**Codes for Chronical Kidney Disease Detection Using Various ML Algorithms**

1. Logical Regression Method

Code:

###Code Starts

# -\*- coding: utf-8 -\*-

"""CKDD\_LogisticRegression.ipynb

Automatically generated by Colab.

Original file is located at

    https://colab.research.google.com/drive/1Aeg79T77j85Z8tN30cturH0gnVFy2ATT

"""

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

data = pd.read\_csv('/content/chronic\_kidney\_disease.csv')

data.head()

data.drop(data.columns[0], axis=1, inplace=True)

data.isnull().sum()

data.duplicated().sum()

# Check for any unwanted or invalid values in categorical columns in the 'data' dataframe

categorical\_columns = data.select\_dtypes(include="object").columns

for column in categorical\_columns:

    print(f"Value counts for {column}:")

    print(data[column].value\_counts())

    print("\n")

Temporary\_data = data.select\_dtypes(exclude="object")

Temporary\_data.describe()

for x in data.select\_dtypes(include="number").columns:

    sns.boxplot(data=data, x=x, color='green')

    plt.show()

def wishker(col):

    q1, q3 = np.percentile(col, [25, 75])

    iqr = q3 - q1

    lbound = q1 - (iqr \* 1.5)

    ubound = q3 + (iqr \* 1.5)

    return lbound, ubound

columns = data.select\_dtypes(include="number").columns.drop(["class", "su"], errors="ignore").to\_list()

for x in columns:

    lbound, ubound = wishker(data[x])

    data[x] = np.where(data[x] < lbound, lbound, data[x])

    data[x] = np.where(data[x] > ubound, ubound, data[x])

    sns.boxplot(data=data, x=x,color='green')

    plt.show()

from sklearn.preprocessing import LabelEncoder

objandcategory = data.select\_dtypes(include=['object', 'category']).columns

for col in objandcategory:

    Instance = LabelEncoder()

    data[col] = Instance.fit\_transform(data[col])

pd.set\_option("display.max\_column", None)

data.head()

plt.figure(figsize=(15, 15))

sns.heatmap(data.corr(), cmap='coolwarm', annot=True, cbar=False)

plt.title("Correlation Matrix")

plt.show()

correlations = data.corr().abs()

selected\_features = correlations.loc[correlations['CKD'] >= 0.29, 'CKD']

selected\_features = selected\_features.index.difference(['CKD'])

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split,GridSearchCV

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Select only the relevant features from the dataset using selected\_features

X = data[selected\_features]

y = data['CKD']

# Splitting the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Logistic Regression without Scaling and Reduced Iterations

logistic\_model = LogisticRegression(max\_iter=10, random\_state=42)  # Limited iterations

logistic\_model.fit(X\_train, y\_train)

# Predictions and evaluation

y\_pred = logistic\_model.predict(X\_test)

accuracy\_before\_tuning = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Logistic Regression Model (Low Accuracy) Accuracy: {accuracy\_before\_tuning \* 100:.2f}%")

# Plotting the confusion matrix using seaborn

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted Negative', 'Predicted Positive'], yticklabels=['True Negative', 'True Positive'])

plt.title('Confusion Matrix - Logistic Regression')

plt.ylabel('True Label')

plt.xlabel('Predicted Label')

plt.show()

X = data[selected\_features]

y = data['CKD']

# Splitting the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define the parameter grid for grid search

param\_grid = {

    'solver': ['liblinear'],

    'max\_iter': [50],

    'C': [0.01, 0.1, 1, 10, 100]  # Same C values for both grid search and plot

}

# Perform grid search

grid\_search = GridSearchCV(LogisticRegression(random\_state=42), param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# Best C value from the grid search

best\_C = grid\_search.best\_params\_['C']

best\_accuracy = grid\_search.best\_score\_

# Train the best model

best\_logistic\_model = grid\_search.best\_estimator\_

# Predictions using the best model

y\_pred\_best = best\_logistic\_model.predict(X\_test)

# Confusion Matrix for the best model

conf\_matrix\_best = confusion\_matrix(y\_test, y\_pred\_best)

# Plot confusion matrix using seaborn

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix\_best, annot=True, fmt='d', cmap='Blues',

            xticklabels=['Predicted Negative', 'Predicted Positive'],

            yticklabels=['True Negative', 'True Positive'])

plt.title(f'Confusion Matrix - Best Logistic Regression (C={best\_C})')

plt.ylabel('True Label')

plt.xlabel('Predicted Label')

plt.show()

# Train and evaluate models over a range of C values

C\_values = [0.01, 0.1, 1, 10, 100]

accuracies = []

for C in C\_values:

    logistic\_model = LogisticRegression(C=C, max\_iter=50, solver='liblinear', random\_state=42)

    logistic\_model.fit(X\_train, y\_train)

    # Predictions and evaluation

    y\_pred = logistic\_model.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    accuracies.append(accuracy)

# Plotting accuracy vs C values

plt.figure(figsize=(8, 6))

plt.plot(C\_values, accuracies, marker='o', linestyle='-', color='b')

plt.title(f'Accuracy vs Regularization Strength (C) for Logistic Regression\nBest C: {best\_C} with Accuracy: {best\_accuracy \* 100:.2f}%')

plt.xlabel('Regularization Strength (C)')

plt.ylabel('Accuracy')

plt.xscale('log')

plt.show()

# Output the results

print(f"Best C value from GridSearch: {best\_C}")

print(f"Tuned Logistic Regression Model Accuracy: {best\_accuracy \* 100:.2f}%")

accuracies = [accuracy\_before\_tuning, best\_accuracy]

models = ['Before Tuning', 'After Tuning']

plt.figure(figsize=(8, 6))

plt.bar(models, accuracies, color=['#4CAF50', '#FFC107'])

plt.title('Accuracy Comparison: Before vs After Hyperparameter Tuning')

plt.ylabel('Accuracy')

plt.ylim(0, 1)

plt.show()

import joblib

joblib.dump(best\_logistic\_model, 'best\_logistic\_model.pkl')

print("Model saved successfully!")

1. Support Vector Machine(SVM)

Code:

# -\*- coding: utf-8 -\*-

"""Data\_pre-processing\_&\_Feature\_Eng.ipynb

Automatically generated by Colab.

Original file is located at

    https://colab.research.google.com/drive/1zz6LNzQnlaVSt7Tm2WEDoVxJDbcqO\_9M

\*\*Sanity check of data\*\*

"""

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sn

import numpy as np

pd.set\_option("display.max\_column", None)

df = pd.read\_excel("/content/Final\_outcome.xlsx")

df.drop(df.columns[0], axis = 1, inplace = True)

df.head()

print(df.isnull().sum())

print(df.duplicated().sum())

print(df.info())

#Checking for any garbage values

for x in df.select\_dtypes(include = "object").columns:

  print(df[x].value\_counts())

"""\*\*EDA\*\*"""

Temporary\_data = df.select\_dtypes(exclude = "object")

Temporary\_data.describe()

for x in df.select\_dtypes(include = "number").columns:

  sn.boxplot(data = df, x = x)

  plt.show()

df.select\_dtypes(include = "number").columns.to\_list()

after\_outlier = df

def wishker(col):

  q1,q3 = np.percentile(col,[25,75])

  iqr = q3 - q1

  lbound = q1 - (iqr \* 1.5)

  ubound = q3 + (iqr \* 1.5)

  return lbound,ubound

columns = after\_outlier.select\_dtypes(include="number").columns.drop(["class","su"], errors = "ignore").to\_list()

for x in columns:

  lbound,ubound = wishker(after\_outlier[x])

  after\_outlier[x] = np.where(after\_outlier[x] < lbound, lbound, after\_outlier[x])

  after\_outlier[x] = np.where(after\_outlier[x] > ubound, ubound , after\_outlier[x])

  sn.boxplot(data = after\_outlier, x = x)

  plt.show()

from sklearn.preprocessing import LabelEncoder

objandcategory = after\_outlier.select\_dtypes(include = ['object', 'category']).columns

for col in objandcategory:

  Instance = LabelEncoder()

  after\_outlier[col] = Instance.fit\_transform(after\_outlier[col])

after\_outlier.head()

after\_outlier.dtypes

"""\*\*Feature Selection\*\*"""

from sklearn.feature\_selection import chi2

independent = after\_outlier.select\_dtypes(include = "number").drop(columns = ["class"], axis = 1)

dependent = after\_outlier["class"]

scores = chi2(independent, dependent)

pd.DataFrame(scores)

#\*\*Higher the Chi value ----> Higher the importance\*\*

chivalues = pd.Series(scores[0], index = independent.columns)

chivalues.sort\_values(ascending = True, inplace = True)

chivalues.plot.bar()

#\*\*Higher the p-value -----> lower the importance\*\*

pvalues = pd.Series(scores[1], index = independent.columns)

pvalues.sort\_values(ascending = True, inplace = True)

pvalues.plot.bar()

plt.figure(figsize = (15,15))

sn.heatmap(after\_outlier.corr(), cmap = 'plasma',  annot = True, cbar = False)

plt.title("Correlation matrix")

plt.show()

features = after\_outlier.corr()

features = abs(features['class'])

features = features[features >= 0.29]

features = features.index[:-1]

features

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import numpy as np

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

X = after\_outlier[features]

y = after\_outlier['class']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.8, random\_state=42)

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

svm\_model = SVC(kernel='poly', degree=10, C=10000, random\_state=42)

svm\_model.fit(X\_train\_scaled, y\_train)

y\_pred = svm\_model.predict(X\_test\_scaled)

accuracy\_first\_code = accuracy\_score(y\_test, y\_pred)

print(f"First Code SVM Model Accuracy: {accuracy\_first\_code \* 100:.2f}%")

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

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

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

scaler\_standard = StandardScaler()

X\_train\_scaled\_standard = scaler\_standard.fit\_transform(X\_train)

X\_test\_scaled\_standard = scaler\_standard.transform(X\_test)

param\_grid = {

    'C': [0.1, 1, 10, 100],

    'kernel': ['linear', 'rbf'],

    'gamma': ['scale', 'auto', 0.1, 1, 10]

}

grid\_search = GridSearchCV(estimator=SVC(random\_state=42), param\_grid=param\_grid, cv=5, verbose=1, n\_jobs=-1)

grid\_search.fit(X\_train\_scaled\_standard, y\_train)

best\_svm\_model = grid\_search.best\_estimator\_

y\_pred\_tuned = best\_svm\_model.predict(X\_test\_scaled\_standard)

accuracy\_second\_code\_tuned = accuracy\_score(y\_test, y\_pred\_tuned)

print(f"Second Code Tuned Model Accuracy: {accuracy\_second\_code\_tuned \* 100:.2f}%")

print("Best hyperparameters:", grid\_search.best\_params\_)

print(f"Accuracy: {accuracy\_second\_code\_tuned \* 100:.2f}%")

print("Confusion Matrix:")

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_tuned)

print(conf\_matrix)

print("Classification Report:")

class\_report = classification\_report(y\_test, y\_pred\_tuned)

print(class\_report)

cm\_first\_code = confusion\_matrix(y\_test, y\_pred)

disp\_first\_code = ConfusionMatrixDisplay(confusion\_matrix=cm\_first\_code)

disp\_first\_code.plot(cmap='Blues', values\_format='d')

plt.title('Confusion Matrix for First Code SVM Model')

plt.show()

cm\_second\_code\_tuned = confusion\_matrix(y\_test, y\_pred\_tuned)

disp\_second\_code\_tuned = ConfusionMatrixDisplay(confusion\_matrix=cm\_second\_code\_tuned)

disp\_second\_code\_tuned.plot(cmap='Blues', values\_format='d')

plt.title('Confusion Matrix for Tuned SVM Model')

plt.show()

plt.figure(figsize=(10, 6))

plt.bar(

    ['SVM Model (First Code)', 'Tuned Model (Second Code)'],

    [accuracy\_first\_code, accuracy\_second\_code\_tuned],

    color=['red', 'orange']

)

plt.ylabel('Accuracy')

plt.title('SVM Model Accuracy Comparison')

plt.ylim(0, 1)

plt.tight\_layout()

plt.xticks(rotation=45, ha='right')

plt.show()

import joblib

# Save the tuned model

joblib.dump(best\_svm\_model, 'svm\_model\_tuned.pkl')

1. K- Nearest Neighbors(KNN)

Code:

1. import numpy as np
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. # Machine learning tools
5. from sklearn.model\_selection import train\_test\_split
6. from sklearn.preprocessing import StandardScaler
7. from sklearn.neighbors import KNeighborsClassifier
8. from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report
9. df = pd.read\_csv('/content/ckd\_dataset.csv')
10. print(df.head())
11. print(df.info())
12. # Check for missing values
13. print(df.isnull().sum())
14. # Display basic statistics of the data
15. print(df.describe())
16. df\_numeric = df.select\_dtypes(include='number')
17. df[df\_numeric.columns] = df\_numeric.fillna(df\_numeric.mean())
18. # Handle non-numeric columns (e.g., filling with the mode)
19. for col in df.select\_dtypes(exclude='number').columns:
20. df[col].fillna(df[col].mode()[0], inplace=True)
21. print(df.isnull().sum())
22. # Replace 'class' with the actual target column name
23. X = df.drop('class', axis=1)  # Features
24. y = df['class']  # Target
25. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)
26. # Check the shapes of the datasets
27. print(X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape)
28. print(X\_train.dtypes)
29. X\_train = pd.get\_dummies(X\_train, drop\_first=True)
30. X\_test = pd.get\_dummies(X\_test, drop\_first=True)
31. # Check the shape to ensure both sets have the same columns
32. print(X\_train.shape, X\_test.shape)
33. print(df.columns)
34. import pandas as pd
35. from sklearn.preprocessing import LabelEncoder
36. # Load the dataset
37. df = pd.read\_csv('/content/ckd\_dataset.csv')
38. # Identify categorical columns
39. categorical\_columns = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'class']
40. # Label encode binary or ordinal categorical columns (those with two values)
41. label\_encoder = LabelEncoder()
42. for col in categorical\_columns:
43. df[col] = label\_encoder.fit\_transform(df[col])
44. print(df.head())
45. scaler = StandardScaler()
46. X\_train = scaler.fit\_transform(X\_train)
47. X\_test = scaler.transform(X\_test)
48. knn = KNeighborsClassifier(n\_neighbors=5)  # Start with k=5
49. knn.fit(X\_train, y\_train)
50. # Make predictions
51. y\_pred = knn.predict(X\_test)
52. # Calculate accuracy
53. accuracy = accuracy\_score(y\_test, y\_pred)
54. print(f"Accuracy: {accuracy:.2f}")
55. # Confusion Matrix and Classification Report
56. conf\_matrix = confusion\_matrix(y\_test, y\_pred)
57. print("Confusion Matrix:\n", conf\_matrix)
58. class\_report = classification\_report(y\_test, y\_pred)
59. print("Classification Report:\n", class\_report)
60. # Test different values of k
61. accuracies = []
62. for k in range(1, 21):
63. knn = KNeighborsClassifier(n\_neighbors=k)
64. knn.fit(X\_train, y\_train)
65. y\_pred\_k = knn.predict(X\_test)
66. accuracies.append(accuracy\_score(y\_test, y\_pred\_k))
67. # Plot accuracy vs. k
68. plt.plot(range(1, 21), accuracies, marker='o')
69. plt.title('Accuracy vs. Number of Neighbors (k)')
70. plt.xlabel('Number of Neighbors (k)')
71. plt.ylabel('Accuracy')
72. plt.show()
73. import joblib
74. # Save the model to a file
75. joblib.dump(knn, 'knn\_ckd\_model.pkl')
76. # Load the model back later
77. # knn = joblib.load('knn\_ckd\_model.pkl')
78. import pickle
79. # Specify the path to your .pkl file
80. file\_path = '/content/knn\_ckd\_model.pkl'
81. # Open and load the .pkl file
82. with open(file\_path, 'rb') as file:
83. data = pickle.load(file)
84. # Display the contents of the .pkl file
85. print(data)

4. Gaussian Process Regression (GPR)

Code:

import pandas as pd

from sklearn.preprocessing import  LabelEncoder

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.gaussian\_process import GaussianProcessRegressor

from sklearn.gaussian\_process.kernels import RBF, ConstantKernel as C

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split, learning\_curve

from sklearn.metrics import roc\_curve, auc, precision\_recall\_curve

import joblib

# age – Age of the patient

# bp – Blood Pressure

# sg – Specific Gravity

# al – Albumin

# su – Sugar

# rbc – Red Blood Cells

# pc – Pus Cells

# pcc – Pus Cell Clumps

# ba – Bacteria

# bgr – Blood Glucose Random

# bu – Blood Urea

# sc – Serum Creatinine

# sod – Sodium

# pot – Potassium

# hemo – Hemoglobin

# pcv – Packed Cell Volume

# wbcc – White Blood Cell Count

# rbcc – Red Blood Cell Count

# htn – Hypertension

# dm – Diabetes Mellitus

# cad – Coronary Artery Disease

# appet – Appetite

# pe – Pedal Edema

# ane – Anemia

# class – Chronic Kidney Disease (CKD) or Non-CKD (Target column)

# Function Definations

# Function to generate bar graph with custom ranges for numeric features

def create\_bar\_graph\_with\_range(column):

    if column == 'age':

        bins = [0, 20, 40, 60, 80, 100]

        labels = ['1-20', '21-40', '41-60', '61-80', '81-100']

    elif column == 'bp':

        bins = [0, 90, 120, 140, 160, 180, 200]

        labels = ['<90', '90-120', '120-140', '140-160', '160-180', '>180']

    elif column == 'bgr':

        bins = [0, 70, 100, 150, 200, 300, 500]

        labels = ['<70', '70-100', '100-150', '150-200', '200-300', '>300']

    else:

        # creating bins using the min and max values

        min\_value = data[column].min()

        max\_value = data[column].max()

        bins = [min\_value, (min\_value + max\_value) / 3, 2 \* (min\_value + max\_value) / 3, max\_value]

        labels = [f'{round(bins[i], 2)} - {round(bins[i+1], 2)}' for i in range(len(bins)-1)]

    # Sorting the bins in ascending order

    bins = sorted(bins)

    # Bin the data and count the values in each bin

    binned\_data = pd.cut(X\_encoded[column], bins=bins, labels=labels, include\_lowest=True)

    bin\_counts = binned\_data.value\_counts()

    # Creating the bar graph

    plt.figure(figsize=(8, 6))

    sns.barplot(x=bin\_counts.index, y=bin\_counts.values, palette="Set3")

    plt.title(f"Bar Graph of {get\_full\_form(column)} with Ranges")

    plt.xlabel('Range')

    plt.ylabel('Count')

    plt.savefig(f'Bar\_Graph\_{get\_full\_form(column)}.png')

    plt.close()

#Function to get full forms of the short forms

def get\_full\_form(short\_form):

    if short\_form == 'age':

        return 'Age of the patient'

    elif short\_form == 'bp':

        return 'Blood Pressure'

    elif short\_form == 'sg':

        return 'Specific Gravity'

    elif short\_form == 'al':

        return 'Albumin'

    elif short\_form == 'su':

        return 'Sugar'

    elif short\_form == 'rbc':

        return 'Red Blood Cells'

    elif short\_form == 'pc':

        return 'Pus Cells'

    elif short\_form == 'pcc':

        return 'Pus Cell Clumps'

    elif short\_form == 'ba':

        return 'Bacteria'

    elif short\_form == 'bgr':

        return 'Blood Glucose Random'

    elif short\_form == 'bu':

        return 'Blood Urea'

    elif short\_form == 'sc':

        return 'Serum Creatinine'

    elif short\_form == 'sod':

        return 'Sodium'

    elif short\_form == 'pot':

        return 'Potassium'

    elif short\_form == 'hemo':

        return 'Hemoglobin'

    elif short\_form == 'pcv':

        return 'Packed Cell Volume'

    elif short\_form == 'wbcc':

        return 'White Blood Cell Count'

    elif short\_form == 'rbcc':

        return 'Red Blood Cell Count'

    elif short\_form == 'htn':

        return 'Hypertension'

    elif short\_form == 'dm':

        return 'Diabetes Mellitus'

    elif short\_form == 'cad':

        return 'Coronary Artery Disease'

    elif short\_form == 'appet':

        return 'Appetite'

    elif short\_form == 'pe':

        return 'Pedal Edema'

    elif short\_form == 'ane':

        return 'Anemia'

    elif short\_form == 'class':

        return 'Chronic Kidney Disease (CKD) or Non-CKD (Target column)'

    else:

        return 'Unknown'

## Section 1

# Loading the Dataset

file\_path = "Final\_outcome.xlsx"

data = pd.read\_excel(file\_path)

#Dropping the first column as it is the indexing

data = data.iloc[:, 1:]

print("Data after removing indexing column:")

print(data.head())

# List of numerical columns to scale

numerical\_columns = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc']

# Initialize the StandardScaler

scaler = StandardScaler()

# Apply scaling to the numerical columns

data[numerical\_columns] = scaler.fit\_transform(data[numerical\_columns])

# Verify scaling

print(data.head())

## Section 2

#Seperating the Data Features and the Target

# Seperating the targeted column to check if the results of the training works or not

target\_column = "class"

# Separating the ckd columns and other features column

X = data.drop(columns=[target\_column])

y = data[target\_column]

# Display the shapes to verify separation

print("Features (X) shape:", X.shape)

print("Target (y) shape:", y.shape)

# Display the first few rows of X and y

print("\nFeatures (X):")

print(X.head())

print("\nTarget (y):")

print(y.head())

##Section 3

##Encoding the Categorical(Text and String Values in the data) data so it can be used in Computation to process

# Applying one-hot encoding directly using pandas get\_dummies

X\_encoded = pd.get\_dummies(X, drop\_first=True)

# Applying Label Encoding to the target

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

# Checking if the encoding is done currectly or not

print("Class Mapping: ", dict(zip(label\_encoder.classes\_, label\_encoder.transform(label\_encoder.classes\_))))

y\_encoded = (y\_encoded == 0).astype(int)  # Reversing the orders because ckd is made 0 and non-ckd as 1 by default

# Displaying the encoded data

print("Encoded Features (X):")

print(X\_encoded.head())

print("\nEncoded Target (y):")

print(y\_encoded)

## Section 4: Plotting the features in Bar Graph

# List of numeric columns from the dataset

# numeric\_columns = ['age', 'bp', 'al', 'su','bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc']

# # Ensure all numeric columns are present in the data

# numeric\_columns = [col for col in numeric\_columns if col in data.columns]

# # Create bar graphs for all numeric columns

# for column in numeric\_columns:

#     create\_bar\_graph\_with\_range(column)

## Section 5: Model Creation and Training

# Step 1: Spliting the dataset into training and testing sets

# Using 50% of the data for training and 50% for testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y\_encoded, test\_size=0.5, random\_state=100)

print(f"\n Training set: X\_train = {X\_train.shape}, y\_train = {y\_train.shape}")

print(f"\n Testing set: X\_test = {X\_test.shape}, y\_test = {y\_test.shape}")

# Step 2: Creating the GPR model

# Defining the kernel (RBF kernel with constant term)

kernel = RBF(1.0, (1e-4, 1e3))

# Creating the GPR model

gpr = GaussianProcessRegressor(kernel=kernel, n\_restarts\_optimizer=3)

# Step 3: Training the model

gpr.fit(X\_train, y\_train)

## Section 6: Evaluating the Model

#Making predictions on the test set

y\_pred, sigma = gpr.predict(X\_test, return\_std=True)

# Since GPR gives continuous predictions, we will need to apply a threshold for classification if needed

# Converting predictions to binary if it's a classification task (like CKD detection)

y\_pred\_class = (y\_pred > 0.7).astype(int)  # Setting the threshold to 0.7

cm =confusion\_matrix(y\_test, y\_pred\_class)

report = classification\_report(y\_test, y\_pred\_class, output\_dict=True, zero\_division=0)

# Printing the predictions and uncertainty

print("Predictions:", y\_pred)

print("Uncertainty (std deviation):", sigma)

# Printing out the evaluation results

print("Confusion Matrix:")

print(cm)

print("\nClassification Report:")

print(report)

# Calculating the accuracy

accuracy = accuracy\_score(y\_test, y\_pred\_class)

print(f"Accuracy: {accuracy:.4f}")

##  PLotting the graphs

#Plotting the accuracy per points in the test data

# Ensuring y\_test and y\_pred are numpy arrays

if not isinstance(y\_test, np.ndarray):

    y\_test = np.array(y\_test)

if not isinstance(y\_pred, np.ndarray):

    y\_pred = np.array(y\_pred)

accuracy\_per\_point = 1 - np.abs(y\_test - y\_pred)

plt.figure(figsize=(10, 6))

plt.plot(range(len(y\_test)), accuracy\_per\_point, marker='o', linestyle='-', color='blue', label='Accuracy per Point')

plt.title("Model Prediction Accuracy for Each Data Point", fontsize=16)

plt.xlabel("Test Data ", fontsize=12)

plt.ylabel("Accuracy (0 to 1)", fontsize=12)

plt.grid(alpha=0.5)

plt.ylim(0, 1.1)

plt.text(0.95, 0.95, f'Overall Accuracy: {accuracy:.4f}', horizontalalignment='right', verticalalignment='top', transform=plt.gca().transAxes, fontsize=12, bbox=dict(facecolor='white', alpha=0.7))

plt.legend()

plt.savefig('model\_accuracy\_plot.png', dpi=300)

# Correlation heatmap

plt.figure(figsize=(12, 8))

correlation\_matrix = X\_encoded.corr()

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title("Correlation Heatmap of Features")

plt.savefig('correlation\_heatmap.png')

# Plot confusion matrix

plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Non-CKD", "CKD"], yticklabels=["Non-CKD", "CKD"])

plt.title("Confusion Matrix", fontsize=16)

plt.xlabel("Predicted Class", fontsize=12)

plt.ylabel("True Class", fontsize=12)

plt.savefig('confusion\_matrix.png')

# ROC Curve

fpr, tpr, \_ = roc\_curve(y\_test, y\_pred)

roc\_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.title("ROC Curve", fontsize=16)

plt.xlabel("False Positive Rate", fontsize=12)

plt.ylabel("True Positive Rate", fontsize=12)

plt.legend(loc="lower right")

plt.savefig('roc\_curve.png')

# Classification report

report\_df = pd.DataFrame(report).transpose()

plt.figure(figsize=(8, 6))

sns.heatmap(report\_df[['precision', 'recall', 'f1-score']].iloc[:-1, :], annot=True, cmap='Blues', fmt='.2f')

plt.title('Classification Report Metrics (Precision, Recall, F1-score)')

plt.xlabel('Metrics')

plt.ylabel('Class')

plt.savefig('Classification\_report.png')

##Prediction Distribution

plt.figure(figsize=(8, 6))

sns.histplot(y\_pred\_class, kde=True, color="blue", bins=30)

plt.title("Prediction Distribution", fontsize=16)

plt.xlabel("Predicted Class", fontsize=12)

plt.ylabel("Frequency", fontsize=12)

plt.savefig('Prediction\_Distribution.png')

### # Error Analysis

errors = y\_test - y\_pred\_class

plt.figure(figsize=(8, 6))

plt.hist(errors, bins=30, color='blue', edgecolor='black')

plt.title("Error Analysis", fontsize=16)

plt.xlabel("Error (True - Predicted)", fontsize=12)

plt.ylabel("Frequency", fontsize=12)

plt.savefig('Error\_analysis.png')

##Exporting the model

joblib.dump(gpr, 'gpr\_model.pkl')

print("Model saved successfully.")