https://data.mendeley.com/datasets/wj9rwkp9c2/1 Rashid, Ahlam (2020), “Diabetes Dataset”, Mendeley Data, V1, doi: 10.17632/wj9rwkp9c2.1

```
!wget https://data.mendeley.com/public-files/datasets/wj9rwkp9c2/files/2eb60cac-96b8-46ea-b971-6415e972afc9/file_downloaded
```

```
--2024-05-02 17:17:24--  https://data.mendeley.com/public-files/datasets/wj9rwkp9c2/files/2eb60cac-96b8-46ea-b971-6415e9
Resolving data.mendeley.com (data.mendeley.com)... 162.159.130.86, 162.159.133.86
Connecting to data.mendeley.com (data.mendeley.com)|162.159.130.86|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://prod-dcd-datasets-public-files-eu-west-1.s3.eu-west-1.amazonaws.com/e205d80e-2bc6-49ed-bfcc-4215b6b516
--2024-05-02 17:17:25--  https://prod-dcd-datasets-public-files-eu-west-1.s3.eu-west-1.amazonaws.com/e205d80e-2bc6-49ed-
Resolving prod-dcd-datasets-public-files-eu-west-1.s3.eu-west-1.amazonaws.com (prod-dcd-datasets-public-files-eu-west-1.
Connecting to prod-dcd-datasets-public-files-eu-west-1.s3.eu-west-1.amazonaws.com (prod-dcd-datasets-public-files-eu-wes
HTTP request sent, awaiting response... 200 OK
Length: 49511 (48K) [application/vnd.ms-excel]
Saving to: ‘file_downloaded’

file_downloaded     100%[===================>]  48.35K  --.-KB/s    in 0.1s

2024-05-02 17:17:26 (341 KB/s) - ‘file_downloaded’ saved [49511/49511]
```

```
!pip install shap
```

```
Collecting shap
  Downloading shap-0.45.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.manylinux_2_17_x86_64.manylinux2014_x86
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 538.2/538.2 kB 4.3 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (2.0.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.66.2)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (24.0)
Collecting slicer==0.0.7 (from shap)
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.58.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2024.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.4.0
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas-
Installing collected packages: slicer, shap
Successfully installed shap-0.45.0 slicer-0.0.7
```

```
pip install --upgrade xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.4)
```

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import shap
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor
from xgboost.sklearn import XGBClassifier
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix,ConfusionMatrixDisplay
```

```
df = pd.read_csv('/content/file_downloaded')
df
```

226 to 250 of 1000 entries Filter

index	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
225	129	34311	F	56	8.5	92	5.7	4.8	1.7	1.3	2.8	0.7	35.0	Y
226	173	24029	M	51	7.3	65	7.0	3.8	3.8	1.0	1.1	1.7	31.0	Y
227	176	24030	M	52	3.0	60	7.0	3.8	3.2	0.8	1.7	1.4	33.0	Y
228	179	24031	M	56	3.4	44	4.8	4.1	1.5	0.8	1.7	1.4	39.0	Y
229	182	24032	F	59	3.8	91	6.0	4.2	2.0	2.3	0.9	1.4	38.0	Y
230	183	34312	M	52	3.0	60	7.0	3.8	3.2	0.8	1.7	1.4	33.0	Y
231	185	24033	F	56	8.5	92	5.7	4.8	1.7	1.3	2.8	0.7	35.0	Y
232	195	34313	M	56	3.4	44	4.8	4.1	1.5	0.8	1.7	1.4	39.0	Y
233	316	39963	M	51	7.3	65	7.0	3.8	3.8	1.0	1.1	1.7	31.0	Y
234	646	34314	F	59	3.8	91	6.0	4.2	2.0	2.3	0.9	1.4	38.0	Y
235	186	34315	F	54	4.3	52	6.7	3.1	1.1	3.1	1.2	0.7	37.0	Y
236	187	47586	M	69	5.9	71	10.4	5.4	1.3	1.7	3.1	0.6	33.0	Y
237	188	24034	F	54	4.3	52	6.7	3.1	1.1	3.1	1.2	0.7	37.0	Y
238	191	24035	M	60	6.6	76	7.8	4.7	2.3	1.4	1.6	0.6	26.0	Y
239	194	24036	M	54	5.9	71	10.4	5.4	1.3	1.7	3.1	0.6	32.0	Y
240	197	24037	M	69	5.9	71	10.4	5.4	1.3	1.7	3.1	0.6	33.0	Y
241	200	24038	M	57	2.0	77	7.0	5.5	1.9	1.0	3.7	0.9	33.0	Y
242	203	24039	M	55	5.0	76	10.2	5.6	4.6	0.8	2.9	31.8	33.9	Y
243	411	34316	M	57	2.0	77	7.0	5.5	1.9	1.0	3.7	0.9	33.0	Y
244	520	34317	M	54	5.9	71	10.4	5.4	1.3	1.7	3.1	0.6	32.0	Y
245	622	9423	M	55	5.0	76	10.2	5.6	4.6	0.8	2.9	31.8	33.9	Y
246	758	34318	M	60	6.6	76	7.8	4.7	2.3	1.4	1.6	0.6	26.0	Y
247	206	24040	F	60	2.0	45	6.8	7.2	2.2	0.8	2.2	1.1	27.0	Y
248	600	34319	F	60	2.0	45	6.8	7.2	2.2	1.0	2.2	1.1	27.0	Y
249	76	9903	M	73	4.3	79	6.0	5.3	1.4	1.5	3.2	0.6	27.0	Y

Show 25 per page

1 2 3 4 5 6 7 8 9 10 11 20 30 40

Next steps: Generate code with df View recommended plots

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   ID         1000 non-null   int64  
 1   No_Pation  1000 non-null   int64  
 2   Gender     1000 non-null   object 
 3   AGE        1000 non-null   int64  
 4   Urea       1000 non-null   float64
 5   Cr         1000 non-null   int64  
 6   HbA1c      1000 non-null   float64
 7   Chol       1000 non-null   float64
 8   TG         1000 non-null   float64
 9   HDL        1000 non-null   float64
 10  LDL        1000 non-null   float64
 11  VLDL       1000 non-null   float64
 12  BMI        1000 non-null   float64
 13  CLASS      1000 non-null   object 
dtypes: float64(8), int64(4), object(2)
memory usage: 109.5+ KB
```

```
df['ID'].nunique()
```

```
800
```

```
df['CLASS'].nunique()
```

```
5
```

```
df['CLASS'].value_counts()
```

```
CLASS
Y     840
N     102
P      53
Y       4
N       1
Name: count, dtype: int64
```

```
df['CLASS'].unique()
```

```
array(['N', 'N ', 'P', 'Y', 'Y '], dtype=object)
```

```
df['CLASS'] = df['CLASS'].str.replace(' ', '')
df['CLASS'].value_counts()
```

```
CLASS
Y    844
N    103
P     53
Name: count, dtype: int64
```

```
df['Gender'].nunique()
```

```
3
```

```
df['Gender'].value_counts()
```

```
Gender
M    565
F    434
f      1
Name: count, dtype: int64
```

```
df['Gender'].unique()
```

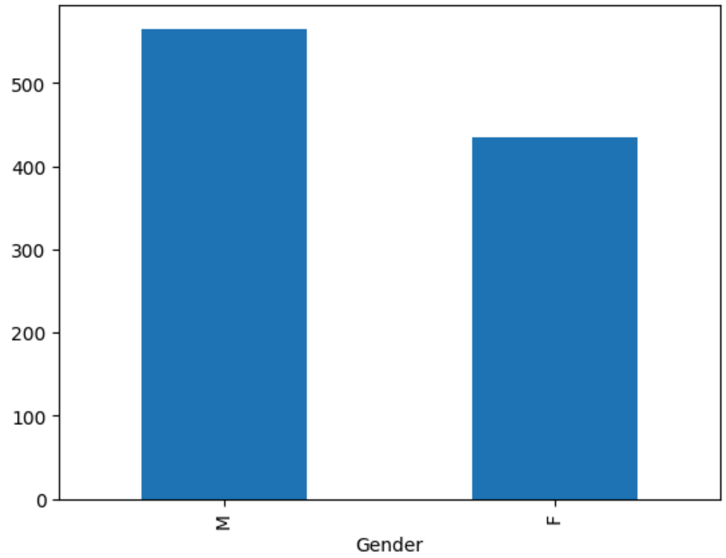
```
array(['F', 'M', 'f'], dtype=object)
```

```
df['Gender'] = df['Gender'].apply(str.upper)
df['Gender'].value_counts()
```

```
Gender
M    565
F    435
Name: count, dtype: int64
```

```
df['Gender'].value_counts().plot(kind = 'bar')
```

```
<Axes: xlabel='Gender'>
```




```
df['AGE'].nunique()
```

```
50
```

```
age_range_buckets = ["[{0} - {1})".format(age, age + 10) for age in range(20, 100, 10)]
age_range_buckets
```

```
['[20 - 30)',
 '[30 - 40)',
 '[40 - 50)',
 '[50 - 60)',
 '[60 - 70)',
 '[70 - 80)',
 '[80 - 90)',
 '[90 - 100)']
```

```
df['age_range'] = pd.cut(x=df['AGE'], bins=8, labels=age_range_buckets)
df
```

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	200	454317	M	71	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0
996	671	876534	M	31	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2
997	669	87654	M	30	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4
998	99	24004	M	38	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5
999	248	24054	M	54	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0

1000 rows × 15 columns

Next steps: Generate code with `df` View recommended plots

```
df['age_range'].value_counts()
```

```
age_range
[60 - 70)     476
[70 - 80)     274
[50 - 60)      79
[40 - 50)      58
[30 - 40)      54
[80 - 90)      36
[90 - 100)     19
[20 - 30)       4
Name: count, dtype: int64
```

```
# Select only numeric columns
numeric_cols = df.select_dtypes(include=[np.number])

# Calculate correlation
correlation_values = numeric_cols.corr()
```

```
correlation_values
```

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	
ID	1.000000	0.064920	-0.065980	-0.094434	-0.102457	-0.007571	0.044390	-(
No_Pation	0.064920	1.000000	-0.088006	-0.019160	0.000692	-0.032057	-0.030171	-(
AGE	-0.065980	-0.088006	1.000000	0.105092	0.054941	0.379136	0.036649	(
Urea	-0.094434	-0.019160	0.105092	1.000000	0.624134	-0.023603	0.001852	(
Cr	-0.102457	0.000692	0.054941	0.624134	1.000000	-0.037412	-0.007097	(
HbA1c	-0.007571	-0.032057	0.379136	-0.023603	-0.037412	1.000000	0.177489	(
Chol	0.044390	-0.030171	0.036649	0.001852	-0.007097	0.177489	1.000000	(
TG	-0.055908	-0.039885	0.148204	0.040980	0.056579	0.218556	0.321789	1
HDL	0.026231	-0.013357	-0.020038	-0.036994	-0.023804	0.028933	0.103814	-(
LDL	-0.064305	-0.003171	0.016105	-0.007301	0.039479	0.011057	0.416665	(
VLDL	0.146142	0.113754	-0.087903	-0.011191	0.009615	0.073462	0.076294	(
BMI	0.049409	0.017719	0.375956	0.045618	0.054746	0.413350	0.013678	(

Next steps: Generate code with `correlation_values` View recommended plots

```
plt.figure(figsize=(15,9))
sns.heatmap(correlation_values,annot = True)
```

```
<Axes: >
```


```
df2 = df.drop(['ID','No_Pation','AGE'], axis=1)
df2
```

	Gender	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS	age_range
0	F	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N	[60 - 70)
1	M	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N	[20 - 30)
2	F	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N	[60 - 70)
3	F	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N	[60 - 70)
4	M	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N	[30 - 40)
...	...	...	...	...	...	...	...	...	...	...	...	...
995	M	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0	Y	[80 - 90)
996	M	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2	Y	[30 - 40)
997	M	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4	Y	[30 - 40)
998	M	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5	Y	[40 - 50)
999	M	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0	Y	[60 - 70)

1000 rows × 12 columns

Next steps: Generate code with df2 View recommended plots

```
le1 = LabelEncoder()
df2['Gender'] =le1.fit_transform(df2['Gender'])
le2 = LabelEncoder()
df2['CLASS'] =le2.fit_transform(df2['CLASS'])
le3 = LabelEncoder()
df2['age_range'] =le3.fit_transform(df2['age_range'])
```

```
le1.classes_
```

```
array(['F', 'M'], dtype=object)
```

```
le2.classes_
```

```
array(['N', 'P', 'Y'], dtype=object)
```

```
le3.classes_
```

```
array(['[20 - 30)', '[30 - 40)', '[40 - 50)', '[50 - 60)', '[60 - 70)',
       '[70 - 80)', '[80 - 90)', '[90 - 100)'], dtype=object)
```

```
df2
```

	Gender	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS	age_range
0	0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0	4
1	1	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	0	0
2	0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0	4
3	0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0	4
4	1	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...
995	1	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0	2	6
996	1	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2	2	1
997	1	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4	2	1
998	1	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5	2	2
999	1	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0	2	4

1000 rows × 12 columns

Next steps: Generate code with df2 View recommended plots

```
scaler = MinMaxScaler()
```

```
columns_to_scale = ['Urea','Cr','HbA1c','Chol','TG','HDL','LDL','VLDL','BMI']
columns_scaled = ['Urea_scaled','Cr_scaled','HbA1c_scaled','Chol_scaled','TG_scaled','HDL_scaled','LDL_scaled','VLDL_scaled'
scale_values = df2[columns_to_scale].values
scaled_array = scaler.fit_transform(scale_values)
df2_scaled = pd.DataFrame(scaled_array, columns=columns_scaled)
df2_scaled
```

	Urea_scaled	Cr_scaled	HbA1c_scaled	Chol_scaled	TG_scaled	HDL_scaled
0	0.109375	0.050378	0.264901	0.407767	0.044444	0.226804
1	0.104167	0.070529	0.264901	0.359223	0.081481	0.092784
2	0.109375	0.050378	0.264901	0.407767	0.044444	0.226804
3	0.109375	0.050378	0.264901	0.407767	0.044444	0.226804
4	0.171875	0.050378	0.264901	0.475728	0.051852	0.061856
...	...	...	...	...	...	...
995	0.273438	0.114610	0.403974	0.728155	0.103704	0.103093
996	0.065104	0.068010	0.754967	0.398058	0.140741	0.051546
997	0.171875	0.094458	0.384106	0.398058	0.059259	0.103093
998	0.138021	0.066751	0.384106	0.514563	0.125926	0.144330
999	0.117188	0.076826	0.397351	0.368932	0.103704	0.092784

1000 rows × 9 columns

Next steps: Generate code with df2_scaled View recommended plots

```
df3 = pd.concat([df2,df2_scaled],axis=1)
df3
```

	Gender	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	...	age_range	Urea
0	0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	...	4	
1	1	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	...	0	
2	0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	...	4	
3	0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	...	4	
4	1	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	...	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
995	1	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0	...	6	
996	1	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2	...	1	
997	1	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4	...	1	
998	1	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5	...	2	
999	1	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0	...	4	

1000 rows × 21 columns

```
X = df3[['Gender','age_range','Urea_scaled','Cr_scaled','HbA1c_scaled','Chol_scaled','TG_scaled','HDL_scaled','LDL_scaled','
y = df3['CLASS'].values
train_set, test_set,train_label,test_label = train_test_split(X,y,test_size=0.20,random_state=0)
```

```
X[0]
```

```
array([0.        , 4.        , 0.109375  , 0.05037783, 0.26490066,
       0.40776699, 0.04444444, 0.22680412, 0.11458333, 0.01146132,
       0.17391304])
```

```
xgb_model = XGBClassifier(objective='multi:softmax', num_class=3)
models = []
models.append(('LR', LogisticRegression(solver ='lbfgs',multi_class='auto')))
models.append(('KNN', KNeighborsClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVC', SVC(gamma='scale')))
models.append(('RFC', RandomForestClassifier(n_estimators=100)))
models.append(('DTR', DecisionTreeRegressor()))
models.append(('XGB', xgb_model))
```

```
results = []
names = []
```

```
for name, model in models:
    kfold = model_selection.KFold(n_splits = 10)
    cv_results = model_selection.cross_val_score(model, X, y,cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "{}: {} ({})". format(name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
# Create an instance of the RandomForestClassifier
RFC_model = RandomForestClassifier()

# Fit the model to your training data
RFC_model_fitted = RFC_model.fit(train_set, train_label)

# Use the fitted model to make predictions on your test set
RFC_model_prediction = RFC_model.predict(test_set)
```

```
XGBClassifier_model = XGBClassifier()
XGBClassifier_model_fitted = XGBClassifier_model.fit(train_set,train_label)
XGBClassifier_model_prediction = XGBClassifier_model.predict(test_set)
```

```
df3['CLASS'].unique() #0 means No diabetes, 1 means predicted, 2 means diabetes
```

```
array([0, 1, 2])
```

```
print('XGB Classifier Accuracy Score:\n', accuracy_score(test_label, XGBClassifier_model_prediction))
print('RFC Accuracy Score:\n', accuracy_score(test_label, RFC_model_prediction))
```

```
XGB Classifier Accuracy Score:
 0.995
```

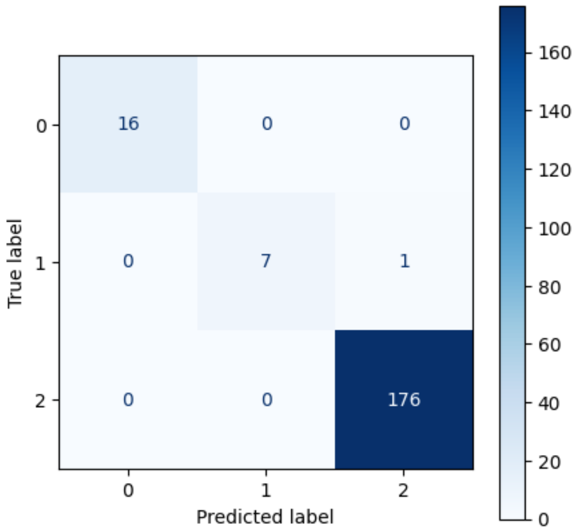
```
RFC Accuracy Score:
 0.985
```

```
print('XGB Classifier Report:\n', classification_report(test_label,XGBClassifier_model_prediction,target_names=['0','1','2']
print('RFC Classification Report:\n', classification_report(test_label, RFC_model_prediction, target_names=['0','1','2']))
```

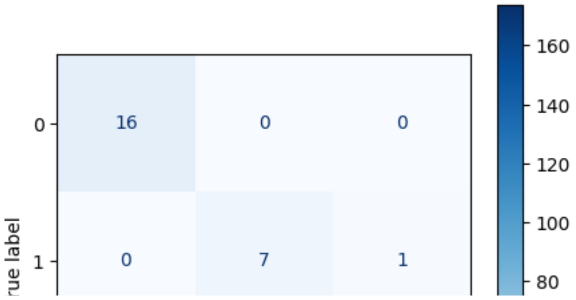
```
XGB Classifier Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.88      0.93         8
           2       0.99      1.00      1.00       176

    accuracy                           0.99       200
   macro avg       1.00      0.96      0.98       200
weighted avg       1.00      0.99      0.99       200

RFC Classification Report:
               precision    recall  f1-score   support

           0       0.89      1.00      0.94        16
           1       1.00      0.88      0.93         8
           2       0.99      0.99      0.99       176

    accuracy                           0.98       200
   macro avg       0.96      0.95      0.96       200
weighted avg       0.99      0.98      0.99       200
```

```
XGBClassifier_matrix = confusion_matrix(test_label,XGBClassifier_model_prediction)
XGBClassifier_confusion_matrix_display = ConfusionMatrixDisplay(XGBClassifier_matrix)
fig, ax = plt.subplots(figsize=(5,5))
XGBClassifier_confusion_matrix_display.plot(cmap=plt.cm.Blues,ax=ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7a5a317276d0>
```




```
# Generate the confusion matrix
RFC_matrix = confusion_matrix(test_label, RFC_model_prediction)

# Create a ConfusionMatrixDisplay instance
RFC_confusion_matrix_display = ConfusionMatrixDisplay(RFC_matrix)

# Plot the confusion matrix
fig, ax = plt.subplots(figsize=(5,5))
RFC_confusion_matrix_display.plot(cmap=plt.cm.Blues, ax=ax)
plt.show()
```


16
0
0
0
7
1
0
1
rue label
160
140
120
100
80