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()
```

```python
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))
```

```python
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!")
```

2. Support Vector Machine(SVM)
   Code:

```python
# -*- 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**
"""
```

```python
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()
```

```python
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)
```

```python
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')
```

3. K- Nearest Neighbors(KNN)
   Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Machine learning tools
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

df = pd.read_csv('/content/ckd_dataset.csv')
print(df.head())

print(df.info())

# Check for missing values
print(df.isnull().sum())

# Display basic statistics of the data
print(df.describe())

df_numeric = df.select_dtypes(include='number')
df[df_numeric.columns] = df_numeric.fillna(df_numeric.mean())

# Handle non-numeric columns (e.g., filling with the mode)
```

```
for col in df.select_dtypes(exclude='number').columns:
    df[col].fillna(df[col].mode()[0], inplace=True)

print(df.isnull().sum())
# Replace 'class' with the actual target column name
X = df.drop('class', axis=1)  # Features
y = df['class']  # Target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Check the shapes of the datasets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

print(X_train.dtypes)

X_train = pd.get_dummies(X_train, drop_first=True)
X_test = pd.get_dummies(X_test, drop_first=True)

# Check the shape to ensure both sets have the same columns
print(X_train.shape, X_test.shape)

print(df.columns)
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv('/content/ckd_dataset.csv')

# Identify categorical columns
categorical_columns = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'class']

# Label encode binary or ordinal categorical columns (those with two values)
label_encoder = LabelEncoder()
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])

print(df.head())

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

```python
knn = KNeighborsClassifier(n_neighbors=5)  # Start with k=5
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Confusion Matrix and Classification Report
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)

class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)

# Test different values of k
accuracies = []
for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred_k))

# Plot accuracy vs. k
plt.plot(range(1, 21), accuracies, marker='o')
plt.title('Accuracy vs. Number of Neighbors (k)')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.show()

import joblib

# Save the model to a file
joblib.dump(knn, 'knn_ckd_model.pkl')

# Load the model back later
# knn = joblib.load('knn_ckd_model.pkl')

import pickle

# Specify the path to your .pkl file
file_path = '/content/knn_ckd_model.pkl'
# Open and load the .pkl file
```

```
    with open(file_path, 'rb') as file:
        data = pickle.load(file)

    # Display the contents of the .pkl file
    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')
```

```python
    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
```

```python
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
```

```python
# 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)
```

```python
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
```

```python
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)
```

```python
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.")
```