```
# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)

# to split the data into train and test
from sklearn.model_selection import train_test_split

# to build linear regression_model
from sklearn.linear_model import LinearRegression


# to check model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# I changed this part
!pip install mlxtend
import joblib
import sys
sys.modules['sklearn.externals.joblib'] = joblib
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
```

```
Requirement already satisfied: mlxtend in /usr/local/lib/python3.10/dist-packages (0.22.0)
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.11.4)
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.25.2)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (2.0.3)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.2.2)
Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (3.7.1)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.4.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from mlxtend) (67.7.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->mlxtend) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->mlxtend) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.2->mlxtend) (3.4.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxten
```

```
# Importing Libraries
import requests
import pandas as pd
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB as Naive_Bayes
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_classification
from xgboost import XGBClassifier
from sklearn.externals import joblib

from IPython.display import display
import pickle
```

```
df = pd.read_csv('/content/parkinsons dataset.csv')
```

Double-click (or enter) to edit

```
df.head()
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimm
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.043
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.061
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.052
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.054
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.064

```
df.tail()
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitt
191	phon_R01_S50_3	209.516	253.017	89.488	0.00564	
192	phon_R01_S50_4	174.688	240.005	74.287	0.01360	
193	phon_R01_S50_5	198.764	396.961	74.904	0.00740	
194	phon_R01_S50_6	214.289	260.277	77.973	0.00567	
195	phon_R01_S50_6	216.289	266.279	77.974	0.00568	

```
print('Number of Features In Dataset :', df.shape[1])
print('Number of Instances In Dataset : ', df.shape[0])
```

```
Number of Features In Dataset : 24
Number of Instances In Dataset :  196
```

```
# Dropping The Name Column
df.drop(['name'], axis=1, inplace=True)
```

```
print('Number of Features In Dataset :', df.shape[1])
print('Number of Instances In Dataset : ', df.shape[0])
```

```
Number of Features In Dataset : 23
Number of Instances In Dataset :  196
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 196 entries, 0 to 195
Data columns (total 23 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   MDVP:Fo(Hz)       196 non-null    float64
 1   MDVP:Fhi(Hz)      196 non-null    float64
 2   MDVP:Flo(Hz)      196 non-null    float64
 3   MDVP:Jitter(%)    196 non-null    float64
 4   MDVP:Jitter(Abs)  196 non-null    float64
 5   MDVP:RAP          196 non-null    float64
 6   MDVP:PPQ          196 non-null    float64
 7   Jitter:DDP        196 non-null    float64
 8   MDVP:Shimmer      196 non-null    float64
 9   MDVP:Shimmer(dB)  196 non-null    float64
 10  Shimmer:APQ3      196 non-null    float64
 11  Shimmer:APQ5      196 non-null    float64
 12  MDVP:APQ          196 non-null    float64
 13  Shimmer:DDA       196 non-null    float64
 14  NHR               196 non-null    float64
 15  HNR               196 non-null    float64
 16  status            196 non-null    int64  
 17  RPDE              196 non-null    float64
 18  DFA               196 non-null    float64
 19  spread1           196 non-null    float64
 20  spread2           196 non-null    float64
 21  D2                196 non-null    float64
 22  PPE               196 non-null    float64
dtypes: float64(22), int64(1)
memory usage: 35.3 KB
```

```
df.describe()
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP
count	196.000000	196.000000	196.000000	196.000000	196.000000	196.00
mean	154.545276	197.457847	116.128964	0.006218	0.000044	0.00
std	41.521110	91.390318	43.496022	0.004836	0.000035	0.00
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.00
25%	117.721000	134.965750	84.044250	0.003460	0.000020	0.00
50%	149.239500	176.212000	104.205000	0.004945	0.000030	0.00
75%	183.653750	224.804250	139.504250	0.007347	0.000060	0.00
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.02

```
df['status'] = df['status'].astype('uint8')
```

```
# Checking For Duplicate Rows In Dataset
print('Number of Duplicated Rows :',df.duplicated().sum())
```

```
Number of Duplicated Rows : 0
```

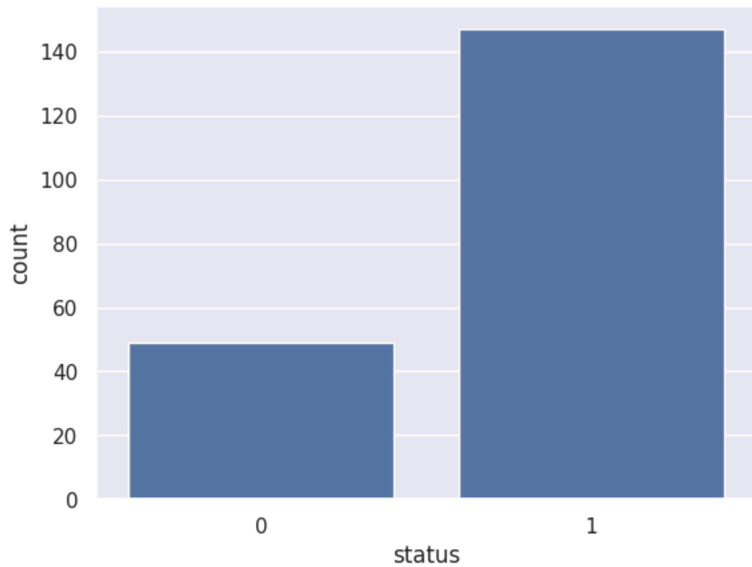
```
# Checking For Missing Values In Dataset
df.isna().sum()
```

```
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
```

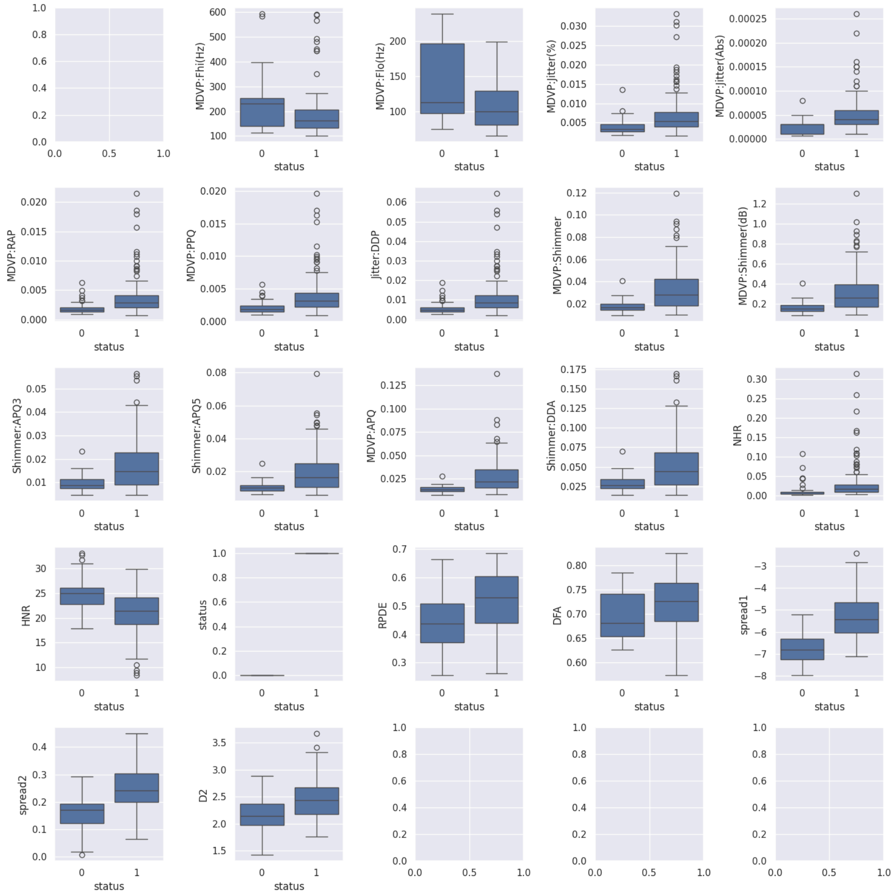
```
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
status              0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
dtype: int64
```

```
#Balance of Data
sns.countplot(x='status',data=df)
```

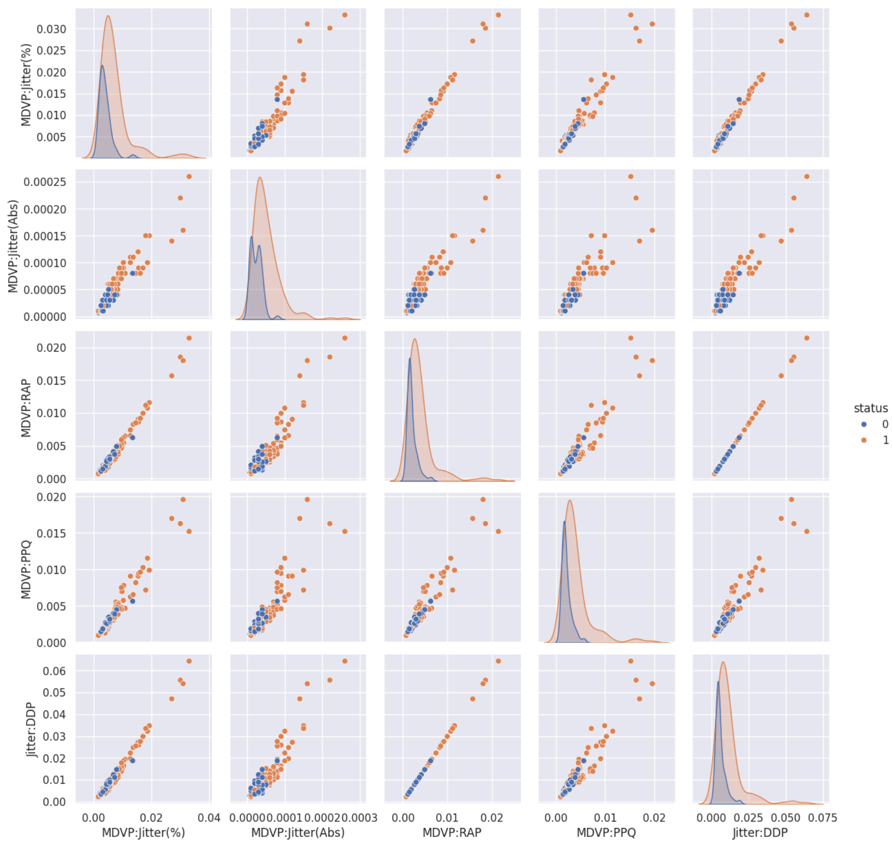
```
<Axes: xlabel='status', ylabel='count'>
```




```
#Box Plot
fig,axes=plt.subplots(5,5,figsize=(15,15))
axes=axes.flatten()

for i in range(1,len(df.columns)-1):
    sns.boxplot(x='status',y=df.iloc[:,i],data=df,orient='v',ax=axes[i])
plt.tight_layout()
plt.show()
```

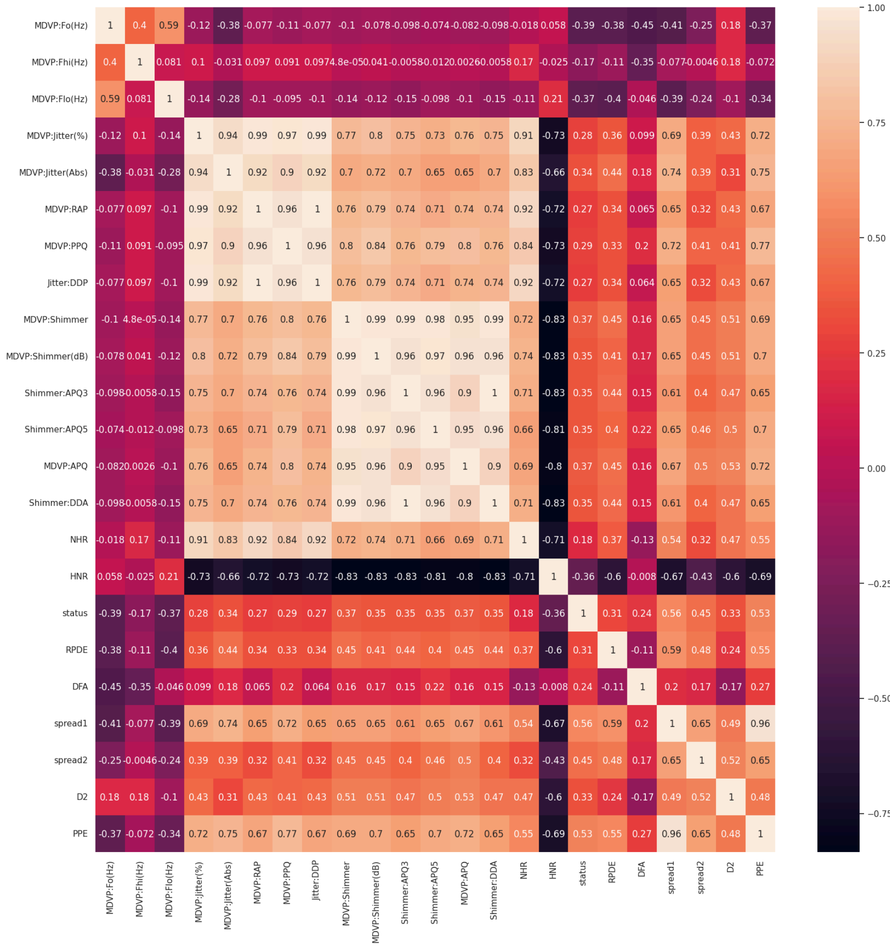

MDVP:Fhi(Hz)
MDVP:Flo(Hz)
MDVP:Jitter(%)
MDVP:Jitter(Abs)
MDVP:RAP
MDVP:PPQ
Jitter:DDP
MDVP:Shimmer
MDVP:Shimmer(dB)
Shimmer:APQ3
Shimmer:APQ5
MDVP:APQ
Shimmer:DDA
NHR
HNR
status
RPDE
DFA
spread1
spread2
D2
status

```
plt.rcParams['figure.figsize'] = (15, 4)
sns.pairplot(df,hue = 'status', vars = ['MDVP:Jitter(%)','MDVP:Jitter(Abs)','MDVP:RAP','MDVP:PPQ', 'Jitter:DDP'] )
plt.show()
```

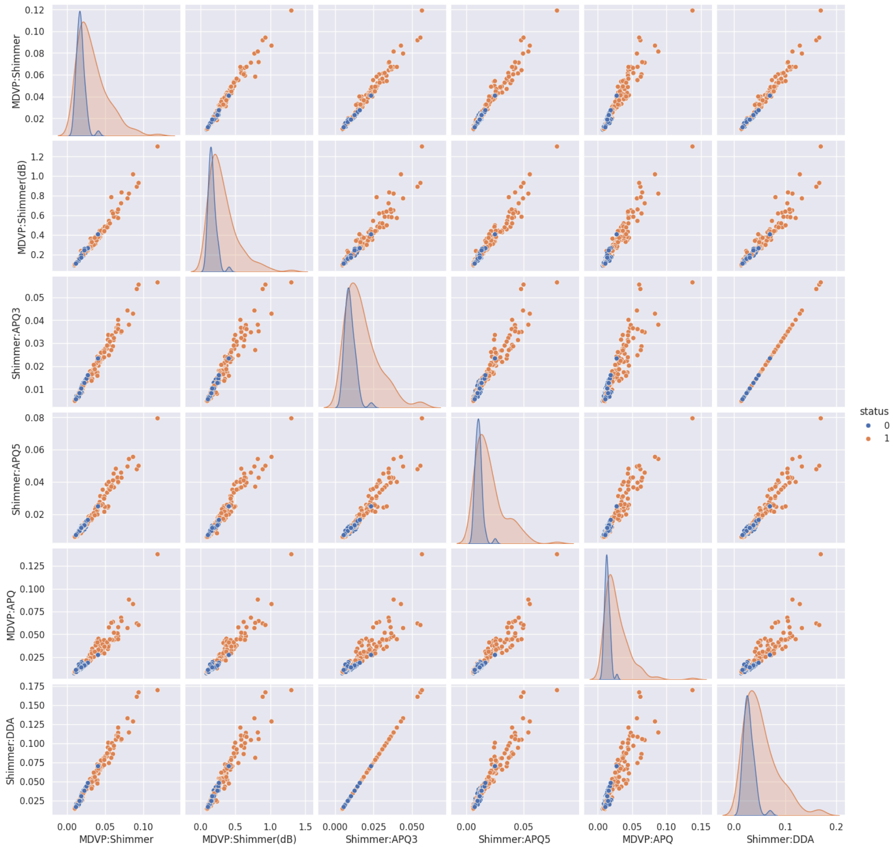



```
fig, ax = plt.subplots(figsize=(20,20))
sns.heatmap(df.corr(),annot=True,ax=ax)
```

```
<Axes: >
```



```
plt.rcParams['figure.figsize'] = (15, 4)
sns.pairplot(df,hue = 'status', vars = ['MDVP:Shimmer','MDVP:Shimmer(dB)','Shimmer:APQ3','Shimmer:APQ5','MDVP:APQ','Shimmer:DDA'] )
plt.show()
```




```
# Exploring Imabalance In Dataset
df['status'].value_counts()
```

```
status
1    147
0     49
Name: count, dtype: int64
```

```
X = df.drop(['status'], axis=1)
y = df['status']

print('Feature (X) Shape Before Balancing :', X.shape)
print('Target (y) Shape Before Balancing :', y.shape)
```

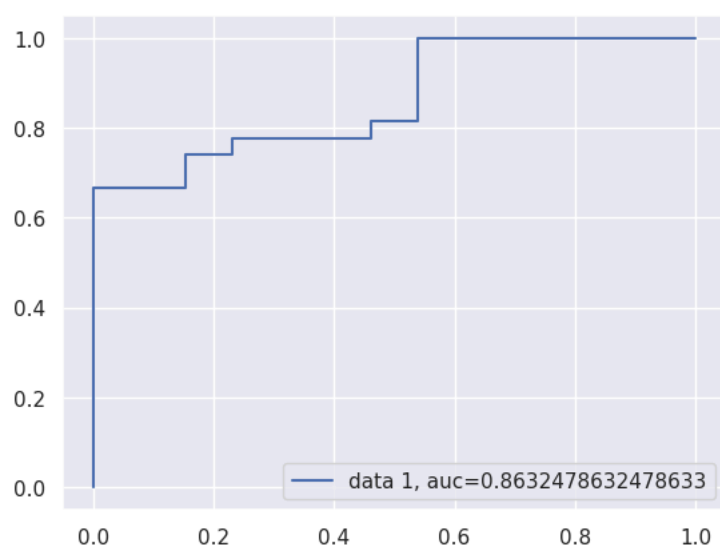
```
Feature (X) Shape Before Balancing : (196, 22)
Target (y) Shape Before Balancing : (196,)
```

```
y=df['status']
cols=['MDVP:RAP','Jitter:DDP','DFA','NHR','MDVP:Fhi(Hz)','status']
x=df.drop(cols,axis=1)
```

```
from sklearn.preprocessing import MinMaxScaler


# Scaling features between -1 and 1  for mormalization
scaler = MinMaxScaler((-1,1))



# define X_features , Y_labels
X_features = scaler.fit_transform(X)
Y_labels = y



# splitting the dataset into traning and testing sets into 80 - 20
train_size=0.80
test_size=0.20
seed=5
from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(X_features, Y_labels , test_size=0.20, random_state=20)



logmodel = LogisticRegression()
logmodel.fit(X_train, y_train)
predlog = logmodel.predict(X_test)


print(classification_report(y_test, predlog))
print("Confusion Matrix:")
confusion_matrix(y_test, predlog)
```

```
              precision    recall  f1-score   support

           0       0.86      0.46      0.60        13
           1       0.79      0.96      0.87        27

    accuracy                           0.80        40
   macro avg       0.82      0.71      0.73        40
weighted avg       0.81      0.80      0.78        40

Confusion Matrix:
array([[ 6,  7],
       [ 1, 26]])
```

```
y_pred_proba = logmodel.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```




```
# Dumping Logistic Regression Model
pickle.dump(logmodel, open('lg.pkl','wb'))
```

```
# Naive Bayes

gnb = Naive_Bayes()
gnb.fit(X_train, y_train)
predgnb = gnb.predict(X_test)

print(classification_report(y_test, predgnb))
```

```
              precision    recall  f1-score   support

           0       0.50      1.00      0.67        13
           1       1.00      0.52      0.68        27

    accuracy                           0.68        40
   macro avg       0.75      0.76      0.67        40
weighted avg       0.84      0.68      0.68        40
```

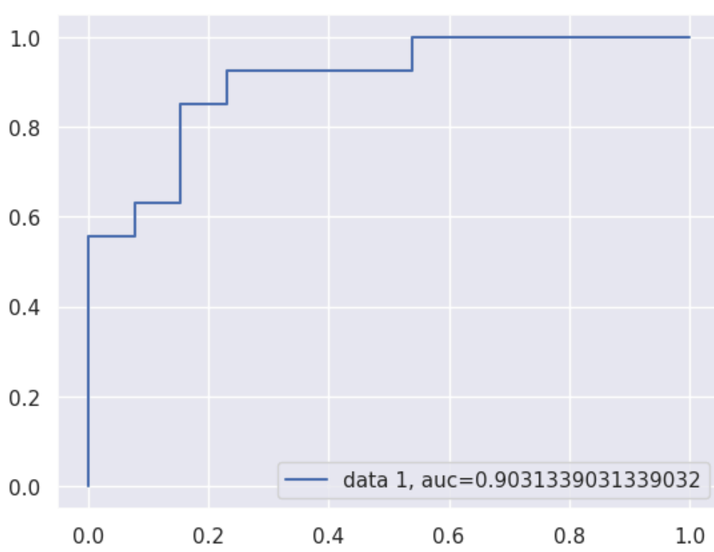
```
print("Confusion Matrix:")
confusion_matrix(y_test, predgnb)
```

```
Confusion Matrix:
array([[13,  0],
       [13, 14]])
```

```
# scores -check how efficiently labels are predicted
accuracy_testing = accuracy_score(y_test, predgnb)
print("Accuracy % :",accuracy_testing*100)
```

```
Accuracy % : 67.5
```

```
y_pred_proba = gnb.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

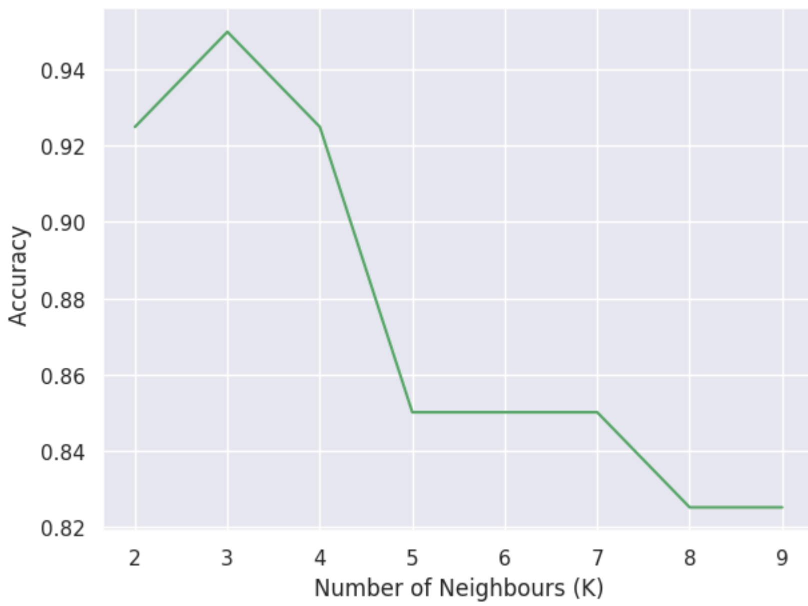



```
pickle.dump(gnb,open ('gnb.pkl','wb'))
```

```
import numpy as np

Ks = 10
mean_acc = []
ConfustionMx = [];
for n in range(2,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc.append(metrics.accuracy_score(y_test, yhat))
print('Neighbor Accuracy List')
print(mean_acc)
```

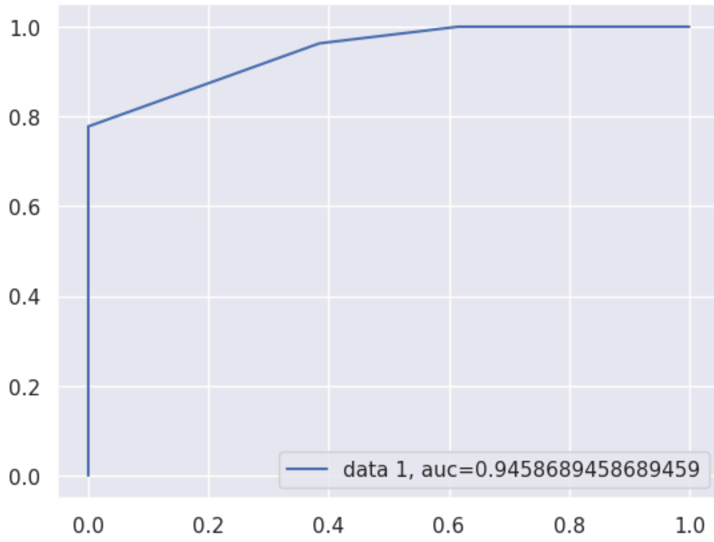
```
Neighbor Accuracy List
[0.925, 0.95, 0.925, 0.85, 0.85, 0.85, 0.825, 0.825]
```

```
plt.plot(range(2,Ks),mean_acc,'g')
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbours (K)')
plt.tight_layout()
plt.show()
```




```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
predKNN = knn.predict(X_test)
```

```
y_pred_proba = knn.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```




```
# Dumping KNN Classifier
pickle.dump(knn, open('knn_clf.pkl','wb'))
```

```
# Defining Parameter Dictionary
param_dict = {'max_depth': range(4,8), 'eta' : [0.1, 0.2, 0.3, 0.4, 0.5],
              'reg_lambda' : [0.8, 0.9, 1, 1.1, 1.2],
              'random_state': [300, 600, 900]}
```

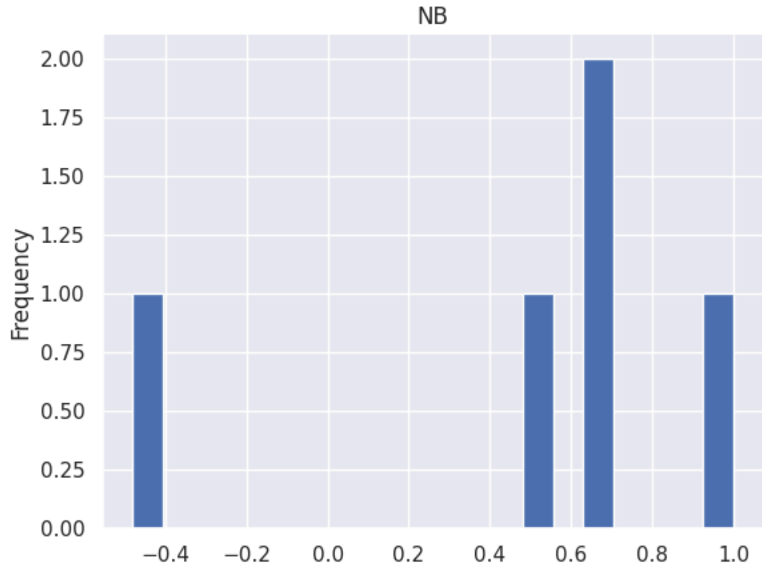
```
from sklearn.metrics import precision_score,recall_score ,accuracy_score, f1_score, r2_score, log_loss

chart = {
        'Metric':["Accuracy", "F1-Score", "Recall", "Precision", "R2-Score"],
        'LR':[accuracy_score(y_test, predlog), f1_score(y_test, predlog), recall_score(y_test, predlog), precision_score(y_test, predlog), r
        'NB':[accuracy_score(y_test, predgnb), f1_score(y_test, predgnb), recall_score(y_test, predgnb), precision_score(y_test, predgnb), r
        'KNN':[accuracy_score(y_test, predKNN), f1_score(y_test, predKNN), recall_score(y_test, predKNN), precision_score(y_test, predKNN),
}
chart = pd.DataFrame(chart)
```

```
display(chart)
```

	Metric	LR	NB	KNN
0	Accuracy	0.800000	0.675000	0.850000
1	F1-Score	0.866667	0.682927	0.892857
2	Recall	0.962963	0.518519	0.925926
3	Precision	0.787879	1.000000	0.862069
4	R2-Score	0.088319	-0.481481	0.316239

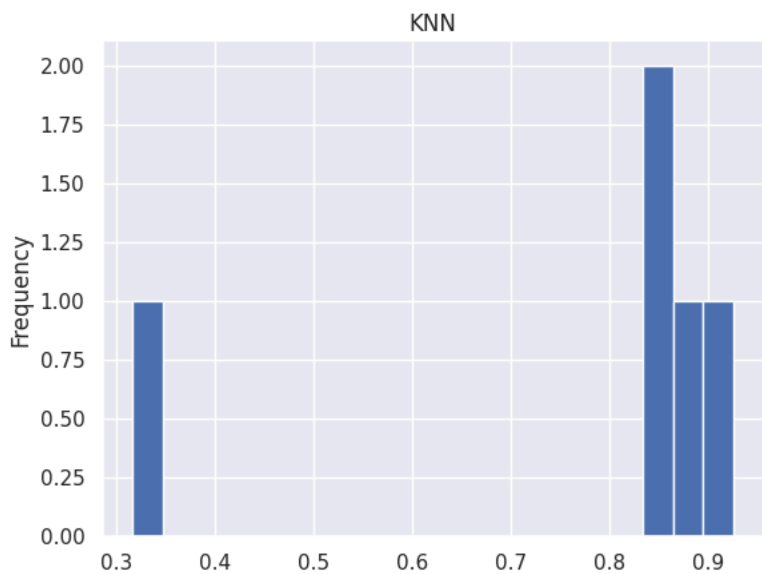
NB

```
# @title NB

from matplotlib import pyplot as plt
chart['NB'].plot(kind='hist', bins=20, title='NB')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



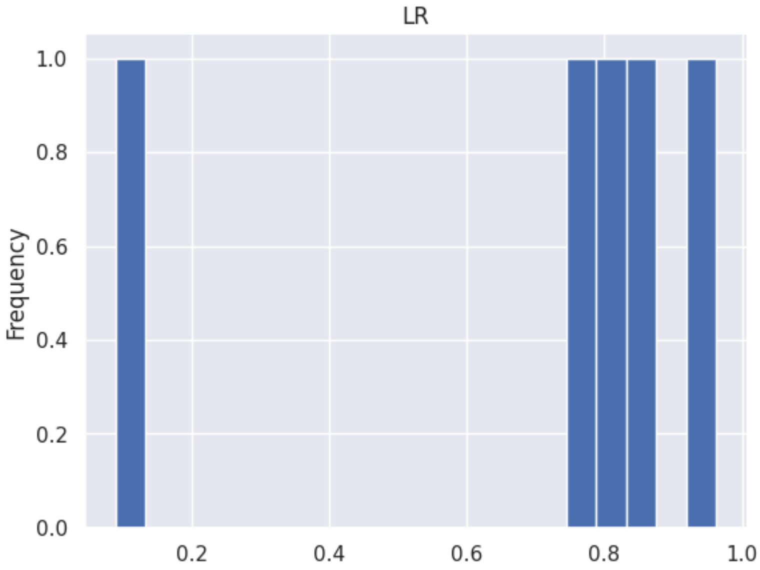
KNN

```
# @title KNN

from matplotlib import pyplot as plt
chart['KNN'].plot(kind='hist', bins=20, title='KNN')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

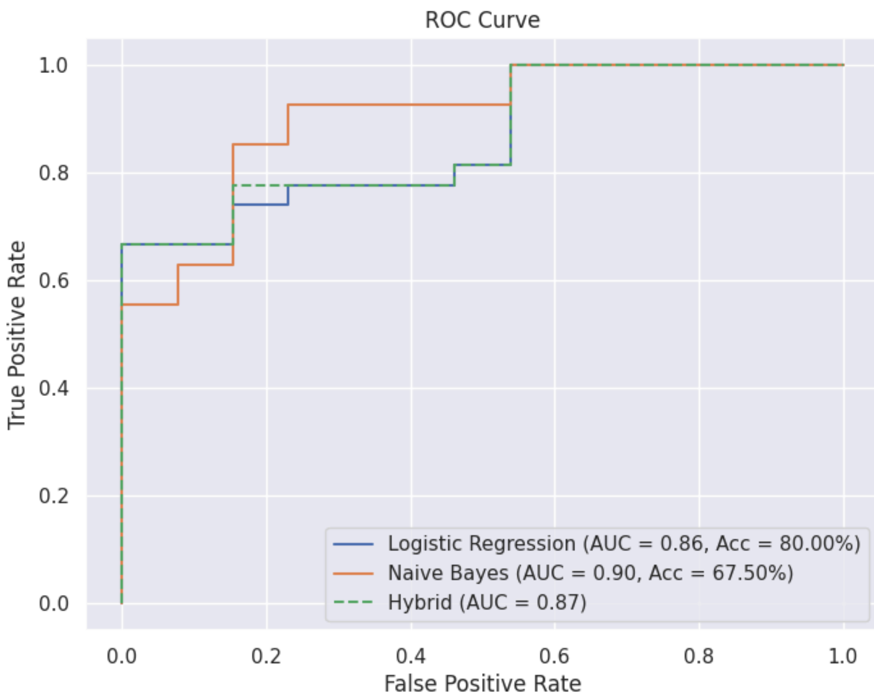


LR

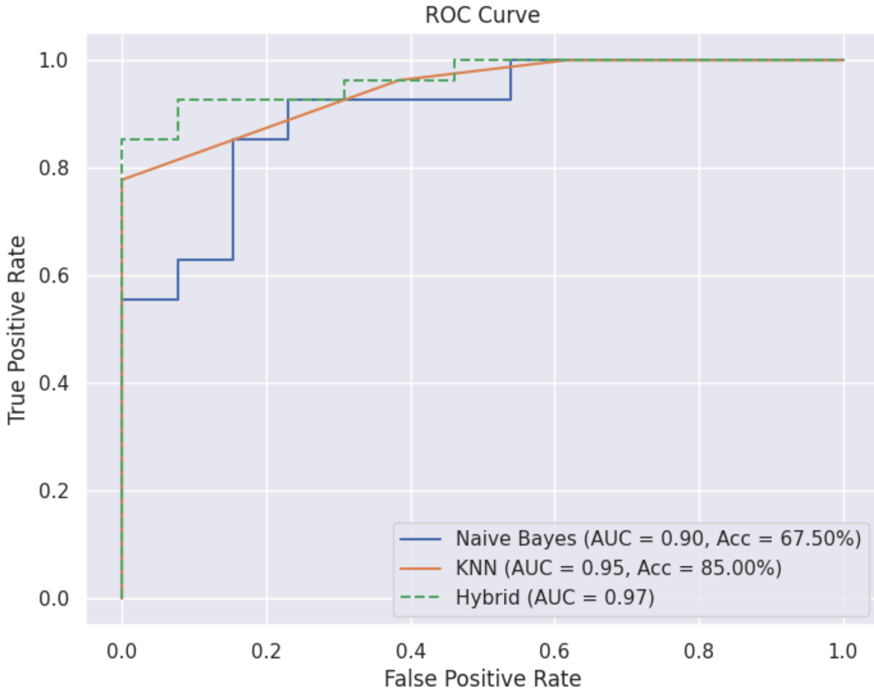
```
# @title LR

from matplotlib import pyplot as plt
chart['LR'].plot(kind='hist', bins=20, title='LR')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



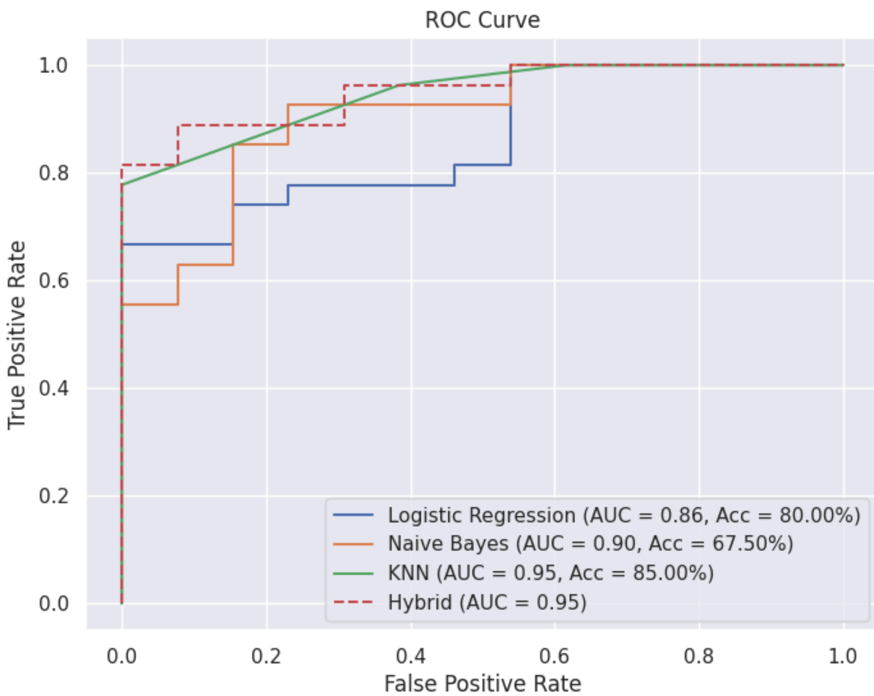

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Initialize Logistic Regression and Naive Bayes classifiers
logmodel = LogisticRegression()
gnb = GaussianNB()

# Fit both models on training data
logmodel.fit(X_train, y_train)
gnb.fit(X_train, y_train)

# Predict probabilities for both models
log_prob = logmodel.predict_proba(X_test)[:, 1]
gnb_prob = gnb.predict_proba(X_test)[:, 1]

# Weighted average of probabilities
hybrid_prob = (log_prob + gnb_prob) / 2

# Compute accuracy for each model
log_accuracy = accuracy_score(y_test, logmodel.predict(X_test))
gnb_accuracy = accuracy_score(y_test, gnb.predict(X_test))

# Compute AUC for each model
log_auc = roc_auc_score(y_test, log_prob)
gnb_auc = roc_auc_score(y_test, gnb_prob)
hybrid_auc = roc_auc_score(y_test, hybrid_prob)

# Plot ROC curves for each model
plt.figure(figsize=(8, 6))
fpr_log, tpr_log, _ = roc_curve(y_test, log_prob)
fpr_gnb, tpr_gnb, _ = roc_curve(y_test, gnb_prob)
fpr_hybrid, tpr_hybrid, _ = roc_curve(y_test, hybrid_prob)

plt.plot(fpr_log, tpr_log, label='Logistic Regression (AUC = {:.2f}, Acc = {:.2f}%)'.format(log_auc, log_accuracy * 100))
plt.plot(fpr_gnb, tpr_gnb, label='Naive Bayes (AUC = {:.2f}, Acc = {:.2f}%)'.format(gnb_auc, gnb_accuracy * 100))
plt.plot(fpr_hybrid, tpr_hybrid, label='Hybrid (AUC = {:.2f})'.format(hybrid_auc), linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```




```
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Initialize Naive Bayes and KNN classifiers
gnb = GaussianNB()
knn = KNeighborsClassifier(n_neighbors=5)

# Fit both models on training data
gnb.fit(X_train, y_train)
knn.fit(X_train, y_train)

# Predict probabilities for both models
gnb_prob = gnb.predict_proba(X_test)[:, 1]
knn_prob = knn.predict_proba(X_test)[:, 1]

# Weighted average of probabilities
hybrid_prob = (gnb_prob + knn_prob) / 2

# Compute accuracy for each model
gnb_accuracy = accuracy_score(y_test, gnb.predict(X_test))
knn_accuracy = accuracy_score(y_test, knn.predict(X_test))

# Compute AUC for each model
gnb_auc = roc_auc_score(y_test, gnb_prob)
knn_auc = roc_auc_score(y_test, knn_prob)
hybrid_auc = roc_auc_score(y_test, hybrid_prob)

# Plot ROC curves for each model
plt.figure(figsize=(8, 6))
fpr_gnb, tpr_gnb, _ = roc_curve(y_test, gnb_prob)
fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_prob)
fpr_hybrid, tpr_hybrid, _ = roc_curve(y_test, hybrid_prob)

plt.plot(fpr_gnb, tpr_gnb, label='Naive Bayes (AUC = {:.2f}, Acc = {:.2f}%)'.format(gnb_auc, gnb_accuracy * 100))
plt.plot(fpr_knn, tpr_knn, label='KNN (AUC = {:.2f}, Acc = {:.2f}%)'.format(knn_auc, knn_accuracy * 100))
plt.plot(fpr_hybrid, tpr_hybrid, label='Hybrid (AUC = {:.2f})'.format(hybrid_auc), linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```


ROC Curve
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Naive Bayes (AUC = 0.90, Acc = 67.50%)
KNN (AUC = 0.95, Acc = 85.00%)
Hybrid (AUC = 0.97)

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Initialize classifiers
logmodel = LogisticRegression()
gnb = GaussianNB()
knn = KNeighborsClassifier(n_neighbors=5)

# Fit all models on training data
logmodel.fit(X_train, y_train)
gnb.fit(X_train, y_train)
knn.fit(X_train, y_train)

# Predict probabilities for all models
log_prob = logmodel.predict_proba(X_test)[:, 1]
gnb_prob = gnb.predict_proba(X_test)[:, 1]
knn_prob = knn.predict_proba(X_test)[:, 1]

# Weighted average of probabilities
hybrid_prob = (log_prob + gnb_prob + knn_prob) / 3

# Compute accuracy for each model
log_accuracy = accuracy_score(y_test, logmodel.predict(X_test))
gnb_accuracy = accuracy_score(y_test, gnb.predict(X_test))
knn_accuracy = accuracy_score(y_test, knn.predict(X_test))

# Compute AUC for each model
log_auc = roc_auc_score(y_test, log_prob)
gnb_auc = roc_auc_score(y_test, gnb_prob)
knn_auc = roc_auc_score(y_test, knn_prob)
hybrid_auc = roc_auc_score(y_test, hybrid_prob)

# Plot ROC curves for each model
plt.figure(figsize=(8, 6))
fpr_log, tpr_log, _ = roc_curve(y_test, log_prob)
fpr_gnb, tpr_gnb, _ = roc_curve(y_test, gnb_prob)
fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_prob)
fpr_hybrid, tpr_hybrid, _ = roc_curve(y_test, hybrid_prob)

plt.plot(fpr_log, tpr_log, label='Logistic Regression (AUC = {:.2f}, Acc = {:.2f}%)'.format(log_auc, log_accuracy * 100))
plt.plot(fpr_gnb, tpr_gnb, label='Naive Bayes (AUC = {:.2f}, Acc = {:.2f}%)'.format(gnb_auc, gnb_accuracy * 100))
plt.plot(fpr_knn, tpr_knn, label='KNN (AUC = {:.2f}, Acc = {:.2f}%)'.format(knn_auc, knn_accuracy * 100))
plt.plot(fpr_hybrid, tpr_hybrid, label='Hybrid (AUC = {:.2f})'.format(hybrid_auc), linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have a DataFrame 'data' with multiple observations for each algorithm
# For demonstration purposes, I'll create a random dataset
import pandas as pd
import numpy as np

algorithms = ['Logistic Regression', 'Naive Bayes', 'KNN', 'Hybrid']
auc_scores = np.random.rand(100, 4)  # Random AUC scores for 100 observations and 4 algorithms

# Create a DataFrame for visualization
data = pd.DataFrame(auc_scores, columns=algorithms)

# Plotting box plots for AUC scores
plt.figure(figsize=(10, 6))
sns.boxplot(data=data)
plt.title('AUC Scores Comparison')
plt.xlabel('Algorithm')
plt.ylabel('AUC')
plt.show()
```