A project report

On

Early Prediction of Chronic Kidney Disease

By

Team Id: SWTID1720164961

Contents

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 Code
    10.2. GitHub & Project Demo Link

1. Introduction

1.1 Project Overview

Chronic Kidney Disease (CKD) is a significant health concern characterized by a gradual decline in kidney function over time. Early detection and intervention are vital to slow disease progression, prevent complications, and improve patient outcomes. Machine learning (ML) offers a powerful approach to predict CKD in its early stages by analyzing large datasets and identifying patterns that may not be evident through traditional methods.

1.2 Objectives

Develop Accurate Prediction Model:
Create a machine learning model for early CKD prediction with high performance metrics.

Identify Key Predictive Features:
Determine significant risk factors contributing to early CKD onset.

Enhance Early Diagnosis:
Enable healthcare providers to identify at-risk patients for timely intervention.

Integrate with Clinical Systems:
Develop a user-friendly interface for seamless integration into healthcare systems.

Improve Patient Management:
Facilitate proactive monitoring and early treatment of at-risk patients.

Educate Healthcare Professionals:
Provide training for effective use of the prediction model.

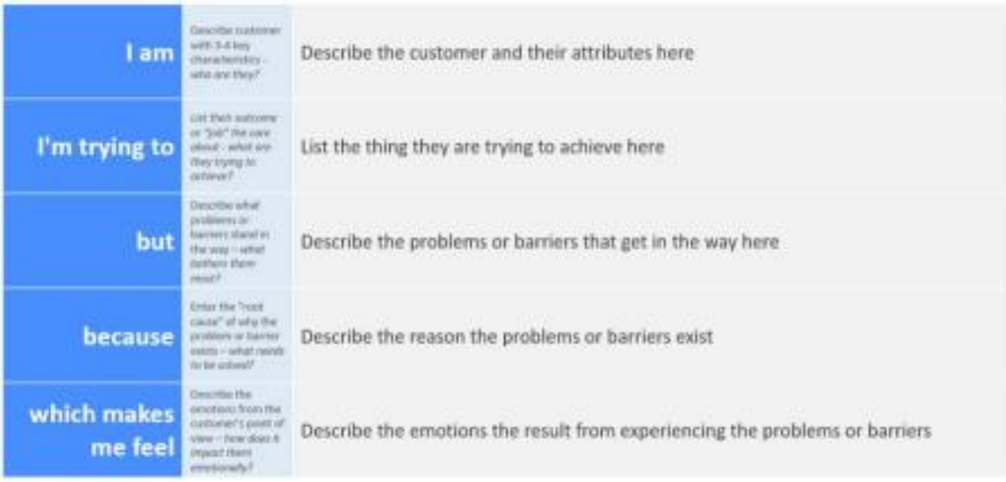
Promote Preventive Healthcare:
Encourage early detection and management to reduce CKD complications.

2. Project Initialization and Planning Phase

2.1 Define Problem Statements :

Develop a machine learning model to accurately detect Chronic Kidney Disease (CKD) in its early stages based on patient demographics, medical history, and clinical test results. The model should provide reliable predictions to assist healthcare providers in timely interventions and patient management, aiming to improve early diagnosis rates and reduce the progression of CKD to advanced stages.

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	I am a patient concerned about my health and well-being	I am trying to understand if I have Chronic Kidney Disease (CKD) at an early stage	But I often experience uncertainty and anxiety about my health condition	Because early detection is crucial for effective treatment and management.	Which makes me feel hopeful yet apprehensive about the future of my health.



Reference: https://miro.com/templates/customer-problem-statement/

Example:

2.2 Project Proposal (Proposed Solution)

The proposal report aims to Develop an AI-driven platform for early detection of chronic kidney disease (CKD) using machine learning algorithms. This platform aims to analyse patient data comprehensively to provide timely diagnoses, improving treatment outcomes and patient care.

Project Overview	
Objective	The primary objective is to develop and deploy a robust machine learning model that can accurately detect Chronic Kidney Disease (CKD) in its early stages using patient data, thereby facilitating early intervention and improving patient outcomes.
Scope	Developing a machine learning model for early detection of Chronic Kidney Disease (CKD) using patient data, from feature selection to deployment and compliance with healthcare regulations
Problem Statement	
Description	Creating a robust machine learning system for early detection of Chronic Kidney Disease (CKD) using patient data to improve healthcare outcomes.
Impact	Enhancing early detection of CKD to improve patient prognosis and healthcare efficiency.
Proposed Solution	
Approach	Utilizing machine learning algorithms to analyze patient data for early detection of CKD.
Key Features	1. Early Detection: Implementing a machine learning model to identify CKD in its early stages allows for timely medical intervention, potentially slowing disease progression. 2. Risk Factor Identification : Utilizing patient demographics, medical history, and clinical biomarkers helps identify individuals at higher risk for CKD, enabling proactive management strategies. 3. Personalized Treatment Plans: Tailoring treatment plans based on individual patient data and disease progression patterns.

Resource Requirements

Resource Type	Description	Specification/Allocation
Hardware		

Computing Resources	CPU/GPU specifications, number of cores	2 x NVIDIA V100 GPUs
Memory	RAM specifications	16 GB
Storage	Disk space for data, models, and logs	1 TB SSD
Software		
Frameworks	Python frameworks	Flask
Libraries	Additional libraries	pandas, numpy
Development Environment	IDE, version control	Jupyter Notebook, Git
Data		
Data	Source, size, format	Kaggle dataset, 401images

2.3 Initial Project Planning

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
1	Data Collection and Preprocessing	US-01	Collect historical shipping data, Clean and preprocess data	8	High	Thrishal Vignesh		
2	Feature Engineering	US-02	Identify and create relevant features	5	High	Bala Chandra		
3	Model Development	US-03	Train initial machine learning models	8	High	Megha Syam		
4	Model Evaluation	US-04	Evaluate model	5	Medium	Praneeth		

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
			performance using cross-validation					
5	Model Improvement	US-05	Optimize model parameters and features	8	High	Megha Syam		
6	Model Deployment	US-06	Deploy the best-performing model for real-time predictions	8	High	Thrishal Vignesh		
7	Continuous Improvement	US-07	Set up monitoring and feedback loops for model updates	5	Medium	Praneeth		

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan & Raw Data Sources Identification

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

Section	Description

Project Overview	To minimize the impact of Chronic Kidney Disease (CKD), it is essential to employ a machine learning model for early detection, which can identify the disease at its earliest stages. Early detection facilitates timely medical intervention, significantly slowing disease progression and improving patient outcomes.
Data Collection Plan	1. Gather patient demographics, medical history, and clinical biomarkers from electronic health records and public health databases. 2. Ensure data quality through cleaning, normalization, and feature engineering while maintaining patient privacy and ethical compliance
Raw Data Sources Identified	1. *Electronic Health Records (EHRs): Comprehensive patient information including demographics, medical history, and clinical test results from Kaggle 2. *Laboratory Test Results: Detailed biomarker data such as serum creatinine, eGFR, and urine albumin-to-creatinine ratio

Raw Data Sources

Dataset 1	Description of the data in this source.	Link of Dataset 1	format	data	Access permissions
Kaggle dataset	This data comprises age bp RBC count WBC count and other major factors to detect Chronic Kidney Disease	https://www.kaggle.com/code/niteshyadav3103/chronic-kidney-disease-prediction-98-accuracy	csv	10 kb	public

3.2 Data Quality Report

The Data Quality Report 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	Data Quality Issue	Severity	Resolution Plan
Kaggle Dataset	Missing values in the columns: Rbc, pc, wc, rc, sod, pct, pv	Moderate	Used mean/median imputation

3.3 Data Exploration and Preprocessing

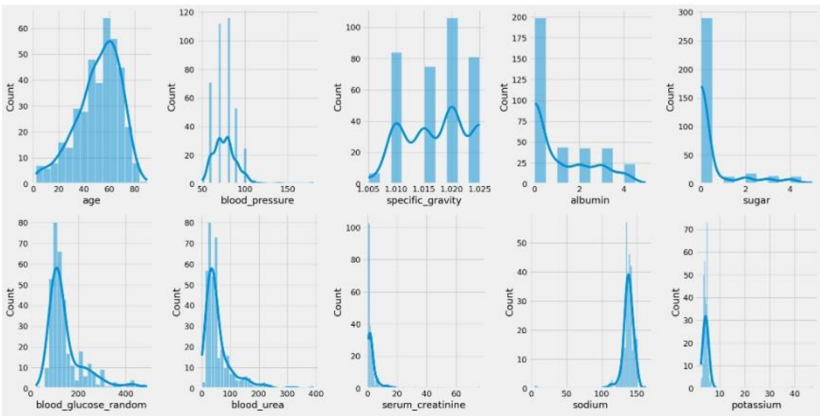
Identifies data sources, assesses quality issues like missing values and duplicates, and implements resolution plans to ensure accurate and reliable analysis.

<table>
<tr><th>Section</th><th>Description</th></tr>
<tr><td>Data Overview</td><td>
400 rows 26 columns

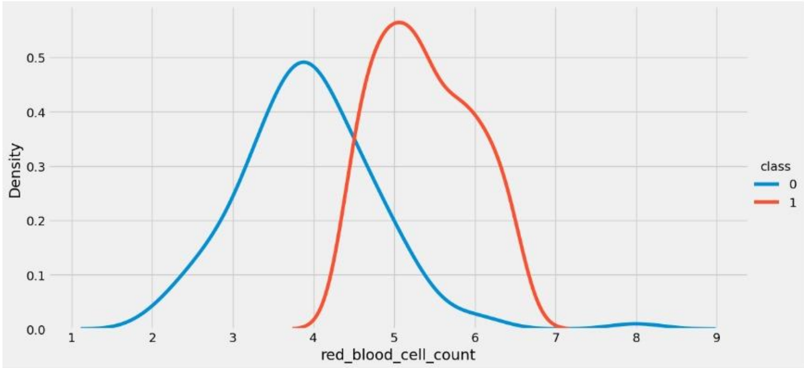
```
[5]: data.head()
```

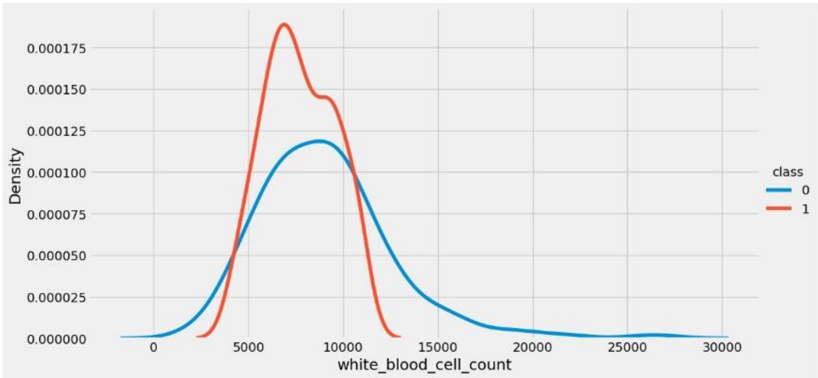
[5]:

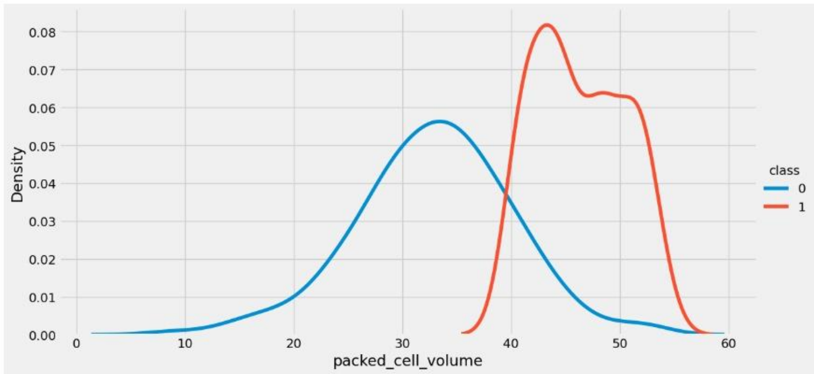
	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	cla
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	
2	2	62.0	80.0	1.010	2.0	3.0	normal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

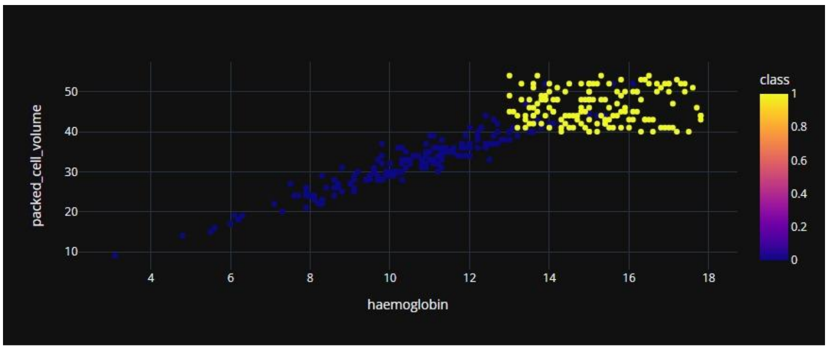
5 rows × 26 columns
</td></tr>
<tr><td>Univariate Analysis</td><td>


</td></tr>
</table>

<table>
<tr><td>Bivariate Analysis</td><td>






</td></tr>
<tr><td>Multivariate Analysis</td><td>


</td></tr>
<tr><td>Outliers and Anomalies</td><td>NA</td></tr>
</table>

<table>
<tr><td colspan="2">

Data Preprocessing Code Screenshots

</td></tr>
<tr><td>Loading Data</td><td>

```
[5]: data.head()
```

[5]:

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

5 rows × 26 columns

</td></tr>
<tr><td>Handling Missing Data</td><td>

```
[66]: # filling null values, we will use two methods, random sampling for higher null values and
      # mean/mode sampling for lower null values

      def random_value_imputation(feature):
          random_sample = data[feature].dropna().sample(data[feature].isna().sum())
          random_sample.index = data[data[feature].isnull()].index
          data.loc[data[feature].isnull(), feature] = random_sample

      def impute_mode(feature):
          mode = data[feature].mode()[0]
          data[feature] = data[feature].fillna(mode)

[67]: # filling num_cols null values using random sampling method

      for col in num_cols:
          random_value_imputation(col)
```

```
[68]: data[num_cols].isnull().sum()

[68]: age                       0
      blood_pressure            0
      specific_gravity          0
      albumin                   0
      sugar                     0
      blood_glucose_random      0
      blood_urea                0
      serum_creatinine          0
      sodium                    0
      potassium                 0
      haemoglobin               0
      packed_cell_volume        0
      white_blood_cell_count    0
      red_blood_cell_count      0
      dtype: int64

[69]: # filling "red_blood_cells" and "pus_cell" using random sampling method and rest of cat_cols using mode imputation

      random_value_imputation('red_blood_cells')
      random_value_imputation('pus_cell')

      for col in cat_cols:
          impute_mode(col)

[71]: data[cat_cols].isnull().sum()
```

</td></tr>
<tr><td>Data Transformation</td><td>NA</td></tr>
<tr><td>Feature Engineering</td><td>NA</td></tr>
<tr><td>Save Processed Data</td><td>NA</td></tr>
</table>

4.1 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.

4.2 Model Selection Report

Model	Description	Hyper parameters	Performance Metric
Rain forest	Ensemble of decision trees; robust, handles complex relationships, reduces overfitting, and provides feature importance for loan approval prediction.	-	**97.5%**
Decision tree	Simple tree structure; interpretable, captures non-linear relationships, suitable for initial insights into loan approval patterns.	-	**73.3%**
KNN	Classifies based on nearest neighbors; adapts well to data patterns, effective for local variations in loan approval criteria.	-	**96.6%**
Gradient boosting	XG Boost builds upon the principles of traditional gradient boosting while introducing several enhancements and optimizations that make it a go-to choice for predictive modeling tasks.	-	**97.5%**

4.3 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:

```
#70% train data, 30% test data
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 0)
```

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of knn

knn_acc = accuracy_score(y_test, knn.predict(X_test))

print(f"Training Accuracy of KNN is {accuracy_score(y_train, knn.predict(X_train))}")
print(f"Test Accuracy of KNN is {knn_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, knn.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, knn.predict(X_test))}")
```

```
# Decision tree classifier
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(X_test))

print(f"Training Accuracy of Decision Tree Classifier is {accuracy_score(y_train, dtc.predict(X_train))}")
print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, dtc.predict(X_test))}")
```

```
# hyper parameter tuning of decision tree

from sklearn.model_selection import GridSearchCV
grid_param = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 10],
    'splitter' : ['best', 'random'],
    'min_samples_leaf' : [1, 2, 3, 5, 7],
    'min_samples_split' : [1, 2, 3, 5, 7],
    'max_features' : ['auto', 'sqrt', 'log2']
}

grid_search_dtc = GridSearchCV(dtc, grid_param, cv = 5, n_jobs = -1, verbose = 1)
grid_search_dtc.fit(X_train, y_train)
```

```
# best estimator

dtc = grid_search_dtc.best_estimator_

# accuracy score, confusion matrix and classification report of decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(X_test))

print(f"Training Accuracy of Decision Tree Classifier is {accuracy_score(y_train, dtc.predict(X_train))}")
print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, dtc.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, dtc.predict(X_test))}")
```

```
# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_features = 'auto',
                                min_samples_leaf = 2, min_samples_split = 3, n_estimators = 130)
rd_clf.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of random forest

rd_clf_acc = accuracy_score(y_test, rd_clf.predict(X_test))

print(f"Training Accuracy of Random Forest Classifier is {accuracy_score(y_train, rd_clf.predict(X_train))}")
print(f"Test Accuracy of Random Forest Classifier is {rd_clf_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, rd_clf.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, rd_clf.predict(X_test))}")
```

```
#Ada boost Classifier
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(base_estimator = dtc)
ada.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of ada boost

ada_acc = accuracy_score(y_test, ada.predict(X_test))

print(f"Training Accuracy of Ada Boost Classifier is {accuracy_score(y_train, ada.predict(X_train))}")
print(f"Test Accuracy of Ada Boost Classifier is {ada_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, ada.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, ada.predict(X_test))}")
```

```
#XG Boost
from xgboost import XGBClassifier

xgb = XGBClassifier(objective = 'binary:logistic', learning_rate = 0.5, max_depth = 5, n_estimators = 150)
xgb.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of xgboost

xgb_acc = accuracy_score(y_test, xgb.predict(X_test))

print(f"Training Accuracy of XgBoost is {accuracy_score(y_train, xgb.predict(X_train))}")
print(f"Test Accuracy of XgBoost is {xgb_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, xgb.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, xgb.predict(X_test))}")
```

Model Validation and Evaluation Report:

Model	Classification Report	Accuracy	Confusion Matrix
Decision Tree	Classification Report :- precision recall f1-score support 0 0.96 1.00 0.98 72 1 1.00 0.94 0.97 48 accuracy 0.97 120 macro avg 0.98 0.97 0.97 120 weighted avg 0.98 0.97 0.97 120	97.5%	Confusion Matrix :- [[72 0] [3 45]]
KNN	Classification Report :- precision recall f1-score support 0 0.70 0.65 0.68 72 1 0.53 0.58 0.55 48 accuracy 0.62 120 macro avg 0.61 0.62 0.62 120 weighted avg 0.63 0.62 0.63 120	62.5%	Confusion Matrix :- [[47 25] [20 28]]
Random Forest	Classification Report :- precision recall f1-score support 0 0.96 1.00 0.98 72 1 1.00 0.94 0.97 48 accuracy 0.97 120 macro avg 0.98 0.97 0.97 120 weighted avg 0.98 0.97 0.97 120	97.5%	Confusion Matrix :- [[72 0] [3 45]]
ADA Boost	Classification Report :- precision recall f1-score support 0 0.96 1.00 0.98 72 1 1.00 0.94 0.97 48 accuracy 0.97 120 macro avg 0.98 0.97 0.97 120 weighted avg 0.98 0.97 0.97 120	97.5%	Confusion Matrix :- [[72 0] [3 45]]
XG Boost	Classification Report :- precision recall f1-score support 0 0.96 1.00 0.98 72 1 1.00 0.94 0.97 48 accuracy 0.97 120 macro avg 0.98 0.97 0.97 120 weighted avg 0.98 0.97 0.97 120	97.5%	Confusion Matrix :- [[72 0] [3 45]]

5. 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.

5.1 Hyperparameter Tuning Documentation :

<table>
<tr>
<td>KNN</td>
<td><pre>#KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

accuracy score, confusion matrix and classification report of knn

knn_acc = accuracy_score(y_test, knn.predict(X_test))

print(f"Training Accuracy of KNN is {accuracy_score(y_train, knn.predict(X_train))}")
print(f"Test Accuracy of KNN is {knn_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, knn.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, knn.predict(X_test))}")</pre></td>
<td><pre>Training Accuracy of KNN is 0.7928571428571428
Test Accuracy of KNN is 0.625</pre></td>
</tr>
<tr>
<td>XG Boost</td>
<td><pre>#XG Boost
from xgboost import XGBClassifier

xgb = XGBClassifier(objective = 'binary:logistic', learning_rate = 0.5, max_depth = 5, n_estimators = 150)
xgb.fit(X_train, y_train)

accuracy score, confusion matrix and classification report of xgboost

xgb_acc = accuracy_score(y_test, xgb.predict(X_test))

print(f"Training Accuracy of XgBoost is {accuracy_score(y_train, xgb.predict(X_train))}")
print(f"Test Accuracy of XgBoost is {xgb_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, xgb.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, xgb.predict(X_test))}")</pre></td>
<td><pre>Training Accuracy of XgBoost is 1.0
Test Accuracy of XgBoost is 0.975</pre></td>
</tr>
</table>

5.2 Performance Metrics Comparison Report

^

<table>
<tr>
<th>Model</th>
<th>Tuned Hyperparameters</th>
<th>Optimal Values</th>
</tr>
<tr>
<td>Decision Tree</td>
<td><pre># hyper parameter tuning of decision tree

from sklearn.model_selection import GridSearchCV
grid_param = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 10],
    'splitter' : ['best', 'random'],
    'min_samples_leaf' : [1, 2, 3, 5, 7],
    'min_samples_split' : [1, 2, 3, 5, 7],
    'max_features' : ['auto', 'sqrt', 'log2']
}

grid_search_dtc = GridSearchCV(dtc, grid_param, cv = 5, n_jobs = -1, verbose = 1)
grid_search_dtc.fit(X_train, y_train)</pre></td>
<td><pre># best parameters and best score

print(grid_search_dtc.best_params_)
print(grid_search_dtc.best_score_)

{'criterion': 'entropy', 'max_depth': 5, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 7, 'splitter': 'best'}
0.9928571428571429</pre></td>
</tr>
<tr>
<td>Random Forest</td>
<td><pre># Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_features = 'auto',
                                min_samples_leaf = 2, min_samples_split = 3, n_estimators = 130)
rd_clf.fit(X_train, y_train)

accuracy score, confusion matrix and classification report of random forest

rd_clf_acc = accuracy_score(y_test, rd_clf.predict(X_test))

print(f"Training Accuracy of Random Forest Classifier is {accuracy_score(y_train, rd_clf.predict(X_train))}")
print(f"Test Accuracy of Random Forest Classifier is {rd_clf_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, rd_clf.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, rd_clf.predict(X_test))}")</pre></td>
<td><pre>C:\Users\thris\anaconda3\Lib\site-packages\sklearn\ensemble_forest.py:424: FutureWarning:

`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'
his parameter as it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.

Training Accuracy of Random Forest Classifier is 1.0
Test Accuracy of Random Forest Classifier is 0.975</pre></td>
</tr>
</table>

<table>
<tr><td>Random Forest</td><td><pre>
Confusion Matrix :-
[[72 0]
 [3 45]]

Classification Report :-
              precision   recall f1-score   support

           0   0.96   1.00   0.98   72
           1   1.00   0.94   0.97   48

    accuracy   0.97   120
   macro avg   0.98   0.97   0.97   120
weighted avg   0.98   0.97   0.97   120
</pre></td></tr>
<tr><td>KNN</td><td><pre>
Confusion Matrix :-
[[47 25]
 [20 28]]

Classification Report :-
              precision   recall f1-score   support

           0   0.70   0.65   0.68   72
           1   0.53   0.58   0.55   48

    accuracy   0.62   120
   macro avg   0.61   0.62   0.62   120
weighted avg   0.63   0.62   0.63   120
</pre></td></tr>
<tr><td>XG Boost</td><td><pre>
Confusion Matrix :-
[[72 0]
 [3 45]]

Classification Report :-
              precision   recall f1-score   support

           0   0.96   1.00   0.98   72
           1   1.00   0.94   0.97   48

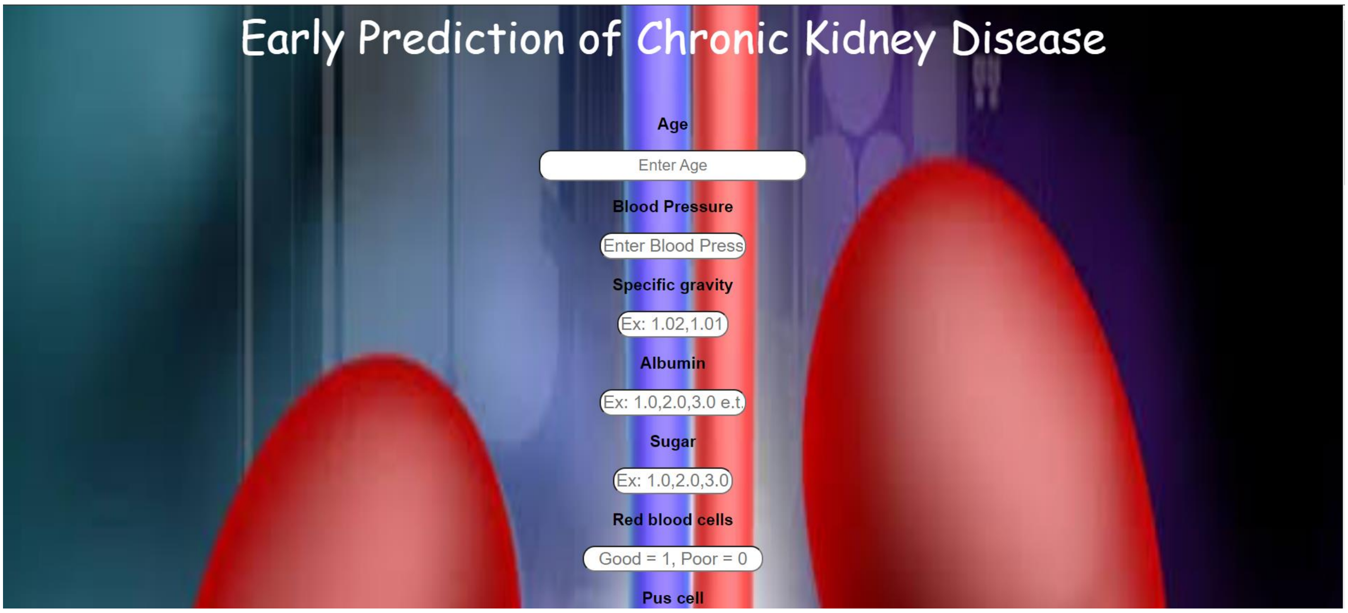
    accuracy   0.97   120
   macro avg   0.98   0.97   0.97   120
weighted avg   0.98   0.97   0.97   120
</pre></td></tr>
</table>

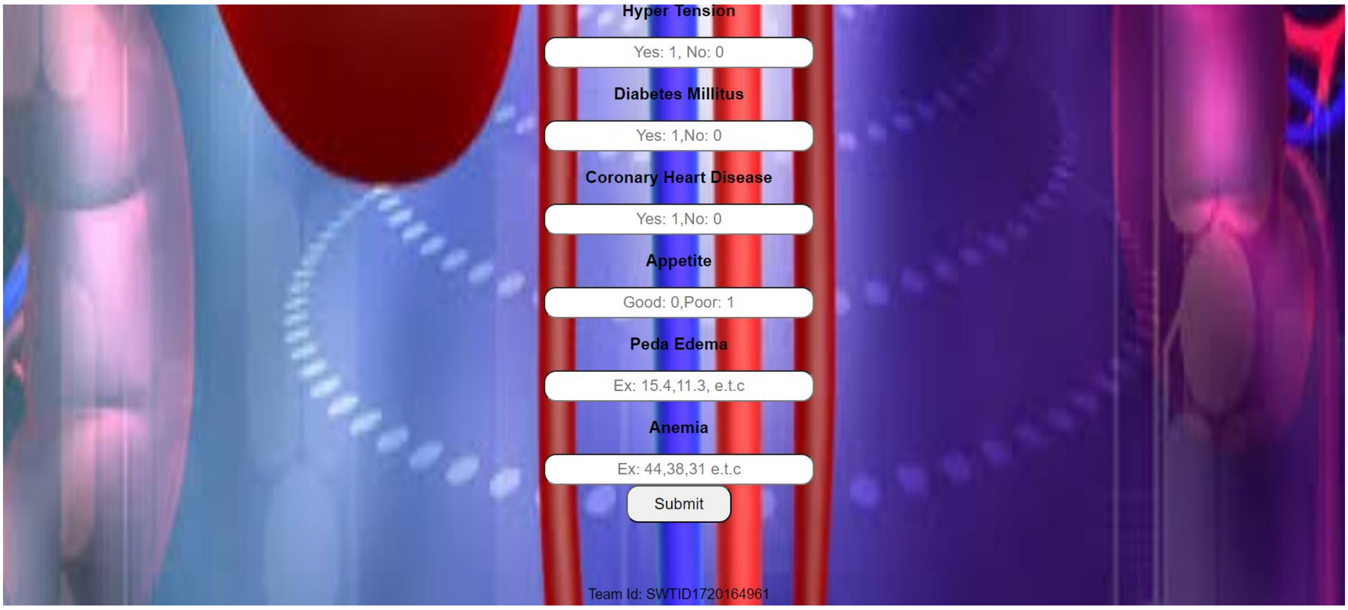
5.3 Final Model Selection Justification:

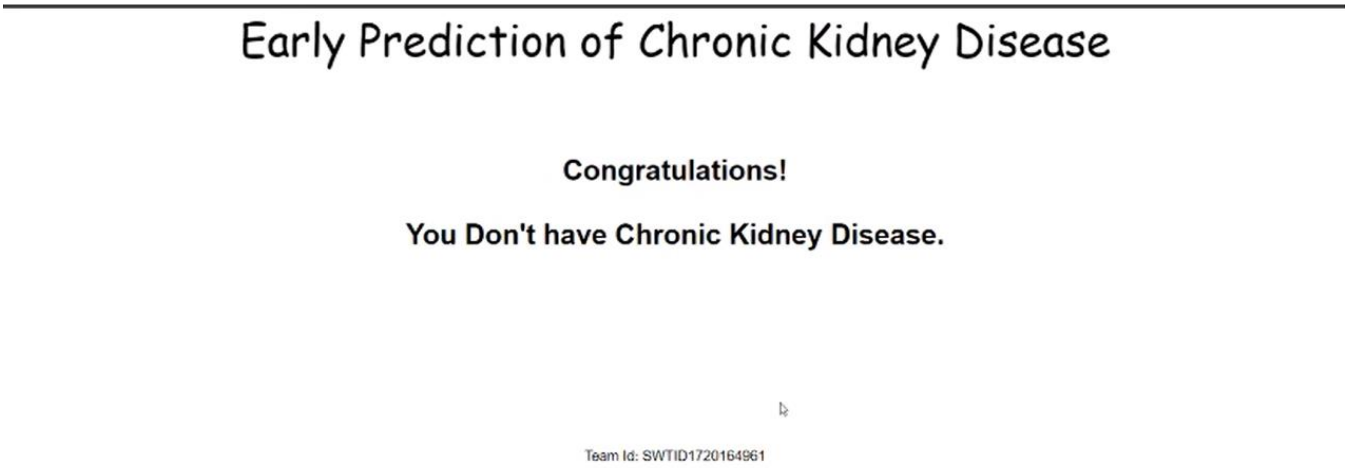
Final Model	Reasoning
Random Forest	The Random Forest was taken for the model due to its higher accuracy, during hyper parameter tuning, it has shown better performance with low risk of over fitting. Therefore, the random forest model was selected as final model for this project.

6. Results

6.1 Output Screenshots







7. Advantages and Disadvantages

Advantages

Early Detection:
Enables timely interventions and prevents severe complications.

Data-Driven Insights:
Facilitates personalized treatment and enhances decision-making.

Cost-Effective:
Reduces the need for expensive treatments and optimizes resource use.

Improved Patient Outcomes:

Enhances quality of life and increases survival rates.

Scalability:
Applicable to large populations and adaptable with new data.

Disadvantages

Data Quality and Availability:
Incomplete or missing data can reduce prediction accuracy.
Potential biases in data may affect model outcomes.

Implementation Challenges:
Integrating ML models into clinical practice requires significant effort and training.

Privacy Concerns:
Handling patient data involves privacy and security risks.

8. Conclusion

The early prediction of Chronic Kidney Disease (CKD) using machine learning holds significant promise for improving patient outcomes and enhancing healthcare delivery. By leveraging patient data and advanced algorithms, machine learning models can accurately identify individuals at risk of developing CKD, enabling timely and personalized interventions.

The key advantages include early detection, data-driven insights, cost-effectiveness, improved patient outcomes, and scalability. However, challenges such as data quality, implementation, and privacy concerns must be addressed to ensure the successful integration of these models into clinical practice.

Overall, the adoption of machine learning for CKD prediction represents a transformative approach in preventive healthcare, fostering proactive patient management and ultimately reducing the burden of chronic kidney disease.

9. Future Scope

The future scope of early CKD prediction using machine learning lies in personalized medicine. By refining algorithms to incorporate genetic, lifestyle, and environmental data, machine learning models can provide highly individualized risk assessments and treatment plans. This approach will enable more precise interventions, improving patient outcomes and reducing the overall impact of CKD. Advances in wearable technology and continuous monitoring can further enhance these models, offering real-time health insights and fostering proactive management of kidney health.

10. Appendix

10.1 Source Code

Model code:

```
# Importing Libraries:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pickle

# for displaying all feature from data:
pd.pandas.set_option('display.max_columns', None)

# Reading data:
data = pd.read_csv('c:/Users/thris/Downloads/chronickidneydisease.csv')

# Dropping unnecessary feature :
data = data.drop('id', axis=1)

# Naming categories
data.columns = ['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar',
'red_blood_cells', 'pus_cell',
              'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_urea',
'serum_creatinine', 'sodium',
              'potassium', 'haemoglobin', 'packed_cell_volume',
'white_blood_cell_count', 'red_blood_cell_count',
              'hypertension', 'diabetes_mellitus', 'coronary_artery_disease',
'appetite', 'peda_edema',
              'aanemia', 'class']

# splitting into numerical and categorical columns
cat_cols = [col for col in data.columns if data[col].dtype == 'object']
num_cols = [col for col in data.columns if data[col].dtype != 'object']

# converting object values to numeric values
data['packed_cell_volume'] = pd.to_numeric(data['packed_cell_volume'], errors='coerce')
data['white_blood_cell_count'] = pd.to_numeric(data['white_blood_cell_count'],
errors='coerce')
data['red_blood_cell_count'] = pd.to_numeric(data['red_blood_cell_count'],
errors='coerce')

# replacing object columns to numerical values
data['diabetes_mellitus'].replace(to_replace={'\tno':'no','\tyes':'yes',' yes':'yes'},
inplace=True)
data['coronary_artery_disease'].replace(to_replace='\tno', value='no', inplace=True)
data['class'].replace(to_replace={'ckd\t': 'ckd', 'notckd': 'not ckd'}, inplace=True)

# filling null values, using random sampling for higher null values and mean/mode
sampling for lower null values
```

```
def random_value_imputation(feature):
    random_sample = data[feature].dropna().sample(data[feature].isna().sum())
    random_sample.index = data[data[feature].isnull()].index
    data.loc[data[feature].isnull(), feature] = random_sample

def impute_mode(feature):
    mode = data[feature].mode()[0]
    data[feature] = data[feature].fillna(mode)

# filling num_cols null values using random sampling method
for col in num_cols:
    random_value_imputation(col)

# filling "red_blood_cells" and "pus_cell" using random sampling method and rest of
cat_cols using mode imputation
random_value_imputation('red_blood_cells')
random_value_imputation('pus_cell')

for col in cat_cols:
    impute_mode(col)

# Label encoding
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for col in cat_cols:
    data[col] = le.fit_transform(data[col])

ind_col = [col for col in data.columns if col != 'class']
dep_col = 'class'

X = data[ind_col]
y = data[dep_col]

# Train Test Split:
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=33)

# RandomForestClassifier:
from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion='entropy', max_depth=11, min_samples_leaf=2,
min_samples_split=3, n_estimators=130)
rd_clf.fit(X_train, y_train)
```

```
# Creating a pickle file for the classifier
filename = 'ckd.pkl'
try:
    with open(filename, 'wb') as file:
        pickle.dump(rd_clf, file)
    print(f"Successfully saved model to {filename}")
except Exception as e:
    print(f"Error saving model to {filename}: {e}")
```

App.py:

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


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

@app.route('/',methods=['GET'])
def home():
    return render_template('"C:/Users/thris/Downloads/index.html"')

@app.route("/predict", methods=['POST'])
def predict():
    if request.method == 'POST':
        age = float(request.form['age'])
        blood_pressure = float(request.form['blood_pressure'])
        specific_gravity = float(request.form['specific_gravity'])
        albumin = float(request.form['albumin'])
        sugar = float(request.form['sugar'])
        red_blood_cells = float(request.form['red_blood_cells'])
        pus_cell = float(request.form['pus_cell'])
        pus_cell_clumps = float(request.form['pus_cell_clumps'])
        bacteria = float(request.form['bacteria'])
        blood_glucose_random = float(request.form['blood_glucose_random'])
        blood_urea = float(request.form['blood_urea'])
        serum_creatinine = float(request.form['serum_creatinine'])
        sodium = float(request.form['sodium'])
        potassium = float(request.form['potassium'])
        haemoglobin = float(request.form['haemoglobin'])
        packed_cell_volume = float(request.form['packed_cell_volume'])
        white_blood_cell_count = float(request.form['white_blood_cell_count'])
        red_blood_cell_count = float(request.form['red_blood_cell_count'])
        hypertension = float(request.form['hypertension'])
        diabetes_mellitus = float(request.form['diabetes_mellitus'])
        coronary_artery_disease = float(request.form['coronary_artery_disease'])
        appetite = float(request.form['appetite'])
```

```
        peda_edema = float(request.form['peda_edema'])
        aanemia = float(request.form['aanemia'])

        values = np.array([[age, blood_pressure, specific_gravity, albumin, sugar,
       red_blood_cells, pus_cell, pus_cell_clumps, bacteria,
       blood_glucose_random, blood_urea, serum_creatinine, sodium,
       potassium, haemoglobin, packed_cell_volume,
       white_blood_cell_count, red_blood_cell_count, hypertension,
       diabetes_mellitus, coronary_artery_disease, appetite,
       peda_edema, aanemia]])
        prediction = model.predict(values)

        return render_template('"C:/Users/thris/Downloads/result.html"',
prediction=prediction)


if __name__ == "__main__":
    app.run(debug=True)
```

10.2 GitHub& Project Demo Link

GitHub: https://github.com/Thrishal18/Early-Predection-of-Chronic-Kidney-disease
Project Demo:
https://drive.google.com/file/d/1OCHcsJJGCOZ51EYahGTXXpUvhiz1D72b/view?usp=sharing