Early Prediction of Chronic Kidney Disease using Machine Learning

Chirag Ajay Jain, Rakshit Kumar, Tanmay Bhatnagar, Musaddiq Ajaz
Vellore Institute of Technology Andhra Pradesh, Amaravati 522237, India
Team ID: SWTID1720090815

I. Introduction

Project Overview

Chronic kidney disease (CKD) is a global health concern, often progressing silently until later stages. Early detection is crucial for timely intervention and improved patient outcomes. Traditional diagnostic methods can be time-consuming or invasive. This project aims to develop machine learning models to predict the risk of developing CKD based on patient data, assisting healthcare professionals in early identification and enabling better disease management.

Deployment: To make the model accessible and user-friendly, it is deployed as a web application using Flask. This will allow users to input medical details and receive predictions about the likelihood of having CKD.

Objectives

1. Develop machine learning models for early CKD prediction using patient data.
2. Evaluate the performance of different models to identify the most effective one.
3. Create a flask app to deploy the model.

II. Project Initialization and Planning Phase

Problem Statement

The project addresses the challenge of early detection of chronic kidney disease. Traditional methods might have limitations in terms of time, invasiveness, model overfitting and accuracy. This project aims to leverage machine learning to develop a model that can effectively predict the risk of CKD based on readily available patient data.

Project Proposal (Proposed Solution)

Acquire a credible dataset, and perform data pre-processing techniques to address missing data, imbalance data, and messy data. In the case of limited data records, data augmentation techniques can be utilized to create a synthetic data, which is later merged with the real-world data. The model will be trained on this data while undergoing parameter tuning and cross-validation training. Four models will be tested and the model with the best recall and accuracy will be used.

Methodology:

1. Data Collection: Gather and understand the dataset.
2. Data Preprocessing: Clean and preprocess the data to ensure it is suitable for modelling.
3. Model Development: Develop and train various machine learning models.
4. Model Evaluation: Evaluate the models to select the best-performing one.
5. Deployment: Deploy the final model as a web application using Flask.

Initial Project Planning

Timeline: Establish a timeline with clear milestones for each phase of the project:

- Data Collection and Exploration: 29 Jun
- Data Preprocessing: 30 Jun – 7 Jul
- Model Development: 7 Jul – 19 Jul
- Model Evaluation: 7 Jul – 19 Jul
- Model Deployment: 7 Jul – 20 Jul

Resource Allocation: Assign roles and responsibilities to team members:

- Data Collection and Pre-processing: Chirag Ajay Jain, Rakshit Kumar
- Model Development: Chirag Ajay Jain
- Model Deployment: Tanmay Bhatnagar, Mussadiq Ajaz
- Project Manager: Chirag Ajay Jain

III. Data Collection and Pre-processing Phase

Data Collection Plan and Raw Data Sources Identified

Data Source: The primary dataset for this project is the CKD dataset available from the UCI Machine Learning Repository. This dataset contains various medical attributes that are crucial for predicting CKD.

Exploration: Performed an initial exploration of the dataset to understand its structure, the types of attributes it contains, and the presence of any missing values.

Data Quality Report

Assessment: The dataset includes all the necessary categorical features to proceed with the ML model development, it contains a many missing observations that can be addressed using statistical handling i.e. Mean and Mode. Unnecessary columns are present within the dataset that are not contributing towards the prediction. These columns are dropped from the dataset either manually or through recursive feature elimination.

Outliers: Outliers are purposely retained as in the medical field, there are always a chance of a rare/anomalous case. Thus, retaining outliers may be beneficial for early prediction.

Statistics and Visualizations:

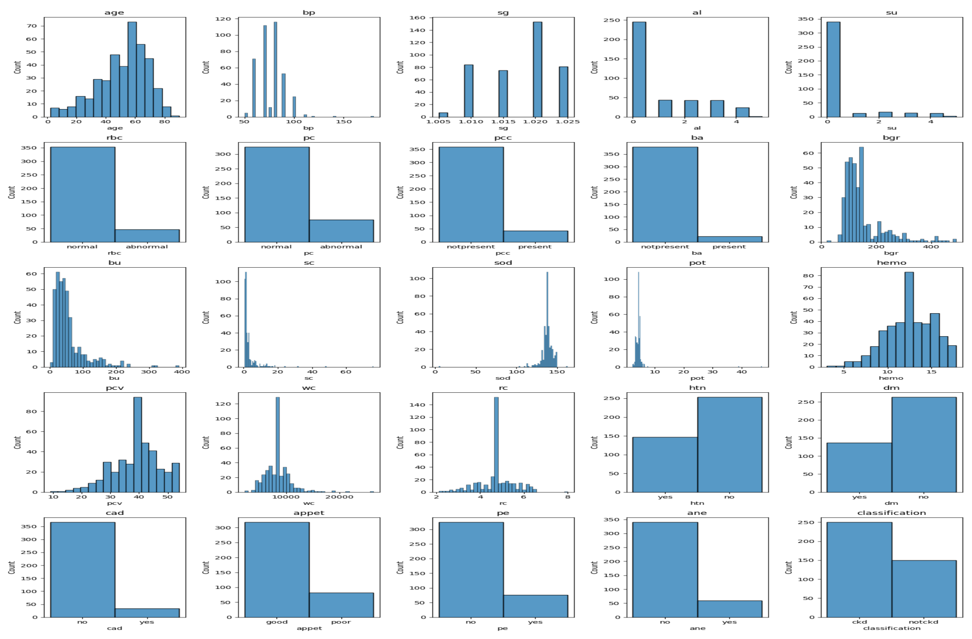

Fig.1 Univariate analysis of each feature

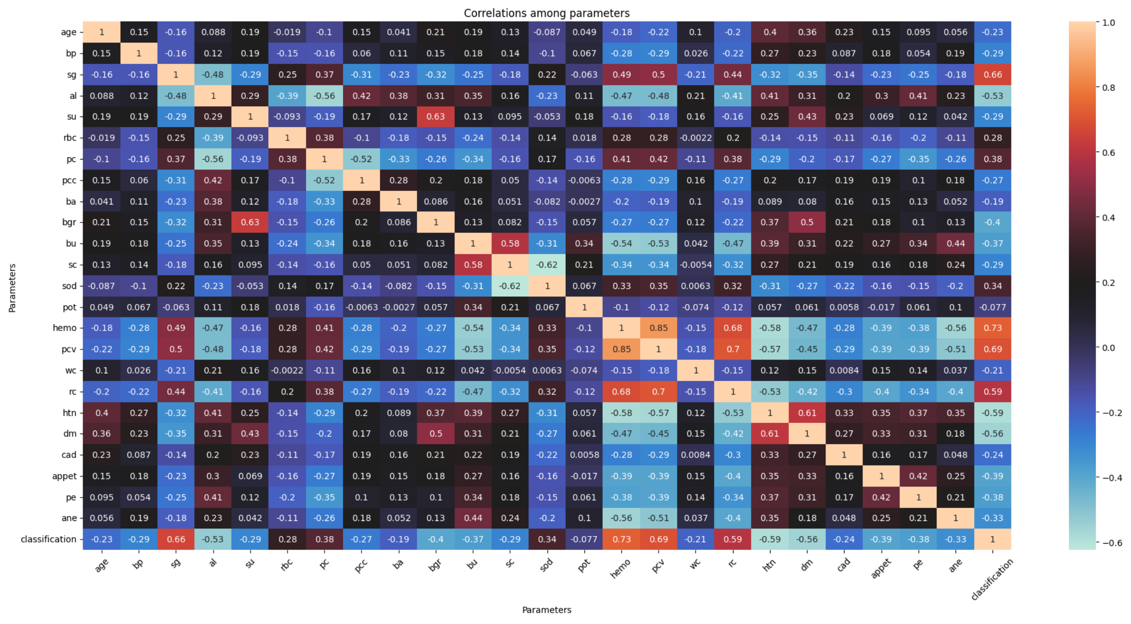

Fig.2 Heatmap of the dataset

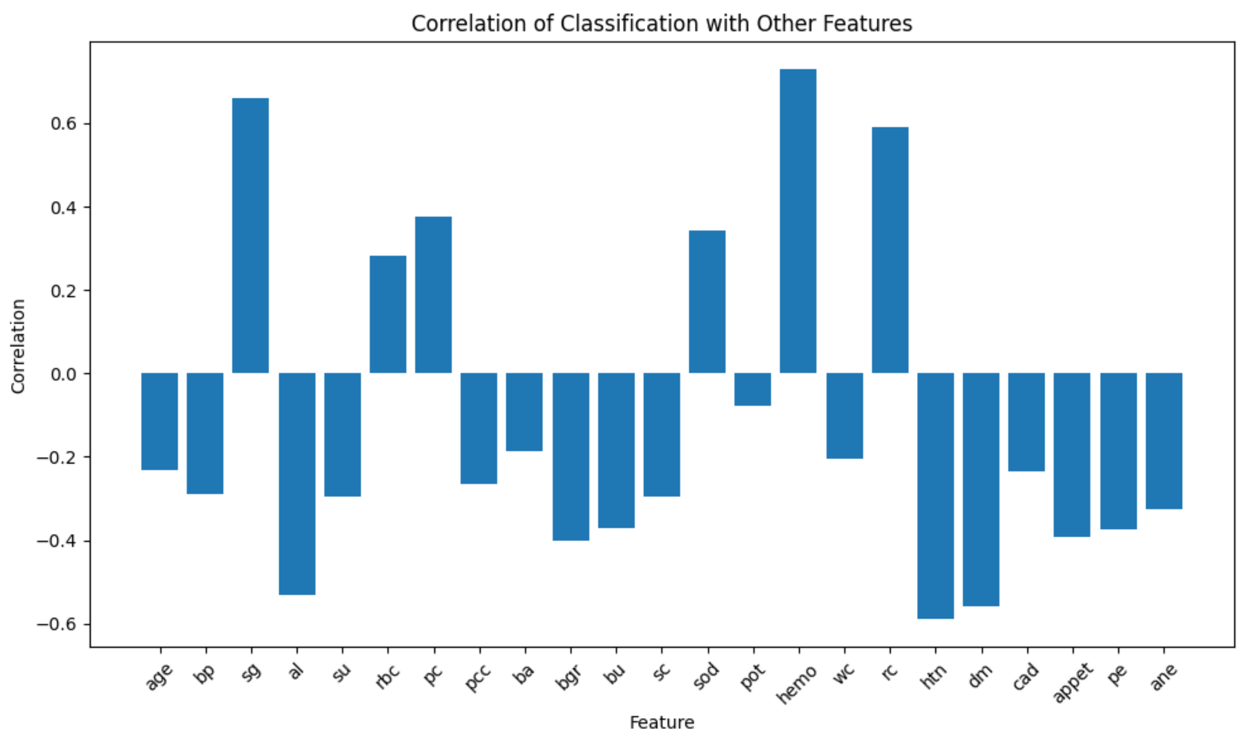

Fig.3 Correlation between classification and other features

<u>Data Pre-processing</u>

Data exploration techniques like visualization and statistical analysis were employed to understand the data distribution, identify relationships between features, and discover potential patterns related to CKD risk.

Label encoding was performed to convert categorical features into numerical features for visualisation and model training.

A synthetic dataset was created using Copulas library mimicking the real-world dataset. It is then merged with that dataset and then separated into train and test set, where test set acts as unseen data and train set is used for training the model.

The merged dataset is then divided into independent and target variable. Data imbalance is handled using SMOTE and the data is finally ready.

IV. Model Development Phase

<u>Feature Selection Report</u>

Feature selection was initially carried out by analysing the Heatmap, off of which *Packed Cell Volume* was discarded. Later on, during model building RFECV (Recursive Feature Elimination) was introduced to eliminated those features that were contributing less towards the prediction of CKD on the basis of correlation.

Feature	Description	Selected (Yes/No)
ID	Index value of records	No
Age	Patient’s age	Yes
Blood Pressure	Blood pressure. It is measured in millimeters of mercury (mmHg)	Yes
Specific Gravity	Specific gravity of Urine. It measures the concentration of particles in urine compared to water.	Yes

Albumin	This is a protein found in blood and urine.	Yes
Sugar	Blood sugar level, refers to the amount of glucose present in the bloodstream.	Yes
Red Blood Cells	These are cells in the blood that carry oxygen throughout the body.	Yes
Pus Cells	White blood cells in urine.	Yes
Pus Cell Clumps	Multiple pus cells grouped together.	Yes
Bacteria	Presence of bacteria in urine	Yes
Blood Glucose Random	Blood sugar test done at a random time, without prior fasting.	Yes
Blood Urea	Urea is a waste by-product in the blood that is filtered by kidneys.	Yes
Serum Creatinine	Creatinine is a waste product in the blood filtered by the kidneys.	Yes
Sodium	An electrolyte that helps regulate fluids in the body.	Yes
Potassium	An electrolyte that plays crucial role in nerve and muscle function.	Yes

Haemoglobin	Protein in RBC that carries oxygen.	Yes
Packed Cell Volume	Percentage of red blood cells in whole blood.	No
White Blood Cell Count	Total number of white blood cells in the bloodstream.	Yes
Red Blood Cell Count	Number of red blood cells in bloodstream.	Yes
Hypertension	Chronic condition where blood pressure remains consistently elevated, which can increase the risk of heart disease and stroke.	Yes
Diabetes Mellitus	Group of metabolic disorders characterized by high blood sugar levels due to problems with insulin production or function	Yes
Coronary Artery Disease	Arteries supplying blood to the heart become narrowed or blocked by plaque buildup.	Yes
Appetite	Person’s desire to eat.	No
Peda Edema	Swelling in feet.	No
Anemia	Blood has lower than normal number of red blood cells or hemoglobin.	No

Classification	Presence or absence of chronic kidney disease	No

---

Model Selection Report

Various machine learning models, such as Random Forest, Logistic Regression, Decision Tree Classifier, and XGBoost Classifier, were chosen for exploration. The rationale behind selecting these models was based on their effectiveness in classification tasks and their suitability for the binary classification.

Initial Model Training Code, Model Validation, and Evaluation Report

Machine learning models were implemented using code (details in Appendix X.1). Cross-validation was used to evaluate model performance and ensure generalizability. Evaluation metrics like accuracy, and recall were employed to assess how well the models predicted the risk of CKD. Several visualisations such as ROC curve and precision recall curve were used to evaluate the model performance.

Model	Baseline Metric	Optimized Metric
Random Forest Classifier	Accuracy: 95.00% Recall: 88.88%	Accuracy: 94.16% Recall: 88.88%
Logistic Regression	Accuracy: 92.50% Recall: 94.44%	Accuracy:93.33% Recall: 91.66%
Decision Tree Classifier	Accuracy: 94.16% Recall: 86.11%	Accuracy: 90.00% Recall: 80.55%
XGBoost Classifier	Accuracy: 92.50 % Recall:88.88%	Accuracy: 93.33% Recall: 94.44%

V. Model Optimization and Tuning Phase

Tuning Documentation

Hyperparameter tuning was performed using GridSearchCV to identify the optimal configuration for each machine learning model. This involved systematically testing different combinations of hyperparameter values and selecting the combination that yielded the best performance on the validation dataset.

Final Model Selection Justification

A model for medical disease prediction must have a high recall score and accuracy as lives are at stake. It is necessary for a model to diagnose people for CKD when they truly have. If a model diagnoses someone with no CKD when they have it, it could be life threatening. A model with good recall score with reasonable accuracy must be selected and thus XGBoost Classifier is selected.

VI. Results

Output Screenshots

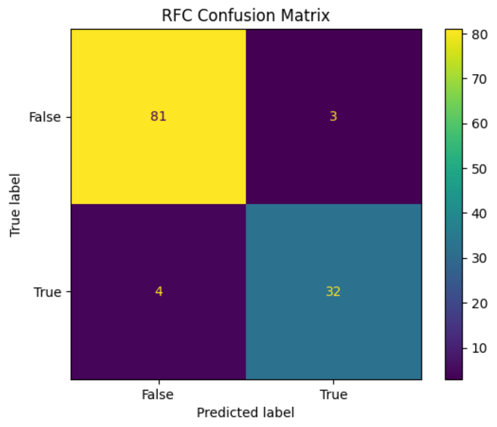

Fig.4 RFC Confusion Matrix

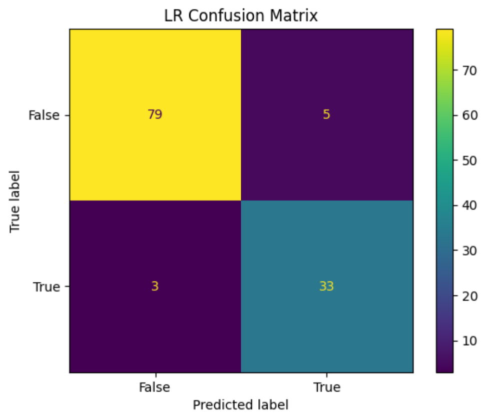

Fig.5 LR Confusion Matrix

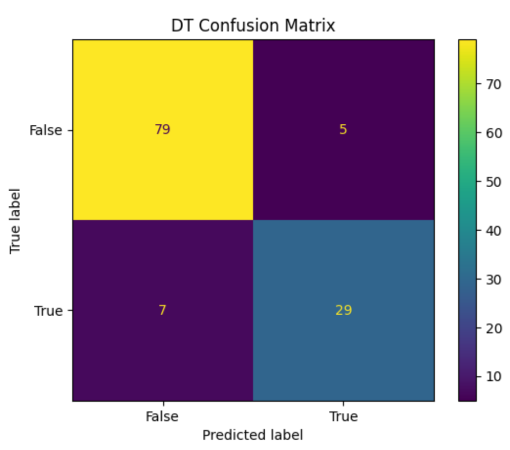

Fig.6 DT Confusion Matrix

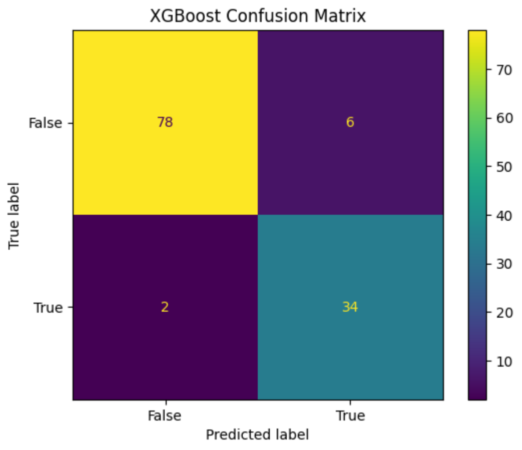

Fig.7 XGBoost Confusion Matrix

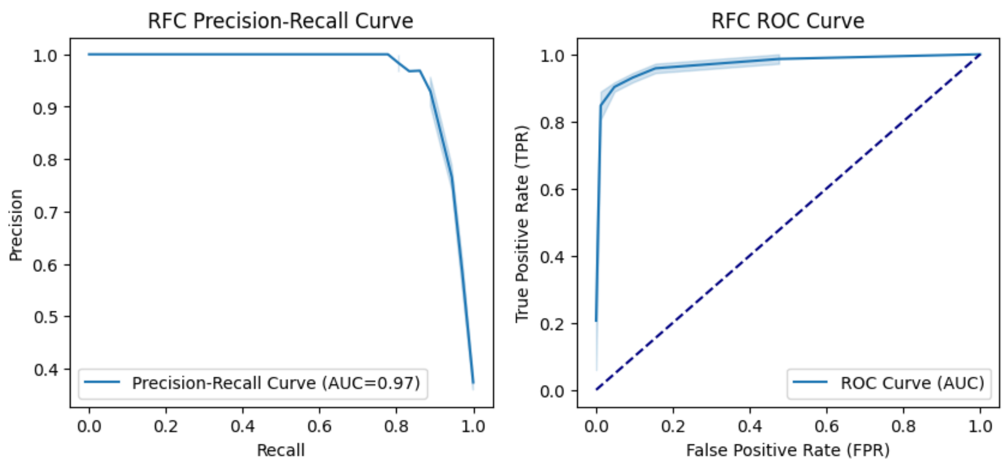

Fig.8 RFC Precision-Recall & ROC Curve

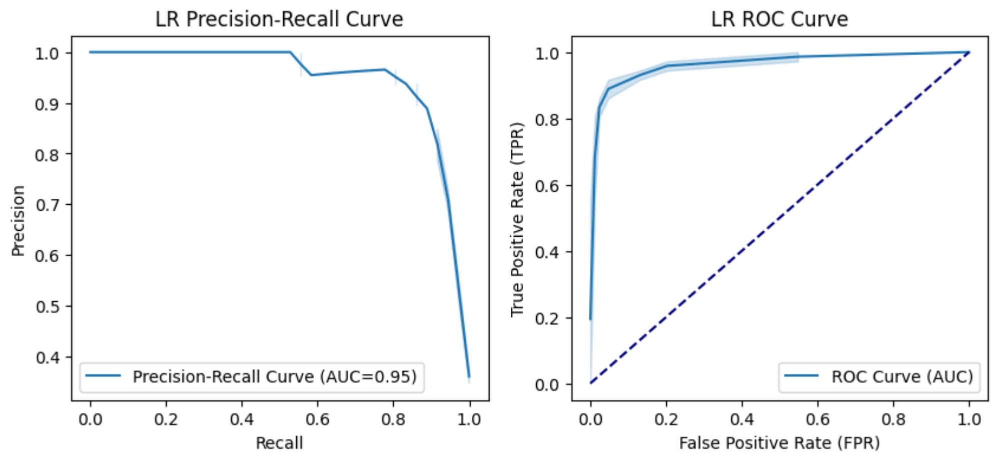

Fig.9 LR Precision-Recall & ROC Curve

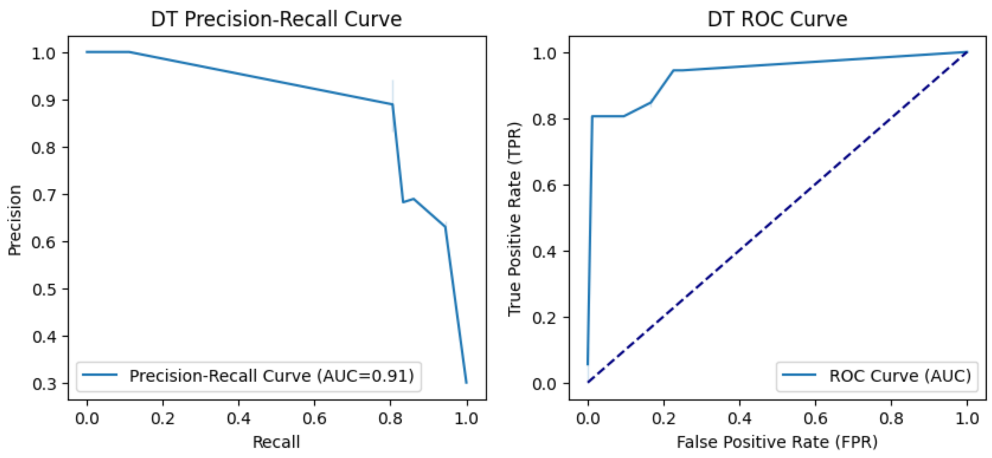

Fig.10 DT Precision-Recall & ROC Curve

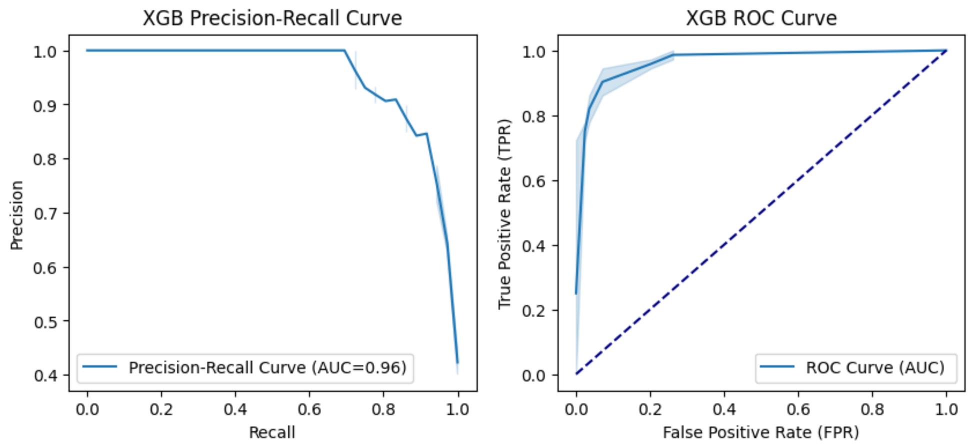

Fig.11 XGBoost Precision-Recall & ROC Curve

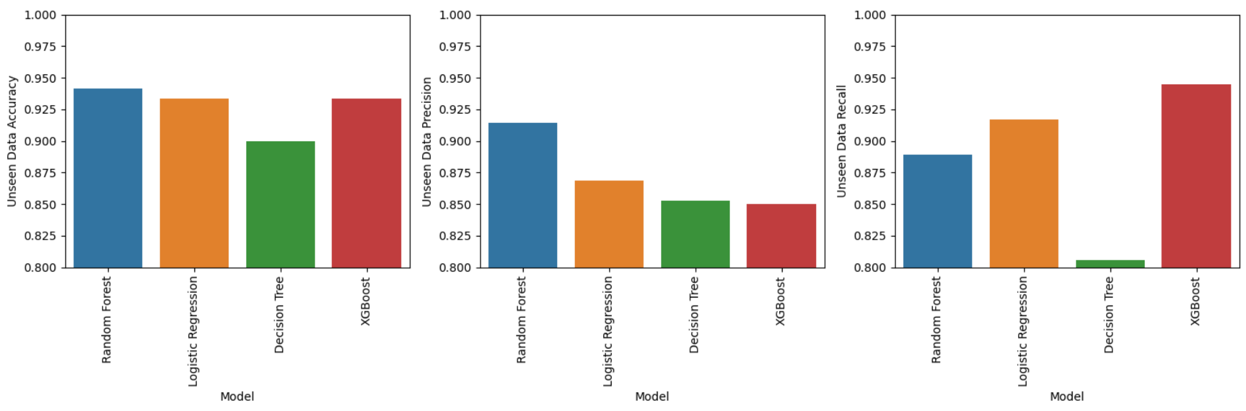

Fig.12 Comparison of Accuracy ,Precision, and Recall of all models on unseen data

VII. Advantages & Disadvantages

Advantages

- Limited real-world data can lead to overfitting, where models perform well on training data (high accuracy) but struggle with unseen data. Our approach tackles this challenge by using cross-validation and synthetic data augmentation.
- **Cross-validation prevents overfitting:** This technique helps the model learn from different data subsets, improving generalizability.
- **Synthetic data augmentation expands the dataset:** By adding synthetic data, the model encounters a broader range of examples, enhancing its ability to generalize.
- **Improved generalization leads to better performance:** By addressing overfitting, our model is more likely to perform accurately on unseen data

Impact: Helps analyse the potential impact of early CKD prediction on patient health and the healthcare system, including reduced healthcare costs and improved quality of life for patients.

Public Health Benefits: Identifying individuals at risk can help allocate resources effectively and inform public health campaigns

Disadvantages

Limited Action ability: While the model predicts CKD, it doesn't provide specific recommendations. This limits its direct impact on patient care.

Model Complexity and Interpretability: Even though the model may be accurate, understanding why it makes a particular prediction can be challenging, especially for complex models. This could hinder trust and adoption in the medical community.
Resource Intensive: Developing and maintaining the model requires ongoing effort and computational resources.

VIII. Conclusion

Summary

Data Preprocessing: The dataset was cleaned, preprocessed, and explored thoroughly. Feature engineering techniques were applied to handle categorical and continuous variables.

Data Imbalance: The dataset exhibited class imbalance, which was addressed using SMOTE oversampling.

Model Selection: Four classification models (Random Forest, Logistic Regression, Decision Tree, and XGBoost) were implemented and evaluated.

Model Performance: All models demonstrated reasonable performance, with XGBoost Classifier showing the best results in terms of accuracy, precision, and recall.

Hyperparameter Tuning: The importance of hyperparameter tuning was recognized, but not fully implemented for all models.

Feature Selection: The use of RFECV for feature selection was a positive step, but further exploration of feature selection techniques is recommended.

Model Performance

Accuracy: All models achieved relatively high accuracy due to the dataset's characteristics. However, accuracy alone might not be the most informative metric for imbalanced datasets.

Precision and Recall: These metrics provide a better understanding of the model's ability to correctly predict positive and negative cases. Random Forest is expected to perform well in terms of both precision and recall.

AUC-ROC: This metric can be used to assess the overall performance of the models, especially in terms of their ability to discriminate between positive and negative classes.

IX. Future Scope

Improvements

Key areas for improvement include:

Comprehensive model evaluation: Calculate and report precision, recall, F1-score, and AUC-ROC for each model.

Hyperparameter tuning: Perform thorough hyperparameter tuning for all models using techniques like GridSearchCV or RandomizedSearchCV.

Feature engineering: Explore additional feature engineering techniques to potentially improve model performance.

Model selection: Consider using ensemble methods or other advanced algorithms to further enhance predictive capabilities.

Feature Engineering Exploration: Experiment with different feature scaling and transformation techniques (e.g., normalization, standardization).

Model Ensemble: Combine predictions from multiple models using techniques like bagging or boosting

Accuracy: Exploring possibilities for improving model accuracy through advanced techniques such as feature engineering and ensemble methods.

Additional Data: Incorporating additional medical data from other sources to enhance prediction accuracy and robustness.

Accessibility

Mobile App: Can move the web app to a mobile application to provide wider accessibility, allowing users to make predictions on the go.

X. Appendix

Source Code

Note: The following code was exported from jupyter notebook as a python script and hence may should error if run in a compiler. Use a jupyter notebook and run each code block.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split,GridSearchCV,KFold
from sklearn.feature_selection import RFECV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import
accuracy_score,confusion_matrix,ConfusionMatrixDisplay,classification_report,
precision_score,recall_score,precision_recall_curve,roc_curve,auc
from copulas.multivariate import GaussianMultivariate
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from pickle import dump

kidney_data= pd.read_csv("chronickidneydisease.csv")
kidney_data.reset_index(drop=True,inplace=True)
kidney_data.head()
kidney_data.drop(columns=['id'],inplace=True)
kidney_data.info()

features =
{'age':'Age','bp':'Blood_Pressure','sg':'Specific_gravity','al':'Albumin','su':'Sugar',

'rbc':'Red_blood_cells','pc':'Pus_cell','pcc':'Pus_cell_clumps','ba':'Bacteria',

'bgr':'Blood_Glucose_Random','bu':'Blood_urea','sc':'Serum_creatinine','sod':'Sodium',

'pot':'Potassium','hemo':'Haemoglobin','pcv':'Packed_cell_volume','wc':'White_blood_ce
ll_count',

'rc':'Red_blood_cell_count','htn':'Hypertension','dm':'Diabetes_mellitus','cad':'Coron
ary_artery_disease',
            'appet':'Appetite','pe':'Peda_edema','ane':'Anemia'}
# Renaming Columns
```

```python
kidney_data.rename(columns=features)
# Identifying categorical and continuous type features for better understanding of unique values and handling missing values.

categorical=pd.DataFrame(columns=['Categorical Columns','Unique Values'])
object_columns=list(kidney_data.columns[kidney_data.dtypes=='O'])
for columns in object_columns:
    unique_val= kidney_data[columns].unique()
    categorical= pd.concat([categorical, pd.DataFrame({'Categorical Columns': [columns], 'Unique Values': [unique_val]})], ignore_index=True)

# Based on the above dataset, we will clean the categorical features to achieve proper values

kidney_data['dm']=kidney_data['dm'].replace('\tno','no').replace('\tyes','yes').replace(' yes','yes')
kidney_data['cad']=kidney_data['cad'].replace('\tno','no')
kidney_data['classification']=kidney_data['classification'].replace('ckd\t','ckd')

Continuous=pd.DataFrame(columns=['Continuous Columns','Unique Values'])
object_columns=list(kidney_data.columns[kidney_data.dtypes!='O'])
for columns in object_columns:
    unique_val= kidney_data[columns].unique()
    Continuous = pd.concat([Continuous, pd.DataFrame({'Continuous Columns': [columns], 'Unique Values': [unique_val]})], ignore_index=True)

Continuous=pd.concat([Continuous,pd.DataFrame({'Continuous Columns':['pcv'],'Unique Values':[categorical.iloc[4]['Unique Values']]})],ignore_index=True)
Continuous=pd.concat([Continuous,pd.DataFrame({'Continuous Columns':['rc'],'Unique Values':[categorical.iloc[6]['Unique Values']]})],ignore_index=True)
Continuous=pd.concat([Continuous,pd.DataFrame({'Continuous Columns':['wc'],'Unique Values':[categorical.iloc[5]['Unique Values']]})],ignore_index=True)
categorical=pd.concat([categorical,pd.DataFrame({'Categorical Columns':['sg'],'Unique Values':[Continuous.iloc[2]['Unique Values']]})],ignore_index=True)
categorical=pd.concat([categorical,pd.DataFrame({'Categorical Columns':['al'],'Unique Values':[Continuous.iloc[3]['Unique Values']]})],ignore_index=True)
categorical=pd.concat([categorical,pd.DataFrame({'Categorical Columns':['su'],'Unique Values':[Continuous.iloc[4]['Unique Values']]})],ignore_index=True)
Continuous.drop(index=[2,3,4],inplace=True)
categorical.drop(index=[4,5,6],inplace=True)

# Converting `pcv`, `rc`, and `wc` to numeric type

kidney_data['pcv']=pd.to_numeric(kidney_data['pcv'],errors='coerce')
kidney_data['rc']=pd.to_numeric(kidney_data['rc'],errors='coerce')
kidney_data['wc']=pd.to_numeric(kidney_data['wc'],errors='coerce')
kidney_data['sg']=kidney_data['sg'].astype(object)
```

```python
kidney_data['al']=kidney_data['al'].astype(object)
kidney_data['su']=kidney_data['su'].astype(object)
# Cleaned and updated features:

kidney_data.isnull().sum()

#Mode of Categorical features
kidney_data['age']=kidney_data['age'].fillna(kidney_data['age'].mode()[0])
kidney_data['rbc']=kidney_data['rbc'].fillna(kidney_data['rbc'].mode()[0])
kidney_data['pc']=kidney_data['pc'].fillna(kidney_data['pc'].mode()[0])
kidney_data['pcc']=kidney_data['pcc'].fillna(kidney_data['pcc'].mode()[0])
kidney_data['ba']=kidney_data['ba'].fillna(kidney_data['ba'].mode()[0])
kidney_data['htn']=kidney_data['htn'].fillna(kidney_data['htn'].mode()[0])
kidney_data['dm']=kidney_data['dm'].fillna(kidney_data['dm'].mode()[0])
kidney_data['cad']=kidney_data['cad'].fillna(kidney_data['cad'].mode()[0])
kidney_data['appet']=kidney_data['appet'].fillna(kidney_data['appet'].mode()[0])
kidney_data['pe']=kidney_data['pe'].fillna(kidney_data['pe'].mode()[0])
kidney_data['ane']=kidney_data['ane'].fillna(kidney_data['ane'].mode()[0])
kidney_data['sg']=kidney_data['sg'].fillna(kidney_data['sg'].mode()[0])
kidney_data['al']=kidney_data['al'].fillna(kidney_data['al'].mode()[0])
kidney_data['su']=kidney_data['su'].fillna(kidney_data['su'].mode()[0])
#Mean of continuous columns
kidney_data['bp']=kidney_data['bp'].fillna(kidney_data['bp'].mean())
kidney_data['bgr']=kidney_data['bgr'].fillna(kidney_data['bgr'].mean())
kidney_data['bu']=kidney_data['bu'].fillna(kidney_data['bu'].mean())
kidney_data['sc']=kidney_data['sc'].fillna(kidney_data['sc'].mean())
kidney_data['sod']=kidney_data['sod'].fillna(kidney_data['sod'].mean())
kidney_data['pot']=kidney_data['pot'].fillna(kidney_data['pot'].mean())
kidney_data['hemo']=kidney_data['hemo'].fillna(kidney_data['hemo'].mean())
kidney_data['pcv']=kidney_data['pcv'].fillna(kidney_data['pcv'].mean())
kidney_data['rc']=kidney_data['rc'].fillna(kidney_data['rc'].mean())
kidney_data['wc']=kidney_data['wc'].fillna(kidney_data['wc'].mean())

kidney_data.isnull().sum()

kidney_data.describe()

num=0
plt.figure(figsize=(15,20))
for col in kidney_data.columns:
  num+=1
  plt.subplot(5,5,num)
  sns.histplot(kidney_data[col])
  plt.title(f'{col}')
  plt.tight_layout()
plt.show()
```

```
num=0
plt.figure(figsize=(15,30))
for col in Continuous['Continuous Columns']:
  num+=1
  plt.subplot(6,4,num)
  sns.swarmplot(kidney_data[col])
  plt.title(f'{col}')
  plt.tight_layout()
plt.show()

numerical_features=list(Continuous['Continuous Columns'])
columns = list(kidney_data.columns)
for i in columns:
  row=6
  col=4
  if kidney_data[i].dtype=='object':
    plt_counter=1
    plt.figure(figsize=(25,45))
    for j in numerical_features:
      plt.subplot(row,col,plt_counter)
      sns.stripplot(data=kidney_data,x=i,y=j,hue='classification')
      plt.title(f'{i} vs {j}' )
      plt_counter+=1
    plt.show()
    row+=1

label_enc=LabelEncoder()
classes={}
for i in categorical.iloc[:]['Categorical Columns']:
    kidney_data[i] = label_enc.fit_transform(kidney_data[i])
    classes[f"{i}"]=label_enc.classes_
kidney_data.head()

plt.figure(figsize=(20,10))
correlations=kidney_data.corr()
sns.heatmap(correlations,annot=True,cmap='icefire')
plt.xlabel("Parameters")
plt.ylabel("Parameters")
plt.title("Correlations among parameters")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

kidney_data.drop(columns = ['pcv'],inplace=True)

correlation_dict = {}
for col in kidney_data.drop(columns=['classification']).columns:
  correlation_dict[col] = kidney_data['classification'].corr(kidney_data[col])
correlation_values = list(correlation_dict.values())
column_names = list(correlation_dict.keys())
plt.figure(figsize=(10, 6))
plt.bar(column_names, correlation_values)
plt.xlabel('Feature')
plt.ylabel('Correlation')
plt.title('Correlation of Classification with Other Features')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

kidney_data.to_csv('kidney_data_processed.csv')

copula = GaussianMultivariate()
copula.fit(kidney_data)
synthetic_data = copula.sample(200)

synthetic_data.age = round(abs(synthetic_data.age)).astype('float64')
synthetic_data.bp =round(abs(synthetic_data.bp)).astype('float64')
synthetic_data.sg =round(abs(synthetic_data.sg)).astype('int64')
synthetic_data.su =round(abs(synthetic_data.su)).astype('int64')
synthetic_data.al =round(abs(synthetic_data.al)).astype('int64')
synthetic_data.pcc =round(abs(synthetic_data.pcc)).astype('int64')
synthetic_data.rbc =round(abs(synthetic_data.rbc)).astype('int64')
synthetic_data.pc =round(abs(synthetic_data.pc)).astype('int64')
synthetic_data.ba =round(abs(synthetic_data.ba)).astype('int64')
synthetic_data.bgr =round(abs(synthetic_data.bgr)).astype('float64')
synthetic_data.bu =round(abs(synthetic_data.bu)).astype('float64')
synthetic_data.sc =round(abs(synthetic_data.sc)).astype('float64')
synthetic_data.sod =round(abs(synthetic_data.sod)).astype('float64')
synthetic_data.pot =round(abs(synthetic_data.pot)).astype('float64')
synthetic_data.hemo =round(abs(synthetic_data.hemo)).astype('float64')
synthetic_data.wc = round(abs(synthetic_data.wc)).astype('float64')
synthetic_data.rc =round(abs(synthetic_data.rc)).astype('float64')
synthetic_data.htn =round(abs(synthetic_data.htn)).astype('int64')
synthetic_data.dm =round(abs(synthetic_data.dm)).astype('int64')
synthetic_data.cad =round(abs(synthetic_data.cad)).astype('int64')
synthetic_data.appet =round(abs(synthetic_data.appet)).astype('int64')
synthetic_data.pe =round(abs(synthetic_data.pe)).astype('int64')
synthetic_data.ane =round(abs(synthetic_data.ane)).astype('int64')
synthetic_data.classification =round(abs(synthetic_data.classification)).astype('int64')

synthetic_data.to_csv("synthetic_data.csv",index=False)

plt.figure(figsize=(20,10))
correlations=synthetic_data.corr()
```

```python
sns.heatmap(correlations,annot=True,cmap='icefire')
plt.xlabel("Parameters")
plt.ylabel("Parameters")
plt.title("Correlations among parameters")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()


columns = list(kidney_data.columns)
columns.remove('classification')

merged_kidney_data = pd.concat([kidney_data,synthetic_data])
test_size = int(0.2*(merged_kidney_data.shape[0]))
test_data = merged_kidney_data.sample(test_size,random_state=42)
merged_kidney_data.drop(test_data.index,inplace=True)

Xtest = test_data.drop("classification",axis=1)
ytest = test_data['classification']

X = merged_kidney_data.drop("classification", axis=1)
y = merged_kidney_data["classification"]
smote = SMOTE(random_state=42)
X_resampled,y_resampled = smote.fit_resample(X, y)

print(y.value_counts())
print(y_resampled.value_counts())

print(X_resampled.shape)
print(y_resampled.shape)

kf=KFold(n_splits=25,random_state=42,shuffle=True) #KFold object
Accuracy,All_prec,All_rec,test_accuracy,test_recall,test_precision = {},{},{},{},{},{}
#Dictionaries to store corresponding values for final analysis


rfc = RandomForestClassifier()

X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled,train_size=0.8,random_state=42)
rfc.fit(X_train,y_train)
y_pred= rfc.predict(X_test)
print(f'Accuracy:{accuracy_score(y_test,y_pred)*100:.4f}%')

y_pred = rfc.predict(Xtest)
print(f'Unseen Data Accuracy:{accuracy_score(ytest,y_pred)*100:.4f}%')
print(f'Unseen Data recall:{recall_score(ytest,y_pred)*100:.4f}%')
```

```
confusion_mat = confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels =
['False', 'True'])
cm_display.plot()
plt.title("RFC Confusion Matrix")
plt.show()

print(classification_report(ytest,y_pred))

param_dist = {
    'n_estimators': [10, 50, 100, 1000],
    'max_depth':[3, 5, 10, 15, 20, None],
    'min_samples_split': [2, 4, 8, 16],
    'min_samples_leaf': [1, 2, 4, 8],
    'max_features': ['auto', 'sqrt', 'log2']
}

grid_search =
GridSearchCV(estimator=rfc,param_grid=param_dist,scoring="accuracy",cv=10)
grid_search.fit(X_train, y_train)



best_params = grid_search.best_params_
max_depth = best_params['max_depth']
n_estimators = best_params['n_estimators']
min_sample_split = best_params['min_samples_split']
min_sample_leaf = best_params['min_samples_leaf']
best_params

best_rfc=RandomForestClassifier(n_estimators=n_estimators,min_samples_split=min_sample_
split, min_samples_leaf=min_sample_leaf,max_depth=max_depth)
rfe_rfc = RFECV(best_rfc,min_features_to_select=20,cv=kf,step=1)

scores=[]
precision= []
recall = []
for train_index, test_index in kf.split(X_resampled,y_resampled):
  X_train, X_test = X_resampled.iloc[train_index], X_resampled.iloc[test_index]
  y_train, y_test = y_resampled.iloc[train_index], y_resampled.iloc[test_index]
  rfe_rfc.fit(X_train, y_train)
  y_pred=rfe_rfc.predict(X_test)
  prec = precision_score(y_test,y_pred)
  rec = recall_score(y_test,y_pred)
  score = accuracy_score(y_test,y_pred)
  precision.append(prec)
  recall.append(rec)
  scores.append(score)
  print(f'Fold Score: {score}')

average_score = sum(scores) / len(scores)
average_precision = sum(precision)/len(precision)
average_recall = sum(recall)/len(recall)
Accuracy['Random Forest'] = average_score
All_prec['Random Forest'] = average_precision
All_rec['Random Forest'] = average_recall
print(f'Average Cross-Validation Score: {average_score*100:.4f}')
print(f'Average Precision score: {average_precision*100:.4f}')
print(f'Average Recall score: {average_recall*100:.4f}')

y_pred = rfe_rfc.predict([[45,90,1,0,1,1,0,0,0,230,57,0,140,6,20,3700,0,0,0,1,0,0,0]])
print(y_pred)

y_pred=rfe_rfc.predict(Xtest)
test_accuracy['Random Forest'] = accuracy_score(ytest,y_pred)
test_recall['Random Forest'] = recall_score(ytest,y_pred)
test_precision['Random Forest'] = precision_score(ytest,y_pred)

confusion_mat=confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels =
['False', 'True'])
cm_display.plot()
plt.title('RFC Confusion Matrix')
plt.show()

print(classification_report(ytest,y_pred))

yscore = rfe_rfc.predict_proba(Xtest)[:,1]
plt.figure(figsize=(10,4))
precision, recall, thresholds = precision_recall_curve(ytest, yscore)
auc_score=auc(recall,precision)
plt.subplot(1,2,1)
sns.lineplot(x=recall, y=precision, label=f'Precision-Recall Curve
(AUC={auc_score:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('RFC Precision-Recall Curve')
fpr, tpr, thresholds = roc_curve(ytest, yscore)
plt.subplot(1,2,2)
sns.lineplot(x=fpr, y=tpr, label='ROC Curve (AUC)')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('RFC ROC Curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.show()

lr = LogisticRegression()

X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled,test_size=0.2,random_state=42)
lr.fit(X_train,y_train)
y_pred= lr.predict(X_test)
print(f'Accuracy:{accuracy_score(y_test,y_pred)*100:.4f}%')

y_pred = lr.predict(Xtest)
print(f'Unseen Data Accuracy:{accuracy_score(ytest,y_pred)*100:.4f}%')
print(f'Unseen Data Recall:{recall_score(ytest,y_pred)*100:.4f}%')

confusion_mat = confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels =
['False', 'True'])
cm_display.plot()
plt.title("LR Confusion Matrix")
plt.show()

print(classification_report(ytest,y_pred))

param_dist = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'lbfgs']
}

grid_search =
GridSearchCV(estimator=lr,param_grid=param_dist,scoring='accuracy',cv=10)
grid_search.fit(X_train,y_train)

best_params = grid_search.best_params_
C = best_params['C']
Solver = best_params['solver']
best_lr = LogisticRegression(C=C,solver=Solver)
rfe_lr = RFECV(best_lr,min_features_to_select=20,cv=kf,step=1)

scores=[]
precision =[]
recall = []
for train_index, test_index in kf.split(X_resampled,y_resampled):
  X_train, X_test = X_resampled.iloc[train_index], X_resampled.iloc[test_index]
  y_train, y_test = y_resampled.iloc[train_index], y_resampled.iloc[test_index]
  rfe_lr.fit(X_train, y_train)
  y_pred=rfe_lr.predict(X_test)
  prec = precision_score(y_test,y_pred)
  rec = recall_score(y_test,y_pred)
  score = rfe_lr.score(X_test, y_test)
  scores.append(score)
  precision.append(prec)
  recall.append(rec)
  print(f'Fold Score: {score}')

average_score = sum(scores) / len(scores)
average_precision = sum(precision)/len(precision)
average_recall = sum(recall)/len(recall)
Accuracy['Linear Regression'] = average_score
All_prec['Linear Regression'] = average_precision
All_rec['Linear Regression '] = average_recall
print(f'Average Cross-Validation Score: {average_score*100:.4f}')
print(f'Average Precision score: {average_precision*100:.4f}')
print(f'Average Recall score: {average_recall*100:.4f}')

y_pred = rfe_lr.predict([[45,90,1,0,1,1,0,0,0,230,57,0,140,6,20,3700,0,0,0,1,0,0,0]])
print(y_pred)
# Testing optimized model on unseen data

y_pred= rfe_lr.predict(Xtest)
test_accuracy['Logistic Regression'] = accuracy_score(ytest,y_pred)
test_recall['Logistic Regression'] = recall_score(ytest,y_pred)
test_precision['Logistic Regression'] = precision_score(ytest,y_pred)

confusion_mat=confusion_matrix(y_pred=y_pred,y_true=ytest)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels =
['False', 'True'])
cm_display.plot()
plt.title('LR Confusion Matrix')
plt.show()

print(classification_report(y_pred=y_pred,y_true=ytest))

yscore = rfe_lr.predict_proba(Xtest)[:,1]
plt.figure(figsize=(10,4))
precision, recall, thresholds = precision_recall_curve(ytest, yscore)
auc_score = auc(recall,precision)
plt.subplot(1,2,1)
sns.lineplot(x=recall, y=precision, label=f'Precision-Recall Curve
(AUC={auc_score:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('LR Precision-Recall Curve')
fpr, tpr, thresholds = roc_curve(ytest, yscore)
plt.subplot(1,2,2)
```

```
sns.lineplot(x=fpr, y=tpr, label='ROC Curve (AUC)')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('LR ROC Curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.show()

dt= DecisionTreeClassifier()

X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled,test_size=0.2,random_state=42)
dt.fit(X_train,y_train)
y_pred= dt.predict(X_test)
print(f'Accuracy:{accuracy_score(y_test,y_pred)*100:.4f}%')

y_pred = dt.predict(Xtest)
print(f'Unseen Data Accuracy:{accuracy_score(ytest,y_pred)*100:.4f}%')
print(f'Unseen Data Recall:{recall_score(ytest,y_pred)*100:.4f}%')

confusion_mat = confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels =
['False', 'True'])
cm_display.plot()
plt.title("DT Confusion Matrix")
plt.show()

print(classification_report(ytest,y_pred))

param_dist = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

grid_search =
GridSearchCV(estimator=dt,param_grid=param_dist,scoring="accuracy",cv=10)
grid_search.fit(X_train,y_train)


best_params = grid_search.best_params_
min_sample_split = best_params['min_samples_split']
min_sample_leaf = best_params['min_samples_leaf']
max_features = best_params['max_features']
max_depth=best_params['max_depth']
best_dt=DecisionTreeClassifier(min_samples_split=min_sample_split,
min_samples_leaf=min_sample_leaf,max_features=max_features,max_depth=max_depth)
rfe_dt = RFECV(best_dt,min_features_to_select=20,cv=kf,step=1)
```

```
scores=[]
precision = []
recall=[]
for train_index, test_index in kf.split(X_resampled,y_resampled):
  X_train, X_test = X_resampled.iloc[train_index], X_resampled.iloc[test_index]
  y_train, y_test = y_resampled.iloc[train_index], y_resampled.iloc[test_index]
  rfe_dt.fit(X_train, y_train)
  y_pred=rfe_dt.predict(X_test)
  score = accuracy_score(y_test,y_pred)
  prec = precision_score(y_test,y_pred)
  rec =recall_score(y_test,y_pred)
  precision.append(prec)
  recall.append(rec)
  scores.append(score)
  print(f'Fold Score: {score}')

average_score = sum(scores) / len(scores)
average_precision = sum(precision)/len(precision)
average_recall = sum(recall)/len(recall)
Accuracy['Decision Tree'] = average_score
All_prec['Decision Tree'] = average_precision
All_rec['Decision Tree'] = average_recall
print(f'Average Cross-Validation Score: {average_score*100:.4f}')
print(f'Average Precision score: {average_precision*100:.4f}')
print(f'Average Recall score: {average_recall*100:.4f}')

y_pred = rfe_dt.predict([[45,90,1,0,1,1,0,0,0,230,57,0,140,6,20,3700,0,0,0,1,0,0,0]])
print(y_pred)
# Testing optimized model on unseen data

y_pred=rfe_dt.predict(Xtest)
test_accuracy['Decision Tree'] = accuracy_score(ytest,y_pred)
test_recall['Decision Tree'] = recall_score(ytest,y_pred)
test_precision['Decision Tree'] = precision_score(ytest,y_pred)

confusion_mat=confusion_matrix(y_pred=y_pred,y_true=ytest)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels =
['False', 'True'])
cm_display.plot()
plt.title('DT Confusion Matrix')
plt.show()

print(classification_report(ytest,y_pred))


yscore = rfe_dt.predict_proba(Xtest)[:,1]
plt.figure(figsize=(10,4))
```

```
precision, recall, thresholds = precision_recall_curve(ytest, yscore)
auc_score = auc(recall,precision)
plt.subplot(1,2,1)
sns.lineplot(x=recall, y=precision, label=f'Precision-Recall Curve
(AUC={auc_score:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('DT Precision-Recall Curve')
fpr, tpr, thresholds = roc_curve(ytest, yscore)
plt.subplot(1,2,2)
sns.lineplot(x=fpr, y=tpr, label='ROC Curve (AUC)')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('DT ROC Curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.show()

xgb_model = XGBClassifier()

X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled,test_size=0.2,random_state=42)
xgb_model.fit(X_train,y_train)
y_pred= xgb_model.predict(X_test)
print(f'Accuracy:{accuracy_score(y_test,y_pred)*100:.4f}%')

y_pred = xgb_model.predict(Xtest)
print(f'Unseen Data Accuracy:{accuracy_score(ytest,y_pred)*100:.4f}%')
print(f'Unseen Data Recall:{recall_score(ytest,y_pred)*100:.4f}%')


confusion_mat = confusion_matrix(ytest,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels =
['False', 'True'])
cm_display.plot()
plt.title("XGBoost Confusion Matrix")
plt.show()

print(classification_report(ytest,y_pred))

param_dist = {
    'learning_rate': [0.001,0.01],
    'n_estimators': [10, 100, 1000],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.3],
    'reg_lambda':[1,10,100],
    'reg_alpha':[0.01,0.05]
}
```

```python
grid_search =
GridSearchCV(estimator=xgb_model,param_grid=param_dist,scoring="accuracy",cv=10)
grid_search.fit(X_train,y_train)

best_params = grid_search.best_params_
learning_rate = best_params['learning_rate']
n_estimators = best_params['n_estimators']
min_child_weight = best_params['min_child_weight']
gamma = best_params['gamma']
reg_lambda = best_params['reg_lambda']
reg_alpha = best_params['reg_alpha']

best_xgb=XGBClassifier(learning_rate=learning_rate,
n_estimators=n_estimators,min_child_weight=min_child_weight,max_depth=max_depth,gamma=
gamma,reg_alpha=reg_alpha,reg_lambda=reg_lambda)
rfe_xgb = RFECV(best_xgb,min_features_to_select=20,cv=kf,step=1)

scores=[]
precision = []
recall = []
for train_index, test_index in kf.split(X_resampled,y_resampled):
  X_train, X_test = X_resampled.iloc[train_index], X_resampled.iloc[test_index]
  y_train, y_test = y_resampled.iloc[train_index], y_resampled.iloc[test_index]
  rfe_xgb.fit(X_train, y_train)
  y_pred = rfe_xgb.predict(X_test)
  score = accuracy_score(y_test,y_pred)
  rec = recall_score(y_test,y_pred)
  prec = precision_score(y_test,y_pred)
  scores.append(score)
  recall.append(rec)
  precision.append(prec)
  print(f'Fold Score: {score}')

average_score = sum(scores) / len(scores)
average_precision = sum(precision)/len(precision)
average_recall = sum(recall)/len(recall)
Accuracy['XGBoost'] = average_score
All_prec['XGBoost'] = average_precision
All_rec['XGBoost'] = average_recall
print(f'Average Cross-Validation Score: {average_score*100:.4f}')
print(f'Average Precision score: {average_precision*100:.4f}')
print(f'Average Recall score: {average_recall*100:.4f}')

y_pred = rfe_xgb.predict([[45,90,1,0,1,1,0,0,0,230,57,0,140,6,20,3700,0,0,0,1,0,0,0]])
print(y_pred)
# Testing optimized model on unseen data
```

```
y_pred = rfe_xgb.predict(Xtest)
test_accuracy['XGBoost'] = accuracy_score(ytest,y_pred)
test_recall['XGBoost'] = recall_score(ytest,y_pred)
test_precision['XGBoost'] = precision_score(ytest,y_pred)

confusion_mat=confusion_matrix(y_pred=y_pred,y_true=ytest)
cm_display = ConfusionMatrixDisplay(confusion_matrix = confusion_mat, display_labels =
['False', 'True'])
cm_display.plot()
plt.title('XGBoost Confusion Matrix')
plt.show()

print(classification_report(ytest,y_pred))

yscore = rfe_xgb.predict_proba(Xtest)[:,1]
plt.figure(figsize=(10,4))
precision, recall, thresholds = precision_recall_curve(ytest, yscore)
auc_score = auc(recall, precision)
plt.subplot(1,2,1)
sns.lineplot(x=recall, y=precision, label=f'Precision-Recall Curve
(AUC={auc_score:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('XGB Precision-Recall Curve')
fpr, tpr, thresholds = roc_curve(ytest, yscore)
plt.subplot(1,2,2)
sns.lineplot(x=fpr, y=tpr, label='ROC Curve (AUC) ')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('XGB ROC Curve')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.show()

model_name = [key for key in Accuracy.keys()]
model_accuracy = [acc for acc in Accuracy.values()]
model_precision = [prec for prec in All_prec.values()]
model_recall = [rec for rec in All_rec.values()]
model_data =
pd.DataFrame([model_accuracy,model_precision,model_recall],index=['Accuracy','Precisio
n','Recall'],columns=model_name)
model_data

plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
sns.barplot(x=model_name,y=model_accuracy,hue=model_name)
plt.ylim(0.8,1)
plt.xlabel('Model')
plt.ylabel('Cross-Validation Accuracy')
plt.xticks(rotation=90)
plt.subplot(1,3,2)
sns.barplot(x=model_name,y=model_precision,hue=model_name)
plt.ylim(0.8,1)
plt.xticks(rotation=90)
plt.xlabel('Model')
plt.ylabel('Cross-Validation Precision')
plt.subplot(1,3,3)
sns.barplot(x=model_name,y=model_recall,hue=model_name)
plt.ylim(0.8,1)
plt.xticks(rotation=90)
plt.xlabel('Model')
plt.ylabel('Cross-Validation Recall')
plt.tight_layout()
plt.show()


test_name = ['Random Forest','Logistic Regression','Decision Tree','XGBoost']
test_accuracy = [acc for acc in test_accuracy.values()]
test_precision = [prec for prec in test_precision.values()]
test_recall = [rec for rec in test_recall.values()]
test_data =
pd.DataFrame([test_accuracy,test_precision,test_recall],index=['Accuracy','Precision',
'Recall'],columns=test_name)
test_data

plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
sns.barplot(x=test_name,y=test_accuracy,hue=test_name)
plt.ylim(0.8,1)
plt.xlabel('Model')
plt.ylabel('Unseen Data Accuracy')
plt.xticks(rotation=90)
plt.subplot(1,3,2)
sns.barplot(x=test_name,y=test_precision,hue=test_name)
plt.ylim(0.8,1)
plt.xticks(rotation=90)
plt.xlabel('Model')
plt.ylabel('Unseen Data Precision')
plt.subplot(1,3,3)
sns.barplot(x=test_name,y=test_recall,hue=test_name)
plt.ylim(0.8,1)
plt.xticks(rotation=90)
plt.xlabel('Model')
plt.ylabel('Unseen Data Recall')
plt.tight_layout()
plt.show()
```

```
dump(rfe_xgb,open('CKD.pkl','wb'))
dump(label_enc,open('Label_Encoder.pkl','wb'))
retained_features = X.columns[rfe_xgb.support_]
for i in retained_features:
    print(features[i])
dump(retained_features,open("Retained_Features.pkl","wb")) #dumping retained features
to discard features that were eliminated by the RFE.
```

GitHub & Project Demo Link

Github Link: https://github.com/Chirag-65-Jain/Early-Prediction-Of-Chronic-Kidney-Disease-Using-Machine-Learning

Demo Link: https://www.youtube.com/embed/K3AmvblSkT4