

Final Project Report

1. Introduction
 - 1.1. Project overviews
 - 1.2. Objectives
2. Project Initialization and Planning Phase
 - 2.1. Define Problem Statement
 - 2.2. Project Proposal (Proposed Solution)
 - 2.3. Initial Project Planning
3. Data Collection and Preprocessing Phase
 - 3.1. Data Collection Plan and Raw Data Sources Identified
 - 3.2. Data Quality Report
 - 3.3. Data Exploration and Preprocessing
4. Model Development Phase
 - 4.1. Feature Selection Report
 - 4.2. Model Selection Report
 - 4.3. Initial Model Training Code, Model Validation and Evaluation Report
5. Model Optimization and Tuning Phase
 - 5.1. Hyperparameter Tuning Documentation
 - 5.2. Performance Metrics Comparison Report
 - 5.3. Final Model Selection Justification
6. Results
 - 6.1. Output Screenshots
7. Advantages & Disadvantages
8. Conclusion
9. Future Scope
10. Appendix
 - 10.1. Source Cod3
 - 10.2. GitHub & Project Demo Link

Early Prediction of Chronic Kidney Disease using Machine Learning

1.Introduction

1.1 Project overviews

Developing a machine learning methodology for diagnosing chronic kidney diseases involves collecting and preprocessing diverse medical data. Models like logistic regression, decision trees, and neural networks are trained and evaluated using metrics such as accuracy and F1-score. Interpretability and compliance with medical standards are crucial, ensuring the model aids in clinical decision-making. Collaboration with healthcare experts ensures relevance and ethical handling of patient data. Continuous iteration and validation refine the model's accuracy and applicability, aiming to enhance diagnostic precision and patient care outcomes in chronic kidney disease management.

1.2 Objectives

About 10% of the population worldwide suffers from (CKD), and millions die each year because they cannot get affordable treatment, with the number increasing in the elderly. The project aims to develop a machine learning methodology for accurate and timely diagnosis of chronic kidney diseases using diverse medical data. It prioritizes model interpretability, clinical relevance, and ethical data handling. Continuous refinement through collaboration and validation seeks to improve diagnostic precision and enhance patient care outcomes in CKD management. Early detection of CKD is crucial and helpful in decreasing medical resources as ESRD patients preserve their health through hemodialysis, peritoneal dialysis or kidney transplantation

2. Project Initialization and Planning Phase

2.1 Define Problem Statements (Customer Problem Statement Template):

One of the issues that healthcare providers encounter when dealing with patients is early recognition of those who are likely to develop CKD, therefore, take the appropriate preventive measures. CKD is frequently characterized by its ability to maintain the absence of symptoms until the later stages, thus making it difficult to diagnose. Late presentation therefore leads to worsened patient health, higher costs of treating such patients, and a drain on the limited health resources. Thus, we require a robust, stable, and fast solution that can predict the development of CKD based on the clinical attributes in its early stage. Our goal is to use advanced artificial neural network approach and formulate a model that can assess patients' data, pinpoint the potential signs of complications, and give alerts to doctors. This solution should fit well into our current healthcare practice and not impose much complexity on the medical staff, it must provide observations that allows early intervention thus increase the quality of the patient's prognosis and decrease the long term expenses incurred in management of CKD.

Problem statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	Patient suffering with CKD at final stage	Cure my disease quickly to look after my family.	The cost for this disease is to high.	Daily labour with low income.	Optimistic about treatment for my disease was done.



2.2 Project Proposal (Proposed Solution) template

This project proposal outlines a solution to address Early Prediction Of Chronic Kidney

Disease Using Machine Learning. Key features include a machine learning-based prediction model and real-time decision-making.

Project Overview	
Objective	This study proposes the following objectives: “To build and test a machine learning model for the early markers and risk factors of CKD with an intent to improve early diagnosing and early management strategies to increase patient’s quality of living and decrease costs incurred in the healthcare setting.
Scope	This work proposes to design an ML system for classifying CKD employing medical data. Subprocesses cover data acquisition, data cleaning, and transformation, variable selection, model construction, and application to the healthcare domain, with emphasis on enhancing diagnostic performance and decision making aid for CKD.
Problem Statement	
Description	This project uses a machine learning approach to propose a method for the diagnosis of chronic kidney disease based on medical history data. In the context of CKD, it aims to improve diagnostic tools and care interventions for better decision-making process and overall patient management.
Impact	The model developed in this project for the diagnosis of chronic kidney diseases will transform the way the health sector prepares for and manages its patient load in the future by increasing detection rates and delivering better results from treatment. Its mission is to lower the healthcare expenses and improve the patients’ quality of life worldwide; this endeavor establishes a best practice for the utilization of AI in disease management.
Proposed Solution	
Approach	In the context of the CKD diagnosis project, specifications are for assembling clinical data on CKD patients, pre-processing this

	data(missing data imputation, normalization), analyzing the features' distribution, selecting appropriate machine learning algorithms, assessing their performance for clinical practice and deploying the solution.
Key Features	The integration of data from the patients' records, the results of the tests, advanced data preprocessing (handling missing values in the dataset, normalization of data), the selection of the critical features of the dataset such as creatinine level or demographic characteristics, and the model selection and tuning such as Decision Trees and Support Vector Machine (SVM). The solution's specific goal is to provide high diagnostic accuracy

Resource Requirements

Resource Type	Description	Specification/Allocation
Hardware		
Computing Resources	CPU/GPU specifications, number of cores	T4 GPU
Memory	RAM specifications	8 GB
Storage	Disk space for data, models, and logs	1 TB SSD
Software		
Frameworks	Python frameworks	Flask
Libraries	Additional libraries	scikit-learn, pandas, numpy, matplotlib, seaborn, pickle, sklearn, collections, missingo
Development Environment	IDE, version control	Jupyter Notebook, vscode, Git

Data		
Data	Source, size, format	Kaggle dataset, 10,000images,CSV,health department,patient datasets

2.3 Initial Project Planning Template

Sprint	Functional Requirement (Epic)	UserStory Number	UserStory /Task	Story Points	Priority	Team Members	Sprint Start Date	SprintEnd Date (Planned)
Sprint-1	Collect Dataset	CKD-1	DATASET	1	High	P.Durga Naga Bhavani		
Sprint-1	Clean the Dataset	CKD-2	Importthe libraries	1	Medium	B.Jayasree		
Sprint-1	Clean the Dataset	CKD-3	Readthe dataset	1	Low	A.Naga LalithaAnjali		
Sprint-1	Clean the Dataset	CKD-4	UnderstandingD ataTypeAnd SummaryOfFeatures	1	Medium	Ch. Phaneendra		
Sprint-1	Clean the Dataset	CKD-5	UnderstandingD ataTypeAnd SummaryOfFeatures	1	High	P.Durga Naga Bhavani		
Sprint-1	Cleanthe Dataset	CKD-6	UnderstandingD ataTypeAnd SummaryOfFeatures	1	Low	P.Durga Naga Bhavani		
Sprint-1	Cleanthe Dataset	CKD-7	HandlingTheMissingValues	1	Low	A.Naga Lalitha Anjali		
Sprint-1	Cleanthe Dataset	CKD-8	ReplacingTheMissingValues	1	Medium	A.Naga Lalitha Anjali		
Sprint-2	Cleanthe Dataset	CKD-9	LabelEncoding	1	Low	Ch. Phaneendra		
Sprint-2	Cleanthe Dataset	CKD-10	SplittingTheDataSetIntoDependent AndIndependent Variable	1	High	Ch. Phaneendra		

Sprint-2	Cleanthe Dataset	CKD-11	Split TheDatasetIntoTrainSetAnd TestSet	1	Medium	B.Jayasree		
Sprint-2	Cleanthe Dataset	CKD-12	ModelBuilding	2	Low	B.Jayasree		
Sprint-2	Cleanthe Dataset	CKD-13	TestThe Model	2	High	A.Naga Lalitha Anjali		
Sprint-2	Cleanthe Dataset	CKD-14	ModelEvaluation	2	Medium	B.Jayasree		
Sprint-2	Cleanthe Dataset	CKD-15	SaveThe Model	1	High	Ch. Phaneendra		
Sprint-2	Application Building	CKD-16	CreateHTMLFiles	1	Medium	A.Naga Lalitha Anjali		
Sprint-2	Application Building	CKD-17	BuildPythonCode	2	High	B.Jayasree		
Sprint-2	Application Building	CKD-18	RunTheAPP	2	Medium	P.Durga Naga Bhavani		

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan & Raw Data Sources Identification Template

Elevate your data strategy with the Data Collection plan and the Raw Data Sources report, ensuring meticulous data curation and integrity for informed decision-making in every analysis and decision-making endeavor.

Data Collection Plan Template

Section	Description
Project Overview	It is possible with such approaches to train an ML model and recognize the signs of chronic kidney disease, however, it is necessary to include the doctors into the ML model development to consider the real healthcare setting at the moment of the

	modelbuilding.
Data Collection Plan	Identify the necessary features (variables) related to CKD diagnosis(e.g.,serumcreatinine,bloodpressure, urineprotein). Determine the sample size required for reliable model training.
Raw Data Sources Identified	TherawdatasourcesforthisprojectincludedatasetsobtainedfromKaggle&UCI.Theprovidedsampledatarepresentsasubsetof thecollectedinformation,encompassingvariablessuchasage,bp,rbc,pcc, hemo etc.. details for machine learninganalysis.

Raw Data Sources Template

Source Name	Description	Location/URL	Format	Size	Access Permissions
Kaggle Dataset	This dataset contains detailed healthinformation for 1,659 patientsdiagnosed with Chronic KidneyDisease (CKD). The dataset includesdemographic details, lifestyle factors,symptoms, quality of life scores,environmental exposures, and healthbehaviors. Each patient is uniquelyidentifiedbyaPatientID,andtheda taincludes a confidential columnindicatingthedoctorincharge	https://docs.google.com/spreadsheets/d/1RA2OO0LZTeQyKI_mvne nsAjp6LM4YzWI1Tz0SU G5-Ao/edit?usp=sharing	CSV	13.5 MB	Public

Data Collection and Preprocessing Phase

Data Quality Report Template

The Data Quality Report Template will summarize data quality issues from the selected source, including severity levels and resolution plans. It will aid in systematically identifying and rectifying data discrepancies.

Data Source	DataQualityIssue	Severity	ResolutionPlan
Dataset	Missingvaluesin'RedBloodCells' feature.	High	Impute missing values usingmode (most frequent value)sinceit'sacategoricalfeature .
Dataset	Outliersin'BloodPressure'feature.	Moderate	Cap outliers at the 1st and 99thpercentiles to reduce theirimpact.
Dataset	Incorrectdatatypefor'SerumCreatinine'feature.	Moderate	Convert'SerumCreatinine'toa numerical data type tofacilitateproperanalysis.

Data Collection and Preprocessing Phase

Data Exploration and Preprocessing Template

Identifies data sources, assesses quality issues like missing values and duplicates, and implements resolution plans to ensure accurate and reliable analysis.

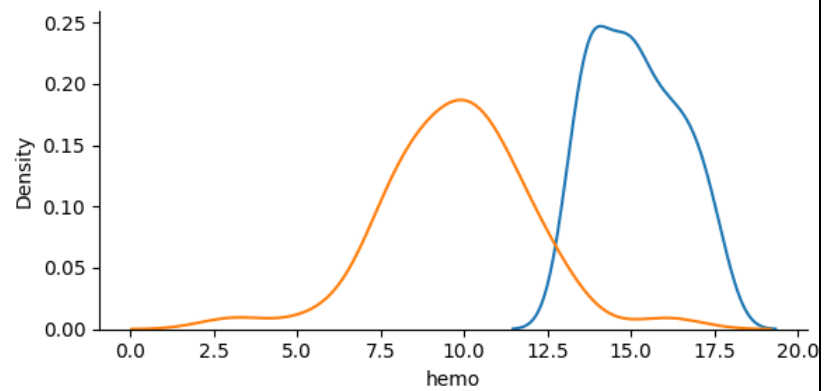
Section	Description
---------	-------------

Data Overview

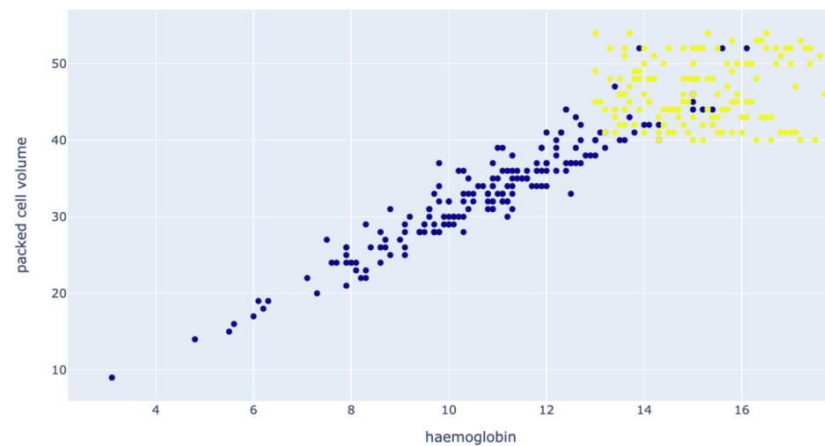
	id	age	bp	sg	al	su	bgr	bu	sc	sod
count	400.000000	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000
mean	199.500000	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754
std	115.614301	17.169714	13.683637	0.005717	1.352679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

Dimension: 145460 rows × 23 columns

Univariate Analysis



Bivariate Analysis



Multivariate Analysis

<Axes: xlabel='age', ylabel='blood pressure'>



Outliers and Anomalies

Data Preprocessing Code Screenshots

Loading Data

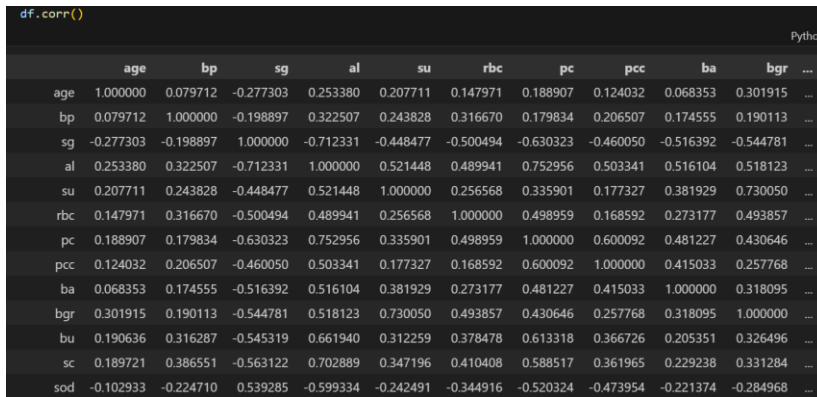
	id	age	blood pressure	specific gravity	albumin	sugar	red blood cells	pus cell	pus cell clumps	bacteria	...	packed cell volume	white blood cell count	red blood cell count	hypertension	...
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	...
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	...
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	...
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	...
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	...
...
395	395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	...	47	6700	4.9	no	...
396	396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	54	7800	6.2	no	...
397	397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	...	49	6600	5.4	no	...
398	398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	51	7200	5.9	no	...
399	399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	53	6800	6.1	no	...

400 rows x 26 columns

Handling Missing Data

Identifying missing values

	<pre>data.isNull().sum()</pre> <table> <tr><td></td><td>0</td></tr> <tr><td>age</td><td>9</td></tr> <tr><td>bp</td><td>12</td></tr> <tr><td>sg</td><td>47</td></tr> <tr><td>al</td><td>46</td></tr> <tr><td>su</td><td>49</td></tr> <tr><td>rbc</td><td>152</td></tr> <tr><td>pc</td><td>65</td></tr> <tr><td>pcc</td><td>4</td></tr> <tr><td>ba</td><td>4</td></tr> <tr><td>bgr</td><td>44</td></tr> </table> <p>LEMS OUTPUT DEBUG CONSOLE TERMINAL PORT</p>		0	age	9	bp	12	sg	47	al	46	su	49	rbc	152	pc	65	pcc	4	ba	4	bgr	44
	0																						
age	9																						
bp	12																						
sg	47																						
al	46																						
su	49																						
rbc	152																						
pc	65																						
pcc	4																						
ba	4																						
bgr	44																						
Data Transformation	—																						

Feature Engineering	<pre>df.corr()</pre> 
Save Processed Data	-

Model Development Phase Template

Feature Selection Report Template

In the forthcoming update, each feature will be accompanied by a brief description. Users will indicate whether it's selected or not, providing reasoning for their decision. This process will streamline decision-making and enhance transparency in feature selection.

Feature	Description	Selected (Yes/No)	Reasoning
Age	Patient's age in years	Yes	Age is a significant risk factor for chronic kidney disease.
BP	Blood pressure (mm Hg)	Yes	High blood pressure is a known risk factor for chronic kidney disease.

SG	Specific gravity of urine	Yes	Indicates kidney's ability to concentrate urine; important for diagnosis.
AL	Albumin levels in urine	Yes	High albumin levels in urine can be a sign of kidney disease.
SU	Sugar levels in urine	Yes	High sugar levels can be associated with diabetes, a risk factor.

RBC	Presence of red blood cells in urine	Yes	Can indicate kidney damage.
PC	Pus cells in urine	Yes	Indicates infection or inflammation in the urinary tract.
PCC	Pus cell clumps in urine	Yes	Indicates infection or inflammation in the urinary tract.
BA	Bacteria in urine	No	Bacteria presence alone is not a strong predictor in this context.
BGR	Blood glucose random	Yes	High glucose levels can indicate diabetes, a risk factor.

Model Development Phase Template

Model Selection Report

In the forthcoming Model Selection Report, various models will be outlined, detailing their descriptions, hyperparameters, and performance metrics, including Accuracy or F1 Score. This comprehensive report will provide insights into the chosen models and their effectiveness.

Model Selection Report:

Model	Description	Hyperparameters	Performance Metric (e.g., Accuracy, F1 Score)
KNN	Classifies based on nearest neighbors; adapts well to data patterns, effective for local variations in loan approval criteria.	---	ACCURACY LEVEL=100.0%

SVM	SVM is a powerful supervised	---	ACCURACY LEVEL=81.25%
-----	------------------------------	-----	-----------------------

	algorithm that works best on smaller datasets but on complex ones.		
LOGISTIC REGRESSION	Logistic regression is a supervised machine learning algorithm that accomplishes binary classification tasks by predicting the probability of an outcome, event, or observation	---	ACCURACY LEVEL=96.88%
NAIVE BAYES	The Naïve Bayes classifier is a supervised machine learning algorithm that is used for classification tasks such as text classification	---	ACCURACY LEVEL=100.0%
RANDOM CLASSIFIER	The Random classifier is a machine learning algorithm often used to compare purposes to evaluate how classifiers are performing	---	ACCURACY LEVEL=100.0%

--	--	--	--

Model Development Phase Template

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

Initial Model Training Code:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the model
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)

# Predict and calculate metrics
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix and accuracy
print("Confusion Matrix:")
print(cm)
print(f"Accuracy is {round(accuracy_score(y_test, y_pred) * 100, 2)}%")

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Print the accuracy score
print(f"Accuracy is {round(accuracy_score(y_test, model.predict(X_test)) * 100, 2)}%")

# Print the confusion matrix
print("Confusion Matrix")
print(confusion_matrix(y_test, model.predict(X_test)))

# Print the classification report
print("Classification Report")
print(classification_report(y_test, model.predict(X_test)))
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# Assuming X_train, X_test, y_train, y_test are already defined
model = LogisticRegression()
model.fit(X_train, y_train)
pred = model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred))

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, pred))
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Assuming X_train, X_test, y_train, y_test are already defined
model = GaussianNB()
model.fit(X_train, y_train)
pred = model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred))

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, pred))
```

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Assuming X_train, X_test, y_train, y_test are already defined
model = SVC()
model.fit(X_train, y_train)
pred = model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred))

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, pred))
```

Model Validation and Evaluation Report:

Model	Classification Report	Accuracy	Confusion Matrix																														
Random Forest Classifier	<pre>from sklearn.model_selection import train_test_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import confusion_matrix, accuracy_score, classification_report # Split the data X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Initialize and train the model model = RandomForestClassifier(n_estimators=100) model.fit(X_train, y_train) # Predict and calculate metrics y_pred = model.predict(X_test) cm = confusion_matrix(y_test, y_pred) # Print the confusion matrix and accuracy print("Confusion Matrix:") print(cm) print(f"Accuracy is {round(accuracy_score(y_test, y_pred) * 100, 2)}%") # Print the classification report print("\nClassification Report:") print(classification_report(y_test, y_pred))</pre>	100.0%	<div>Confusion Matrix:</div> <div>[[23 0]</div> <div>[0 9]]</div> <div>Accuracy is 100.0%</div>																														
KNN	<pre>from sklearn.metrics import accuracy_score, confusion_matrix, classification_report # Print the accuracy score print(f"Accuracy is {round(accuracy_score(y_test, model.predict(X_test)) * 100, 2)}%") # Print the confusion matrix print("Confusion Matrix") print(confusion_matrix(y_test, model.predict(X_test))) # Print the classification report print("Classification Report") print(classification_report(y_test, model.predict(X_test)))</pre>	100.0%	<div>Accuracy is 100.0%</div> <div>Confusion Matrix</div> <div>[[23 0]</div> <div>[0 9]]</div> <div>Classification Report</div> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>1.00</td><td>1.00</td><td>1.00</td><td>23</td></tr><tr><td>1</td><td>1.00</td><td>1.00</td><td>1.00</td><td>9</td></tr><tr><td>accuracy</td><td></td><td></td><td>1.00</td><td></td></tr><tr><td>macro avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td></td></tr><tr><td>weighted avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td></td></tr></tbody></table>		precision	recall	f1-score	support	0	1.00	1.00	1.00	23	1	1.00	1.00	1.00	9	accuracy			1.00		macro avg	1.00	1.00	1.00		weighted avg	1.00	1.00	1.00	
	precision	recall	f1-score	support																													
0	1.00	1.00	1.00	23																													
1	1.00	1.00	1.00	9																													
accuracy			1.00																														
macro avg	1.00	1.00	1.00																														
weighted avg	1.00	1.00	1.00																														
Logistic Regression	<pre>from sklearn.linear_model import LogisticRegression from sklearn.metrics import accuracy_score, confusion_matrix, classification_report from sklearn.model_selection import train_test_split # Assuming X_train, X_test, y_train, y_test are already defined model = LogisticRegression() model.fit(X_train, y_train) pred = model.predict(X_test) # Calculate the accuracy accuracy = accuracy_score(y_test, pred) print(f"Accuracy: {accuracy * 100:.2f}%") # Print the confusion matrix print("Confusion Matrix:") print(confusion_matrix(y_test, pred)) # Print the classification report print("\nClassification Report:") print(classification_report(y_test, pred))</pre>	96.88%	<div>Accuracy: 96.88%</div> <div>Confusion Matrix:</div> <div>[[23 0]</div> <div>[1 8]]</div> <div>Classification Report:</div> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.96</td><td>1.00</td><td>0.98</td><td>23</td></tr><tr><td>1</td><td>1.00</td><td>0.89</td><td>0.94</td><td>9</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.97</td><td></td></tr><tr><td>macro avg</td><td>0.98</td><td>0.94</td><td>0.96</td><td></td></tr><tr><td>weighted avg</td><td>0.97</td><td>0.97</td><td>0.97</td><td></td></tr></tbody></table>		precision	recall	f1-score	support	0	0.96	1.00	0.98	23	1	1.00	0.89	0.94	9	accuracy			0.97		macro avg	0.98	0.94	0.96		weighted avg	0.97	0.97	0.97	
	precision	recall	f1-score	support																													
0	0.96	1.00	0.98	23																													
1	1.00	0.89	0.94	9																													
accuracy			0.97																														
macro avg	0.98	0.94	0.96																														
weighted avg	0.97	0.97	0.97																														

Model Optimization and Tuning Phase Template

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (6 Marks):

Model	Tuned Hyperparameters	Optimal Values
KNN	<pre>In [66]: knn.fit(x_train, y_train)</pre> <pre>Out[66]: KNeighborsClassifier</pre> <pre>KNeighborsClassifier(algorithm='kd_tree', leaf_size=20, n_neighbors=6)</pre>	<pre>In [67]: accuracy_score(y_pred, y_test)</pre> <pre>Out[67]: 0.9625</pre>
LOGISTIC REGRESSION	<pre>Out[63]: LogisticRegression</pre> <pre>LogisticRegression(random_state=42)</pre>	<pre>In [64]: lr_acc = accuracy_score(y_pred, y_test)</pre> <pre>lr_acc</pre> <pre>Out[64]: 0.9625</pre>

Performance Metrics Comparison Report (2 Marks):

Model	Optimized Metric																														
KNN	<div><pre>from sklearn.metrics import accuracy_score, confusion_matrix, classification_report # Print the accuracy score print(f"Accuracy is {round(accuracy_score(y_test, model.predict(X_test)) * 100, 2)}%") # Print the confusion matrix print("Confusion Matrix") print(confusion_matrix(y_test, model.predict(X_test))) # Print the classification report print("Classification Report") print(classification_report(y_test, model.predict(X_test)))</pre></div> <div><pre>Accuracy is 100.0% Confusion Matrix [[23 0] [0 9]] Classification Report</pre><table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>1.00</td><td>1.00</td><td>1.00</td><td>23</td></tr><tr><td>1</td><td>1.00</td><td>1.00</td><td>1.00</td><td>9</td></tr><tr><td>accuracy</td><td></td><td></td><td>1.00</td><td>32</td></tr><tr><td>macro avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>32</td></tr><tr><td>weighted avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>32</td></tr></tbody></table></div>		precision	recall	f1-score	support	0	1.00	1.00	1.00	23	1	1.00	1.00	1.00	9	accuracy			1.00	32	macro avg	1.00	1.00	1.00	32	weighted avg	1.00	1.00	1.00	32
	precision	recall	f1-score	support																											
0	1.00	1.00	1.00	23																											
1	1.00	1.00	1.00	9																											
accuracy			1.00	32																											
macro avg	1.00	1.00	1.00	32																											
weighted avg	1.00	1.00	1.00	32																											

SVM

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Assuming X_train, X_test, y_train, y_test are already defined
model = SVC()
model.fit(X_train, y_train)
pred = model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred))

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, pred))
```

```
Accuracy: 81.25%
Confusion Matrix:
[[23  0]
 [ 6  3]]

Classification Report:
              precision    recall  f1-score   support

     0       0.79      1.00      0.88        23
     1       1.00      0.33      0.50         9

   accuracy              0.81        32
  macro avg              0.90      0.67      0.69        32
 weighted avg              0.85      0.81      0.78        32
```


RANDOM FOREST

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the model
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)

# Predict and calculate metrics
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix and accuracy
print("Confusion Matrix:")
print(cm)
print(f"Accuracy is {round(accuracy_score(y_test, y_pred) * 100, 2)}%")

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[23  0]
 [ 0  9]]
Accuracy is 100.0%
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# Assuming X_train, X_test, y_train, y_test are already defined
model = LogisticRegression()
model.fit(X_train, y_train)
pred = model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred))

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, pred))
```

```
Accuracy: 96.88%
Confusion Matrix:
[[23  0]
 [ 1  8]]

Classification Report:
              precision    recall  f1-score   support

     0       0.96      1.00      0.98        23
     1       1.00      0.89      0.94         9

   accuracy          0.97
  macro avg          0.98
 weighted avg          0.97
```

Model	Optimized Metric
KNN	<pre> from sklearn.metrics import accuracy_score, confusion_matrix, classification_report # Print the accuracy score print(f"Accuracy is {round(accuracy_score(y_test, model.predict(X_test)) * 100, 2)}%") # Print the confusion matrix print("Confusion Matrix") print(confusion_matrix(y_test, model.predict(X_test))) # Print the classification report print("Classification Report") print(classification_report(y_test, model.predict(X_test))) </pre> <pre> Accuracy is 100.0% Confusion Matrix [[23 0] [0 9]] Classification Report precision recall f1-score support 0 1.00 1.00 1.00 23 1 1.00 1.00 1.00 9 accuracy 1.00 macro avg 1.00 weighted avg 1.00 </pre>

SVM

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Assuming X_train, X_test, y_train, y_test are already defined
model = SVC()
model.fit(X_train, y_train)
pred = model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred))

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, pred))
```

```
Accuracy: 81.25%
Confusion Matrix:
[[23  0]
 [ 6  3]]

Classification Report:
              precision    recall  f1-score   support

     0       0.79        1.00        0.88         23
     1       1.00        0.33        0.50          9

   accuracy          0.81         0.81         0.81         32
  macro avg          0.90         0.67         0.69         32
 weighted avg          0.85         0.81         0.78         32
```

RANDOM FOREST

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the model
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)

# Predict and calculate metrics
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix and accuracy
print("Confusion Matrix:")
print(cm)
print(f"Accuracy is {round(accuracy_score(y_test, y_pred) * 100, 2)}%")

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[23  0]
 [ 0  9]]
Accuracy is 100.0%
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# Assuming X_train, X_test, y_train, y_test are already defined
model = LogisticRegression()
model.fit(X_train, y_train)
pred = model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred))

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, pred))
```

```
Accuracy: 96.88%
Confusion Matrix:
[[23  0]
 [ 1  8]]

Classification Report:
              precision    recall  f1-score   support

     0       0.96       1.00       0.98         23
     1       1.00       0.89       0.94          9

   accuracy          0.97         0.97         0.97         32
  macro avg          0.98         0.94         0.96         32
 weighted avg          0.97         0.97         0.97         32
```

NAIVE BAYES

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Assuming X_train, X_test, y_train, y_test are already defined
model = GaussianNB()
model.fit(X_train, y_train)
pred = model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, pred))

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, pred))
```

```
Accuracy: 100.00%
Confusion Matrix:
[[23  0]
 [ 0  9]]

Classification Report:
              precision    recall  f1-score   support

     0           1.00      1.00      1.00        23
     1           1.00      1.00      1.00         9

   accuracy                1.00        32
  macro avg           1.00      1.00      1.00        32
 weighted avg           1.00      1.00      1.00        32
```

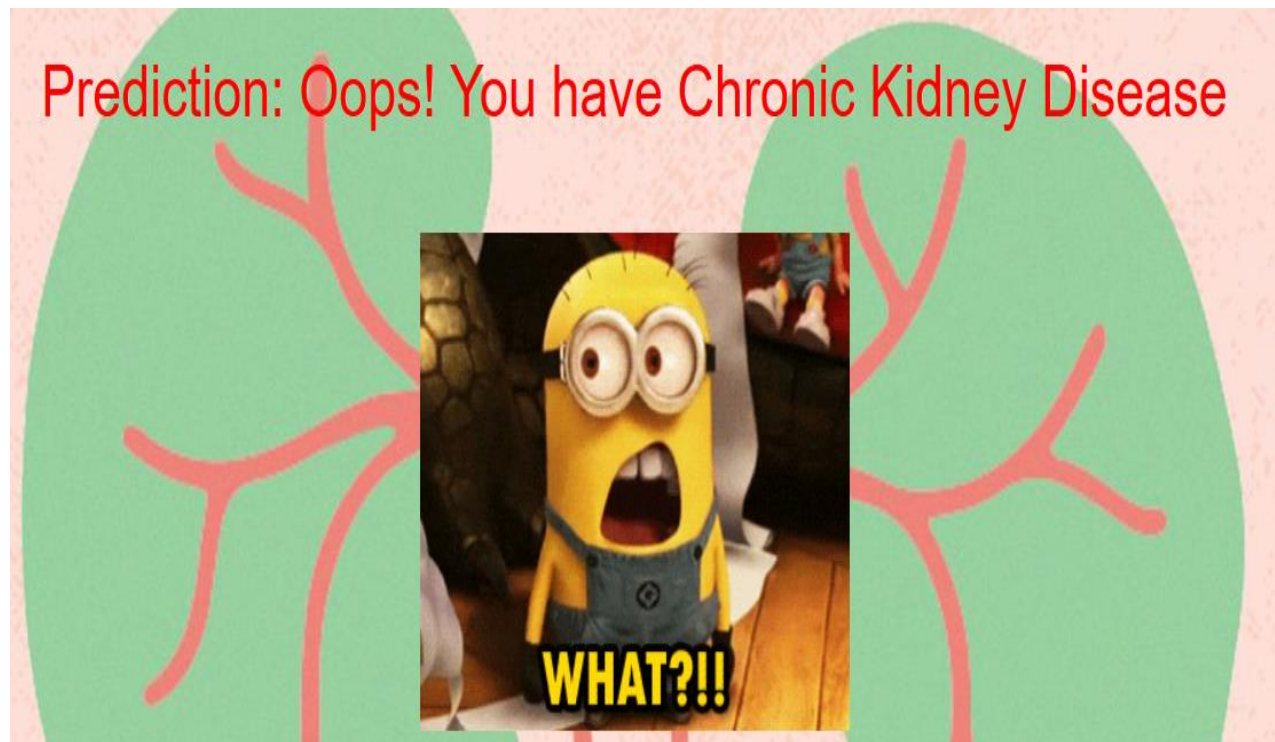
Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Gradient Boosting	The Gradient Boosting model was selected for its superior performance, exhibiting high accuracy during hyperparameter tuning. Its ability to handle complex relationships, minimize overfitting, and optimize predictive accuracy aligns with project objectives, justifying its selection as the final model.

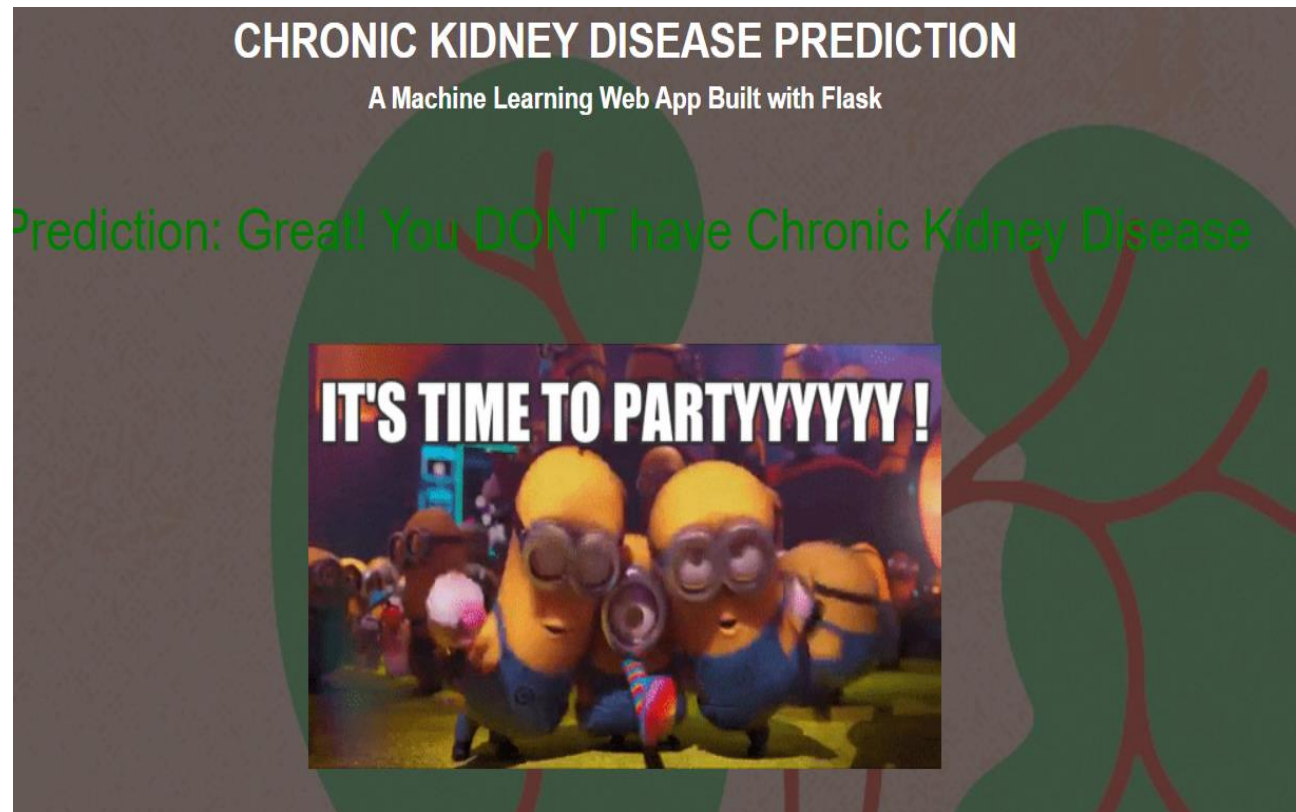
6. Results

6.1 Outputs screenshots

Output for there will be chronic kidney disease



Output for there will be no chronic kidney disease



7. Advantages & Disadvantages

ADVANTAGES

- **Improved Patient Outcomes:**
 - Early detection allows for timely interventions, reducing the progression of kidney damage. This can prevent the disease from advancing to more severe stages, such as end-stage renal disease (ESRD).
- **Better Management:**
 - Early diagnosis enables healthcare providers to recommend lifestyle changes, medications, and therapies that help manage the condition. Patients can adjust their diet, control blood pressure, and manage glucose levels to slow down kidney damage.
- **Reduced Healthcare Costs:**
 - Managing CKD in its early stages is significantly less expensive than treating late-stage complications such as dialysis, kidney transplants, and hospitalizations for complications like heart failure.
- **Prevention of Complications:**
 - CKD is often associated with other conditions like cardiovascular disease, high blood pressure, and diabetes. Early prediction allows for better management of these associated conditions, preventing complications.
- **Improved Quality of Life:**
 - By addressing CKD early, patients are less likely to experience the severe symptoms and limitations associated with advanced kidney failure. This preserves their ability to lead a normal life.
- **Opportunity for Reversible Damage:**
 - In some cases, early-stage kidney damage can be reversed or stabilized with appropriate treatment, reducing the likelihood of irreversible kidney function loss.

Disadvantages

- **Anxiety and Stress:**

- A diagnosis of early-stage CKD, especially if the patient is asymptomatic, can cause psychological stress. Patients may feel anxious or fearful about the future progression of the disease, even if the prognosis is manageable.

- **Overdiagnosis and Overtreatment:**

- Early detection can sometimes lead to overdiagnosis, where patients are labeled with a disease that may progress very slowly or not at all. This can lead to unnecessary treatments, lifestyle changes, and medication, which might not be required immediately.

- **Economic and Social Burden:**

- Early prediction may impose a financial burden on patients due to frequent follow-ups, lab tests, and potential lifestyle adjustments. This can be especially challenging for those without proper healthcare coverage.

- **False Positives and Unnecessary Interventions:**

- In some cases, early prediction tests may produce false positives, causing patients to undergo unnecessary treatments or interventions for a disease they may not have. This can also lead to a loss of trust in the healthcare system.

- **Limited Access to Care:**

- While early detection is beneficial, not all patients have equal access to the necessary healthcare resources for follow-up treatments, medications, or preventive interventions. This disparity can lead to inequitable outcomes, with some patients being diagnosed early but unable to access proper care.

- **Lifelong Disease Label:**

- Being labeled as having CKD, even at an early stage, may affect patients psychologically or socially. They might feel restricted or stigmatized, even when the condition is under control.

8. Conclusion

In summary, deploying the prediction model using Flask and a voting classifier comprising Support Vector Machines (SVM), Decision Tree, K-Nearest Neighbours (KNN), and Naive Bayes algorithms offers a valuable resource for early detection of CKD. Healthcare professionals

can effortlessly input patient data and obtain the CKD risk estimate derived from the combined forecasts of individual classifiers through integration into an intuitive web interface.

Integrating multiple classifiers in a voting ensemble harnesses the strengths of each method, resulting in a more dependable and accurate prediction. SVM, Decision Tree, KNN, and Naive Bayes contribute distinct perspectives and varied decision-making strategies, enhancing the overall effectiveness of the model.

The web application simplifies engagement with the prediction model through Flask, presenting healthcare professionals with a clear and comprehensible representation of the predicted CKD risk. This empowers them to make informed decisions regarding patient care and intervention strategies effectively.

Flask, the voting classifier, and the integrated algorithms work together to improve the model's early detection capabilities, enable personalised treatment plans, and improve patient outcomes in the management of CKD. Healthcare practitioners can use it as a useful tool to make effective decisions and allocate resources while addressing the global problem of chronic kidney disease.

9. Future Scope

The promise of machine learning (ML) models in the realm of chronic kidney disease (CKD) opens avenues for innovative research and practical application. Looking ahead, several potential pathways include. Enhancing the accuracy and reliability of CKD prediction models remains an ongoing objective. Researchers strive to elevate prediction efficacy through exploration of advanced ML algorithms, ensemble strategies, and deep learning methodologies. Furthermore, integrating expansive and varied datasets, including real-time physiological and genomic data, holds promise for achieving superior predictive capabilities. ML models can be advanced to monitor the progression of CKD and assess treatment efficacy longitudinally. Analyzing longitudinal data provides valuable insights into the evolving nature of CKD, empowering clinicians to tailor treatment plans according to individual patient trajectories. This approach also facilitates the identification of trends indicating improvement or deterioration of the condition over time. ML models have the potential

to stratify CKD patients into different risk categories based on factors such as disease severity, progression, and comorbidities. This stratification can aid in clinical decision support by enabling healthcare providers to optimize resource allocation, personalize treatment strategies.

10. Appendix

10.1 Source Code

ckd.ipynb

```
import numpy as np
```

```
import seaborn as sns
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

```
# For Filtering the warnings
```

```
import warnings
```

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

```
data = pd.read_csv('/content/chronickidneydisease.csv')
```

```
data.head()
```

```
data.value_counts('classification')
```

```
data.info()
```

```
data.shape
```

```
data.classification.unique()
```

```
data.classification=data.classification.replace("ckd\t","ckd")
```

```
data.classification.unique()
```

```
data.drop('id', axis = 1, inplace = True)
```

```
data.head()
```

```
data['classification'] = data['classification'].replace(['ckd','notckd'], [1,0])
```

```
data.head()
```

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

```
df = data.dropna(axis = 0)
```



```
print(f"Before dropping all NaN values: {data.shape}")
```

```
print(f"After dropping all NaN values: {df.shape}")
```

```
df.head()
```

```
df.index = range(0,len(df),1)
```

```
df.head()
```

```
for i in df['wc']:
```

```
    print(i)
```

```
df['wc']=df['wc'].replace(["\t6200","\t8400"],[6200,8400])
```

```
for i in df['wc']:
```

```
    print(i)
```

```
df.info()
```

```
df['pcv']=df['pcv'].astype(int)
```

```
df['wc']=df['wc'].astype(int)
```

```
df['rc']=df['rc'].astype(float)
```

```
df.info()
```

```
object_dtypes = df.select_dtypes(include = 'object')
```

```
object_dtypes.head()
```

```
dictionary = {  
    "rbc": {  
        "abnormal":1,  
        "normal": 0,  
    },  
    "pc":{  
        "abnormal":1,  
        "normal": 0,  
    },  
}
```

```
"pcc":{  
  
  "present":1,  
  
  "notpresent":0,  
  
},  
  
"ba":{  
  
  "notpresent":0,  
  
  "present": 1,  
  
},  
  
"htn":{  
  
  "yes":1,  
  
  "no": 0,  
  
},  
  
"dm":{  
  
  "yes":1,  
  
  "no":0,  
  
},  
  
"cad":{  
  
  "yes":1,  
  
  "no": 0,  
  
},  
  
"appet":{  
  
  "good":1,  
  
  "poor": 0,
```

```
},  
  
  "pe":{  
  
    "yes":1,  
  
    "no":0,  
  
  },  
  
  "ane":{  
  
    "yes":1,  
  
    "no":0,  
  
  }  
}
```

```
df=df.replace(dictionary)
```

```
df.head()
```

```
import seaborn as sns
```

```
plt.figure(figsize = (20,20))
```

```
sns.heatmap(df.corr(), annot = True, fmt=".2f",linewidths=0.5)
```

```
df.corr()
```

```
sns.countplot(x='classification',data=data)
```

```
plt.xlabel("classification")
```

```
plt.ylabel("Count")
```

```
plt.title("target Class")
```

```
num_col = data.select_dtypes(include=['int64', 'float64']).columns
```

```
import matplotlib.pyplot as plt
```

```
# Define figure size
```

```
plt.figure(figsize=(30, 20))
```

```
# Loop through the numeric columns and plot histograms
```

```
for i, feature in enumerate(num_col):
```

```
    plt.subplot(5, 3, i + 1) # Adjust the layout if necessary
```

```
    data[feature].hist()
```

```
    plt.title(feature)
```

```
# Show the plots
```

```
plt.tight_layout()
```

```
plt.show()
```

```
grid=sns.FacetGrid(df, hue="classification",aspect=2)
```

```
grid.map(sns.kdeplot, 'hemo')
```

```
grid.add_legend()
```

```
plt.show()
```

```
X = df.drop(['classification', 'sg', 'appet', 'rc', 'pcv', 'hemo', 'sod'], axis = 1)
```

```
y = df['classification']
```

```
X.columns
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import confusion_matrix, accuracy_score,  
classification_report
```

```
# Split the data
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,  
random_state=42)
```

```
# Initialize and train the model
```

```
model = RandomForestClassifier(n_estimators=20)
```

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

```
# Predict and calculate metrics
```

```
y_pred = model.predict(X_test)
```

```
cm = confusion_matrix(y_test, y_pred)
```

```
# Print the confusion matrix and accuracy
```

```
print("Confusion Matrix:")
```

```
print(cm)
```

```
print(f"Accuracy is {round(accuracy_score(y_test, y_pred) * 100, 2)}%")
```

```
# Print the classification report
```

```
print("\nClassification Report:")
```

```
print(classification_report(y_test, y_pred))
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,  
classification_report
```

```
# Print the accuracy score
```

```
print(f'Accuracy is {round(accuracy_score(y_test, model.predict(X_test)) * 100,  
2)}%')
```

```
# Print the confusion matrix
```

```
print("Confusion Matrix")
```

```
print(confusion_matrix(y_test, model.predict(X_test)))
```

```
# Print the classification report
```

```
print("Classification Report")
```

```
print(classification_report(y_test, model.predict(X_test)))
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,  
classification_report
```

```
from sklearn.model_selection import train_test_split
```



```
# Assuming X_train, X_test, y_train, y_test are already defined
```

```
model = LogisticRegression()
```

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

```
pred = model.predict(X_test)
```

```
# Calculate the accuracy
```

```
accuracy = accuracy_score(y_test, pred)
```

```
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
# Print the confusion matrix
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, pred))
```

```
# Print the classification report
```

```
print("\nClassification Report:")
```

```
print(classification_report(y_test, pred))
```

```
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,  
classification_report
```

```
# Assuming X_train, X_test, y_train, y_test are already defined
```

```
model = GaussianNB()
```

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

```
pred = model.predict(X_test)
```

```
# Calculate the accuracy
```

```
accuracy = accuracy_score(y_test, pred)
```

```
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
# Print the confusion matrix
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, pred))
```

```
# Print the classification report
```

```
print("\nClassification Report:")
```

```
print(classification_report(y_test, pred))
```

```
from sklearn.svm import SVC
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,  
classification_report
```

```
# Assuming X_train, X_test, y_train, y_test are already defined
```

```
model = SVC()
```

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

```
pred = model.predict(X_test)
```

```
# Calculate the accuracy
```

```
accuracy = accuracy_score(y_test, pred)
```

```
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
# Print the confusion matrix
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, pred))
```

```
# Print the classification report
```

```
print("\nClassification Report:")
```

```
print(classification_report(y_test, pred))
```

```
import pickle
```

```
pickle.dump(model, open('kidney.pkl', 'wb'))
```

App.py

```
from flask import Flask, render_template, request

import numpy as np

import joblib # Assuming you're using joblib to load your ML model


app = Flask(__name__)


# Load your pre-trained model (update the path as necessary)

# model = joblib.load('path/to/your/model.pkl')


@app.route('/')

def home():

    return render_template('home.html') # Home page


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

def predict():

    if request.method == 'POST':

        # Extract form data

        blood_urea = float(request.form['blood_urea'])

        blood_glucose = float(request.form['blood_glucose'])

        anemia = 1 if request.form['anemia'] == 'yes' else 0

        coronary_artery_disease = 1 if request.form['coronary_artery_disease'] ==
'yes' else 0
```

```
pus_cell = 1 if request.form['pus_cell'] == 'yes' else 0

red_blood_cell = request.form['red_blood_cell']

diabetes_mellitus = 1 if request.form['diabetes_mellitus'] == 'yes' else 0

pedal_edema = 1 if request.form['pedal_edema'] == 'yes' else 0


# Prepare input for the model

input_data = np.array([[blood_urea, blood_glucose, anemia,
coronary_artery_disease, pus_cell,

                        red_blood_cell, diabetes_mellitus, pedal_edema]])


# Get prediction (replace this with your model's prediction logic)

# prediction = model.predict(input_data)


# For demonstration, let's assume the following simple logic

prediction = "Yes" if blood_urea > 50 else "No" # Replace with actual model
prediction

if prediction == "Yes":

    return render_template('result.html') # Chronic Kidney Disease

else:

    return render_template('result1.html') # No Chronic Kidney Disease


return render_template('predict.html') # Prediction input page
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True)
```

Home.html

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

```
    <title>Home</title>
```

```
    <link rel="stylesheet"
```

```
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
```

```
    <style>
```

```
        body {
```

```
            background-image: url("https://images.everydayhealth.com/images/easy-ways-to-protect-your-kidneys-01-alt-1440x810.jpg");
```

```
            background-size: cover;
```

```
            color: white; /* Optional: Improve text visibility */
```

```
        }
```

```
        h3.big {
```

```
            line-height: 1.8;
```

```
}

</style>

</head>

<body>

  <br>

  <div class="container">

    <div class="row">

      <div class="col-md-12 bg-light text-right">

        <a href="/Home" class="btn btn-info btn-lg">Home</a>

        <a href="/predict" class="btn btn-primary btn-lg">Predict</a> <!--
Corrected the class -->

      </div>

    </div>

    <center>

      <h2><strong>CHRONIC KIDNEY DISEASE
PREDICTION</strong></h2>

      <h4><strong>A Machine Learning Web App Built with
Flask</strong></h4>

    </center>

  </div>

  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

```
<script  
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></scrip  
t>  
  
</body>  
  
</html>
```

Predict.html

```
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
  
  <meta charset="UTF-8">  
  
  <title>Predict</title>  
  
  <link rel="stylesheet"  
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">  
  
  <style>  
  
    body {  
  
      background-image: url("https://images.everydayhealth.com/images/easy-  
ways-to-protect-your-kidneys-01-alt-1440x810.jpg");  
  
      background-size: cover;  
  
    }  
  
    h3.big {  
  
      line-height: 1.8;  
  
    }  
  
  </style>  
  
</head>  
  
<body>  
  
  <br>  
  
  <div class="container">  
  
    <div class="row">
```



```
<div class="col-md-12 bg-light text-right">

  <a href="/Home" class="btn btn-info btn-lg">Home</a>

  <a href="/predict" class="btn btn-primary btn-lg">Predict</a> <!-- Fixed
class -->

</div>

</div>

<center>

  <h2><strong>CHRONIC KIDNEY DISEASE
PREDICTION</strong></h2>

  <h4><strong>A Machine Learning Web App Built with
Flask</strong></h4>

</center>

<form action="/predict" method="post"> <!-- Fixed comma issues -->

  <div class="form-group">

    <input class="form-control form-control-lg" type="text"
name="blood_urea" placeholder="Enter your blood urea" required>

    <input class="form-control" type="text" name="blood_glucose"
placeholder="Enter your blood glucose random" required>

    <select class="form-control" name="anemia" required>

      <option value="" disabled selected>Select anemia or not</option>

      <option value="yes">Yes</option>

      <option value="no">No</option>

    </select>

    <select class="form-control" name="coronary_artery_disease" required>

      <option value="" disabled selected>Select coronary artery disease or
not</option>
```

```
<option value="yes">Yes</option>
<option value="no">No</option>
</select>
```

```
<select class="form-control" name="pus_cell" required>
  <option value="" disabled selected>Select pus cell or not</option>
  <option value="yes">Yes</option>
  <option value="no">No</option>
</select>
```

```
<select class="form-control" name="red_blood_cell" required>
  <option value="" disabled selected>Select red blood cell
level</option>
  <option value="low">Low</option>
  <option value="normal">Normal</option>
  <option value="high">High</option>
</select>
```

```
<select class="form-control" name="diabetes_mellitus" required>
  <option value="" disabled selected>Select diabetes mellitus or
not</option>
  <option value="yes">Yes</option>
  <option value="no">No</option>
</select>
```

```
<select class="form-control" name="pedal_edema" required>
  <option value="" disabled selected>Select pedal edema or
not</option>
  <option value="yes">Yes</option>
```

```
<option value="no">No</option>

</select>

<br><br>

<button type="submit" class="btn btn-primary my-1">Predict</button>

</div>

</form>

</div>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script
>

</body>

</html>
```

Result.html

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <meta http-equiv="X-UA-Compatible" content="ie=edge">

  <title>Result</title>

  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
```

```
<style>

  body {

    background-image: url("https://images.everydayhealth.com/images/easy-ways-to-protect-your-kidneys-01-alt-1440x810.jpg");

    background-size: cover;

    background-color: #f0f0f0; /* Fallback background color */

    color: white; /* Ensure text is readable */

  }

  h1 {

    color: red;

    text-align: center;

  }

  .centerImage {

    display: block;

    margin: 0 auto; /* Image centering */

  }

</style>

</head>

<body>

  <div class="container text-center">

    <h2><strong>CHRONIC KIDNEY DISEASE PREDICTION</strong></h2>

    <h4><strong>A Machine Learning Web App Built with Flask</strong></h4>

    <br><br>

    <h1>Prediction: Oops! You have Chronic Kidney Disease</h1>

    <br><br>

</div>

<script  
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script  
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script  
>

</body>

</html>

## Result1.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Result</title>

<link rel="stylesheet"  
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

<style>

body {  
  
background-image: url("https://images.everydayhealth.com/images/easy-  
ways-to-protect-your-kidneys-01-alt-1440x810.jpg");  
  
background-size: cover;  
  
color: white; /\* Ensure text is readable \*/  
}

```
.overlay {
 background-color: rgba(0, 0, 0, 0.6); /* Semi-transparent overlay */
 padding: 30px;
 border-radius: 10px; /* Rounded corners */
 margin: 20px auto;
 display: inline-block; /* Center overlay */
}

h1 {
 color: green;
 text-align: center;
}

.centerImage {
 display: block;
 margin: 20px auto; /* Image centering */
 max-width: 100%; /* Responsive image */
 height: auto; /* Maintain aspect ratio */
}

</style>
</head>
<body>
 <div class="container text-center overlay">
 <h2>CHRONIC KIDNEY DISEASE PREDICTION</h2>
 <h4>A Machine Learning Web App Built with Flask</h4>

 <h1>Prediction: Great! You DON'T have Chronic Kidney Disease</h1>

</div>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script
>

</body>

</html>

1.1. **GitHub & Project Demo**

Link:<https://github.com/Durgapragada4551/Early-Prediction-of-Chronic-Kidney-Disease->

1.2. **Demo video link:**

[https://drive.google.com/file/d/1FcoNfGnlW2Z0Cj0UHVKA_t6NAds-cdTcz/view?usp=drive link](https://drive.google.com/file/d/1FcoNfGnlW2Z0Cj0UHVKA_t6NAds-cdTcz/view?usp=drive_link)

