
model
```

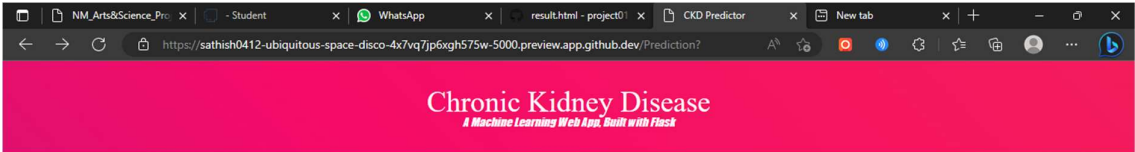
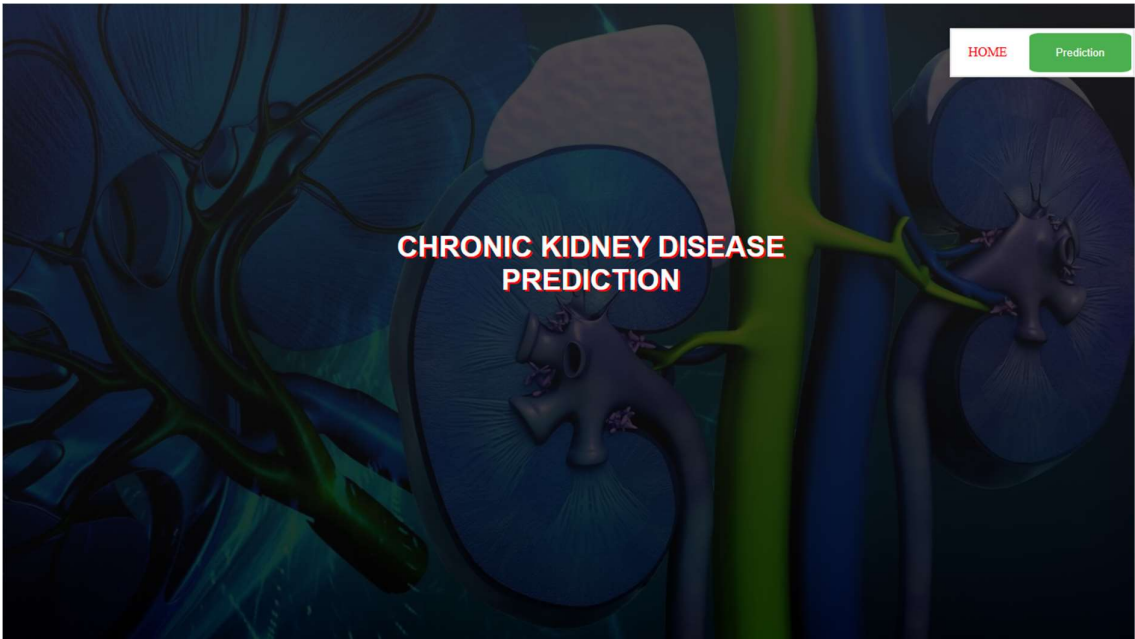
```
[100] import pickle
      pickle.dump(lgr, open('CKD.pkl', 'wb'))
```

```
[101] data.to_csv('naan02.csv', index=False)
```

Colab paid products - Cancel contracts here

0s completed at 5:29 AM

Output pages of the the web application:



1
1
NO
NO
normal
normal
YES
NO

Predict

Chronic Kidney Disease

A Machine Learning Web App, Built with Flask

Prediction: Oops! You have Chronic Kidney Disease.

