```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
# Load the dataset
data = pd.read_csv('kidney_disease.csv')
# Display the first few rows of the dataset
print(data.head())
```

```
   id   age    bp     sg   al   su     rbc        pc         pcc          ba  \
0   0  48.0  80.0  1.020  1.0  0.0     NaN    normal  notpresent  notpresent
1   1   7.0  50.0  1.020  4.0  0.0     NaN    normal  notpresent  notpresent
2   2  62.0  80.0  1.010  2.0  3.0  normal    normal  notpresent  notpresent
3   3  48.0  70.0  1.005  4.0  0.0  normal  abnormal     present  notpresent
4   4  51.0  80.0  1.010  2.0  0.0  normal    normal  notpresent  notpresent

   ...  pcv    wc   rc  htn   dm  cad appet   pe  ane classification
0  ...   44  7800  5.2  yes  yes   no  good   no   no            ckd
1  ...   38  6000  NaN   no   no   no  good   no   no            ckd
2  ...   31  7500  NaN   no  yes   no  poor   no  yes            ckd
3  ...   32  6700  3.9  yes   no   no  poor  yes  yes            ckd
4  ...   35  7300  4.6   no   no   no  good   no   no            ckd

[5 rows x 26 columns]
```

```
# Check for missing values
print("Missing values in each column:")
print(data.isnull().sum())
```

```
Missing values in each column:
id                  0
age                 9
bp                 12
sg                 47
al                 46
su                 49
rbc               152
pc                 65
pcc                 4
ba                  4
bgr                44
bu                 19
sc                 17
sod                87
pot                88
hemo               52
pcv                70
wc                105
rc                130
htn                 2
dm                  2
cad                 2
appet               1
pe                  1
ane                 1
classification      0
dtype: int64
```

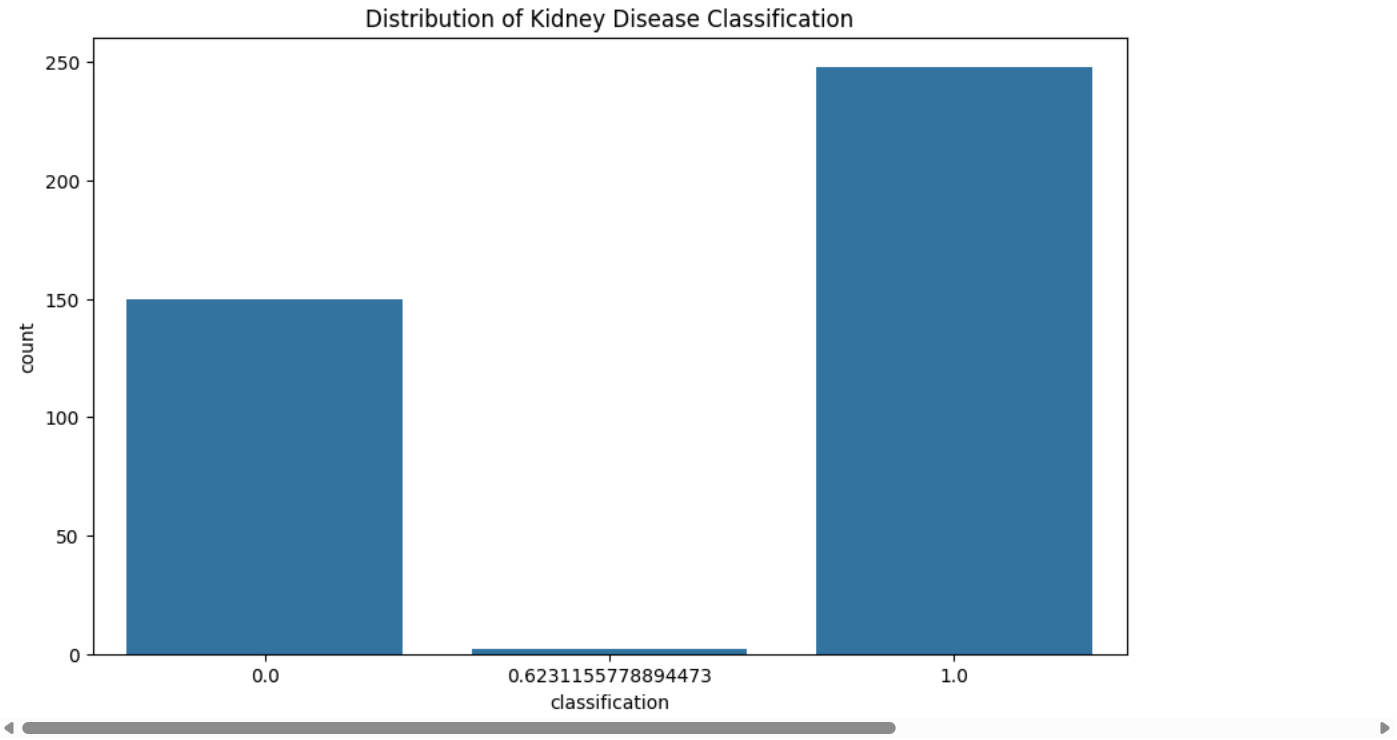
```
# Convert categorical variables to numeric using Label Encoding
categorical_columns = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'classification']
for column in categorical_columns:
    data[column] = data[column].map({
        'normal': 0, 'abnormal': 1,
        'notpresent': 0, 'present': 1,
        'yes': 1, 'no': 0,
        'good': 1, 'poor': 0,
        'ckd': 1, 'notckd': 0
    })
```

```
# Fill missing values (example: fill with mean for numerical columns)
data.fillna(data.mean(numeric_only=True), inplace=True)
```

```
# Statistical summary
print(data.describe())
```

```
                id         age          bp          sg          al          su  \
count  400.000000  400.000000  400.000000  400.000000  400.000000  400.000000
mean   199.500000   51.483376   76.469072    1.017408    1.016949    0.450142
std    115.614301   16.974966   13.476298    0.005369    1.272318    1.029487
min      0.000000    2.000000   50.000000    1.005000    0.000000    0.000000
25%     99.750000   42.000000   70.000000    1.015000    0.000000    0.000000
50%    199.500000   54.000000   78.234536    1.017408    1.000000    0.000000
75%    299.250000   64.000000   80.000000    1.020000    2.000000    0.450142
max    399.000000   90.000000  180.000000    1.025000    5.000000    5.000000

              rbc          pc         pcc          ba  ...         sod  \
count  400.000000  400.000000  400.000000  400.000000  ...  400.000000
mean     0.189516    0.226866    0.106061    0.055556  ...  137.528754
std      0.308983    0.383750    0.306755    0.228199  ...    9.204273
min      0.000000    0.000000    0.000000    0.000000  ...    4.500000
25%      0.000000    0.000000    0.000000    0.000000  ...  135.000000
50%      0.000000    0.000000    0.000000    0.000000  ...  137.528754
75%      0.189516    0.226866    0.000000    0.000000  ...  141.000000
max      1.000000    1.000000    1.000000    1.000000  ...  163.000000

              pot        hemo         htn          dm         cad       appet  \
count  400.000000  400.000000  400.000000  400.000000  400.000000  400.000000
mean     4.627244   12.526437    0.369347    0.341837    0.085859    0.794486
std      2.819783    2.716171    0.482023    0.470146    0.279100    0.404077
min      2.500000    3.100000    0.000000    0.000000    0.000000    0.000000
25%      4.000000   10.875000    0.000000    0.000000    0.000000    1.000000
50%      4.627244   12.526437    0.000000    0.000000    0.000000    1.000000
75%      4.800000   14.625000    1.000000    1.000000    0.000000    1.000000
max     47.000000   17.800000    1.000000    1.000000    1.000000    1.000000

               pe         ane  classification
count  400.000000  400.000000      400.000000
mean     0.190476    0.150376        0.623116
std      0.392677    0.357440        0.483998
min      0.000000    0.000000        0.000000
25%      0.000000    0.000000        0.000000
50%      0.000000    0.000000        1.000000
75%      0.000000    0.000000        1.000000
max      1.000000    1.000000        1.000000

[8 rows x 23 columns]
```

```
# Data visualization
plt.figure(figsize=(10, 6))
sns.countplot(x='classification', data=data)
plt.title('Distribution of Kidney Disease Classification')
plt.show()
```




```
# Load the dataset
data = pd.read_csv('kidney_disease.csv')
# Display the first few rows of the dataset
print(data.head())
# Check for missing values
print("Missing values in each column:")
print(data.isnull().sum())
# Replace invalid entries with NaN
data.replace({'?': np.nan, '\t?': np.nan}, inplace=True)
# Convert relevant columns to numeric, forcing errors to NaN
numeric_columns = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc']
for column in numeric_columns:
    data[column] = pd.to_numeric(data[column], errors='coerce')
```

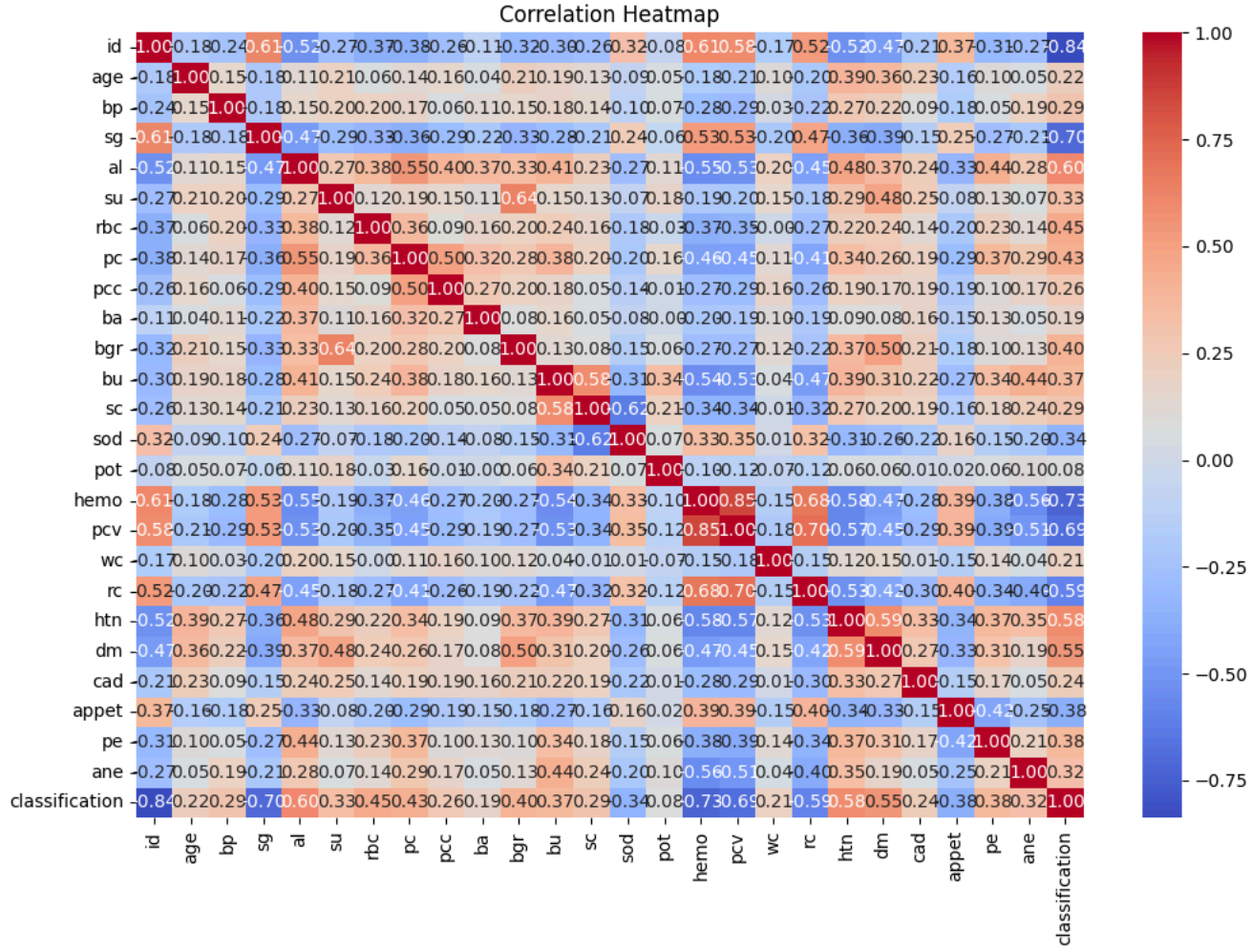
```
   id   age    bp     sg   al   su     rbc        pc         pcc          ba  \
0   0  48.0  80.0  1.020  1.0  0.0     NaN    normal  notpresent  notpresent
1   1   7.0  50.0  1.020  4.0  0.0     NaN    normal  notpresent  notpresent
2   2  62.0  80.0  1.010  2.0  3.0  normal    normal  notpresent  notpresent
3   3  48.0  70.0  1.005  4.0  0.0  normal  abnormal     present  notpresent
4   4  51.0  80.0  1.010  2.0  0.0  normal    normal  notpresent  notpresent

   ...  pcv    wc   rc  htn   dm  cad appet   pe  ane classification
0  ...   44  7800  5.2  yes  yes   no  good   no   no            ckd
1  ...   38  6000  NaN   no   no   no  good   no   no            ckd
2  ...   31  7500  NaN   no  yes   no  poor   no  yes            ckd
3  ...   32  6700  3.9  yes   no   no  poor  yes  yes            ckd
4  ...   35  7300  4.6   no   no   no  good   no   no            ckd

[5 rows x 26 columns]
Missing values in each column:
id                  0
age                 9
bp                 12
sg                 47
al                 46
su                 49
rbc               152
pc                 65
pcc                 4
ba                  4
bgr                44
bu                 19
sc                 17
sod                87
pot                88
hemo               52
pcv                70
wc                105
```

```
rc                130
htn                 2
dm                  2
cad                 2
appet               1
pe                  1
ane                 1
classification      0
dtype: int64
```

```
# Convert categorical variables to numeric using Label Encoding
categorical_columns = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'classification']
for column in categorical_columns:
    data[column] = data[column].map({
        'normal': 0, 'abnormal': 1,
        'notpresent': 0, 'present': 1,
        'yes': 1, 'no': 0,
        'good': 1, 'poor': 0,
        'ckd': 1, 'notckd': 0
    })
```

```
# Fill missing values (example: fill with mean for numerical columns)
data.fillna(data.mean(numeric_only=True), inplace=True)
# Statistical summary
print(data.describe())
# Correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

```
                id         age          bp          sg          al          su  \
count   400.000000  400.000000  400.000000  400.000000  400.000000  400.000000
mean    199.500000   51.483376   76.469072    1.017408    1.016949    0.450142
std     115.614301   16.974966   13.476298    0.005369    1.272318    1.029487
min       0.000000    2.000000   50.000000    1.005000    0.000000    0.000000
25%      99.750000   42.000000   70.000000    1.015000    0.000000    0.000000
50%     199.500000   54.000000   78.234536    1.017408    1.000000    0.000000
75%     299.250000   64.000000   80.000000    1.020000    2.000000    0.450142
max     399.000000   90.000000  180.000000    1.025000    5.000000    5.000000

               rbc          pc         pcc          ba  ...         pcv  \
count   400.000000  400.000000  400.000000  400.000000  ...  400.000000
mean      0.189516    0.226866    0.106061    0.055556  ...   38.884498
std       0.308983    0.383750    0.306755    0.228199  ...    8.151081
min       0.000000    0.000000    0.000000    0.000000  ...    9.000000
25%       0.000000    0.000000    0.000000    0.000000  ...   34.000000
50%       0.000000    0.000000    0.000000    0.000000  ...   38.884498
75%       0.189516    0.226866    0.000000    0.000000  ...   44.000000
max       1.000000    1.000000    1.000000    1.000000  ...   54.000000

                wc          rc         htn          dm         cad  \
count    400.000000  400.000000  400.000000  400.000000  400.000000
mean    8406.122449    4.707435    0.369347    0.341837    0.085859
std     2523.219976    0.840314    0.482023    0.470146    0.279100
min     2200.000000    2.100000    0.000000    0.000000    0.000000
25%     6975.000000    4.500000    0.000000    0.000000    0.000000
50%     8406.122449    4.707435    0.000000    0.000000    0.000000
75%     9400.000000    5.100000    1.000000    1.000000    0.000000
max    26400.000000    8.000000    1.000000    1.000000    1.000000

            appet          pe         ane  classification
count   400.000000  400.000000  400.000000      400.000000
mean      0.794486    0.190476    0.150376        0.623116
std       0.404077    0.392677    0.357440        0.483998
min       0.000000    0.000000    0.000000        0.000000
25%       1.000000    0.000000    0.000000        0.000000
50%       1.000000    0.000000    0.000000        1.000000
75%       1.000000    0.000000    0.000000        1.000000
max       1.000000    1.000000    1.000000        1.000000

[8 rows x 26 columns]
```



```
# Prepare data for modeling
X = data.drop('classification', axis=1)  # Features
y = data['classification']  # Target variable

# Ensure the target variable is of integer type
y = y.astype(int)

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

# Initialize and train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        52

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80

Confusion Matrix:
[[28  0]
 [ 0 52]]
```

```
!pip install streamlit
```

Show hidden output

```
%%writefile app.py
import pandas as pd
import numpy as np
import streamlit as st
from sklearn.ensemble import RandomForestClassifier

# Load the dataset
data = pd.read_csv('kidney_disease.csv')

# Data preprocessing
data.replace({'?': np.nan, '\t?': np.nan}, inplace=True)
numeric_columns = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc']
for column in numeric_columns:
    data[column] = pd.to_numeric(data[column], errors='coerce')

categorical_columns = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'classification']
for column in categorical_columns:
    data[column] = data[column].map({
        'normal': 0, 'abnormal': 1,
        'notpresent': 0, 'present': 1,
        'yes': 1, 'no': 0,
        'good': 1, 'poor': 0,
        'ckd': 1, 'notckd': 0
    })

data.fillna(data.mean(numeric_only=True), inplace=True)

```

```
# Prepare data for modeling
X = data.drop('classification', axis=1)
y = data['classification'].astype(int)

# Train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X, y)

# Streamlit app
st.title("Kidney Disease Prediction App")

# User input for prediction
st.header("Input Patient Data")
age = st.number_input("Age", min_value=1, max_value=120)
bp = st.number_input("Blood Pressure", min_value=0)
sg = st.number_input("Specific Gravity", min_value=1.0, max_value=1.5, step=0.01)
al = st.number_input("Albumin", min_value=0, max_value=5)
su = st.number_input("Sugar", min_value=0, max_value=5)
rbc = st.selectbox("Red Blood Cells", ["normal", "abnormal"])
pc = st.selectbox("Pus Cells", ["normal", "abnormal"])
pcc = st.selectbox("Pus Cell Clumps", ["notpresent", "present"])
ba = st.selectbox("Bacteria", ["notpresent", "present"])
bgr = st.number_input("Blood Glucose Random", min_value=0)
bu = st.number_input("Blood Urea", min_value=0)
sc = st.number_input("Serum Creatinine", min_value=0)
sod = st.number_input("Sodium", min_value=0)
pot = st.number_input("Potassium", min_value=0)
hemo = st.number_input("Hemoglobin", min_value=0)
pcv = st.number_input("Packed Cell Volume", min_value=0)
wc = st.number_input("White Blood Cell Count", min_value=0)
rc = st.number_input("Red Cell Count", min_value=0)
htn = st.selectbox("Hypertension", ["yes", "no"])
dm = st.selectbox("Diabetes Mellitus", ["yes", "no"])
cad = st.selectbox("Coronary Artery Disease", ["yes", "no"])
appet = st.selectbox("Appetite", ["good", "poor"])
pe = st.selectbox("Pedal Edema", ["yes", "no"])
ane = st.selectbox("Anemia", ["yes", "no"])

# Prepare input data for prediction
input_data = pd.DataFrame({
    'age': [age],
    'bp': [bp],
    'sg': [sg],
    'al': [al],
    'su': [su],
    'rbc': 0 if rbc == "normal" else 1,
    'pc': 0 if pc == "normal" else 1,
    'pcc': 0 if pcc == "notpresent" else 1,
    'ba': 0 if ba == "notpresent" else 1,
    'bgr': [bgr],
    'bu': [bu],
    'sc': [sc],
    'sod': [sod],
    'pot': [pot],
    'hemo': [hemo],
    'pcv': [pcv],
    'wc': [wc],
    'rc': [rc],
    'htn': 1 if htn == "yes" else 0,
    'dm': 1 if dm == "yes" else 0,
    'cad': 1 if cad == "yes" else 0,
    'appet': 1 if appet == "good" else 0,
    'pe': 1 if pe == "yes" else 0,
    'ane': 1 if ane == "yes" else 0
})

# Prediction
if st.button("Predict"):
    prediction = model.predict(input_data)
    prediction_label = "Chronic Kidney Disease" if prediction[0] == 1 else "Not Chronic Kidney Disease"
```

```
    st.success(f"The model predicts: {prediction_label}")
```

```
Overwriting app.py
```

```
import pandas as pd
import numpy as np
import streamlit as st
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Load the dataset
data = pd.read_csv('kidney_disease.csv')

# Data preprocessing
data.replace({'?': np.nan, '\t?': np.nan}, inplace=True)
numeric_columns = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc']
for column in numeric_columns:
    data[column] = pd.to_numeric(data[column], errors='coerce')

categorical_columns = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'classification']
for column in categorical_columns:
    data[column] = data[column].map({
        'normal': 0, 'abnormal': 1,
        'notpresent': 0, 'present': 1,
        'yes': 1, 'no': 0,
        'good': 1, 'poor': 0,
        'ckd': 1, 'notckd': 0
    })

data.fillna(data.mean(numeric_only=True), inplace=True)

# Prepare data for modeling
X = data.drop('classification', axis=1)
y = data['classification'].astype(int)

# Train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X, y)

# Streamlit app
st.title("Kidney Disease Prediction App")

# User input for prediction
st.header("Input Patient Data")
age = st.number_input("Age", min_value=1, max_value=120)
bp = st.number_input("Blood Pressure", min_value=0)
sg = st.number_input("Specific Gravity", min_value=1.0, max_value=1.5, step=0.01)
al = st.number_input("Albumin", min_value=0, max_value=5)
su = st.number_input("Sugar", min_value=0, max_value=5)
rbc = st.selectbox("Red Blood Cells", ["normal", "abnormal"])
pc = st.selectbox("Pus Cells", ["normal", "abnormal"])
pcc = st.selectbox("Pus Cell Clumps", ["notpresent", "present"])
ba = st.selectbox("Bacteria", ["notpresent", "present"])
bgr = st.number_input("Blood Glucose Random", min_value=0)
bu = st.number_input("Blood Urea", min_value=0)
sc = st.number_input("Serum Creatinine", min_value=0)
sod = st.number_input("Sodium", min_value=0)
pot = st.number_input("Potassium", min_value=0)
hemo = st.number_input("Hemoglobin", min_value=0)
pcv = st.number_input("Packed Cell Volume", min_value=0)
wc = st.number_input("White Blood Cell Count", min_value=0)
rc = st.number_input("Red Cell Count", min_value=0)
htn = st.selectbox("Hypertension", ["yes", "no"])
dm = st.selectbox("Diabetes Mellitus", ["yes", "no"])
cad = st.selectbox("Coronary Artery Disease", ["yes", "no"])
appet = st.selectbox("Appetite", ["good", "poor"])
pe = st.selectbox("Pedal Edema", ["yes", "no"])
ane = st.selectbox("Anemia", ["yes", "no"])
```

```
65
66 # Prepare input data for prediction
67 input_data = pd.DataFrame({
68     'age': [age],
69     'bp': [bp],
70     'sg': [sg],
71     'al': [al],
72     'su': [su],
73     'rbc': 0 if rbc == "normal" else 1,
74     'pc': 0 if pc == "normal" else 1,
75     'pcc': 0 if pcc == "notpresent" else 1,
76     'ba': 0 if ba == "notpresent" else 1,
77     'bgr': [bgr],
78     'bu': [bu],
79     'sc': [sc],
80     'sod': [sod],
81     'pot': [pot],
82     'hemo': [hemo],
83     'pcv': [pcv],
84     'wc': [wc],
85     'rc': [rc],
86     'htn': 1 if htn == "yes" else 0,
87     'dm': 1 if dm == "yes" else 0,
88     'cad': 1 if cad == "yes" else 0,
89     'appet': 1 if appet == "good" else 0,
90     'pe': 1 if pe == "yes" else 0,
91     'ane': 1 if ane == "yes" else 0
92 })
93
94 # Prediction
95 if st.button("Predict"):
96     prediction = model.predict(input_data)
97     prediction_label = "Chronic Kidney Disease" if prediction[0] == 1 else "Not Chronic Kidney Disease"
98     st.success(f"The model predicts: {prediction_label}")
```

```
2025-05-16 01:19:27.925 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.934 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.942 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.946 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.950 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.953 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.957 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.960 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.964 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.967 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.972 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.976 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.982 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.984 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.995 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:27.999 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:28.002 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:28.006 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:28.009 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:28.011 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:28.014 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:28.017 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:28.021 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:28.024 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-16 01:19:28.030 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
```