**Columns:**

**Bp** → Blood Pressure

**Sg** → Specific Gravity

**Al** → Albumin

**Su** → Sugar

**Rbc** → Red Blood Cells

**Bu** → Blood Urea

**Sc** → Serum Creatinine

**Sod** → Sodium

**Pot** → Potassium

**Hemo** → Hemoglobin

**Wbcc** → White Blood Cell Count

**Rbcc** → Red Blood Cell Count

**Htn** → Hypertension

#Step1

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

#import pandas as pd

df=pd.read\_csv("C:\\Users\\LENOVO\\Desktop\\kidney\\kidney.csv")

#step2

df.isnull().sum()

#step3

df['Bp'].value\_counts()

#step4

df.columns

#step5

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

#step6

# Finding the columns data types ie continous or discrete

for i in df.columns:

print("\*\*\*\*\*\*\* Column Name is =",i," \*\*\*\*\*\*\*\*\*\*\*\*")

print(df[i].value\_counts())

#step7

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Create a figure with multiple subplots

fig, axes = plt.subplots(2, 3, figsize=(12, 6)) # 1 row, 2 columns

# Plot the first boxplot for 'Sg'

sns.boxplot(ax=axes[0][0], data=df['Sg'], orient='h')

axes[0][0].set\_title('Box Plot for Sg')

# Plot the second boxplot for 'Bu'

sns.boxplot(ax=axes[0][1], data=df['Bu'], orient='h')

axes[0][1].set\_title('Box Plot for Bu')

sns.boxplot(ax=axes[0][2], data=df['Sod'], orient='h')

axes[0][2].set\_title('Box Plot for Sod')

sns.boxplot(ax=axes[1][0], data=df['Pot'], orient='h')

axes[1][0].set\_title('Box Plot for Pot')

sns.boxplot(ax=axes[1][1], data=df['Hemo'], orient='h')

axes[1][1].set\_title('Box Plot for Hemo')

sns.boxplot(ax=axes[1][2], data=df['Rbcc'], orient='h')

axes[1][2].set\_title('Box Plot for Rbcc')

# Adjust layout to avoid overlap

plt.tight\_layout()

# Show the plot

plt.show()

#step8

#finding outlier for other column

sns.boxplot(df.Bu,orient='h')

plt.title("Box Plot")

plt.show()

sns.boxplot(df.Sod,orient='h')

plt.title("Box Plot")

plt.show()

sns.boxplot(df.Hemo,orient='h')

plt.title("Box Plot")

plt.show()

sns.boxplot(df.Rbcc,orient='h')

plt.title("Box Plot")

plt.show()

#step9

from feature\_engine.outliers import Winsorizer

win=Winsorizer(capping\_method='gaussian',tail='both',fold=1.5,variables=['Bu'])

Outlier\_Win\_Gau=win.fit\_transform(df[['Bu']])

#step10

sns.boxplot(data=Outlier\_Win\_Gau['Bu'], orient='h')

plt.show()

#step11

df['Bu']=Outlier\_Win\_Gau['Bu']

#step12

#removing outliers in 'Sod'columns

win=Winsorizer(capping\_method='gaussian',tail='both',fold=1.5,variables=['Sod'])

Outlier\_Win\_Gau=win.fit\_transform(df[['Sod']])

df['Sod']=data=Outlier\_Win\_Gau['Sod']

#step13

sns.boxplot(data=Outlier\_Win\_Gau['Sod'], orient='h')

plt.show()

#step14

#removing outliers in 'hemo'columns

win=Winsorizer(capping\_method='gaussian',tail='both',fold=1.5,variables=['Hemo'])

Outlier\_Win\_Gau=win.fit\_transform(df[['Hemo']])

df['Hemo']=data=Outlier\_Win\_Gau['Hemo']

sns.boxplot(data=Outlier\_Win\_Gau['Hemo'], orient='h')

plt.show()

#step15

#removing outliers for all columns

from feature\_engine.outliers import Winsorizer

win=Winsorizer(capping\_method='gaussian',tail='both',fold=1.5,variables=['Sod'])

Outlier\_Win\_Gau=win.fit\_transform(df[['Sod']])

df['Sod']=data=Outlier\_Win\_Gau['Sod']

win=Winsorizer(capping\_method='gaussian',tail='both',fold=1.5,variables=['Pot'])

Outlier\_Win\_Gau=win.fit\_transform(df[['Pot']])

df['Pot']=data=Outlier\_Win\_Gau['Pot']

win=Winsorizer(capping\_method='gaussian',tail='both',fold=1.5,variables=['Sg'])

Outlier\_Win\_Gau=win.fit\_transform(df[['Sg']])

df['Sg']=data=Outlier\_Win\_Gau['Sg']

win=Winsorizer(capping\_method='gaussian',tail='both',fold=1.5,variables=['Bu'])

Outlier\_Win\_Gau=win.fit\_transform(df[['Bu']])

df['Bu']=data=Outlier\_Win\_Gau['Bu']

win=Winsorizer(capping\_method='gaussian',tail='both',fold=1.5,variables=['Rbcc'])

Outlier\_Win\_Gau=win.fit\_transform(df[['Rbcc']])

df['Rbcc']=data=Outlier\_Win\_Gau['Rbcc']

#step16

fig, axes = plt.subplots(2,3,figsize=(10, 12), sharey=True)

fig.suptitle("After removing Outliers")

sns.boxplot(df['Sg'],orient='h',ax=axes[0,0],color='blue')

axes[0,0].set\_title("")

sns.boxplot(df['Bu'],orient='h',ax=axes[0,1],color='blue')

axes[0,1].set\_title("")

sns.boxplot(df['Pot'],orient='h',ax=axes[0,2],color='blue')

axes[0,2].set\_title("")

sns.boxplot(df['Rbcc'],orient='h',ax=axes[1,0],color='blue')

axes[1,0].set\_title("")

sns.boxplot(df['Sod'],orient='h',ax=axes[1,1],color='blue')

axes[1,1].set\_title("")

sns.boxplot(df['Hemo'],orient='h',ax=axes[1,2],color='blue')

axes[1,2].set\_title("")

plt.subplots\_adjust(hspace=0.3)

plt.savefig("outlier1.png")

plt.show()

#step17

import seaborn as sns

import matplotlib.pyplot as plt

plt.style.use('fivethirtyeight')

plt.figure(figsize=(12, 10))

sns.heatmap(df.corr(),annot=True,fmt = ".2f",cmap='viridis')

plt.show()

#step18

# --- Seperating Dependent Features ---

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

x = df.drop(['Class'], axis=1)

y = df['Class']

# --- Splitting Dataset ---

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

#ExtraTreesClassifier

#step19

from sklearn.ensemble import ExtraTreesClassifier

et= ExtraTreesClassifier()

et.fit(x\_train, y\_train)

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score, recall\_score, f1\_score, classification\_report

ypred\_train=et.predict(x\_train) # Train Prediction

ypred\_test=et.predict(x\_test) # Test Prediction

print("Train Accuracy:",accuracy\_score(y\_train,ypred\_train))

print("Test Accuracy:",accuracy\_score(y\_test,ypred\_test))

# Calculate Precision

precision = precision\_score(y\_test, ypred\_test, average='weighted')

# Calculate Recall

recall = recall\_score(y\_test, ypred\_test, average='weighted')

# Calculate F1 Score

f1 = f1\_score(y\_test, ypred\_test, average='weighted')

# Print the results

print("Precision=",precision)

print("Recall=",recall)

print("F1 Score=",f1)

# Print classification report for a detailed overview

print("\nClassification Report:")

print(classification\_report(y\_test, ypred\_test))

from sklearn.metrics import confusion\_matrix

print("Confusion Matrix is:")

print(confusion\_matrix(y\_test,ypred\_test))

#step20

from sklearn.linear\_model import LogisticRegression

l=LogisticRegression()

l.fit(x\_train,y\_train)

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score, recall\_score, f1\_score, classification\_report

ypred\_train=l.predict(x\_train) # Train Prediction

ypred\_test=l.predict(x\_test) # Test Prediction

print("Train Accuracy:",accuracy\_score(y\_train,ypred\_train))

print("Test Accuracy:",accuracy\_score(y\_test,ypred\_test))

# Calculate Precision

precision = precision\_score(y\_test, ypred\_test, average='weighted')

# Calculate Recall

recall = recall\_score(y\_test, ypred\_test, average='weighted')

# Calculate F1 Score

f1 = f1\_score(y\_test, ypred\_test, average='weighted')

# Print the results

print("Precision=",precision)

print("Recall=",recall)

print("F1 Score=",f1)

# Print classification report for a detailed overview

print("\nClassification Report:")

print(classification\_report(y\_test, ypred\_test))

from sklearn.metrics import confusion\_matrix

print("Confusion Matrix is:")

print(confusion\_matrix(y\_test,ypred\_test))

#step21

from sklearn.linear\_model import LogisticRegression

l=LogisticRegression()

l.fit(x\_train,y\_train)

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score, recall\_score, f1\_score, classification\_report

ypred\_train=l.predict(x\_train) # Train Prediction

ypred\_test=l.predict(x\_test) # Test Prediction

print("Train Accuracy:",accuracy\_score(y\_train,ypred\_train))

print("Test Accuracy:",accuracy\_score(y\_test,ypred\_test))

# Calculate Precision

precision = precision\_score(y\_test, ypred\_test, average='weighted')

# Calculate Recall

recall = recall\_score(y\_test, ypred\_test, average='weighted')

# Calculate F1 Score

f1 = f1\_score(y\_test, ypred\_test, average='weighted')

# Print the results

print("Precision=",precision)

print("Recall=",recall)

print("F1 Score=",f1)

# Print classification report for a detailed overview

print("\nClassification Report:")

print(classification\_report(y\_test, ypred\_test))

from sklearn.metrics import confusion\_matrix

print("Confusion Matrix is:")

print(confusion\_matrix(y\_test,ypred\_test))

#step23

#Decision Tree

from sklearn.tree import DecisionTreeClassifier

d=DecisionTreeClassifier(criterion='entropy',splitter='best')

d.fit(x\_train, y\_train)

ypred\_train=d.predict(x\_train) # Train Prediction

ypred\_test=d.predict(x\_test) # Test Prediction

print("Train Accuracy:",accuracy\_score(y\_train,ypred\_train))

print("Test Accuracy:",accuracy\_score(y\_test,ypred\_test))

# Calculate Precision

precision = precision\_score(y\_test, ypred\_test, average='weighted')

# Calculate Recall

recall = recall\_score(y\_test, ypred\_test, average='weighted')

# Calculate F1 Score

f1 = f1\_score(y\_test, ypred\_test, average='weighted')

# Print the results

print("Precision=",precision)

print("Recall=",recall)

print("F1 Score=",f1)

# Print classification report for a detailed overview

print("\nClassification Report:")

print(classification\_report(y\_test, ypred\_test))

from sklearn.metrics import confusion\_matrix

print("Confusion Matrix is:")

print(confusion\_matrix(y\_test,ypred\_test))

#step24

import matplotlib.pyplot as plt

import numpy as np

algorithms=['ExtraTreesClassifier','LogisticRegression','RandomForestClassifier','StackingClassifier','VotingClassifier','BaggingClassifier','HistGradientBoostingClassifier','DecisionTreeClassifier','KNeighborsClassifier','GaussianNB','AdaBoostClassifier','xgboost','MLPClassifier']

Train\_accuracy=[1,.95,1,1,.99,.99,1,1,.79,.94,1,1,.61]

plt.bar(algorithms,Train\_accuracy,color=["red","green","yellow","orange","blue"])

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

plt.xlabel("Algorithms")

plt.ylabel("Train Accuracy")

plt.title("Train accuracy comparision")

plt.savefig("bar.jpg")

plt.show()

#step25

import matplotlib.pyplot as plt

import numpy as np

algorithms=['ExtraTreesClassifier','LogisticRegression','RandomForestClassifier','StackingClassifier','VotingClassifier','BaggingClassifier','HistGradientBoostingClassifier','DecisionTreeClassifier','KNeighborsClassifier','GaussianNB','AdaBoostClassifier','xgboost','MLPClassifier']

Test\_accuracy=[1,.96,1,.98,1,.98,.98,.98,.73,.96,.98,.98,.65]

plt.bar(algorithms,Test\_accuracy,color=["red","green","yellow","orange","blue"])

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

plt.xlabel("Algorithms")

plt.ylabel("Train Accuracy")

plt.title("Test accuracy comparision")

plt.savefig("bar.jpg")

plt.show()

#step26