National College of Ireland

Enhancing Chronic Kidney Disease Prediction through Machine Learning

MSc Research Project Data Analytics

Revanth Vijay Kumar
Student ID: x21218374

School of Computing
National College of Ireland

Supervisor: Dr. Syed Muslim Jameel

National College of Ireland

Project Submission Sheet

Student Name:	Revanth Vijay Kumar
Student ID:	x21218374
Program:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Syed Muslim Jameel
Submission Due Date:	14/08/2023
Project Title:	Enhancing Chronic Kidney Disease Prediction through Machine Learning
Word Count:	145
Page Count:	27

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Revanth Vijay Kumar
Date:	12/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Revanth Vijay Kumar
X21218374

1. Introduction

This configuration manual is used as supporting document for the research project "Enhancing Chronic Kidney Disease Prediction through Machine Learning". Several machine learning algorithms were constructed, even with its versions were evaluated, and compared to see which one performs better at predicting chronic kidney disease. In addition to highlighting key portions of the code, this document gives an overview of the computational environment utilized to carry out this research and contains some of the graphs and outputs that were created as a result.

2. Specifications

The following sub-sections contain a list of the specifications and prerequisites needed to create and execute the files created for this research project.

2.1 Hardware specifications:

Hardware	Configuration
System	ASUSTek COMPUTER INC.
Processor	Intel® Core™ i7-1065G7 CPU @ 1.30GHz, 1498 Mhz, 4 Cores, 8 Logical Processors
Operating System	Microsoft Windows 10 Home (64 bit)
Installed Physical Memory (RAM)	16 GB
Total Physical Memory	15.7 GB
Available Physical Memory	4.79 GB
Total Virtual Memory	46.7 GB
Available Virtual Memory	30.8 GB
Hard Drive	952 GB
Graphic Card	Intel® Iris® Plus Graphics

2.2 Device Specifications:

OS Name	Microsoft Windows 10 Home
Version	10.0.19044 Build 19044
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	LAPTOP-T35D2GUB
System Manufacturer	ASUSTeK COMPUTER INC.
System Model	ZenBook UX393JA_UX393JA
System Type	x64-based PC
System SKU	
Processor	Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 1498 Mhz, 4 Core(s), 8 Logical Processor(s)
BIOS Version/Date	American Megatrends Inc. UX393JA.304, 28/01/2021
SMBIOS Version	3.2
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	ASUSTeK COMPUTER INC.
BaseBoard Product	UX393JA
BaseBoard Version	1.0
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	United Kingdom
Hardware Abstraction Layer	Version = "10.0.19041.1806"
Username	LAPTOP-T35D2GUB\aggui
Time Zone	GMT Summer Time
Installed Physical Memory (RAM)	16.0 GB
Total Physical Memory	15.7 GB
Available Physical Memory	4.79 GB
Total Virtual Memory	46.7 GB
Available Virtual Memory	30.8 GB
Page File Space	31.0 GB

2.3 Software specifications:

Resources	Specification
Operating System (OS)	Windows 10
Main Memory (RAM)	8GB
Hard disk	256GB SSD and 1TB HDD
Programming Language	Python
Platform	Jupyter Notebook

2.4 Python Libraries:

- Pandas
- NumPy
- Seaborn
- Matplotlib
- IPython.display
- Scikit-learn
- sklearn.preprocessing
- sklearn.impute
- sklearn.model_selection
- sklearn.metrics
- sklearn.neighbors
- Imblearn
- imblearn.over_sampling.

2.5 Code and Output

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import chi2, mutual_info_classif

pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)


df = pd.read_csv('/content/kidney_disease.csv')
# set index to id
df.set_index('id', inplace=True)
display(df)
```

id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4
...	...	...	...	...	...	...	...	...	...	...	...	...
395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	140.0	49.0	0.5
396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	75.0	31.0	1.2
397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	100.0	26.0	0.6
398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	114.0	50.0	1.0
399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	131.0	18.0	1.1

400 rows × 25 columns

```
# check column attribute
columns = df.columns
columns_numerical = ["age", "bp", "sg", "al", "su", "bgr", "bu", "sc", "sod", "pot", "hemo", "pcv", "wc", "rc"]
columns_categorical = ["rbc", "pc", "pcc", "ba", "htn", "dm", "cad", "appet", "pe", "ane", "classification"]

print("attribute Numerical: {}".format(columns_numerical))
print("attribute Categorical: {}".format(columns_categorical))
```

```
attribute Numerical: ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc']
attribute Categorical: ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'classification']
```

```
# check misssing value
missing_value_numerical = df[columns_numerical].isnull().sum()
missing_value_categorical = df[columns_categorical].isnull().sum()

display(missing_value_numerical)
display(missing_value_categorical)
```

```
age      9
bp      12
sg      47
al      46
su      49
bgr     44
bu      19
sc      17
sod     87
pot     88
hemo    52
```

```
# check data type
data_type = df.dtypes
print(data_type)

df['dm'] = df['dm'].replace(to_replace={'\tno':'no', '\tyes':'yes', ' yes':'yes'})
df['cad'] = df['cad'].replace(to_replace='\tno', value='no')
df['classification'] = df['classification'].replace(to_replace='ckd\t', value='ckd')
df['pcv'] = df['pcv'].replace(to_replace={'\t?':np.nan, '\t43':43})
df['wc'] = df['wc'].replace(to_replace={'\t?':np.nan, '\t6200':6200, '\t8400':8400})
df['rc'] = df['rc'].replace(to_replace={'\t?':np.nan, '\t3.9':3.9, '\t4.0':4.0, '\t5.2':5.2})

for i in columns_categorical:
    print(i)
    print(df[i].unique())

for i in columns_numerical:
    print(i)
    print(df[i].unique())
```

```
age               float64
bp                float64
sg                float64
al                float64
su                float64
rbc                object
pc                 object
pcc                object
ba                 object
bgr               float64
bu                float64
sc                float64
sod               float64
pot               float64
hemo              float64
pcv                object
wc                 object
rc                 object
htn                object
dm                 object
cad                object
appet              object
pe                 object
ane                object
classification     object
dtype: object
rbc
[nan 'normal' 'abnormal']
pc
['normal' 'abnormal' nan]
pcc
['notpresent' 'present' nan]
ba
['notpresent' 'present' nan]
htn
['yes' 'no' nan]
dm
['yes' 'no' nan]
cad
['no' 'yes' nan]
appet
['good' 'poor' nan]
pe
['no' 'yes' nan]
ane
['no' 'yes' nan]
classification
['ckd' 'notckd']
age
[48.  7. 62. 51. 60. 68. 24. 52. 53. 50. 63. 40. 47. 61. 21. 42. 75. 69.
 nan 73. 70. 65. 76. 72. 82. 46. 45. 35. 54. 11. 59. 67. 15. 55. 44. 26.
 64. 56.  5. 74. 38. 58. 71. 34. 17. 12. 43. 41. 57.  8. 39. 66. 81. 14.
 27. 83. 30.  4.  3.  6. 32. 80. 49. 90. 78. 19.  2. 33. 36. 37. 23. 25.
 20. 29. 28. 22. 79.]
bp
[ 80.  50.  70.  90.  nan 100.  60. 110. 140. 180. 120.]
sg
```

```
    [1.02  1.01  1.005 1.015   nan 1.025]
```

```
# change df classficication to 0 and 1
df['classification'] = df['classification'].replace(to_replace={'ckd':1, 'notckd':0})


print("Dataset Original")
display(df)
nan_values = df[columns_categorical].isnull().sum()

simpleImputer = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')
df[columns_categorical] = simpleImputer.fit_transform(df[columns_categorical])

print("After being imputed with Categorical")
display(df)

encoder = LabelEncoder()
df[columns_categorical] = df[columns_categorical].apply(encoder.fit_transform)

print("After being encoded")
display(df)

imputer = KNNImputer(n_neighbors=5)
df[columns_numerical] = imputer.fit_transform(df[columns_numerical])

print("After being imputed with Numerical data")
display(df)
```

Dataset Original

id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hem
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	NaN	NaN	15
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN	11
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN	NaN	9
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5	11
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN	NaN	11
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	140.0	49.0	0.5	150.0	4.9	15
396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	75.0	31.0	1.2	141.0	3.5	16
397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	100.0	26.0	0.6	137.0	4.4	15
398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	114.0	50.0	1.0	135.0	4.9	14
399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	131.0	18.0	1.1	141.0	3.5	15

400 rows × 25 columns

After being imputed with Categorical

id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hem
0	48.0	80.0	1.020	1.0	0.0	normal	normal	notpresent	notpresent	121.0	36.0	1.2	NaN	NaN	15
1	7.0	50.0	1.020	4.0	0.0	normal	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN	11
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN	NaN	9
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5	11
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN	NaN	11
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	140.0	49.0	0.5	150.0	4.9	15
396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	75.0	31.0	1.2	141.0	3.5	16

```
# plot label target (classification)
value_1_classification = df[df['classification'] == 1].shape[0]
value_0_classification = df[df['classification'] == 0].shape[0]
# change df classficication to 0 and 1
print("data classification 1 (kidney disease): {}".format(value_1_classification))
print(" data classification 0 (not kidney disease){}".format(value_0_classification))
```

```
data classification 1 (kidney disease): 250
 data classification 0 (not kidney disease)150
```

id

```
# check missing value
missing_value_numerical = df[columns_numerical].isnull().sum()
missing_value_categorical = df[columns_categorical].isnull().sum()

print("Missing Value Numerical: {}".format(missing_value_numerical))
print("Missing Value Categorical: {}".format(missing_value_categorical))
```

```
Missing Value Numerical: age     0
bp      0
sg      0
al      0
su      0
bgr     0
bu      0
sc      0
sod     0
pot     0
hemo    0
pcv     0
wc      0
rc      0
dtype: int64
Missing Value Categorical: rbc               0
pc                0
pcc               0
ba                0
htn               0
dm                0
cad               0
appet             0
```

```
pe                0
ane               0
classification    0
dtype: int64
```

```
X = df.drop(columns=['classification'])
y = df['classification']
```

```
X
```

id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	ht
0	48.0	80.0	1.020	1.0	0.0	1	1	0	0	121.0	36.0	1.2	137.6	4.20	15.4	44.0	7800.0	5.20	
1	7.0	50.0	1.020	4.0	0.0	1	1	0	0	113.0	18.0	0.8	137.4	4.00	11.3	38.0	6000.0	4.96	
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	53.0	1.8	133.8	4.20	9.6	31.0	7500.0	3.80	
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	56.0	3.8	111.0	2.50	11.2	32.0	6700.0	3.90	
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	26.0	1.4	138.4	3.98	11.6	35.0	7300.0	4.60	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
395	55.0	80.0	1.020	0.0	0.0	1	1	0	0	140.0	49.0	0.5	150.0	4.90	15.7	47.0	6700.0	4.90	
396	42.0	70.0	1.025	0.0	0.0	1	1	0	0	75.0	31.0	1.2	141.0	3.50	16.5	54.0	7800.0	6.20	
397	12.0	80.0	1.020	0.0	0.0	1	1	0	0	100.0	26.0	0.6	137.0	4.40	15.8	49.0	6600.0	5.40	
398	17.0	60.0	1.025	0.0	0.0	1	1	0	0	114.0	50.0	1.0	135.0	4.90	14.2	51.0	7200.0	5.90	
399	58.0	80.0	1.025	0.0	0.0	1	1	0	0	131.0	18.0	1.1	141.0	3.50	15.8	53.0	6800.0	6.10	

400 rows × 24 columns

```
from imblearn.over_sampling import SMOTE

num_additional_samples = 50

# Calculate the target number of samples after augmentation
target_samples = len(y) + num_additional_samples

# Calculate the sampling strategy to achieve the target number of samples
sampling_strategy = {class_label: target_samples for class_label in set(y)}

# Apply SMOTE to generate the specified number of additional synthetic samples
smote = SMOTE(sampling_strategy=sampling_strategy, random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
/usr/local/lib/python3.10/dist-packages/imblearn/utils/_validation.py:313: UserWarning: After over-sampling, the number of samples
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/imblearn/utils/_validation.py:313: UserWarning: After over-sampling, the number of samples
  warnings.warn(
```

```
X = X_resampled
y = y_resampled
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(" X_train: {}".format(X_train.shape))
print(" X_test: {}".format(X_test.shape))
print(" y_train: {}".format(y_train.shape))
print(" y_test: {}".format(y_test.shape))


print("Dataset X_train")
display(X_train)
print("Dataset X_test")
display(X_test)
print("Dataset y_train")
display(y_train)
print("Dataset y_test")
display(y_test)
```

```
 X_train: (720, 24)
 X_test: (180, 24)
 y_train: (720,)
 y_test: (180,)
Dataset X_train
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	
10	50.000000	60.000000	1.01000	2.000000	4.000000	1	0	1	0	490.000000	55.000000	4.0000
334	24.000000	80.000000	1.02500	0.000000	0.000000	1	1	0	0	125.000000	28.200000	0.8800
244	64.000000	90.000000	1.01500	3.000000	2.000000	1	0	1	0	463.000000	64.000000	2.8000
678	43.181836	69.090821	1.02000	0.000000	0.000000	1	1	0	0	117.636329	25.545459	0.7454
306	52.000000	80.000000	1.02000	0.000000	0.000000	1	1	0	0	128.000000	30.000000	1.2000
...	...	...	...	...	...	...	...	...	...	...	...	
106	50.000000	90.000000	1.01400	1.600000	0.400000	1	1	0	0	89.000000	118.000000	6.1000
270	23.000000	80.000000	1.02500	0.000000	0.000000	1	1	0	0	111.000000	34.000000	1.1000
860	46.686736	60.460026	1.01477	2.861992	0.092005	0	0	0	0	153.029385	35.564090	1.2736
435	34.874672	61.771107	1.02000	0.000000	0.000000	1	1	0	0	105.280676	46.114447	0.9937
102	17.000000	60.000000	1.01000	0.000000	0.000000	1	1	0	0	92.000000	32.000000	2.1000

720 rows × 24 columns

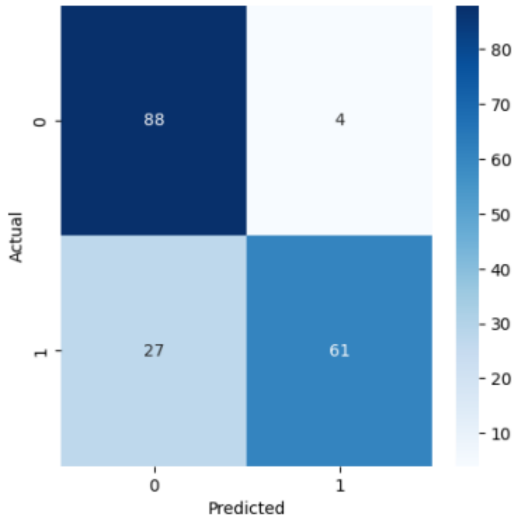
```
Dataset X_test
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	
70	61.000000	80.000000	1.015000	0.000000	4.000000	1	1	0	0	360.000000	19.000000	0.70
827	69.000000	65.313546	1.014686	0.281187	2.312877	1	1	0	0	216.165144	33.438965	23.17
231	60.000000	90.000000	1.015000	1.400000	1.000000	1	1	0	0	269.000000	51.000000	2.80
588	59.791747	67.440362	1.020000	0.000000	0.000000	1	1	0	0	103.419729	30.071566	1.07
39	82.000000	80.000000	1.010000	2.000000	2.000000	1	1	0	0	140.000000	70.000000	3.40
...	...	...	...	...	...	...	...	...	...	...	...	
897	74.127553	79.635563	1.010073	0.032799	0.000000	1	1	0	0	132.291549	96.924912	2.78
578	44.144714	63.710572	1.023145	0.000000	0.000000	1	1	0	0	123.742114	43.031543	0.85
779	69.781546	80.000000	1.015683	0.683078	0.273231	1	1	0	0	130.027716	54.541574	5.96
25	61.000000	60.000000	1.025000	0.000000	0.000000	1	1	0	0	108.000000	75.000000	1.90
84	59.000000	70.000000	1.010000	3.000000	0.000000	1	0	0	0	76.000000	186.000000	15.00

180 rows × 24 columns

```
model_knn_original = KNeighborsClassifier(n_neighbors=5)
model_knn_original.fit(X_train, y_train)

y_pred = model_knn_original.predict(X_test)

df_result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

```
cm_knn_original = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm_knn_original, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
# classification report
cr_knn_original = classification_report(y_test, y_pred)
print(cr_knn_original)
print(model_knn_original.score(X_train, y_train))
```

```
              precision    recall  f1-score   support

           0       0.77      0.96      0.85        92
           1       0.94      0.69      0.80        88

    accuracy                           0.83       180
   macro avg       0.85      0.82      0.82       180
weighted avg       0.85      0.83      0.82       180

0.9
```

```
cv_knn_original = cross_val_score(model_knn_original, X_train, y_train, cv=5)
cv_knn_original_precision = cross_val_score(model_knn_original, X_train, y_train, cv=5, scoring='precision')
cv_knn_original_recall = cross_val_score(model_knn_original, X_train, y_train, cv=5, scoring='recall')
```

```
cv_knn_original_f1 = cross_val_score(model_knn_original, X_train, y_train, cv=5, scoring='f1')
# check data cv each fold
print("Cross Validation Score Accuracy: {}".format(cv_knn_original))
print("Cross Validation Score Accuracy Mean: {}".format(cv_knn_original.mean()))
```

```
Cross Validation Score Accuracy: [0.81944444 0.85416667 0.77777778 0.82638889 0.78472222]
Cross Validation Score Accuracy Mean: 0.8125
```

```
# feature scaling with minmaxscaler

scaler = MinMaxScaler()
print("Dataset before Scaling")
display(X)
X[columns_numerical] = scaler.fit_transform(X[columns_numerical])
print("Dataset after Scaling")
display(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(" X_train: {}".format(X_train.shape))
print(" X_test: {}".format(X_test.shape))
print(" y_train: {}".format(y_train.shape))
print(" y_test: {}".format(y_test.shape))
```

Dataset before Scaling

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc
0	48.000000	80.000000	1.020000	1.000000	0.000000	1	1	0	0	121.000000	36.000000	1.200000
1	7.000000	50.000000	1.020000	4.000000	0.000000	1	1	0	0	113.000000	18.000000	0.800000
2	62.000000	80.000000	1.010000	2.000000	3.000000	1	1	0	0	423.000000	53.000000	1.800000
3	48.000000	70.000000	1.005000	4.000000	0.000000	1	0	1	0	117.000000	56.000000	3.800000
4	51.000000	80.000000	1.010000	2.000000	0.000000	1	1	0	0	106.000000	26.000000	1.400000
...	...	...	...	...	...	...	...	...	...	...	...	...
895	38.930639	70.000000	1.010208	0.097106	0.000000	1	1	0	0	119.820858	23.953594	0.876297
896	9.728786	62.015560	1.010504	3.798444	0.302334	0	0	0	1	104.118111	63.271214	1.010078
897	74.127553	79.635563	1.010073	0.032799	0.000000	1	1	0	0	132.291549	96.924912	2.787245
898	65.544483	72.833289	1.005944	1.283329	0.000000	0	0	0	0	205.613412	30.077745	1.400000
899	52.709156	65.849661	1.010000	0.000000	0.000000	0	1	0	0	222.960813	45.637264	1.914706

900 rows × 24 columns

Dataset after Scaling

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	
0	0.522727	0.230769	0.750000	0.200000	0.000000	1	1	0	0	0.211538	0.088575	0.010582	0.839
1	0.056818	0.000000	0.750000	0.800000	0.000000	1	1	0	0	0.194444	0.042362	0.005291	0.838
2	0.681818	0.230769	0.250000	0.400000	0.600000	1	1	0	0	0.856838	0.132221	0.018519	0.815
3	0.522727	0.153846	0.000000	0.800000	0.000000	1	0	1	0	0.202991	0.139923	0.044974	0.671
4	0.556818	0.230769	0.250000	0.400000	0.000000	1	1	0	0	0.179487	0.062901	0.013228	0.844
...	...	...	...	...	...	...	...	...	...	...	...	...	
895	0.419666	0.153846	0.260404	0.019421	0.000000	1	1	0	0	0.209019	0.057647	0.006300	0.813
896	0.087827	0.092427	0.275195	0.759689	0.060467	0	0	0	1	0.175466	0.158591	0.008070	0.825
897	0.819631	0.227966	0.253644	0.006560	0.000000	1	1	0	0	0.235666	0.244993	0.031577	0.812
898	0.722096	0.175641	0.047221	0.256666	0.000000	0	0	0	0	0.392336	0.073370	0.013228	0.833
899	0.576240	0.121920	0.250000	0.000000	0.000000	0	1	0	0	0.429403	0.113318	0.020036	0.744

900 rows × 24 columns

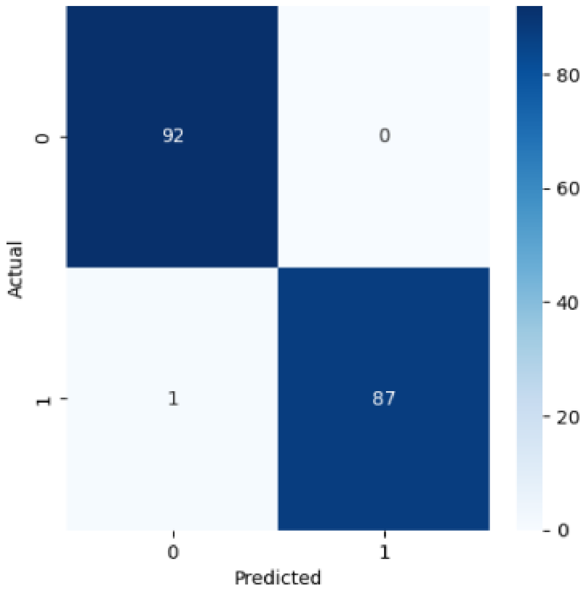
```
model_knn_scaling = KNeighborsClassifier(n_neighbors=5)
model_knn_scaling.fit(X_train, y_train)

y_pred = model_knn_scaling.predict(X_test)

df_result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
display(df_result)
```

	Actual	Predicted
70	1	1
827	1	1
231	1	1
588	0	0
39	1	1
...	...	...
897	1	1
578	0	0
779	1	1
25	1	1
84	1	1

180 rows × 2 columns

```
cm_knn_scaling = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm_knn_scaling, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.show()
```




```
cr_knn_scaling = classification_report(y_test, y_pred)
print(cr_knn_scaling)
print(model_knn_scaling.score(X_train, y_train))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        92
           1       1.00      0.99      0.99        88

    accuracy                           0.99       180
   macro avg       0.99      0.99      0.99       180
weighted avg       0.99      0.99      0.99       180

0.9944444444444445
```

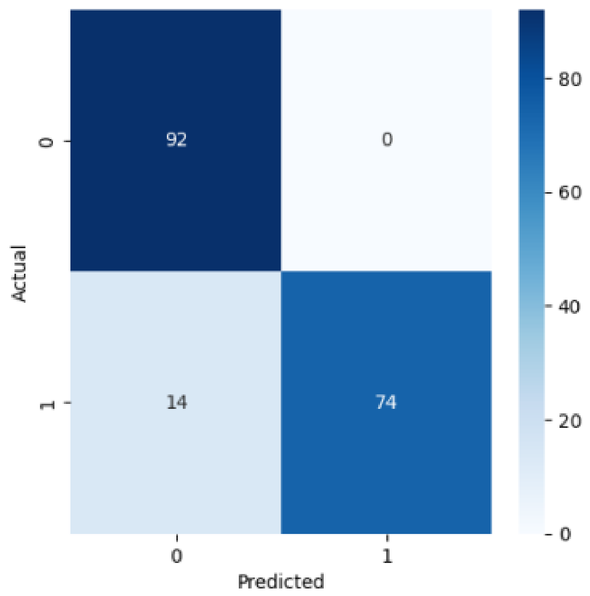
```
cv_knn_scaling = cross_val_score(model_knn_scaling, X_train, y_train, cv=5)
cv_knn_scaling_precision = cross_val_score(model_knn_scaling, X_train, y_train, cv=5, scoring='precision')
cv_knn_scaling_recall = cross_val_score(model_knn_scaling, X_train, y_train, cv=5, scoring='recall')
cv_knn_scaling_f1 = cross_val_score(model_knn_scaling, X_train, y_train, cv=5, scoring='f1')
print("Cross Validation Score Accuracy: {}".format(cv_knn_scaling))
print("Cross Validation Score Accuracy Mean: {}".format(cv_knn_scaling.mean()))
```

```
Cross Validation Score Accuracy: [0.99305556 0.97916667 0.99305556 0.99305556 1.        ]
Cross Validation Score Accuracy Mean: 0.9916666666666666
```

```
from sklearn.naive_bayes import BernoulliNB, GaussianNB, MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import chi2, mutual_info_classif
from sklearn.decomposition import PCA


model_bnb_original = GaussianNB()
model_bnb_original.fit(X_train, y_train)

y_pred = model_bnb_original.predict(X_test)

df_result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
display(df_result)
```

70	1	1
827	1	1
231	1	1

```
cm_bnb_original = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm_bnb_original, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
cr_bnb_original = classification_report(y_test, y_pred)
print(cr_bnb_original)
print(model_bnb_original.score(X_train, y_train))
```

```
              precision    recall  f1-score   support

           0       0.87      1.00      0.93        92
           1       1.00      0.84      0.91        88

    accuracy                           0.92       180
   macro avg       0.93      0.92      0.92       180
weighted avg       0.93      0.92      0.92       180

0.9527777777777777
```

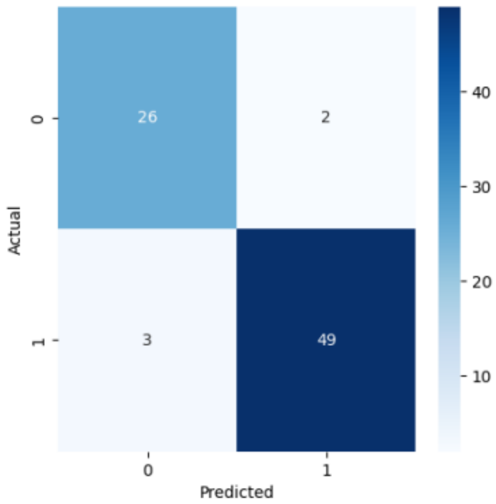
```
cv_bnb_original = cross_val_score(model_bnb_original, X_train, y_train, cv=5)
cv_bnb_original_precision = cross_val_score(model_bnb_original, X_train, y_train, cv=5, scoring='precision')
cv_bnb_original_recall = cross_val_score(model_bnb_original, X_train, y_train, cv=5, scoring='recall')
cv_bnb_original_f1 = cross_val_score(model_bnb_original, X_train, y_train, cv=5, scoring='f1')
print("Cross Validation Score Accuracy: {}".format(cv_bnb_original))
print("Cross Validation Score Accuracy Mean: {}".format(cv_bnb_original.mean()))
```

```
Cross Validation Score Accuracy: [0.96527778 0.96527778 0.97222222 0.9375     0.92361111]
Cross Validation Score Accuracy Mean: 0.9527777777777778
```

```python
model_dt_original = DecisionTreeClassifier(max_depth=5)
model_dt_original.fit(X_train, y_train)

y_pred = model_dt_original.predict(X_test)

df_result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
display(df_result)
```

id		
209	1	1
280	0	1
33	1	1
210	1	1
93	1	1
...	...	...
246	1	1

```python
cm_dt_original = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm_dt_original, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
cr_dt_original = classification_report(y_test, y_pred)
print(cr_dt_original)
print(model_bnb_original.score(X_train, y_train))
```

```
              precision    recall  f1-score   support

           0       0.90      0.93      0.91        28
           1       0.96      0.94      0.95        52

    accuracy                           0.94        80
   macro avg       0.93      0.94      0.93        80
weighted avg       0.94      0.94      0.94        80

0.934375
```

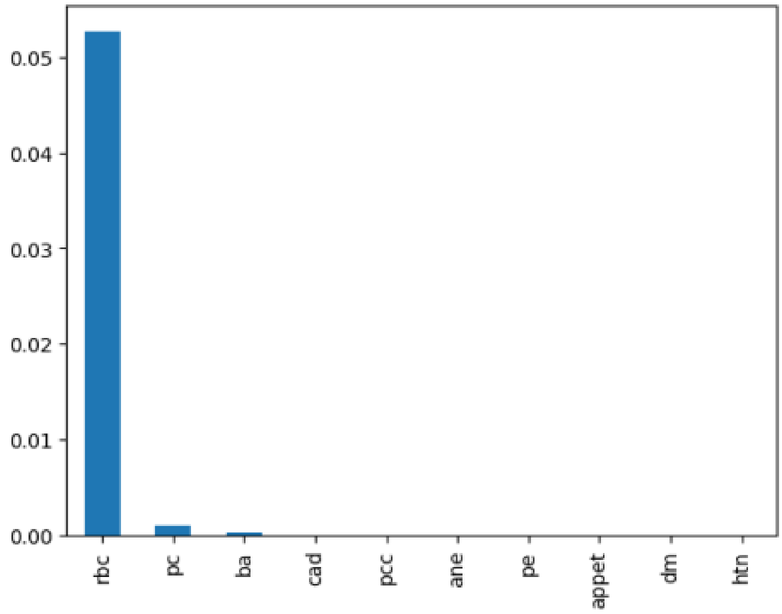
```
cv_dt_original = cross_val_score(model_dt_original, X_train, y_train, cv=5)
cv_dt_original_precision = cross_val_score(model_dt_original, X_train, y_train, cv=5, scoring='precision')
cv_dt_original_recall = cross_val_score(model_dt_original, X_train, y_train, cv=5, scoring='recall')
cv_dt_original_f1 = cross_val_score(model_dt_original, X_train, y_train, cv=5, scoring='f1')
print("Cross Validation Score Accuracy: {}".format(cv_dt_original))
print("Cross Validation Score Accuracy Mean: {}".format(cv_dt_original.mean()))
```

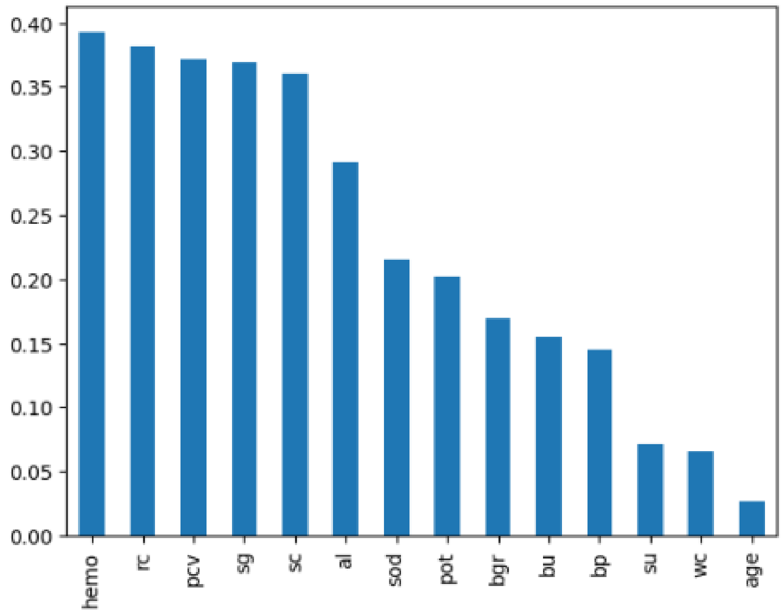
```
Cross Validation Score Accuracy: [0.96875  0.96875  0.953125 0.9375   0.9375  ]
Cross Validation Score Accuracy Mean: 0.953125
```

```
# Feature Selection
# Chi-square is used to determine the relationship between two categorical variables
# column_categorical without classication
column_categorical_without_classification = columns_categorical.copy()
column_categorical_without_classification.remove('classification')
X_column_categorical = X[column_categorical_without_classification]
y_column_categorical = y

X_column_numerical = X[columns_numerical]
```

[redacted]

```
chi_scores = chi2(X_column_categorical, y_column_categorical)
p_value = pd.Series(chi_scores[1], index = X_column_categorical.columns)
p_value.sort_values(ascending = False, inplace = True)
p_value.plot.bar()
plt.show()


# Mutual information is used to measure the relationship between a numeric attribute and a categorical attribute.
mutual_infomation = mutual_info_classif(X_column_numerical, y_column_categorical)
mutual_infomation = pd.Series(mutual_infomation, index = X_column_numerical.columns)
mutual_infomation.sort_values(ascending = False, inplace = True)
mutual_infomation.plot.bar()
plt.show()

list_chi =  p_value[p_value > 0.05].index.tolist()
list_mutual = mutual_infomation[mutual_infomation < 0.1].index.tolist()
print("List Chi Square: {}".format(list_chi))
print("List Mutual Information: {}".format(list_mutual))
```






```
List Chi Square: ['rbc']
List Mutual Information: ['su', 'wc', 'age']
```

```
# drop columns
list_drop = list_chi + list_mutual
print("List Drop: {}".format(list_drop))
X = X.drop(columns=list_drop)
display(X)
```

```
List Drop: ['rbc', 'su', 'wc', 'age']
```

id	bp	sg	al	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv
0	0.230769	0.75	0.2	1	0	0	0.211538	0.088575	0.010582	0.839748	0.038202	0.836735	0.777778
1	0.000000	0.75	0.8	1	0	0	0.194444	0.042362	0.005291	0.838486	0.033708	0.557823	0.644444
2	0.230769	0.25	0.4	1	0	0	0.856838	0.132221	0.018519	0.815773	0.038202	0.442177	0.488889
3	0.153846	0.00	0.8	0	1	0	0.202991	0.139923	0.044974	0.671924	0.000000	0.551020	0.511111
4	0.230769	0.25	0.4	1	0	0	0.179487	0.062901	0.013228	0.844795	0.033258	0.578231	0.577778
...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	0.230769	0.75	0.0	1	0	0	0.252137	0.121951	0.001323	0.917981	0.053933	0.857143	0.844444
396	0.153846	1.00	0.0	1	0	0	0.113248	0.075738	0.010582	0.861199	0.022472	0.911565	1.000000
397	0.230769	0.75	0.0	1	0	0	0.166667	0.062901	0.002646	0.835962	0.042697	0.863946	0.888889
398	0.076923	1.00	0.0	1	0	0	0.196581	0.124519	0.007937	0.823344	0.053933	0.755102	0.933333
399	0.230769	1.00	0.0	1	0	0	0.232906	0.042362	0.009259	0.861199	0.022472	0.863946	0.977778

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Dimensi X_train: {}".format(X_train.shape))
print("Dimensi X_test: {}".format(X_test.shape))
print("Dimensi y_train: {}".format(y_train.shape))
print("Dimensi y_test: {}".format(y_test.shape))
```

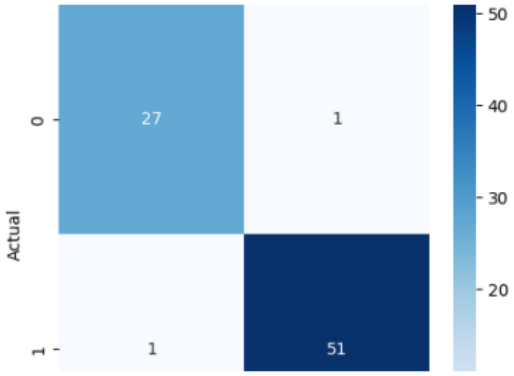
```
Dimensi X_train: (320, 20)
Dimensi X_test: (80, 20)
Dimensi y_train: (320,)
Dimensi y_test: (80,)
```

```
model_knn_feature_selection = KNeighborsClassifier(n_neighbors=5)
model_knn_feature_selection.fit(X_train, y_train)

y_pred = model_knn_feature_selection.predict(X_test)

df_result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
display(df_result)
```

id	Actual	Predicted
209	1	0
280	0	1
33	1	1
210	1	1
93	1	1
...	...	...
246	1	1
227	1	1
369	0	0
176	1	1
289	0	0

80 rows × 2 columns

```
cm_knn_feature_selection = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm_knn_feature_selection, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
cr_knn_feature_selection = classification_report(y_test, y_pred)

print(cr_knn_feature_selection)
print(model_knn_feature_selection.score(X_train, y_train))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        28
           1       0.98      0.98      0.98        52

    accuracy                           0.97        80
   macro avg       0.97      0.97      0.97        80
weighted avg       0.97      0.97      0.97        80

0.99375
```

```
cv_knn_feature_selection = cross_val_score(model_knn_feature_selection, X_train, y_train, cv=5)
cv_knn_feature_selection_precision = cross_val_score(model_knn_feature_selection, X_train, y_train, cv=5, scoring='precision')
cv_knn_feature_selection_recall = cross_val_score(model_knn_feature_selection, X_train, y_train, cv=5, scoring='recall')
cv_knn_feature_selection_f1 = cross_val_score(model_knn_feature_selection, X_train, y_train, cv=5, scoring='f1')
print("Cross Validation Score Accuracy: {}".format(cv_knn_feature_selection))
print("Cross Validation Score Accuracy Mean: {}".format(cv_knn_feature_selection.mean()))
```

```
Cross Validation Score Accuracy: [1.      0.96875 0.9375  1.      1.     ]
Cross Validation Score Accuracy Mean: 0.98125
```

```
# Dimensionality Reduction

pca = PCA(n_components=3)
pca.fit(X)
print("Data PCA")
display(X)
X_pca = pca.transform(X)
print("Data PCA")
df_X_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2', 'PC3'])
display(df_X_pca)
```

Data PCA

id	bp	sg	al	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv
0	0.230769	0.75	0.2	1	0	0	0.211538	0.088575	0.010582	0.839748	0.038202	0.836735	0.777778
1	0.000000	0.75	0.8	1	0	0	0.194444	0.042362	0.005291	0.838486	0.033708	0.557823	0.644444
2	0.230769	0.25	0.4	1	0	0	0.856838	0.132221	0.018519	0.815773	0.038202	0.442177	0.488889
3	0.153846	0.00	0.8	0	1	0	0.202991	0.139923	0.044974	0.671924	0.000000	0.551020	0.511111
4	0.230769	0.25	0.4	1	0	0	0.179487	0.062901	0.013228	0.844795	0.033258	0.578231	0.577778
...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	0.230769	0.75	0.0	1	0	0	0.252137	0.121951	0.001323	0.917981	0.053933	0.857143	0.844444
396	0.153846	1.00	0.0	1	0	0	0.113248	0.075738	0.010582	0.861199	0.022472	0.911565	1.000000
397	0.230769	0.75	0.0	1	0	0	0.166667	0.062901	0.002646	0.835962	0.042697	0.863946	0.888889
398	0.076923	1.00	0.0	1	0	0	0.196581	0.124519	0.007937	0.823344	0.053933	0.755102	0.933333
399	0.230769	1.00	0.0	1	0	0	0.232906	0.042362	0.009259	0.861199	0.022472	0.863946	0.977778

```
X = df_X_pca
y = df['classification']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Dimensi X_train: {}".format(X_train.shape))
print("Dimensi X_test: {}".format(X_test.shape))
print("Dimensi y_train: {}".format(y_train.shape))
print("Dimensi y_test: {}".format(y_test.shape))
```

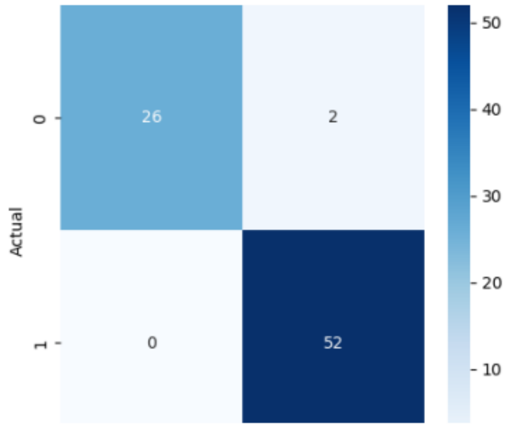
```
Dimensi X_train: (320, 3)
Dimensi X_test: (80, 3)
Dimensi y_train: (320,)
Dimensi y_test: (80,)
 ...     ...     ...     ...
```

```
model_knn_pca = KNeighborsClassifier(n_neighbors=5)
model_knn_pca.fit(X_train, y_train)

y_pred = model_knn_pca.predict(X_test)

df_result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
display(df_result)
```

id	Actual	Predicted
209	1	1
280	0	1
33	1	1
210	1	1
93	1	1
...	...	...
246	1	1
227	1	1
369	0	0
176	1	1
289	0	0

80 rows × 2 columns

```
cm_knn_pca = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm_knn_pca, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
cr_knn_pca = classification_report(y_test, y_pred)
print(cr_knn_pca)
print(model_knn_pca.score(X_train, y_train))
```

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.96        28
           1       0.96      1.00      0.98        52

    accuracy                           0.97        80
   macro avg       0.98      0.96      0.97        80
weighted avg       0.98      0.97      0.97        80

0.99375
```

```
cv_knn_pca = cross_val_score(model_knn_pca, X_train, y_train, cv=5)
cv_knn_pca_precision = cross_val_score(model_knn_pca, X_train, y_train, cv=5, scoring='precision')
cv_knn_pca_recall = cross_val_score(model_knn_pca, X_train, y_train, cv=5, scoring='recall')
cv_knn_pca_f1 = cross_val_score(model_knn_pca, X_train, y_train, cv=5, scoring='f1')
print("Cross Validation Score Accuracy: {}".format(cv_knn_pca))
print("Cross Validation Score Accuracy Mean: {}".format(cv_knn_pca.mean()))
```

```
Cross Validation Score Accuracy: [1.       0.984375 0.984375 0.984375 1.      ]
Cross Validation Score Accuracy Mean: 0.990625
```

```
model_bnb_pca = GaussianNB()
model_bnb_pca.fit(X_train, y_train)
```

```
y_pred = model_bnb_pca.predict(X_test)
```

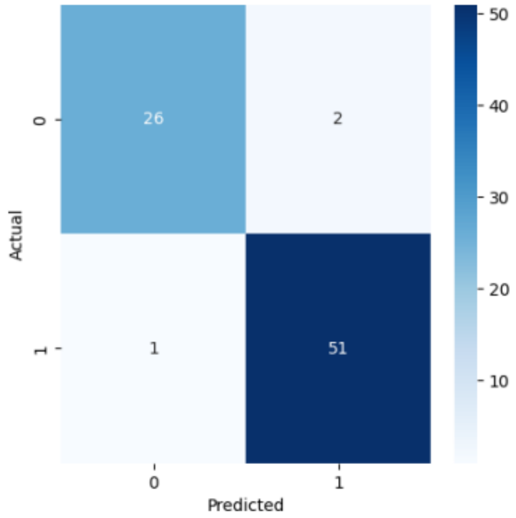
```
df_result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
display(df_result)
```

id	Actual	Predicted
209	1	0
280	0	1
33	1	1
210	1	1
93	1	1
...	...	...
246	1	1
227	1	1
369	0	0
176	1	1
289	0	0

80 rows × 2 columns

```
cm_bnb_pca = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm_bnb_pca, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.show()
```




```
cr_bnb_pca = classification_report(y_test, y_pred)
print(cr_bnb_pca)
print(model_bnb_pca.score(X_train, y_train))
```

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.95        28
           1       0.96      0.98      0.97        52

    accuracy                           0.96        80
   macro avg       0.96      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80

0.978125
```

```
cv_bnb_pca = cross_val_score(model_bnb_pca, X_train, y_train, cv=5)
cv_bnb_pca_precision = cross_val_score(model_bnb_pca, X_train, y_train, cv=5, scoring='precision')
cv_bnb_pca_recall = cross_val_score(model_bnb_pca, X_train, y_train, cv=5, scoring='recall')
cv_bnb_pca_f1 = cross_val_score(model_bnb_pca, X_train, y_train, cv=5, scoring='f1')
print("Cross Validation Score Accuracy: {}".format(cv_bnb_pca))
print("Cross Validation Score Accuracy Mean: {}".format(cv_bnb_pca.mean()))
```

```
Cross Validation Score Accuracy: [1.       0.96875  0.984375 0.984375 0.96875 ]
Cross Validation Score Accuracy Mean: 0.98125
```

```
df_result_acc_precision_recall_knn_original = pd.DataFrame({'Accuracy': [cr_knn_original.split()[15]], 'Precision': [cr_knn_original.spli
df_result_acc_precision_recall_knn_cv = pd.DataFrame({'Accuracy': [cv_knn_original.mean()], 'Precision': [cv_knn_original_precision.mean(
df_result_acc_precision_recall_bnb_original = pd.DataFrame({'Accuracy': [cr_bnb_original.split()[15]], 'Precision': [cr_bnb_original.spli
df_result_acc_precision_recall_bnb_cv = pd.DataFrame({'Accuracy': [cv_bnb_original.mean()], 'Precision': [cv_bnb_original_precision.mean(
df_result_acc_precision_recall_dt_original = pd.DataFrame({'Accuracy': [cr_dt_original.split()[15]], 'Precision': [cr_dt_original.split()
df_result_acc_precision_recall_dt_cv = pd.DataFrame({'Accuracy': [cv_dt_original.mean()], 'Precision': [cv_dt_original_precision.mean()],
df_result_acc_precision_recall_knn_scaling = pd.DataFrame({'Accuracy': [cr_knn_scaling.split()[15]], 'Precision': [cr_knn_scaling.split()
df_result_acc_precision_recall_knn_scaling_cv = pd.DataFrame({'Accuracy': [cv_knn_scaling.mean()], 'Precision': [cv_knn_scaling_precision
df_result_acc_precision_recall_knn_feature_selection = pd.DataFrame({'Accuracy': [cr_knn_feature_selection.split()[15]], 'Precision': [cr
df_result_acc_precision_recall_knn_pca = pd.DataFrame({'Accuracy': [cr_knn_pca.split()[15]], 'Precision': [cr_knn_pca.split()[25]], 'Reca
df_result_acc_precision_recall_knn_pca_cv = pd.DataFrame({'Accuracy': [cv_knn_pca.mean()], 'Precision': [cv_knn_pca_precision.mean()], 'R
df_result_acc_precision_recall_bnb_pca = pd.DataFrame({'Accuracy': [cr_bnb_pca.split()[15]], 'Precision': [cr_bnb_pca.split()[25]], 'Reca
df_result_acc_precision_recall_bnb_pca_cv = pd.DataFrame({'Accuracy': [cv_bnb_pca.mean()], 'Precision': [cv_bnb_pca_precision.mean()], 'R
```

```
df_result_acc_precision_recall = pd.concat([df_result_acc_precision_recall_knn_original, df_result_acc_precision_recall_knn_cv, df_result
```

```
df_result_acc_precision_recall.index = ['KNN Original', 'KNN CV', 'BNB Original', 'BNB CV', 'DT Original', 'DT CV', 'KNN Scaling', 'KNN S
```

```
df_result_acc_precision_recall['Accuracy'] = pd.to_numeric(df_result_acc_precision_recall['Accuracy'], errors='coerce')
```

```
df_result_acc_precision_recall.sort_values(by=['Accuracy'], inplace=True, ascending=False)
display(df_result_acc_precision_recall)
```

Result:

Model_Name	Accuracy	Precision	Recall	F1-Score
KNN PCA CV	0.990625	0.990122	0.995000	0.992468
BNB Original	0.990000	0.990000	0.990000	0.990000
BNB PCA CV	0.981250	0.990122	0.979744	0.984608
KNN Scaling CV	0.971875	0.994737	0.959872	0.976822
KNN Scaling	0.970000	0.970000	0.970000	0.970000
KNN Feature Selection	0.970000	0.970000	0.970000	0.970000
KNN PCA	0.970000	0.980000	0.970000	0.970000
BNB PCA	0.960000	0.960000	0.960000	0.960000
DT CV	0.953125	0.970060	0.959744	0.956764
DT Original	0.940000	0.940000	0.940000	0.940000
BNB CV	0.934375	0.994737	0.899103	0.943962
KNN Original	0.730000	0.730000	0.720000	0.730000