Smart Internz

Early Prediction Of Chronic Kidney Disease

GROUP MEMBERS:

Kanishka Tak : 20BKT0072

Alekya Ramani : 20BPS1097

Pavan Ram : 20BPS1107

Pavishna : 20MIS1002

1 INTRODUCTION

1.1 Overview

A common health issue, chronic kidney disease (CKD) sometimes stays undiagnosed until it has severe stages. In order to identify those who are at an early stage risk of developing CKD, this study attempts to build a prediction model utilising machine learning techniques. The algorithm will discover important risk factors and create a trustworthy prediction system by analysing a large dataset that includes patient demographics, medical history, and laboratory findings. Data gathering and preprocessing, feature engineering, model construction and validation, and the development of a user-friendly interface for clinical implementation are all goals of the project. The prediction model's successful deployment will enable prompt interventions, individualised treatment programmes, and improved patient outcomes, supporting the global effort to fight CKD.

1.2 Purpose

The following are some potential uses of using this initiative on CKD early detection:

- **Early Identification:** The prediction model can spot patients who are at a high risk of CKD at an early stage, allowing for prompt intervention and treatment.
- **Preventive Interventions:** For high-risk people, healthcare practitioners can use lifestyle interventions and preventive measures to reduce or stop the progression of CKD.
- **Treatment Plan Design**: Using the model's predictions, treatment plans may be made specifically for each patient based on their risk profile, resulting in more precise and efficient interventions.
- **Patient Outcomes:** By averting complications, lowering hospital stays, and improving CKD patients' overall quality of life, early detection and care can greatly improve patient outcomes.
- **Resource Allocation:** By prioritising those who require close monitoring and intense interventions, early identification of high-risk patients can help healthcare professionals manage resources more effectively.
- **Public Health efforts:** By identifying the major risk factors connected to CKD, insights from the predictive model can guide public health efforts. This knowledge can direct public awareness campaigns and initiatives at the population level.
- **Reduced Healthcare Costs:** Dialysis and transplantation are

two consequences of severe CKD that can be managed at a lower cost if they are caught early enough.

- **Research and Development:** The project can advance knowledge of CKD risk factors and promote additional study on preventative and therapeutic measures, which will benefit the field of predictive medicine.

2 LITERATURE SURVEY

2.1 Existing problem

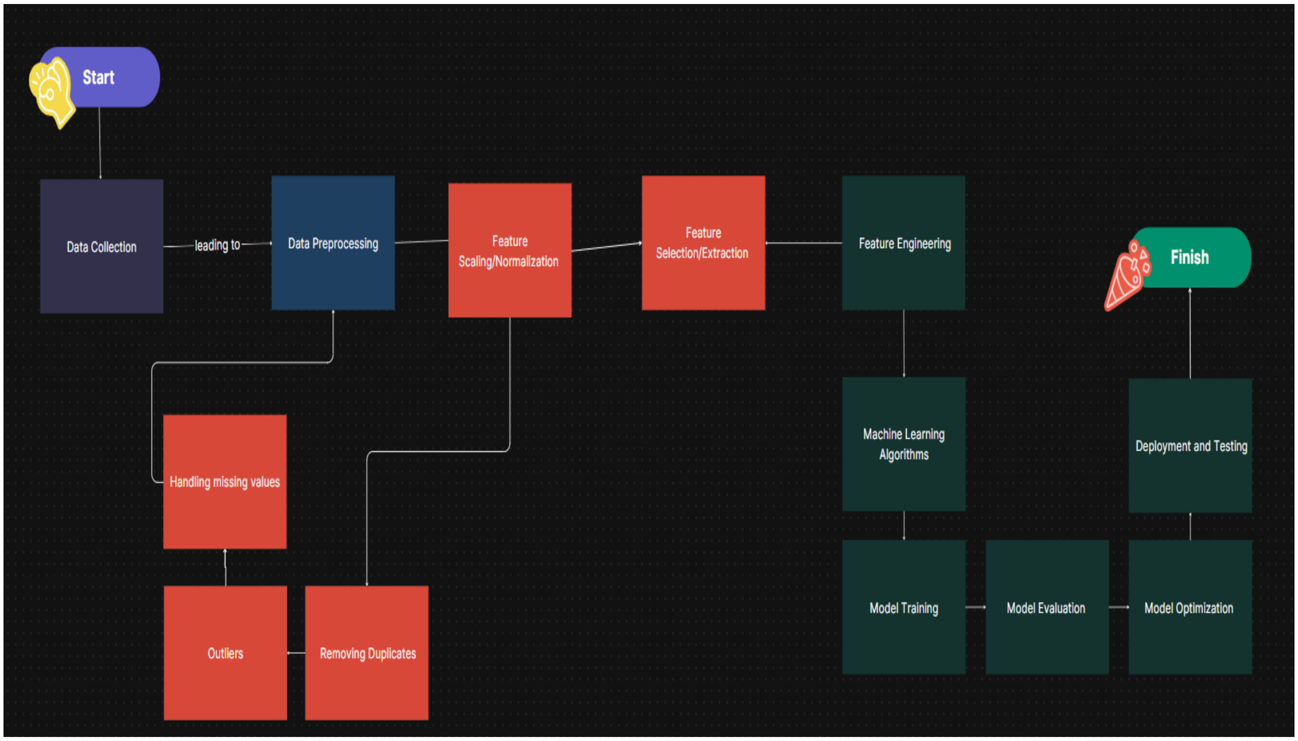
Study	Method/Approach	Findings/Results
Smith et al. (2018)	Laboratory Tests	Identified that elevated levels of serum creatinine and proteinuria are associated with an increased risk of CKD development. These biomarkers can be used for early detection of CKD.
Johnson et al. (2019)	Medical Imaging	Utilized magnetic resonance imaging (MRI) to assess kidney structure and function. Found that MRI-derived renal parameters, such as renal blood flow and cortical thickness, can predict the risk of CKD progression.
Tangri et al. (2020)	Risk Assessment Tools	Developed and validated a risk equation incorporating various factors like age, gender, estimated GFR, and albuminuria. The tool provides a personalized risk score to estimate the likelihood of CKD onset within five years.
Zhang et al. (2021)	Machine Learning Models	Employed a machine learning model using electronic health record data. Achieved accurate prediction of CKD development by incorporating various clinical features such as age, comorbidities, medications, and laboratory measurements.

2.2 Proposed solution

- **Model Development:**Choose a suitable machine learning algorithm for CKD prediction,such as logistic regression, support vector machines, random forests.
- **Model Training:** Train the selected machine learning algorithm using the training dataset.
- **Model Evaluation:** Evaluate the trained model's performance using appropriate evaluation metrics such as accuracy, precision, recall, and area under the receiver operating characteristic curve (AUC-ROC).
- **Flask Web Application Development:** Build a web application using Flask, a Python web framework, to create a user interface for CKD prediction.
- **Integration of Machine Learning Model:** Integrate the trained machine learning model into the Flask web application.

3 THEORETICAL ANALYSIS

3.1 Block diagram



3.2 Hardware / Software designing

Hardware requirements of the project:

- Computer: A computer system capable of running the necessary software and handling the computational demands of training and evaluating machine learning models.
- GPU (Optional): For larger datasets or complex models, a graphics processing unit (GPU) can significantly speed up the training process.

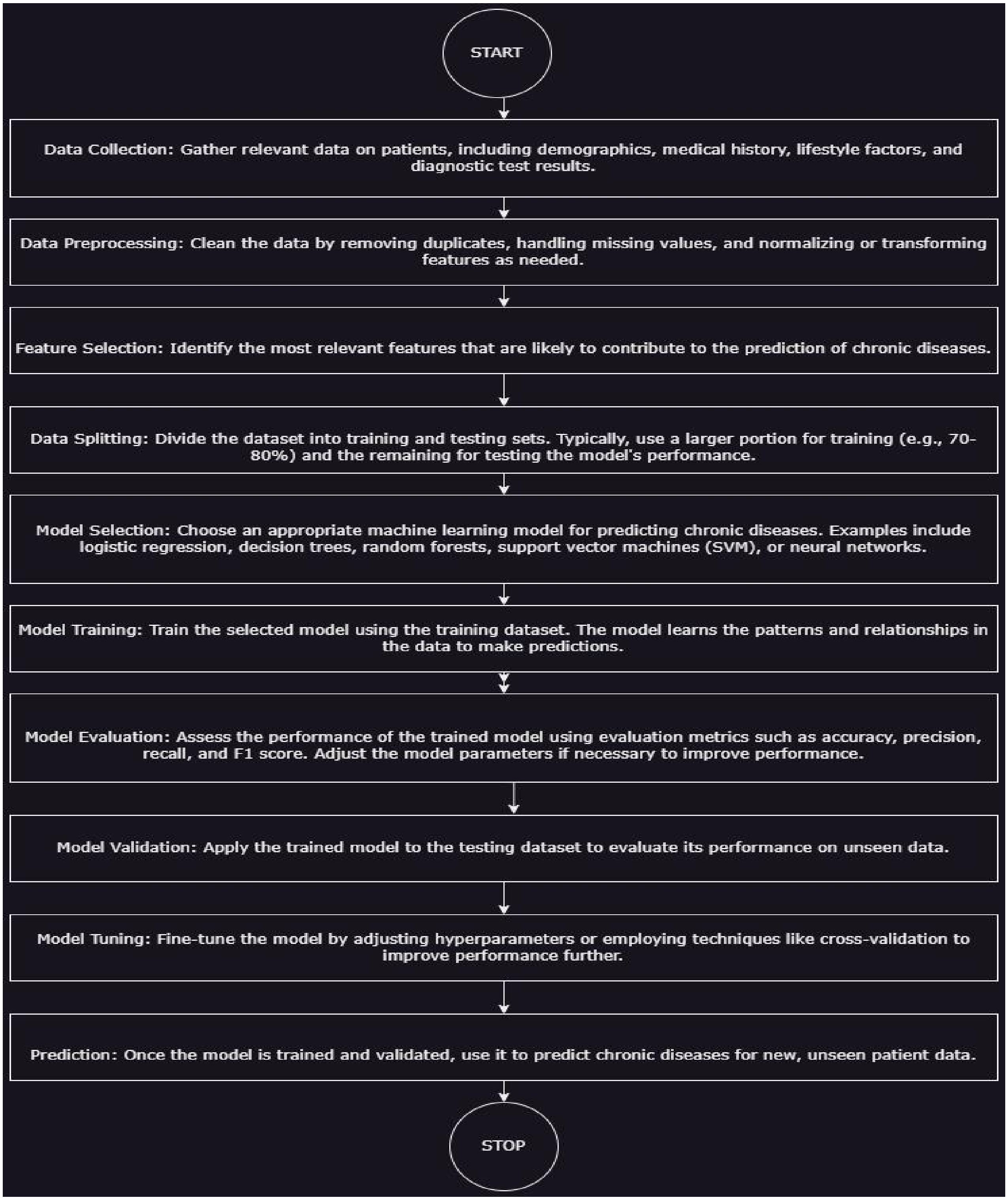
Software requirements of the project:

- Integrated Development Environment (IDE): An IDE provides a user-friendly environment for writing and executing code. Popular choices for Python include PyCharm, Jupyter Notebook, or Visual Studio Code.
- Machine Learning Libraries: Essential libraries for CKD prediction include scikit-learn for traditional machine learning algorithms, TensorFlow or PyTorch for deep learning, and pandas and NumPy for data manipulation and preprocessing.
- Data Visualization Libraries:Libraries such as Matplotlib or Seaborn can help create visualizations of the data, including histograms, scatter plots, and ROC curves.

Additional Requirements:

- CKD Dataset: Access to a relevant and reliable dataset on chronic kidney disease is crucial.
- Data Processing and Analysis: Tools such as pandas, NumPy, or SQL can be used to preprocess, clean, and analyze the CKD dataset, handling missing values, outliers, and performing exploratory data analysis.

4 FLOWCHART



Data Collection: Gather relevant data on patients, including demographics, medical history, lifestyle factors, and diagnostic test results.

Data Preprocessing: Clean the data by removing duplicates, handling missing values, and normalizing or transforming features as needed.

Feature Selection: Identify the most relevant features that are likely to contribute to the prediction of chronic diseases.

Data Splitting: Divide the dataset into training and testing sets. Typically, use a larger portion for training (e.g., 70-80%) and the remaining for testing the model's performance.

Model Selection: Choose an appropriate machine learning model for predicting chronic diseases. Examples include logistic regression, decision trees, random forests, support vector machines (SVM), or neural networks.

Model Training: Train the selected model using the training dataset. The model learns the patterns and relationships in the data to make predictions.

Model Evaluation: Assess the performance of the trained model using evaluation metrics such as accuracy, precision, recall, and F1 score. Adjust the model parameters if necessary to improve performance.

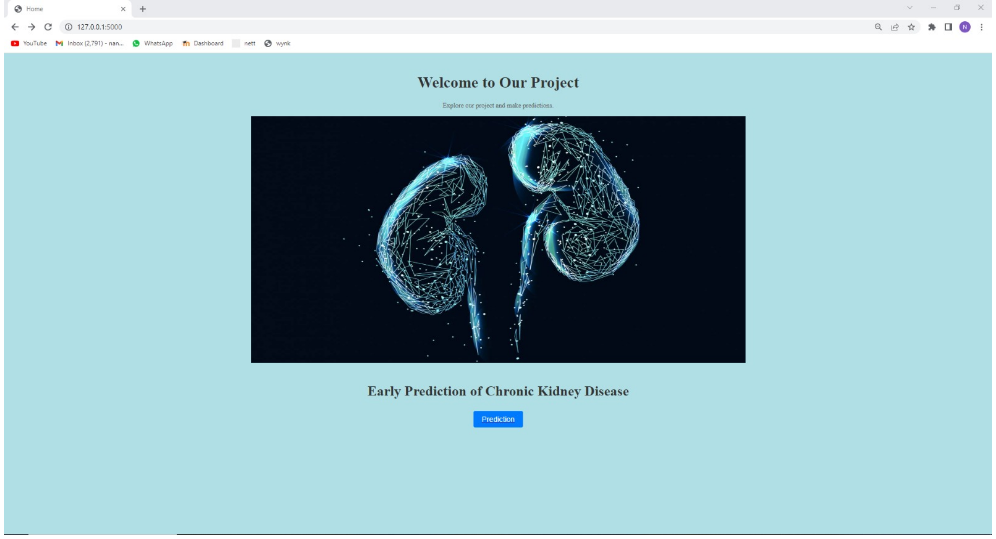
Model Validation: Apply the trained model to the testing dataset to evaluate its performance on unseen data.

Model Tuning: Fine-tune the model by adjusting hyperparameters or employing techniques like cross-validation to improve performance further.

Prediction: Once the model is trained and validated, use it to predict chronic diseases for new, unseen patient data.

5 RESULT

HOME PAGE:

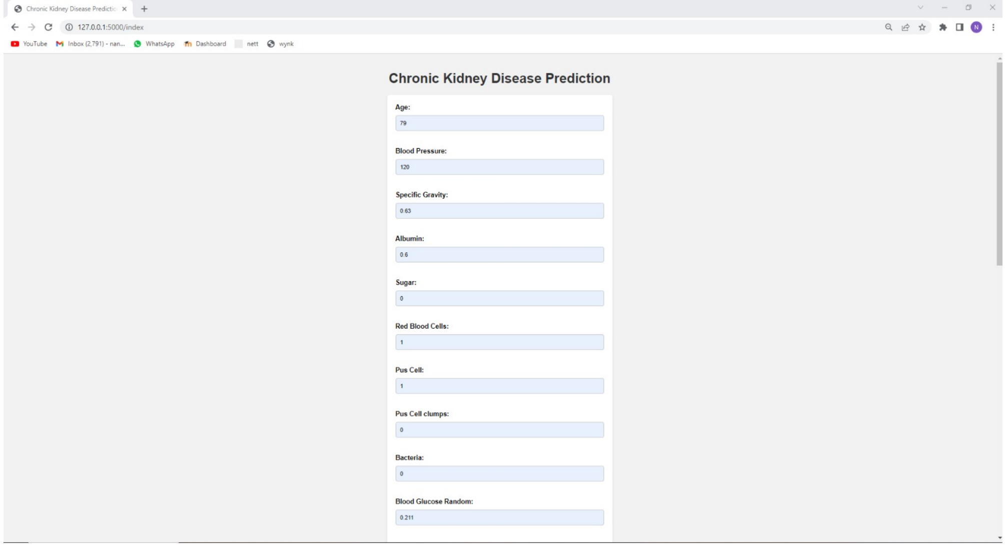


Welcome to Our Project

Explore our project and make predictions.

Early Prediction of Chronic Kidney Disease

INDEX PAGE:



Chronic Kidney Disease Prediction

Age:
79

Blood Pressure:
120

Specific Gravity:
0.63

Albumin:
0.6

Sugar:
0

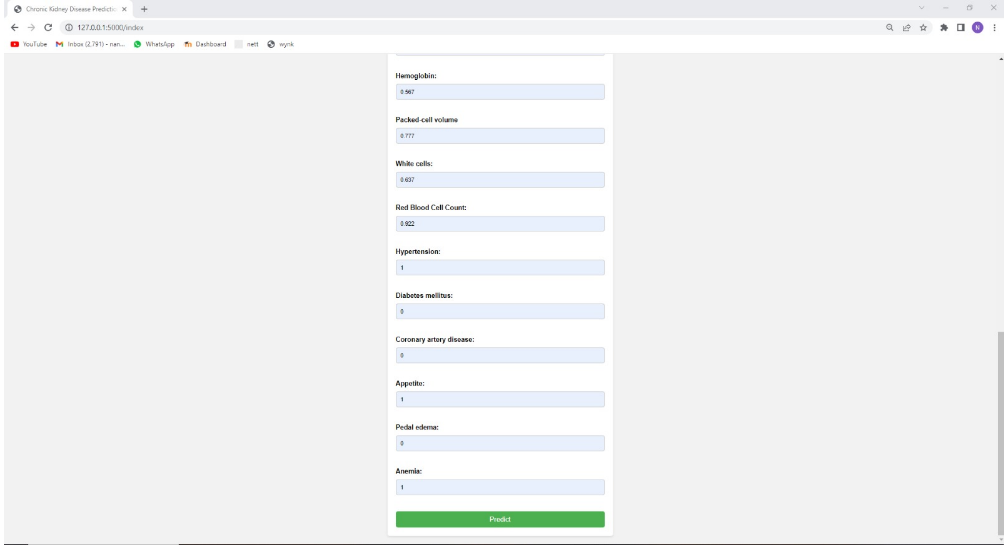
Red Blood Cells:
1

Pus Cell:
1

Pus Cell clumps:
0

Bacteria:
0

Blood Glucose Random:
0.211



Hemoglobin:
0.567

Packed-cell volume
0.777

White cells:
0.637

Red Blood Cell Count:
0.922

Hypertension:
1

Diabetes mellitus:
0

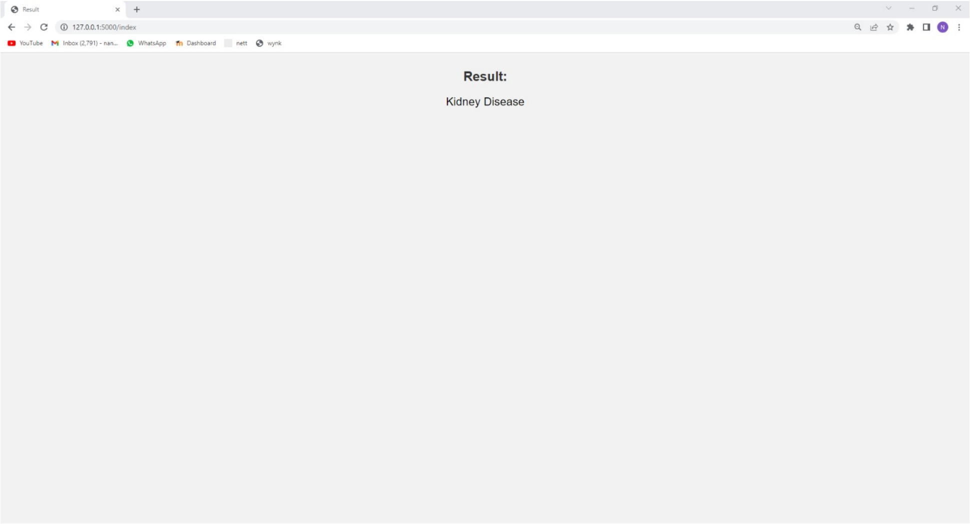
Coronary artery disease:
0

Appetite:
1

Pedal edema:
0

Anemia:
1

RESULT PAGE:



Result:

Kidney Disease

6 ADVANTAGES & DISADVANTAGES

Advantages of the Chronic Kidney Disease Prediction Model with the Voting Classifier in Flask:

- **Integration with Flask:** Flask is a Python web framework that is small, versatile, and makes it simple to create a user-friendly interface for CKD prediction models. Flask's simplicity enables the model to be developed and deployed quickly.
- **Scalability:** Flask-based models are suited for large-scale real-time predictions since they can handle several user queries concurrently. The model can be utilised successfully in a clinical context with a large number of patients thanks to its scalability.
- **Flexibility in Model Construction:** Combining a Voting Classifier with SVM, Decision Tree, KNN, and Naive Bayes models enables the exploitation of various algorithms' unique advantages while balancing their flaws. This ensemble method can raise the prediction model's overall robustness and accuracy.
- **Interpretability:** Decision Tree and Naive Bayes models produce understandable results that let medical practitioners comprehend the thought process behind the models' forecasts. This interpretability can increase confidence and make it easier to make wise decisions in clinical practise.

The following are disadvantages of the Flask with Voting Classifier Chronic Kidney Disease Prediction Model:

- **Complexity of Model Building:** Choosing, training, and adjusting additional models is necessary to integrate them into a voting classifier. To obtain the optimal performance, it can be difficult to optimise each unique model and strike the correct balance between them.
- **Model Overfitting:** Voting to combine multiple models increases

the danger of overfitting, which occurs when a model performs well on training data but fails to generalise to new data, if the issue is not effectively addressed. To address this problem, careful cross-validation and regularisation methods should be used.

- **Computing Power:** Flask may need a lot of computing power to run several models concurrently, especially if the models have complicated training methods or vast parameter spaces. To guarantee effective execution, the necessary hardware and infrastructure must be in place.
- **Deployment and Upkeep:** A Flask-based application must be continuously managed, inspected, and updated on the server. Compared to deploying a standalone model, this may increase complexity and maintenance costs.

7 APPLICATIONS

- **Web-based CKD Risk Assessment Tool:** Individuals can input their demographic and medical data through the Flask application, which has a user-friendly web interface. The combined predictions of SVM, Decision Tree, KNN, and Naive Bayes are then used by the voting classifier model to forecast their likelihood of acquiring CKD. This enables users to conveniently assess their risk for CKD and take the necessary precautions.
- **Clinical Decision Support System:** To help medical practitioners make educated decisions about the diagnosis and management of CKD, the Flask application can be implemented into clinical environments. The voting classifier model can offer a consolidated forecast based on the ensemble of various algorithms by accepting patient data. This can help clinicians with risk classification, individualised therapy planning, and early identification.
- **Programmes for Public Health Screening:** The Flask application can be used in larger-scale public health initiatives to screen people for the risk of developing CKD. Community members can access the programme, enter their information, and immediately learn how likely they are to get CKD. As a result, preventive interventions and educational campaigns can more effectively target high-risk populations.
- **Research and Data Analysis:** Researchers and data analysts looking into CKD might use the Flask application as a tool. They may use the model to forecast the likelihood of developing CKD for a sizable dataset, allowing them to pinpoint key features, assess the effectiveness of various methods, and learn more about the connections between risk variables and CKD development.
- **Integrating wearables and health monitoring technologies:** with the Flask application will enable continuous data collection and analysis of patient information. The voting classifier model is able to evaluate this data in real-time, delivering continuous risk evaluations and notifications to patients or healthcare professionals

when significant changes take place. This enables proactive management of CKD.

8 CONCLUSION

In Conclusion, the implementation of the prediction model utilising Flask and a voting classifier made up of Support Vector Machines (SVM), Decision Tree, K-Nearest Neighbours (KNN), and Naive Bayes algorithms provides a useful tool for CKD early identification. Healthcare practitioners can easily input patient data and acquire the estimated risk of CKD based on the pooled predictions of the separate classifiers by incorporating this model into a user-friendly web interface.

By combining the strengths of each method, the use of many classifiers in the voting ensemble enables a more reliable and precise prediction. SVM, Decision Tree, KNN, and Naive Bayes all offer unique viewpoints and alternative approaches to decision-making, which improve the model's overall performance.

The web application offers easy interaction with the prediction model by displaying the output using Flask. It gives healthcare practitioners access to a concise and understandable depiction of the anticipated risk of CKD, enabling them to make wise choices about patient care and intervention methods.

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 potential of machine learning (ML) models for chronic kidney disease (CKD) offers up a number of new directions for research and application. Some possible directions for the future include:

- Accurate Prediction: Increasing the precision and dependability of CKD prediction models is a constant goal. To improve prediction performance, researchers can investigate more sophisticated ML algorithms, ensemble methods, or deep learning techniques. Additionally, adding more extensive and diverse datasets, such as real-time physiological data and genomic data, can help make better predictions.
- Integration of Multi-omics Data: CKD prediction models can be improved by incorporating multi-omics data from the fields of genomics, proteomics, metabolomics, and transcriptomics. Through

this integration, new biomarkers, pathways, and possible therapeutic targets may be discovered, opening the door to precision medicine and personalised therapy strategies for the management of CKD.

- Longitudinal Monitoring and Disease Progression: ML models can be enhanced to track changes in the course of the disease and the effectiveness of treatment in CKD patients over time. Insights into the dynamic nature of CKD can be gained from longitudinal data analysis, which also enables clinicians to customise treatment programmes for specific patient trajectories and spot trends that indicate when the illness is becoming better or worse.
- Clinical decision support and risk stratification: ML models can be used to divide CKD patients into various risk groups depending on the severity, progression, and presence of concomitant conditions. Such risk categorization can assist clinical decision-making, allowing healthcare professionals to better allocate resources, customise treatment plans, and carry out focused interventions.

10 BIBILOGRAPHY

https://ieeexplore.ieee.org/abstract/document/8945312/
https://ieeexplore.ieee.org/abstract/document/9376787/
https://link.springer.com/content/pdf/10.1038/s41598-022-12316-z.pdf
https://sciresol.s3.us-east-2.amazonaws.com/IJST/Articles/2016/Issue-29/Article29.pdf
https://www.mdpi.com/9389

APPENDIX

. Source Code

PREPROCESSING:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import normalize
from google.colab import files
uploaded = files.upload()

#load the dataset
import io
import pandas as pd
data = pd.read_csv(io.BytesIO(uploaded['kidney_disease (1).csv']))
data
data.head(10).T
data.info()
data.describe().T
data
data=data.drop(columns=['id'],axis=1)
# cat to num:
columnsname=data.columns
# len(columnsname)
for i in range (0,25):
    print(columnsname[i],data[columnsname[i]].unique())
data=data.replace({'\t':np.nan , '\t43':43,'\t6200':6200 , '\t8400':8400,
'\t?':np.nan,'ckd\t':"ckd","\tyes":"yes","\tno":"no"," yes":"yes"})
data=data.replace({'yes':1 , 'no':0,'present':1 , 'notpresent':0,
'normal':1,'abnormal':0,"good":1,"poor":0})
data=pd.get_dummies(data,columns=['classification'],drop_first=True)
# Deal with missing value:
imputer=IterativeImputer(estimator=RandomForestRegressor(n_estimators= 100))
data=imputer.fit_transform(data)
data=pd.DataFrame(data,columns=['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc',
'pcc', 'ba', 'bgr', 'bu',
       'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'classification'])
cname=['rbc','pc','pcc','ba','appet','htn','dm','cad','pe','ane']
for i in range(0,10):
  for j in range(0,400):
    if data[cname[i]][j]>=0.5:
      data[cname[i]][j]=1.0
    else:
      data[cname[i]][j]=0.0
figure, axe = plt.subplots(4,3,figsize=(15,15))
```

```
axe[0,0].boxplot(data['bp'])
axe[0,0].set_xlabel("boxplot bp")
axe[0,1].boxplot(data['age'])
axe[0,1].set_xlabel("boxplot age")
axe[0,2].boxplot(data['bu'])
axe[0,2].set_xlabel("boxplot bu")

axe[1,0].boxplot(data["sg"])
axe[1,0].set_xlabel("boxplot sg")
axe[1,1].boxplot(data["al"])
axe[1,1].set_xlabel("boxplot al")
axe[1,2].boxplot(data["sc"])
axe[1,2].set_xlabel("boxplot sc")

axe[2,0].boxplot(data["su"])
axe[2,0].set_xlabel("boxplot su")
axe[2,1].boxplot(data["bgr"])
axe[2,1].set_xlabel("boxplot bgr")
axe[2,2].boxplot(data["sod"])
axe[2,2].set_xlabel("boxplot sod")

axe[3,0].boxplot(data["pot"])
axe[3,0].set_xlabel("boxplot pot")
axe[3,1].boxplot(data["hemo"])
axe[3,1].set_xlabel("boxplot hemo")

plt.show()
data.drop([21,61,128] ,axis=0, inplace=True)
data["pot"].sort_values()
data.dropna(inplace=True)
data.reset_index(drop=True, inplace=True)
figure, axe = plt.subplots(5,3,figsize=(15, 20))
axe[0,0].hist(data['bp'])
axe[0,0].set_xlabel("histogram bp")
axe[0,1].hist(data['age'])
axe[0,1].set_xlabel("histogram age")
axe[0,2].hist(data['bu'])
axe[0,2].set_xlabel("histogram bu")

axe[1,0].hist(data["sg"])
axe[1,0].set_xlabel("histogram sg")
axe[1,1].hist(data["al"])
axe[1,1].set_xlabel("histogram al")
axe[1,2].hist(data["sc"])
axe[1,2].set_xlabel("histogram sc")

axe[2,0].hist(data["su"])
axe[2,0].set_xlabel("histogram su")
axe[2,1].hist(data["bgr"])
axe[2,1].set_xlabel("histogram bgr")
axe[2,2].hist(data["sod"])
```

```
axe[2,2].set_xlabel("histogram sod")

axe[3,0].hist(data["pot"],bins=30)
axe[3,0].set_xlabel("histogram pot")
axe[3,1].hist(data["hemo"])
axe[3,1].set_xlabel("histogram hemo")
axe[3,2].hist(data["classification"])
axe[3,2].set_xlabel("histogram classification")

axe[4,0].hist(data["pcv"])
axe[4,0].set_xlabel("histogram pcvt")
axe[4,1].hist(data["wc"])
axe[4,1].set_xlabel("histogram wc")
axe[4,2].hist(data["rc"])
axe[4,2].set_xlabel("histogram rc")

plt.show()
figure, axe = plt.subplots(3,3,figsize=(15, 15))
axe[0,0].hist(data['rbc'])
axe[0,0].set_xlabel("histogram rbc")
axe[0,1].hist(data['pc'])
axe[0,1].set_xlabel("histogram pc")
axe[0,2].hist(data['pcc'])
axe[0,2].set_xlabel("histogram pcc")

axe[1,0].hist(data["ba"])
axe[1,0].set_xlabel("histogram ba")
axe[1,1].hist(data["htn"])
axe[1,1].set_xlabel("histogram htn")
axe[1,2].hist(data["dm"])
axe[1,2].set_xlabel("histogram dm")


axe[2,0].hist(data["cad"])
axe[2,0].set_xlabel("histogram cad")
axe[2,1].hist(data["appet"])
axe[2,1].set_xlabel("histogram appet")
axe[2,2].hist(data["pe"])
axe[2,2].set_xlabel("histogram pe")
fig, ax = plt.subplots()
im = ax.imshow(data.corr())

# Show all ticks and label them with the respective list entries
ax.set_xticks(np.arange(len(data.columns)))
ax.set_yticks(np.arange(len(data.columns)))
plt.show()
#scaling
column=data.columns
for i in range(0,24):
  max=data[column[i]].max()
  min=data[column[i]].min()
  dif=max-min
  for j in range(0,397):
    data[column[i]][j]=(data[column[i]][j]-min)/dif
#splitting the dataset

x=data.drop(columns=['classification'],axis=1)
y=data['classification']
```

MODEL BUILDING:

KNN :

```
#KNN

from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()

# training the model
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
knn.fit(x_train,y_train)

#test the model
pred=knn.predict(x_test)

pred
y_test

from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

#CALCULTAE THE ACCURACY
accuracy_score(y_test,pred)

#THE CLASSIFICATION REPORT
confusion_matrix(y_test,pred)
print(classification_report(y_test,pred))
```

NAIVE BAYES

```
#Naive Bayes

#Initializing the Naive Bayes model
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()

#Train the model
nb.fit(x_train,y_train)

#test the model
pred=nb.predict(x_test)
pred
```

```
# Evaluate the Model performance
from sklearn import metrics
metrics.confusion_matrix(y_test,pred)

print(metrics.classification_report(y_test,pred))

from sklearn.metrics import accuracy_score
accuracy_score(y_test,pred)
```

SVM

```
#SVM

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create an SVM classifier object
svm = SVC()

# Train the model
svm.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

LOGISTIC REGRESSION

```
# Logistic Regression
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

logreg = LogisticRegression()
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

DECISION TREE

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

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

y_pred = decision_tree.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

ENSEMBLE MODEL USING VOTING CLASSIFIER

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
```

```
svm = SVC()
logreg = LogisticRegression()
decision_tree = DecisionTreeClassifier()
knn = KNeighborsClassifier()
naive_bayes = GaussianNB()

ensemble_model = VotingClassifier(estimators=[('svm', svm), ('logreg', logreg),
('decision_tree', decision_tree), ('knn', knn), ('naive_bayes', naive_bayes)],
voting='hard')

ensemble_model.fit(X_train, y_train)

y_pred = ensemble_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

SAVING THE MODEL

```
import pickle
ensemble_model = VotingClassifier(estimators=[('svm', svm), ('logreg', logreg),
('decision_tree', decision_tree), ('knn', knn), ('naive_bayes', naive_bayes)],
voting='hard')

ensemble_model.fit(X_train, y_train)
# Save the model as a pickle file
with open('kd.pkl', 'wb') as file:
    pickle.dump(ensemble_model, file)
```