**BTech/ III Year CSE/ V Semester**

**19CSE305/ Machine Learning**

**Project Review (Report)**

****

|  |  |  |
| --- | --- | --- |
| **S.no** | **Name** | **Roll number** |
| **1** | **Chunduru Rohith Kumar** | **CB.EN.U4CSE21612** |
| **2** |  |  |
| **3** |  |  |
| **4** |  |  |
| **5** |  |  |

Problem Statement: Based on the available information we will predict whether the person will have stroke or not.

Available information (DATASET):

<https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset>

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths.  
This dataset is used to predict whether a patient is likely to get stroke based on the input parameters like gender, age, various diseases, and smoking status. Each row in the data provides relevant information about the patient

Attribute Information

1) id: unique identifier  
2) gender: "Male", "Female" or "Other"  
3) age: age of the patient  
4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension  
5) heart\_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease  
6) ever\_married: "No" or "Yes"  
7) work\_type: "children", "Govt\_jov", "Never\_worked", "Private" or "Self-employed"  
8) Residence\_type: "Rural" or "Urban"  
9) avg\_glucose\_level: average glucose level in blood  
10) bmi: body mass index  
11) smoking\_status: "formerly smoked", "never smoked", "smokes" or "Unknown"\*  
12) stroke: 1 if the patient had a stroke or 0 if not

**Feature Selection:**

In most of the case Id is not necessary for the prediction. So we can drop the id attribute from the dataset. Other all features(attributes are important).

**Preprocessing** **and Feature Engineering**:

Check for the null values in any of the attributes and fill the values with the mean or mode or median.

In our dataset, there are more null values in the bmi feature. By checking for the outliers using IQR(Inter Quartile Range) we found that there are outliers. The bmi values which are less Q1-1.5\*IQR or greater than Q3+1.5\*IQR.

Since there are outliers we filled the null values of bmi attribute with median of the features. When there are outliers median is most suitable then mean for filling up the null values.

**StandardScaling:**

Feature scaling becomes necessary when dealing with datasets containing features that have different ranges, units of measurement, or orders of magnitude. In such cases, the variation in feature values can lead to biased model performance or difficulties during the learning process.

By applying feature scaling, the dataset’s features can be transformed to a more consistent scale, making it easier to build accurate and effective machine learning models. Scaling facilitates meaningful comparisons between features, improves model convergence, and prevents certain features from overshadowing others based solely on their magnitude.

**Oversampling:**

Since our dataset is imbalanced, One approach to addressing the problem of class imbalance is to randomly resample the training dataset. The two main approaches to randomly resampling an imbalanced dataset are to delete examples from the majority class, called undersampling, and to duplicate examples from the minority class, called oversampling. In our case we used Oversampling.

**One hot encoding:**

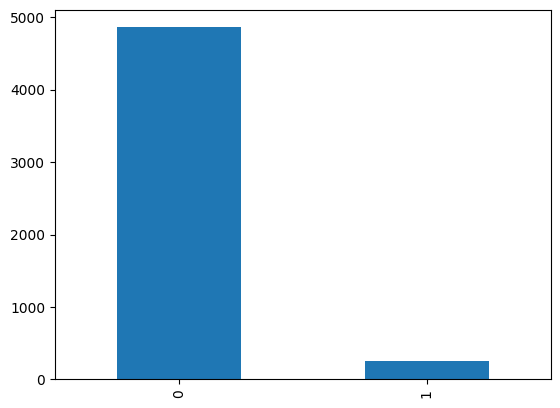
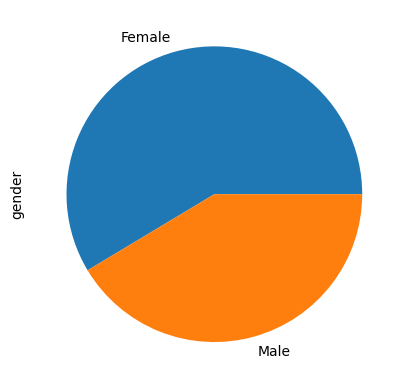
Since our dataset contains categorical data, we transformed these categorical data into a binary vector representation that can used as input for models.

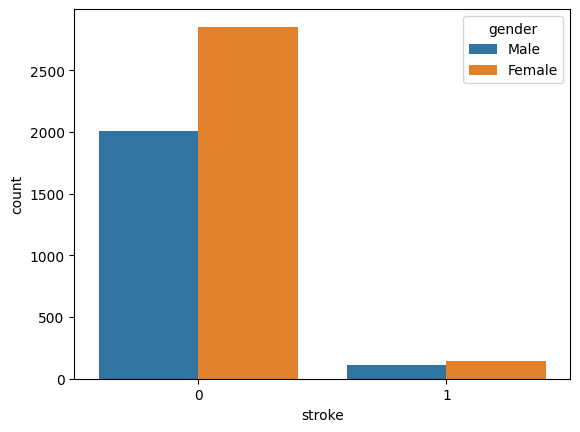
Gender attribute contains categorical data so after one hot encoding it was replaced by gender\_male\_1 if value is 1 indicates male 0 indicates female.

Work\_type, Residence ,Smoking\_status are also contains categorical data, So we make those dummies.

**Visualisation:**

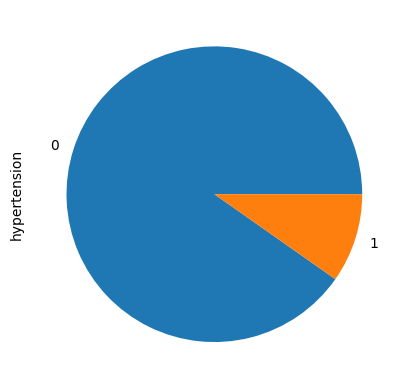
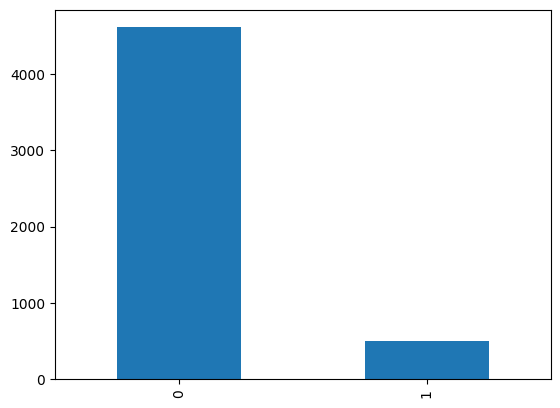
1. Gender



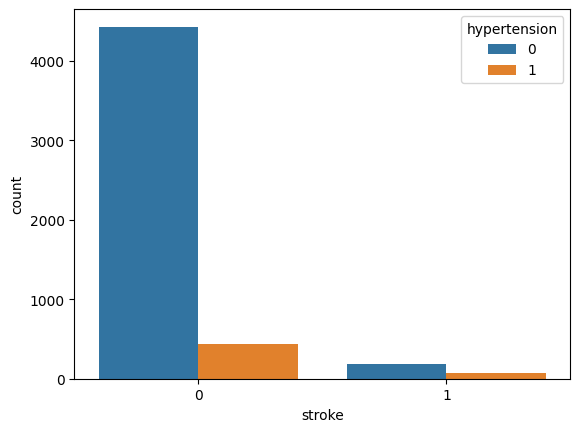
 Comparison with the target attribute stroke

**Inference:** There are more females having less probability of getting stroke.

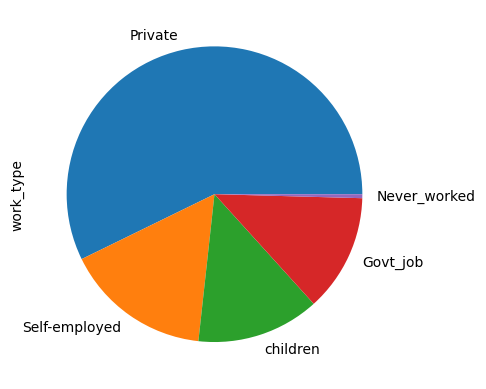
1. Hypertension attribute

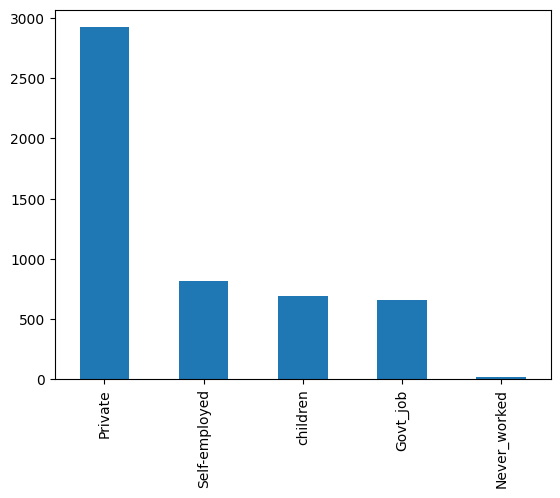


Comparison with the target attribute stroke:

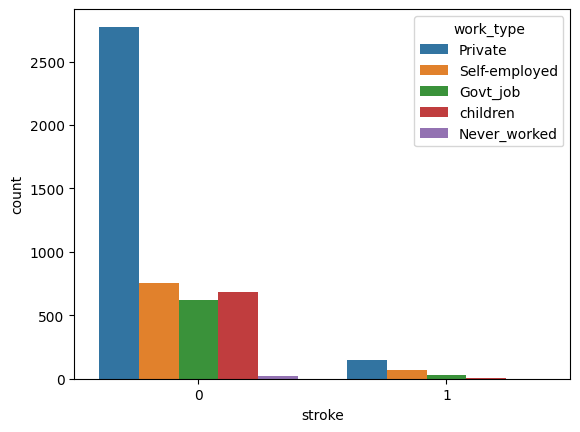


Inference: Probability of getting stroke is more for people having hypertension than people not having hypertension.

1. Work\_type attribute

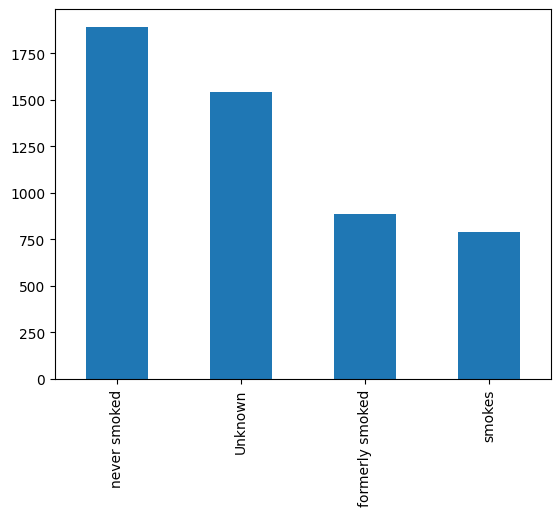
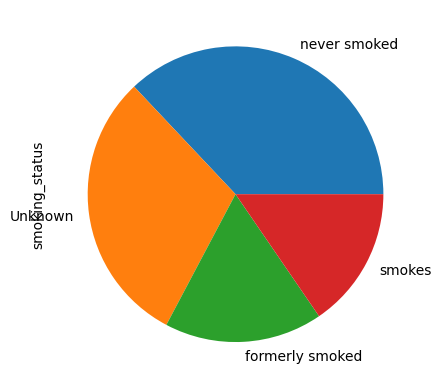


Comparison with the target attribute:



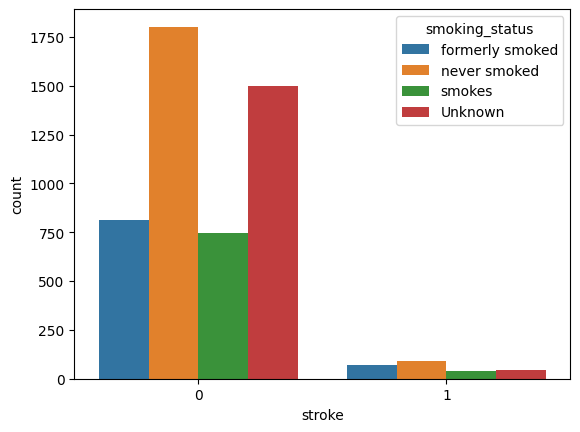
Inference: The majority of the people who got stroke are private workers.

1. Smoking\_status



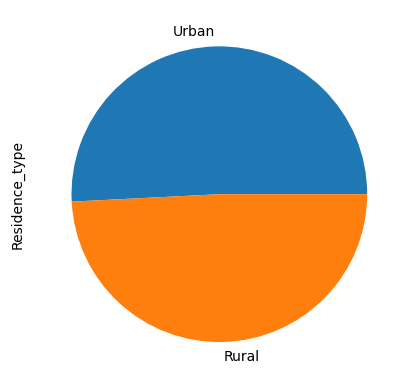
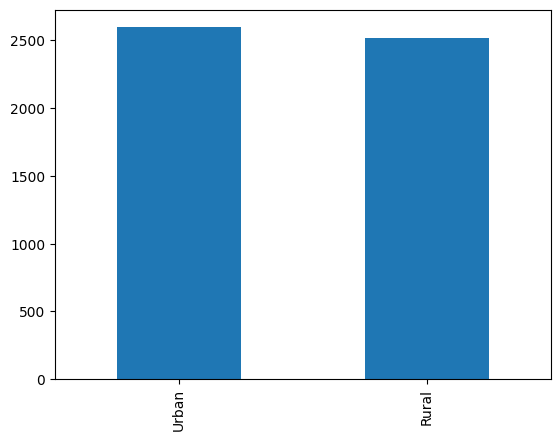
Comparison with target attribute

stroke:

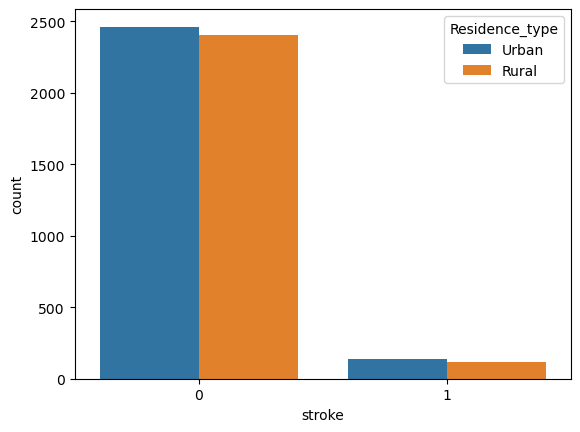


* Inference: Based on the plot we can that those who formerly smoked got more strokes The people who smoked and never smoked has a somewhat same probability of getting stroke

1. Residence\_type:



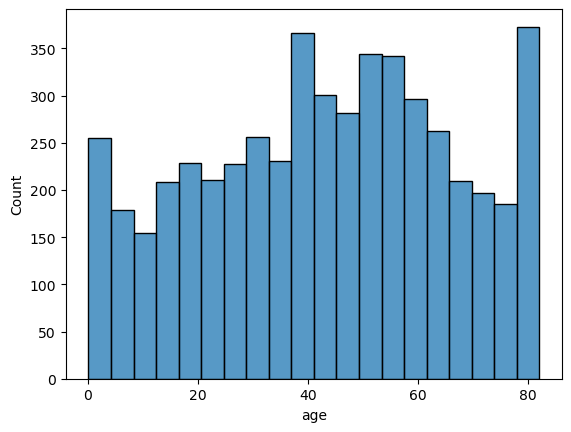
Comparison with target attribute:



* Inference: Based on the analysis the people who live in Urban areas were reported with more strokes

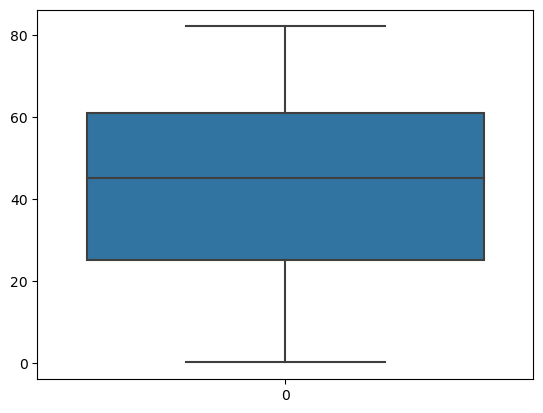
1. Age attribute

Since it is not categorical data, we used histogram to find the age of majority people.



Inference: It follows normal distribution

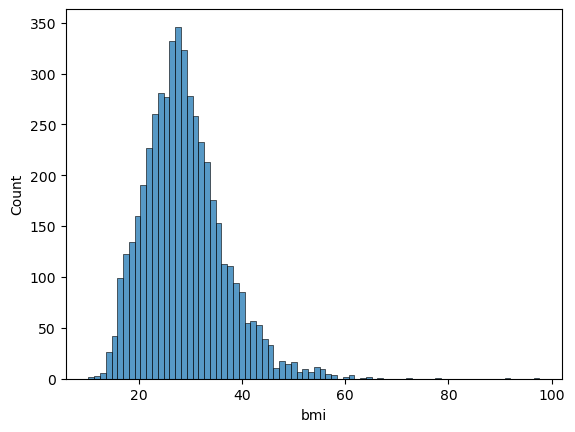
For visualisation of outliers we can use boxplot.



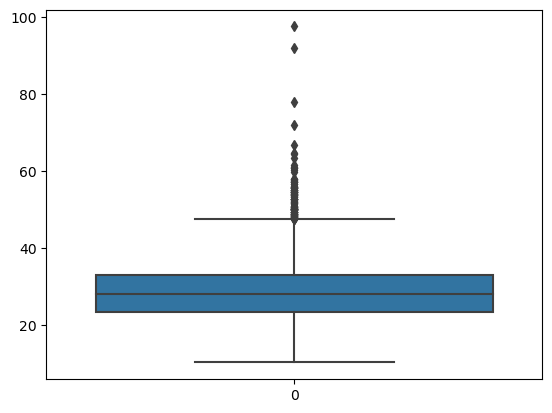
Inference: There are no outliers according to the box plot.

1. BMI attribute

Continuous data, so we can visualise using the histogram.

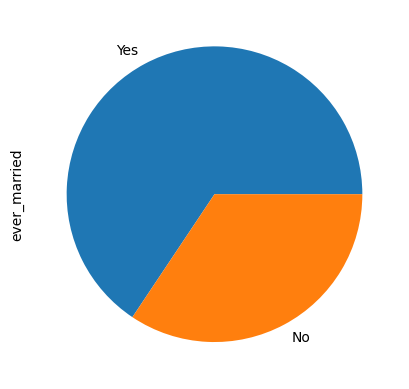
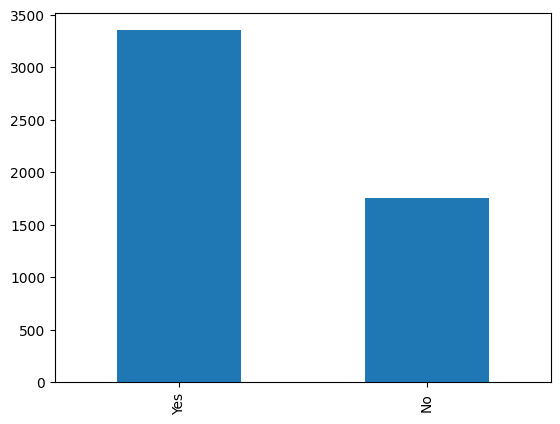


Box plot for outliers:

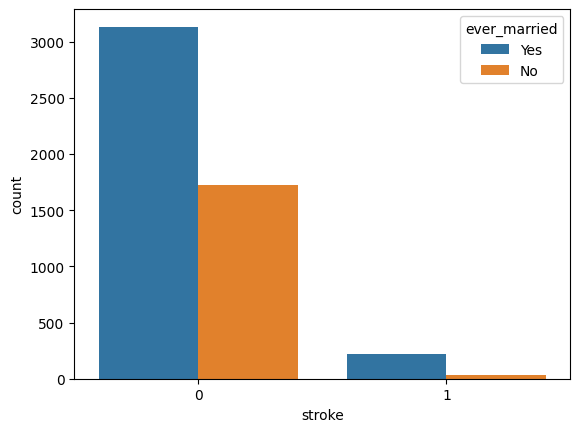


Inference: Based on the histogram and boxplot we see that there are many outliers in bmi.

1. Ever\_married attribute



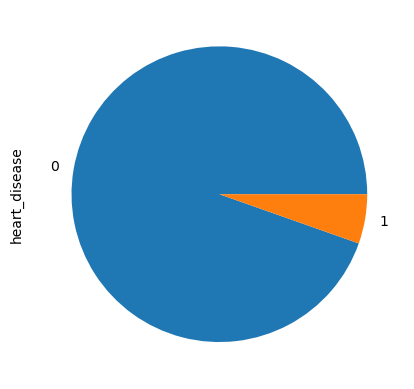
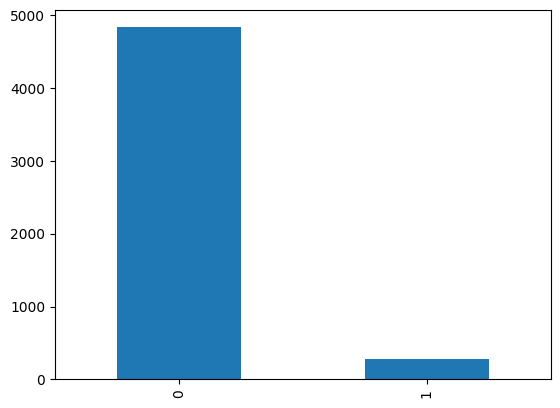
Comparison with the target attribute:



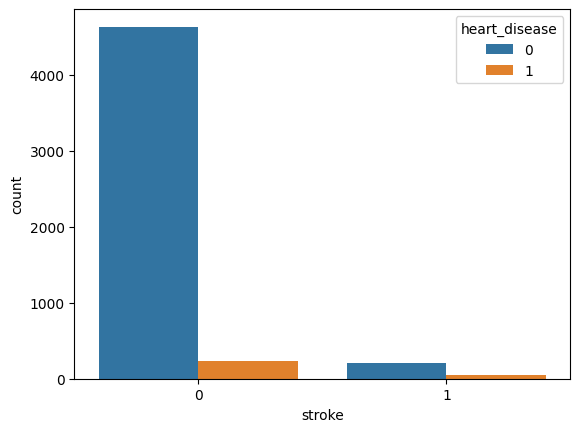
Inference: This plotting shows that the number of "Married people with Strokes" is approximately 10 to 12 times the no. "Unmarried people with Strokes".

This shows most of the Married people got Strokes compared to Unmarried people.

1. Heart\_disease attribute



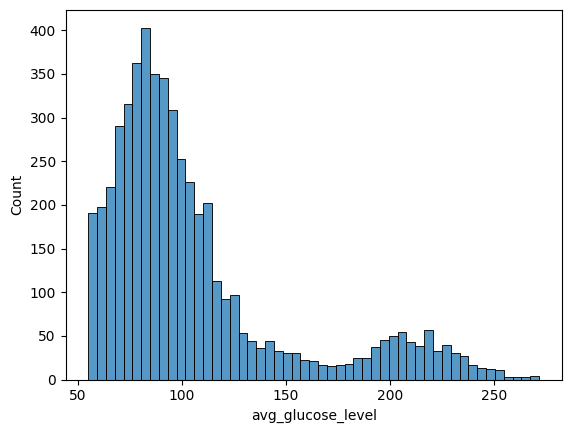
Comparison with target attribute:



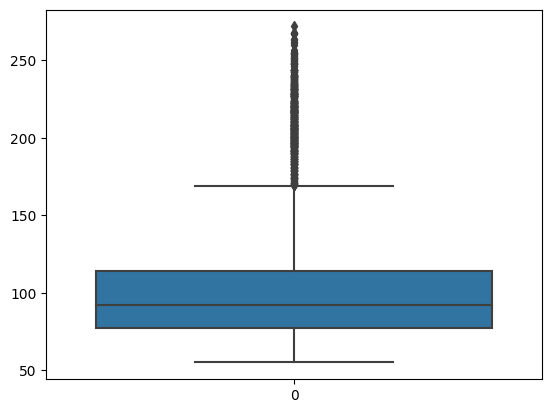
Inference:This plotting shows that the number of "people with Strokes but no heart disease" is approximately 6 to 8 times the number of "people with Strokes and also heart disease". This shows most of the people with no heart disease are suffering with Strokes compared to the once who have Heart Disease.

1. Average\_glucose\_level attribute

Numerical data, visualise using histogram



Visualisation using box plot check for outliers.



Inference: There are many outliers present based on the boxplot and histogram

**Exploratory Data Analysis(EDA):**

* 1. Gender:

Counts

Female: 2994

Male :2115

Other 1

We have a ‘other’ gender and since there is only 1 instance we will remove it. There are more females as compared to males

* 1. Target Attribute : Stroke

Counts

0: 4861 1 :249

Percentage of people who actually got a stroke : 4.9.Our dataset is highly skewed since only around 5% of the instances got stroke.We will be needing to perform necessary transformations to improve samples of minority class

* 1. Hyper – tension:

Counts

0: 4611

1: 498

* 1. Work Type :

Private 2925

Self-employed 819

children 687

Govt\_job 657

Never\_worked 22

* 1. Smoking Status:

Counts

never smoked 1892

Unknown 1544

formerly smoked 885

smokes 789

* 1. Residence Type:

Counts

Urban 2596

Rural 2513

We have almost equal percentage of people from urban and rural.

* 1. BMI

Null values – 201

NULL values hold 3.93 % of the instances in the dataframe.

The distribution is right skewed with skewness 1.05

People who got stroke and their BMI is NA: 40

People who got stroke and their BMI is given: 249

Percentage of people with stroke in Nan values to the overall dataset: 16.04

Among the 201 bmi NULL values 40 values in them got stroke

Thus we cant drop NULL values

Since there are outliers present we can’t perform mean imputation as mean is affected

by the outliers

Hence we impute it with median values

Median of the values:28.1

* 1. Age:

All rows have age.

* 1. Average\_glucose\_level:

Positively skewed with skewness 1.5

Outliers count 627.

* 1. Heart Disease

Count

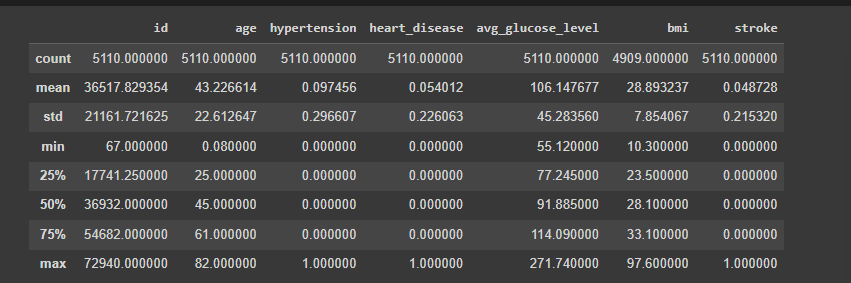
0 – 4833

1 – 276

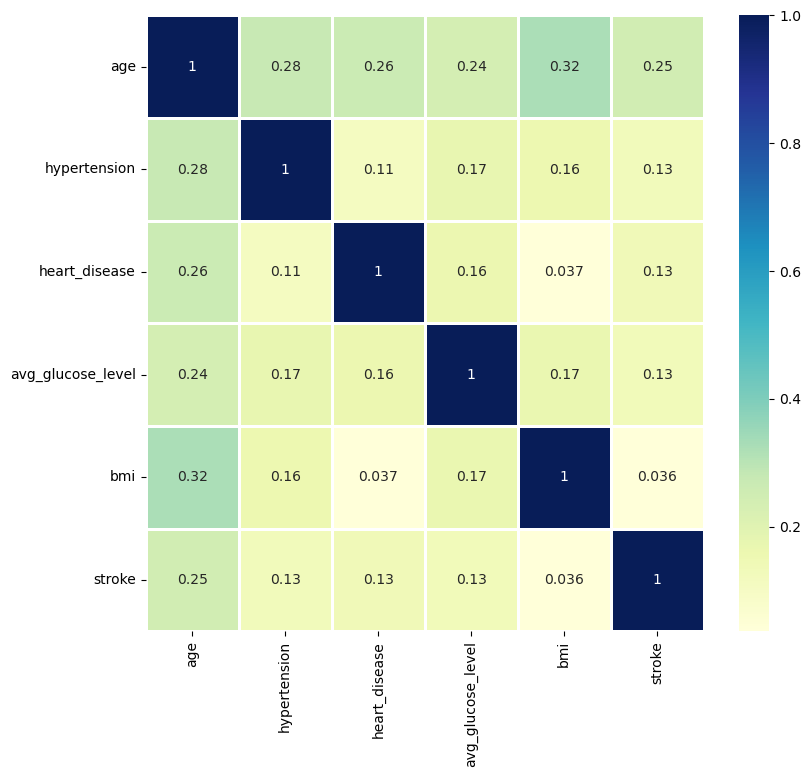
* 1. Ever married:

Count

Yes – 3353

No – 1756

**CORRELATION BETWEEN THE ATTRIBUTES**



There is a weak correlation between the attributes as per the plotted heatmap

The highest correlation found was between age and bmi - 0.32

Rest all correlations were less than 0.32

We could not drw any statistical insight from heatmap

**SPITTING THE DATA-**

We splitted the data for training and testing purpose using sklearn.select\_model library

**MODEL 1 – DECISION TREE**

By using the sklearn.tree library we can implement the decision tree with DecisionTreeClassifier

# Import the Decision Tree Classifier module

from sklearn.tree import DecisionTreeClassifier

# Libraries for calculating performance metrics

from sklearn import metrics

import numpy as np

from sklearn.metrics import auc, roc\_auc\_score, roc\_curve, precision\_score, recall\_score, f1\_score, accuracy\_score

import pandas as pd

# Create the classifier object

clf = DecisionTreeClassifier()

# Training the classifier

clf = clf.fit(X\_train, y\_train)

# Define a function to get user input

def get\_user\_input():

    # Get input from user

    print(df.columns)

    user\_input = input("Please enter your data in the given format: ")

    # Convert user input to a list

    use = user\_input.split()

    # Create a pandas series from the user input

    user\_data = pd.Series(use, index=X\_test.columns)

    # Append the user data to the test dataset

    gg= X\_test.append(user\_data, ignore\_index=True)

    # Return the last row of the test dataset

    print(gg.tail(1))

    return gg.tail(1)

# Get user data

u = get\_user\_input()

# Predict result using the user's data

y = clf.predict(u)

# Print the prediction.

print("The predicted output is:", "STORKE" if y==1 else "NO STROKE")

y\_pred=clf.predict(X\_test)

For User input (based on the attributes after one hot encoding done) we can add the user input to the X\_test dataframe and we can predict the output only for the appended input values using predict function.

**METRICS:**

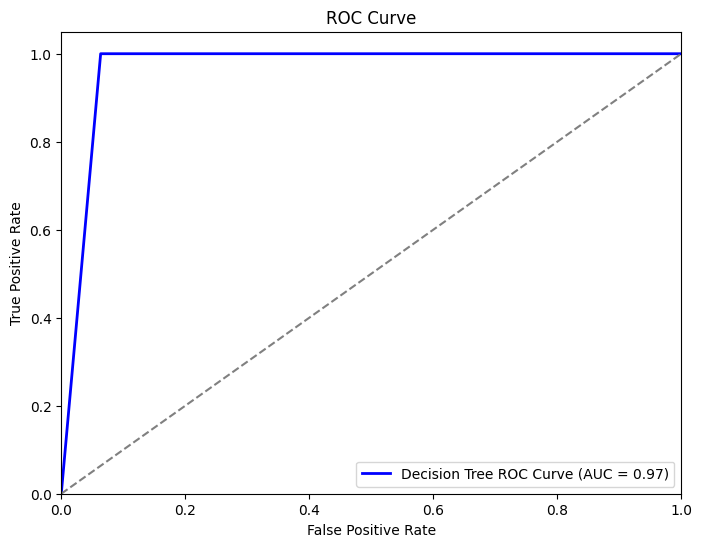
Accuracy: 0.97

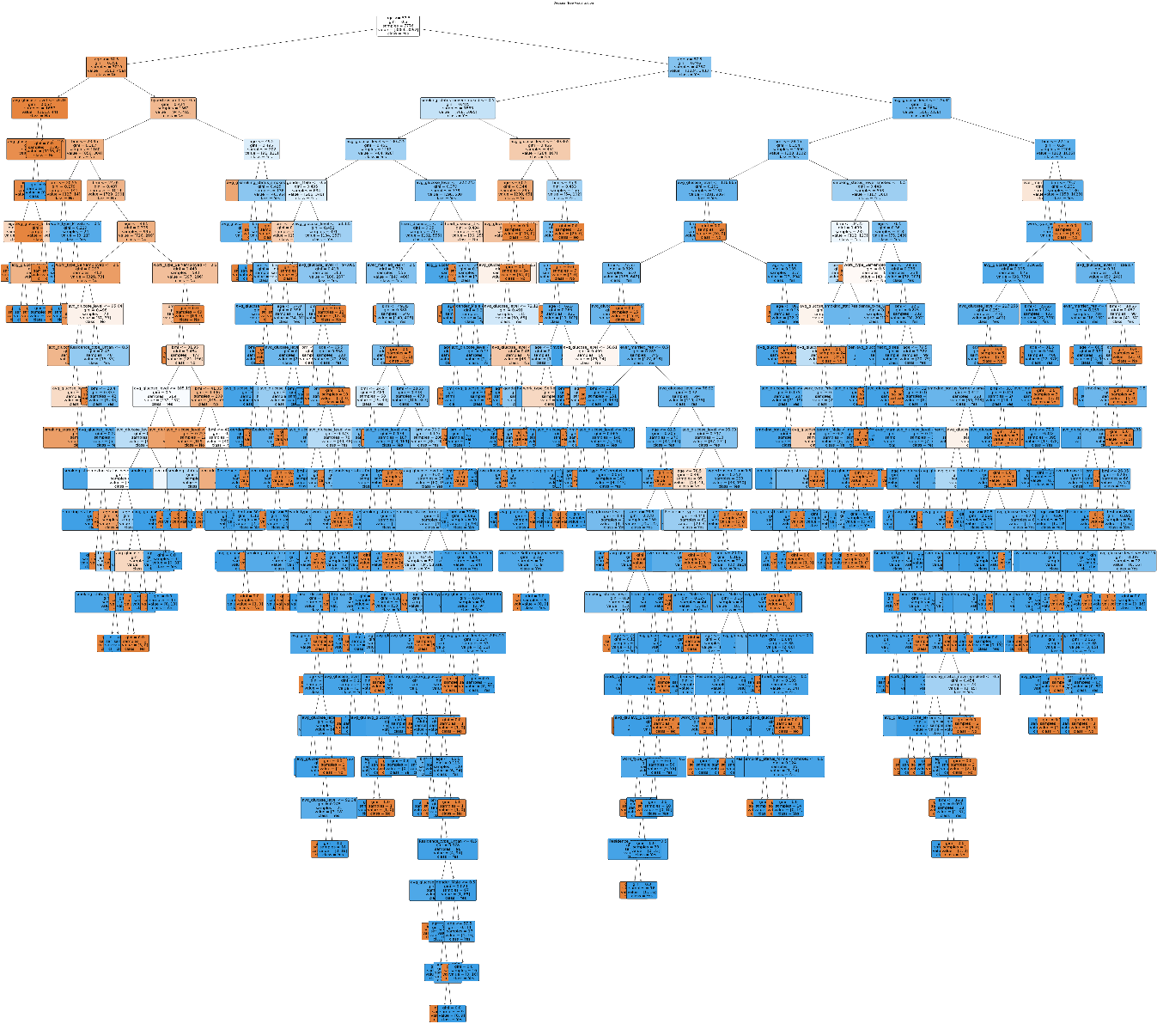
Precision: 0.94

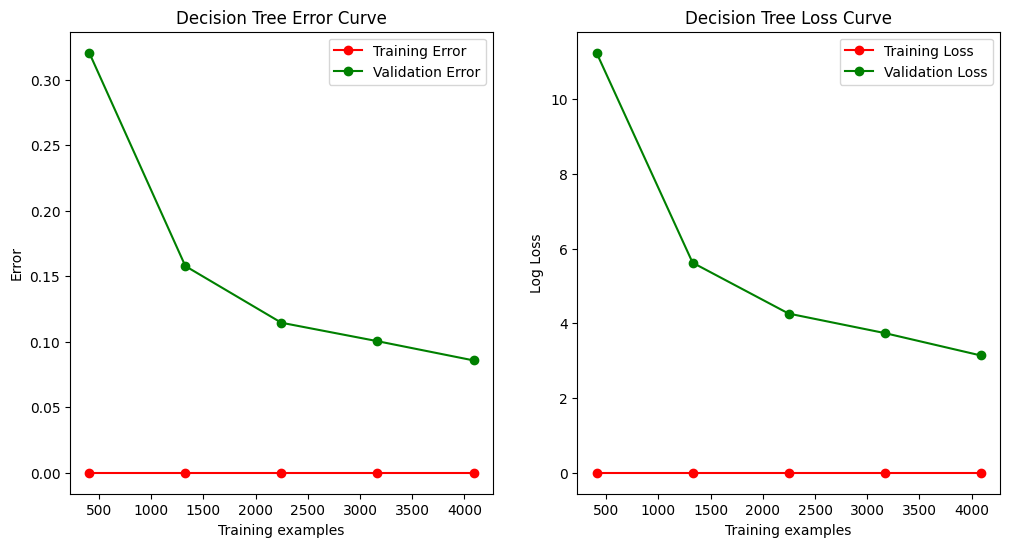
Recall: 1.00

F1 Score: 0.97

ROC AUC Score: 0.97

**ROC CURVE:**

Visualisation

**ERROR AND LOSS CURVES.**

**MODEL 2 – RANDOM FORESTS**

from sklearn.ensemble import RandomForestClassifier

# Create the classifier object

rf\_clf = RandomForestClassifier(n\_estimators = 10)

# Train the model using the training sets

rf\_clf.fit(X\_train, y\_train)

# performing predictions on the test dataset

y\_pred\_rf = rf\_clf.predict(X\_test)

# Define a function to get user input

def get\_user\_input():

    # Get input from user

    print(df.columns)

    user\_input = input("Please enter your data in the given format: ")

    # Convert user input to a list

    use = user\_input.split()

    # Create a pandas series from the user input

    user\_data = pd.Series(use, index=X\_test.columns)

    # Append the user data to the test dataset

    gg= X\_test.append(user\_data, ignore\_index=True)

    # Return the last row of the test dataset

    print(gg.tail(1))

    return gg.tail(1)

# Get user data

u = get\_user\_input()

# Predict result using the user's data

y = clf.predict(u)

# Print the prediction.

print("The predicted output is:", y)

**Visulaising**

from sklearn import tree

for i, t in enumerate(rf\_clf.estimators\_):

    plt.figure(figsize=(12, 8))

    tree.plot\_tree(t, filled=True, feature\_names=list(X\_train.columns), class\_names=['No', 'Yes'], rounded=True)

    plt.title(f"Decision Tree Visualization (Tree {i+1})")

    plt.show()

**Metrics:**

Accuracy: 0.99

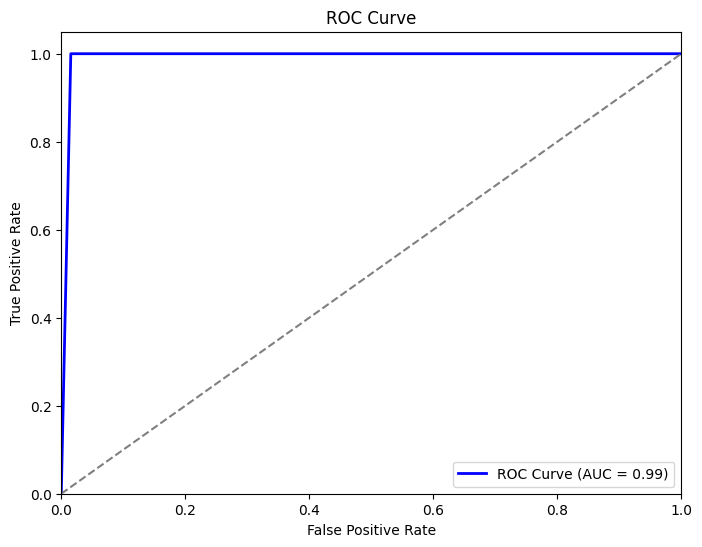
Precision: 0.98

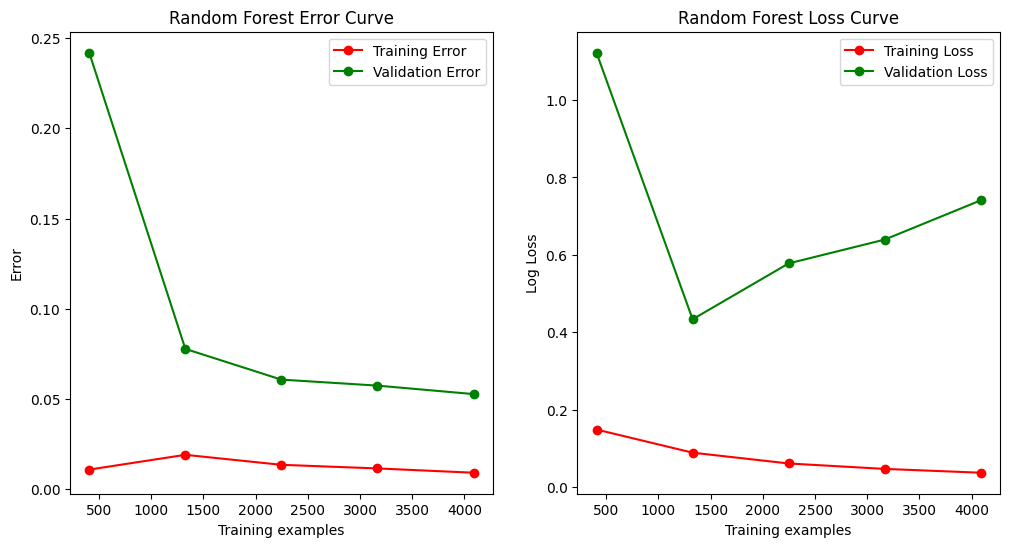
Recall: 1.00

F1 Score: 0.99

ROC AUC Score: 1.00

**ROC CURVE :**

****

**ERROR AND LOSS CURVES** ****

**MODEL 3–KNN**

#importing the necessary libraries

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report,accuracy\_score,confusion\_matrix

from sklearn.metrics import auc,roc\_auc\_score,roc\_curve,precision\_score,recall\_score,f1\_score

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

# Create the classifier object

knn = KNeighborsClassifier(n\_neighbors = 2)

# Training the classifier

knn.fit(X\_train,y\_train)

#predicting result using the test dataset

y\_pred\_knn = knn.predict(X\_test)

y\_pred\_prob\_knn = knn.predict\_proba(X\_test)[:, 1]

**Visulaising and metrics**

# Plot the confusion matrix

plot\_confusion\_matrix(y\_test, y\_pred\_knn, classes=['No Stroke', 'Stroke'], cmap=plt.cm.Blues)

print('Accuracy:',accuracy\_score(y\_test, y\_pred\_knn))

print('Precision:', precision\_score(y\_test, y\_pred\_knn))

print('Recall:', recall\_score(y\_test, y\_pred\_knn))

print('F1 Score:', f1\_score(y\_test, y\_pred\_knn))

print('ROC AUC Score:', roc\_auc\_score(y\_test, y\_pred\_prob\_knn))

error\_rate = []

accuracy\_rate = []

# Will take some time

for i in range(1,40):

    knn = KNeighborsClassifier(n\_neighbors=i)

    knn.fit(X\_train,y\_train)

    pred\_i = knn.predict(X\_test)

    error\_rate.append(np.mean(pred\_i != y\_test))

    accuracy\_rate.append(accuracy\_score(y\_test, pred\_i))

# Plotting the error and loss curve

plt.figure(figsize=(10,5))

plt.plot(range(1,40), error\_rate, color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

plt.show()

# Plotting the KNN graph

plt.figure(figsize=(10,5))

plt.plot(range(1,40), accuracy\_rate, color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersize=10)

plt.title('Accuracy Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Accuracy Rate')

plt.show()

fpr\_dt, tpr\_dt, thresholds\_dt = roc\_curve(y\_test, y\_pred\_knn)

# Calculate the ROC AUC score for the Decision Tree model

roc\_auc\_dt = roc\_auc\_score(y\_test, y\_pred\_knn)

# P

plt.figure(figsize=(8, 6))

plt.plot(fpr\_dt, tpr\_dt, color='b', lw=2, label=f'KNN ROC Curve (AUC = {roc\_auc\_dt:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

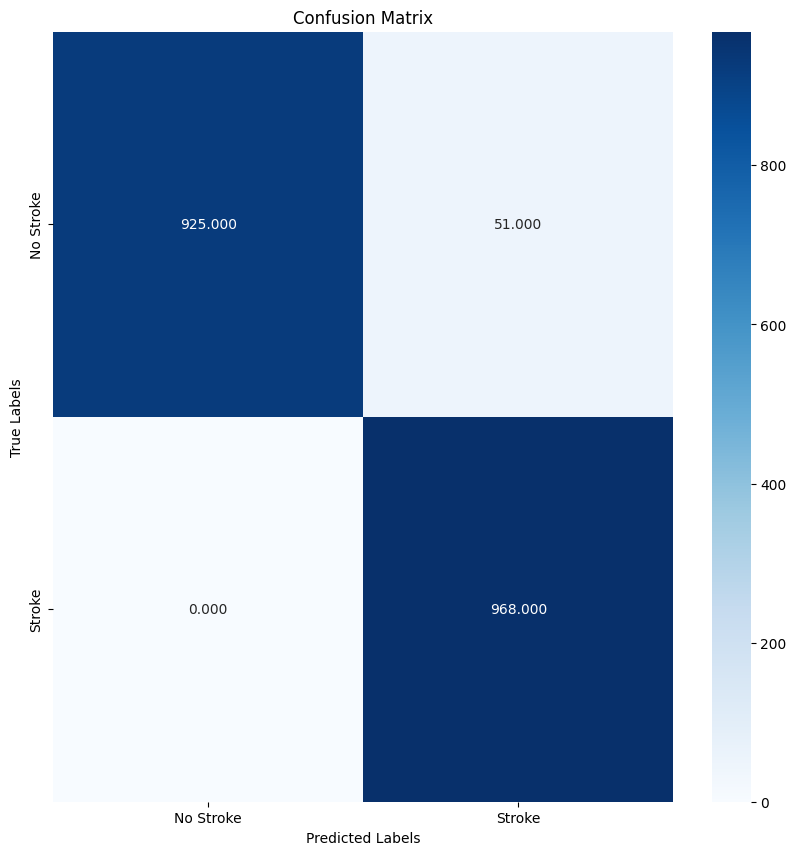
plt.xlabel('False Positive Rate')

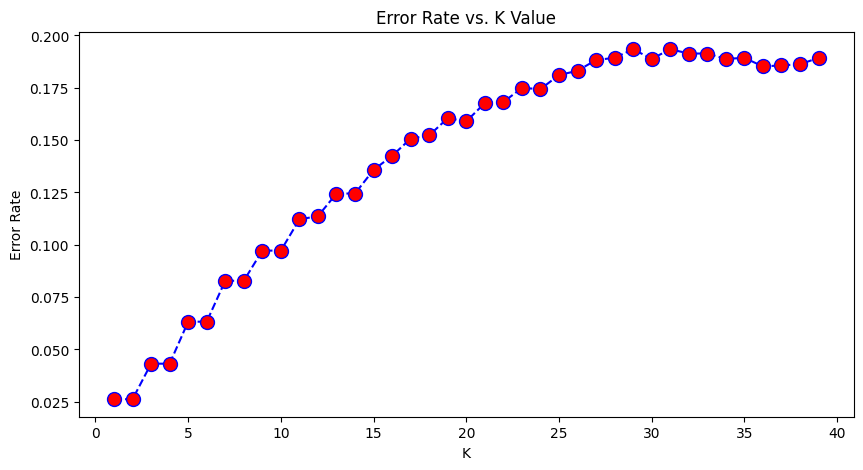
plt.ylabel('True Positive Rate')

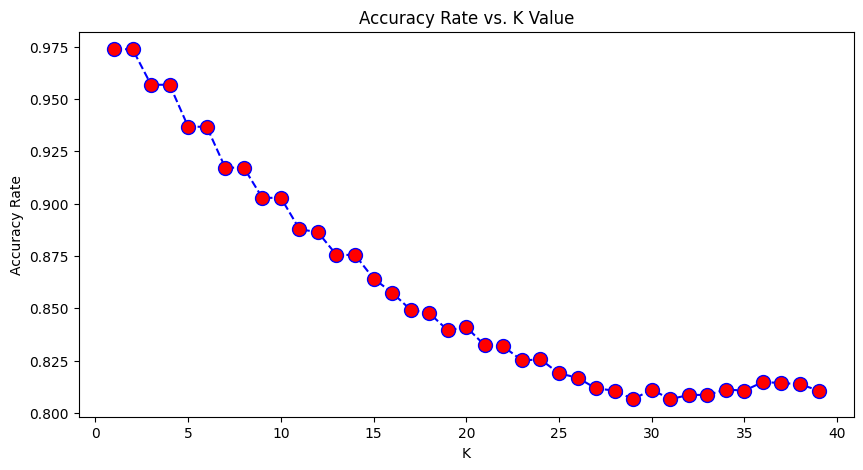
plt.title('ROC Curve')

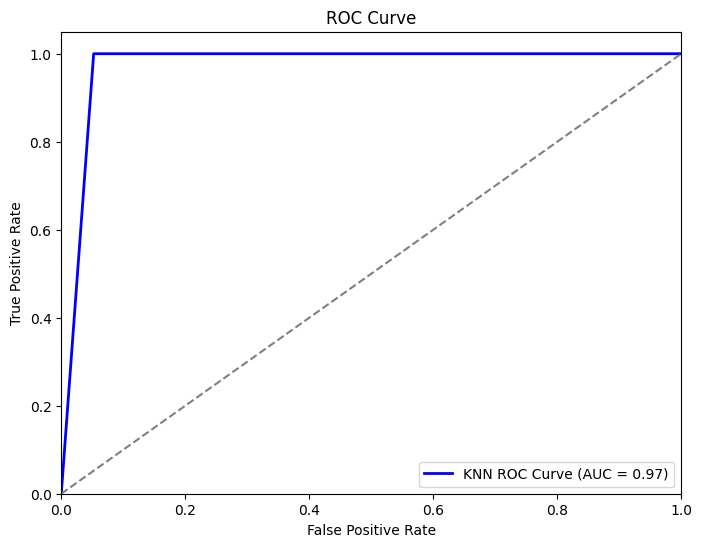
plt.legend(loc='lower right')

plt.show()









**Metrics:**



**MODEL 4:ARTIFICIAL NEURAL NETWORK**

**CODE:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the data

data = pd.read\_csv("healthcare-dataset-stroke-data.csv")

# Create the pair plot

sns.pairplot(data, diag\_kind="kde")

plt.title("Pairplot of Healthcare Dataset Stroke Data with respect to stroke\_1")

# Adjust the layout

plt.tight\_layout()

# Save the plot

plt.savefig("pairplot.png")

from sklearn import metrics

from sklearn.metrics import auc,roc\_auc\_score,roc\_curve,precision\_score,recall\_score,f1\_score,accuracy\_score

from sklearn.metrics import confusion\_matrix, accuracy\_score, roc\_curve, auc

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report,accuracy\_score,confusion\_matrix

from sklearn.metrics import auc,roc\_auc\_score,roc\_curve,precision\_score,recall\_score,f1\_score

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

model = keras.Sequential([

    layers.Input(shape=(X\_train.shape[1],), name='input\_layer'),

    layers.Dense(128, activation='LeakyReLU', name='dense\_1'),

    layers.Dropout(0.3, name='dropout\_1'),

    layers.Dense(64, activation='LeakyReLU', name='dense\_2'),

    layers.Dropout(0.3, name='dropout\_2'),

    layers.Dense(1, activation='sigmoid', name='output\_layer')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

# Evaluate the model

y\_pred = (model.predict(X\_test) > 0.5).astype("int32")

keras.utils.plot\_model(model, to\_file='model\_visualization.png', show\_shapes=True, show\_layer\_names=True)

# Get the input from the user

user\_input = {

    'age': float(input('Enter age: ')),

    'avg\_glucose\_level': float(input('Enter average glucose level: ')),

    'bmi': float(input('Enter bmi: ')),

    'gender\_Male': int(input('Enter gender (1 for Male, 0 for Female): ')),

    'hypertension\_1': int(input('Enter hypertension (1 for Yes, 0 for No): ')),

    'heart\_disease\_1': int(input('Enter heart disease (1 for Yes, 0 for No): ')),

    'ever\_married\_Yes': int(input('Enter ever married (1 for Yes, 0 for No): ')),

    'work\_type\_Never\_worked': int(input('Enter work type Never worked (1 for Yes, 0 for No): ')),

    'work\_type\_Private': int(input('Enter work type Private (1 for Yes, 0 for No): ')),

    'work\_type\_Self-employed': int(input('Enter work type Self-employed (1 for Yes, 0 for No): ')),

    'work\_type\_children': int(input('Enter work type children (1 for Yes, 0 for No): ')),

    'Residence\_type\_Urban': int(input('Enter residence type Urban (1 for Yes, 0 for No): ')),

    'smoking\_status\_formerly smoked': int(input('Enter smoking status formerly smoked (1 for Yes, 0 for No): ')),

    'smoking\_status\_never smoked': int(input('Enter smoking status never smoked (1 for Yes, 0 for No): ')),

    'smoking\_status\_smokes': int(input('Enter smoking status smokes (1 for Yes, 0 for No): ')),

}

# Convert the user input into a pandas DataFrame

user\_input\_df = pd.DataFrame([user\_input])

# Use the trained model to predict the output

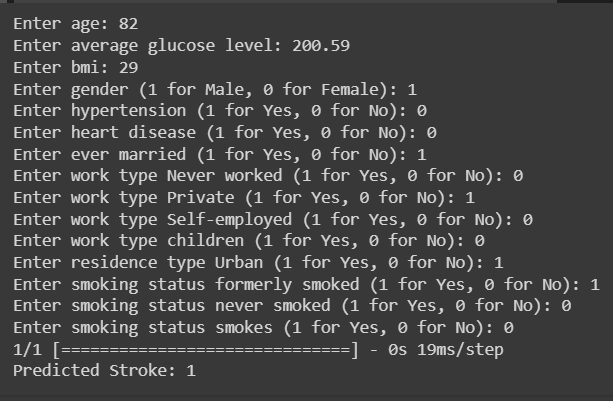
user\_pred\_ann = model.predict(user\_input\_df)

if(user\_pred\_ann > 0.5):

  print('Predicted Stroke:', 1)

else:

  print('Predicted Stroke:', 0)

****

print("Accuracy of our model on test data : " , model.evaluate(X\_test,y\_test)[1]\*100 , "%")

epochs = [i for i in range(50)]

fig , ax = plt.subplots(1,2)

train\_acc = history.history['accuracy']

train\_loss = history.history['loss']

test\_acc = history.history['val\_accuracy']

test\_loss = history.history['val\_loss']

fig.set\_size\_inches(20,6)

ax[0].plot(epochs , train\_loss , label = 'Training Loss')

ax[0].plot(epochs , test\_loss , label = 'Testing Loss')

ax[0].set\_title('Training & Testing Loss')

ax[0].legend()

ax[0].set\_xlabel("Epochs")

ax[1].plot(epochs , train\_acc , label = 'Training Accuracy')

ax[1].plot(epochs , test\_acc , label = 'Testing Accuracy')

ax[1].set\_title('Training & Testing Accuracy')

ax[1].legend()

ax[1].set\_xlabel("Epochs")

plt.show()

**EVALUATION METRICS:**

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, auc, roc\_auc\_score,confusion\_matrix

# Evaluate the model

y\_pred = (model.predict(X\_test) > 0.5).astype("int32")

# Plot the confusion matrix big and with colors

def plot\_confusion\_matrix(y\_true, y\_pred, classes, cmap=plt.cm.Blues):

  cm = confusion\_matrix(y\_true, y\_pred)

  fig, ax = plt.subplots(figsize=(10, 10))

  sns.heatmap(cm, annot=True, fmt=".3f", ax=ax, cmap=cmap)

  ax.set\_title('Confusion Matrix')

  ax.set\_xlabel('Predicted Labels')

  ax.set\_ylabel('True Labels')

  ax.set\_xticklabels(classes)

  ax.set\_yticklabels(classes)

  plt.show()

# Plot the confusion matrix

plot\_confusion\_matrix(y\_test, y\_pred, classes=['No Stroke', 'Stroke'], cmap=plt.cm.Blues)

# Print the confusion matrix

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

# Print the metrics

print(f"Accuracy: {accuracy:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1 Score: {f1:.2f}")

**AREA OF ROC:**

y\_prob = model.predict(X\_test)

from sklearn.metrics import roc\_curve, roc\_auc\_score

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

roc\_auc = roc\_auc\_score(y\_test, y\_prob)

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.show()

GRAPH FOR ERROR CURVE AND LOSS CURVE

# Train the model and obtain the training history

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

# Extract training and validation loss and accuracy from the history

train\_loss = history.history['loss']

val\_loss = history.history['val\_loss']

train\_accuracy = history.history['accuracy']

val\_accuracy = history.history['val\_accuracy']

# Create the error curve (1 - accuracy)

train\_error = [1 - acc for acc in train\_accuracy]

val\_error = [1 - acc for acc in val\_accuracy]

# Plot Error Curve

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(train\_error, 'o-', color="r", label="Training Error")

plt.plot(val\_error, 'o-', color="g", label="Validation Error")

plt.title("Neural Network Error Curve")

plt.xlabel("Epochs")

plt.ylabel("Error")

plt.legend(loc="best")

# Plot Loss Curve

plt.subplot(1, 2, 2)

plt.plot(train\_loss, 'o-', color="r", label="Training Loss")

plt.plot(val\_loss, 'o-', color="g", label="Validation Loss")

plt.title("Neural Network Loss Curve")

plt.xlabel("Epochs")

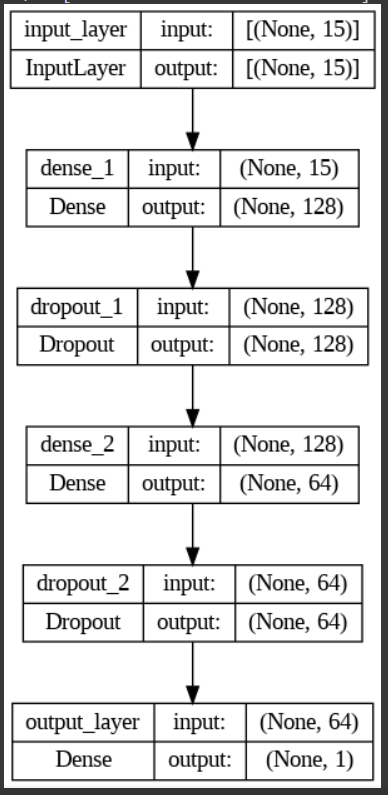
plt.ylabel("Loss")

plt.legend(loc="best")

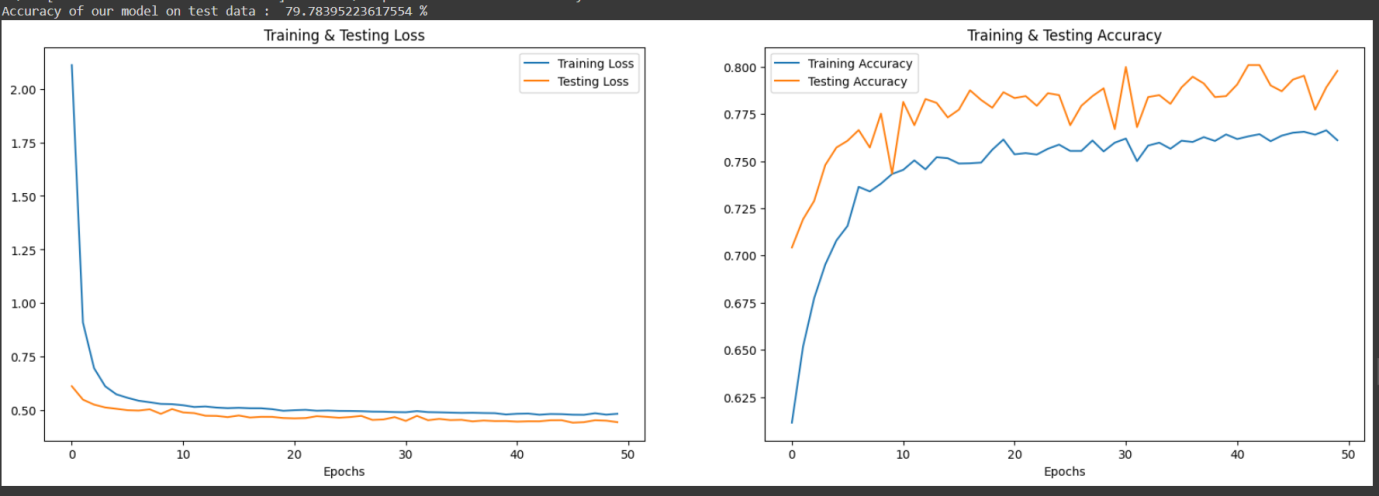
plt.show()

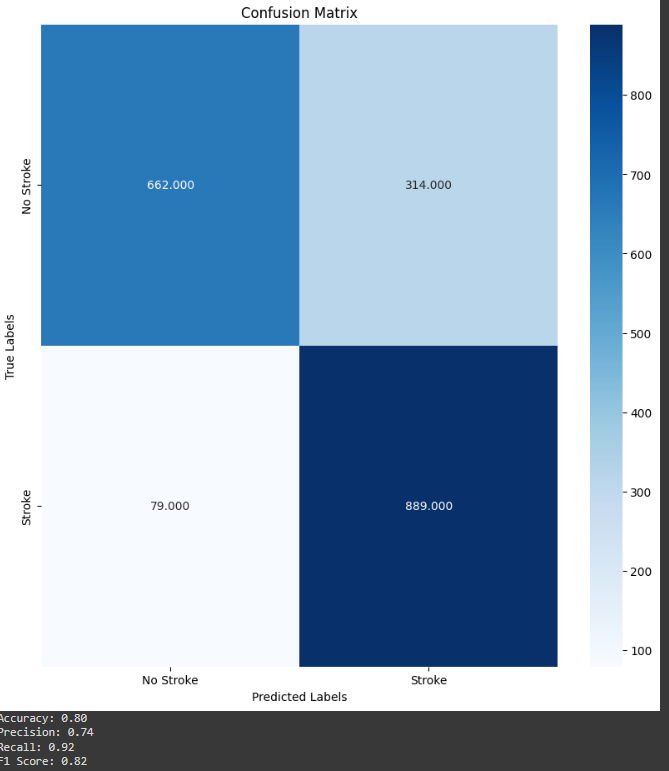
**VISUALIZATION:**

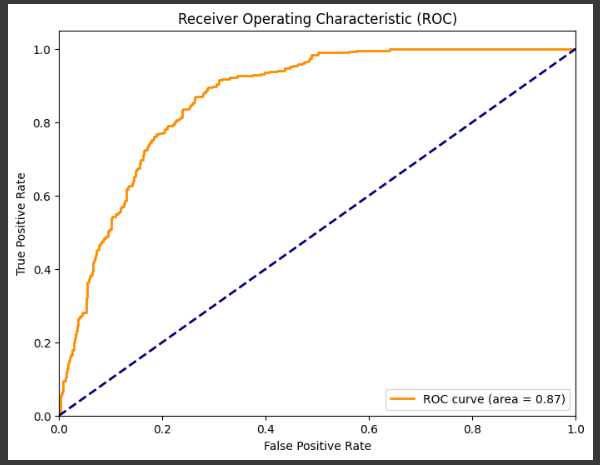
**ARTIFICIAL NEURAL NETWORK:**

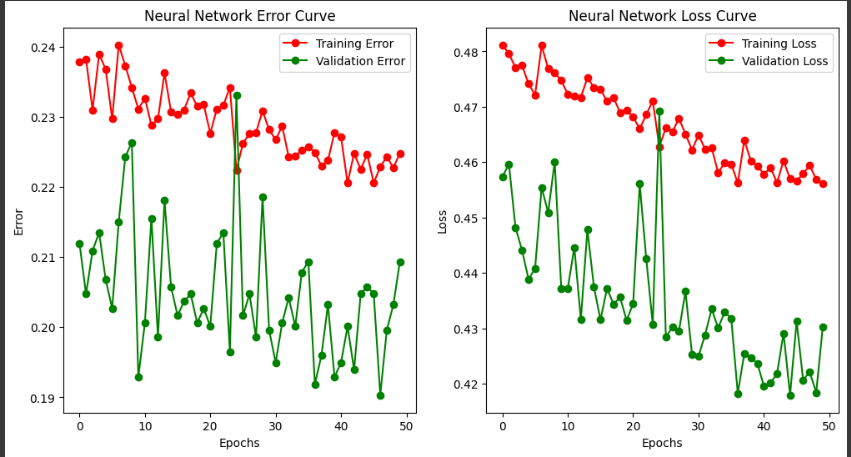
****

**LOSS AND ACCURACY CURVES OF TRAINING AND TESTING:**

****

****

****

****

**MODEL 5:LOGISTIC REGRESSION**

**CODE:**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, accuracy\_score, roc\_curve, auc

import matplotlib.pyplot as plt

# Create and train the logistic regression model

classifier = LogisticRegression(random\_state=0, max\_iter=1000)

classifier.fit(X\_train, y\_train)

# Make predictions

y\_pred\_lr = classifier.predict(X\_test)

loss = classifier.score(X\_test, y\_test)

# Plot the logistic regression graph

plt.figure()

plt.plot(X\_test, y\_pred\_lr, 'o-', color='b')

plt.xlabel('Feature')

plt.ylabel('Predicted probability')

plt.title('Logistic regression graph')

plt.show()

print("Loss:", loss)

# Get the input from the user

user\_input = {

    'age': float(input('Enter age: ')),

    'avg\_glucose\_level': float(input('Enter average glucose level: ')),

    'bmi': float(input('Enter bmi: ')),

    'gender\_Male': int(input('Enter gender (1 for Male, 0 for Female): ')),

    'hypertension\_1': int(input('Enter hypertension (1 for Yes, 0 for No): ')),

    'heart\_disease\_1': int(input('Enter heart disease (1 for Yes, 0 for No): ')),

    'ever\_married\_Yes': int(input('Enter ever married (1 for Yes, 0 for No): ')),

    'work\_type\_Never\_worked': int(input('Enter work type Never worked (1 for Yes, 0 for No): ')),

    'work\_type\_Private': int(input('Enter work type Private (1 for Yes, 0 for No): ')),

    'work\_type\_Self-employed': int(input('Enter work type Self-employed (1 for Yes, 0 for No): ')),

    'work\_type\_children': int(input('Enter work type children (1 for Yes, 0 for No): ')),

    'Residence\_type\_Urban': int(input('Enter residence type Urban (1 for Yes, 0 for No): ')),

    'smoking\_status\_formerly smoked': int(input('Enter smoking status formerly smoked (1 for Yes, 0 for No): ')),

    'smoking\_status\_never smoked': int(input('Enter smoking status never smoked (1 for Yes, 0 for No): ')),

    'smoking\_status\_smokes': int(input('Enter smoking status smokes (1 for Yes, 0 for No): ')),

}

# Convert the user input into a pandas DataFrame

user\_input\_df = pd.DataFrame([user\_input])

# Use the trained logistic regression model to predict the output

user\_pred\_ann = model.predict(user\_input\_df)

# Threshold the prediction at 0.5

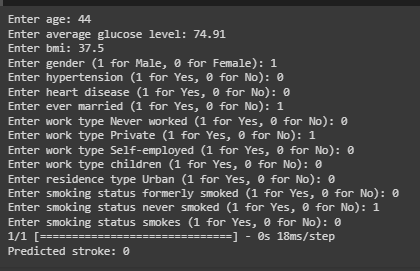
user\_pred = user\_pred\_ann > 0.5

if(user\_pred == True):

  print('Predicted stroke: 1')

else:

  print('Predicted stroke: 0')

****

**EVALUATION METRICS:**

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, auc, roc\_auc\_score,confusion\_matrix

import matplotlib.pyplot as plt

# Get the confusion matrix

cm = confusion\_matrix(y\_test, y\_pred\_lr)

# Calculate the other evaluation metrics

accuracy = accuracy\_score(y\_test, y\_pred\_lr)

precision = precision\_score(y\_test, y\_pred\_lr)

recall = recall\_score(y\_test, y\_pred\_lr)

f1\_score = f1\_score(y\_test, y\_pred\_lr)

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_lr)

# Print the evaluation metrics

def plot\_confusion\_matrix(y\_true, y\_pred, classes, cmap=plt.cm.Blues):

  cm = confusion\_matrix(y\_true, y\_pred\_lr)

  fig, ax = plt.subplots(figsize=(10, 10))

  sns.heatmap(cm, annot=True, fmt=".3f", ax=ax, cmap=cmap)

  ax.set\_title('Confusion Matrix')

  ax.set\_xlabel('Predicted Labels')

  ax.set\_ylabel('True Labels')

  ax.set\_xticklabels(classes)

  ax.set\_yticklabels(classes)

  plt.show()

# Plot the confusion matrix

plot\_confusion\_matrix(y\_test, y\_pred\_lr, classes=['No Stroke', 'Stroke'], cmap=plt.cm.Blues)

# Print the confusion matrix

print(cm)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 score:", f1\_score)

print("ROC AUC:", roc\_auc)

**AREA OF ROC CURVE:**

from sklearn.metrics import roc\_curve, roc\_auc\_score

import matplotlib.pyplot as plt

# Get the predictions from the logistic regression model

y\_pred\_lr = classifier.predict(X\_test)

# Calculate the ROC curve and AUC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_lr)

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_lr)

# Plot the ROC curve

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

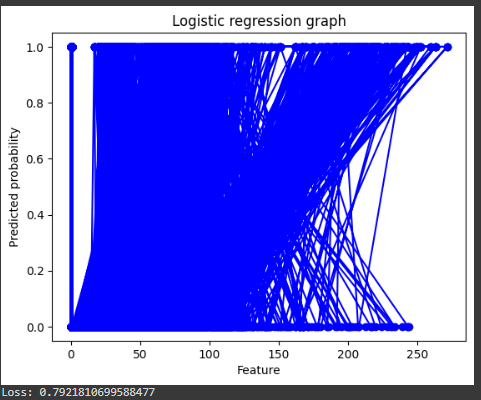
plt.title('Receiver Operating Characteristic')

plt.legend(loc='lower right')

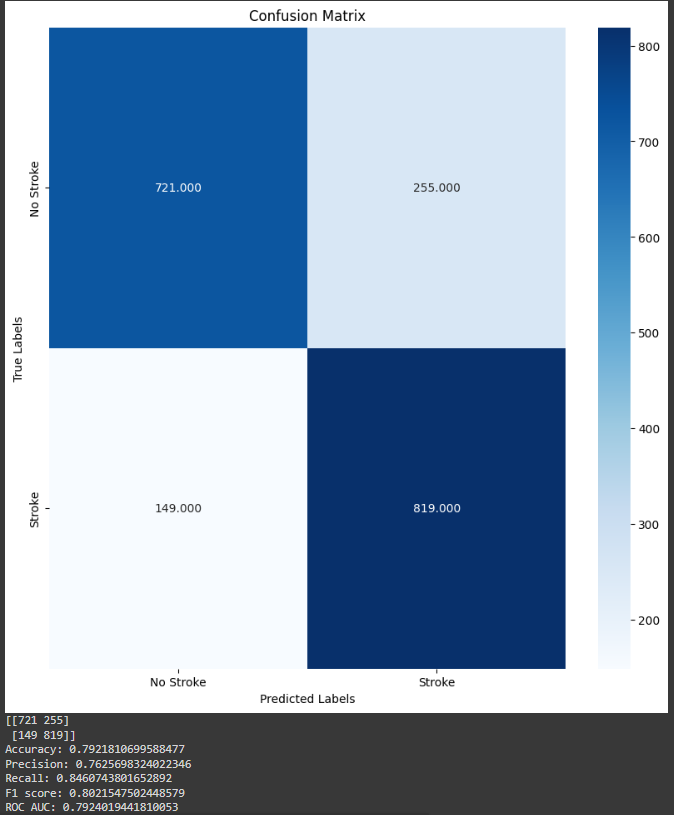
plt.show()

**VISUALIZATION:**

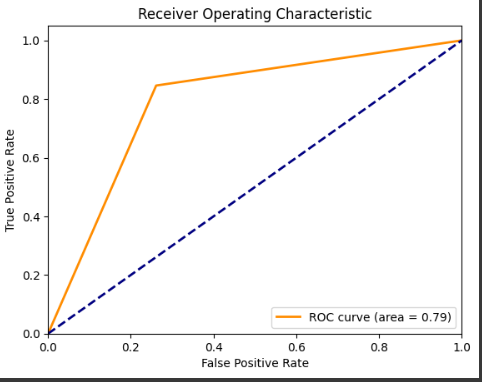
**LOGISTIC REGRESSION**

****

**EVALUATION METRICS:**

****

**AREA OF ROC CURVE:**

****

**MODEL 6 Naive Bayes**

**Model Selection: Naive Bayes for Brain Stroke Prediction**

**In the pursuit of developing an effective predictive model for brain stroke occurrence, the choice of an appropriate algorithm plays a pivotal role. After careful consideration of various factors, the Naive Bayes algorithm emerged as a fitting candidate for this particular task.**

**1. Simplicity and Speed:**

**The Naive Bayes algorithm is renowned for its simplicity. Its underlying assumption of conditional independence among features, given the class label, not only simplifies the model but also contributes to computational efficiency. In the context of brain stroke prediction, where timely decision-making is crucial, the algorithm's speed becomes a valuable asset.**

**2. Interpretability:**

**Interpretability is paramount in medical applications. Naive Bayes provides a clear and intuitive understanding of the relationship between features and the likelihood of a stroke. This interpretability is vital for medical professionals who need insights into the factors contributing to stroke prediction.**

**3. Handling Categorical Data:**

**Medical datasets often encompass categorical features, such as gender, work type, and smoking status. Naive Bayes handles these types of data gracefully, making it suitable for datasets with a mix of numerical and categorical features.**

**4. Few Hyperparameters:**

**Naive Bayes typically involves fewer hyperparameters compared to more complex models. This is advantageous when working with limited data and facilitates a more straightforward tuning process.**

**5. Assumption of Independence:**

**While the assumption of independence among features may not perfectly align with the intricacies of real-world data, it can serve as a reasonable approximation. In specific health-related contexts, certain features may indeed exhibit a degree of independence, justifying the application of the Naive Bayes model.**

**Challenges and Future Considerations:**

**While the Naive Bayes model offers notable advantages, it is imperative to acknowledge its limitations. The assumption of feature independence may not always hold, and in instances where the model's accuracy falls short, exploring alternative models becomes a logical next step.  
  
  
Advantages**

* It is not only a simple approach but also a fast and accurate method for prediction.
* Naive Bayes has a very low computation cost.
* It can efficiently work on a large dataset.
* It performs well in case of discrete response variable compared to the continuous variable.
* It can be used with multiple class prediction problems.
* It also performs well in the case of text analytics problems.
* When the assumption of independence holds, a Naive Bayes classifier performs better compared to other models like logistic regression.

**Disadvantages**

* The assumption of independent features. In practice, it is almost impossible that model will get a set of predictors which are entirely independent.
* If there is no training tuple of a particular class, this causes zero posterior probability. In this case, the model is unable to make predictions. This problem is known as Zero Probability/Frequency Problem.

**Using module part -1**

import pandas as pd

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, precision\_score, recall\_score, f1\_score, roc\_auc\_score

data = df

features = [

'age',

'avg\_glucose\_level',

'bmi',

'gender\_Male',

'hypertension\_1',

'heart\_disease\_1',

'ever\_married\_Yes',

'work\_type\_Never\_worked',

'work\_type\_Private',

'work\_type\_Self-employed',

'work\_type\_children',

'Residence\_type\_Urban',

'smoking\_status\_formerly smoked',

'smoking\_status\_never smoked',

'smoking\_status\_smokes'

]

target = 'stroke\_1'

X = data[features]

y = data[target]

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

gnb\_clf = GaussianNB(var\_smoothing=1e-9)

gnb\_clf.fit(X\_train, y\_train)

gnb\_scores = cross\_val\_score(gnb\_clf, X, y, cv=5)

print(f"Gaussian Naive Bayes Cross-Validation Accuracy: {gnb\_scores.mean()}")

user\_input = input("Enter the values for the features (separated by spaces): ").split()

input\_df = pd.DataFrame([user\_input], columns=features)

gnb\_predictions = gnb\_clf.predict(input\_df)

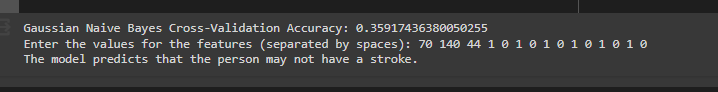
if gnb\_predictions[0] == 1:

  print("The model predicts that the person may have a stroke.")

else:

  print("The model predicts that the person may not have a stroke.")

gnb\_y\_pred = gnb\_clf.predict(X\_test)

  
  
  
  
**Metrics:**

# Calculate the evaluation metrics for the Gaussian Naive Bayes model

gnb\_accuracy = accuracy\_score(y\_test, gnb\_y\_pred)

gnb\_precision = precision\_score(y\_test, gnb\_y\_pred)

gnb\_recall = recall\_score(y\_test, gnb\_y\_pred)

gnb\_f1 = f1\_score(y\_test, gnb\_y\_pred)

gnb\_roc\_auc = roc\_auc\_score(y\_test, gnb\_clf.predict\_proba(X\_test)[:, 1])

# Print the evaluation metrics

print("\nGaussian Naive Bayes Evaluation Metrics:")

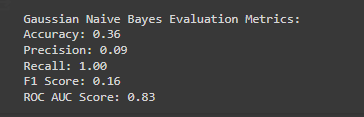
print(f"Accuracy: {gnb\_accuracy:.2f}")

print(f"Precision: {gnb\_precision:.2f}")

print(f"Recall: {gnb\_recall:.2f}")

print(f"F1 Score: {gnb\_f1:.2f}")

print(f"ROC AUC Score: {gnb\_roc\_auc:.2f}")



Roc curve:

import pandas as pd

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, roc\_curve

fpr\_gnb, tpr\_gnb, thresholds\_gnb = roc\_curve(y\_test, gnb\_y\_pred)

roc\_auc\_gnb = roc\_auc\_score(y\_test, gnb\_y\_pred)

plt.figure(figsize=(8, 6))

plt.plot(fpr\_gnb, tpr\_gnb, color='b', lw=2, label='Gaussian Naive Bayes (AUC = {roc\_auc\_gnb:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

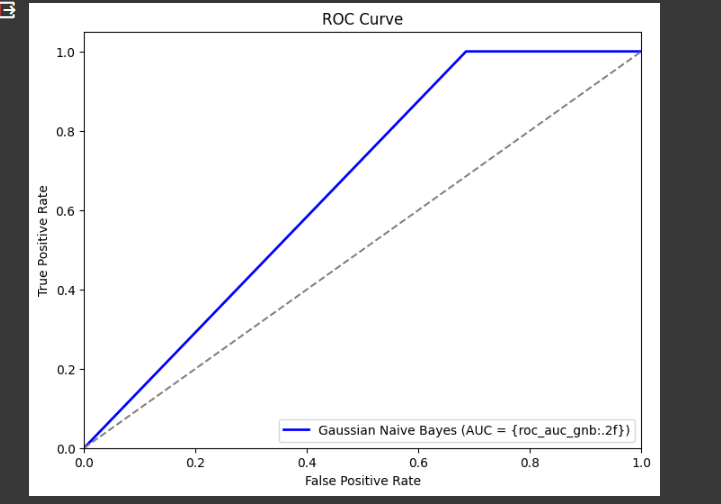
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc='lower right')

plt.show()



ERROR and LOSS curve  
import pandas as pd

from sklearn.model\_selection import train\_test\_split, learning\_curve

from sklearn.naive\_bayes import GaussianNB

import matplotlib.pyplot as plt

import numpy as np

# Load the dataset

data=df

# Define the features and target variable

features = [

'age',

'avg\_glucose\_level',

'bmi',

'gender\_Male',

'hypertension\_1',

'heart\_disease\_1',

'ever\_married\_Yes',

'work\_type\_Never\_worked',

'work\_type\_Private',

'work\_type\_Self-employed',

'work\_type\_children',

'Residence\_type\_Urban',

'smoking\_status\_formerly smoked',

'smoking\_status\_never smoked',

'smoking\_status\_smokes'

]

target = 'stroke\_1'

X = data[features]

y = data[target]

# Split the data into training and test sets

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

# Create a Gaussian Naive Bayes classifier

gnb\_clf = GaussianNB()

# Calculate the learning curve

train\_sizes, train\_scores, test\_scores = learning\_curve(gnb\_clf, X\_train, y\_train, cv=5, scoring='accuracy')

train\_scores\_mean = 1 - np.mean(train\_scores, axis=1)

test\_scores\_mean = 1 - np.mean(test\_scores, axis=1)

# Calculate the loss curve

train\_sizes, train\_loss, test\_loss = learning\_curve(gnb\_clf, X\_train, y\_train, cv=5, scoring='neg\_log\_loss')

train\_loss\_mean = -np.mean(train\_loss, axis=1)

test\_loss\_mean = -np.mean(test\_loss, axis=1)

# Plot the error and loss curve

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r", label="Training Error")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g", label="Validation Error")

plt.title("Gaussian Naive Bayes Error Curve")

plt.xlabel("Training examples")

plt.ylabel("Error")

plt.legend(loc="best")

# Plot the loss curve

plt.subplot(1, 2, 2)

plt.plot(train\_sizes, train\_loss\_mean, 'o-', color="r", label="Training Loss")

plt.plot(train\_sizes, test\_loss\_mean, 'o-', color="g", label="Validation Loss")

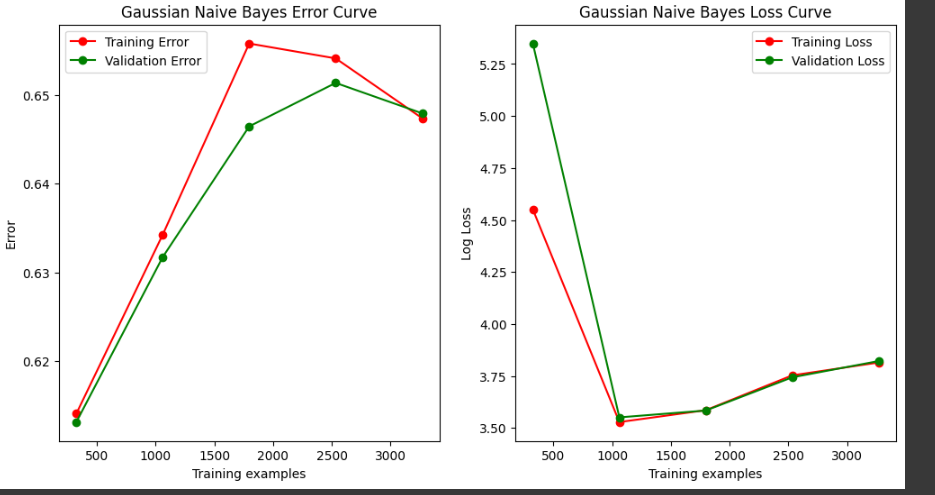
plt.title("Gaussian Naive Bayes Loss Curve")

plt.xlabel("Training examples")

plt.ylabel("Log Loss")

plt.legend(loc="best")

plt.show()



Naive Bayes part 2

import pandas as pd

# Load the stroke data into a DataFrame

data = pd.read\_csv('/content/healthcare-dataset-stroke-data.csv')

available\_columns = data.columns.tolist()

chosen\_column = input("Enter the column name (gender, age, hypertension, heart\_disease, ever\_married, work\_type, Residence\_type, avg\_glucose\_level, bmi, smoking\_status): ")

chosen\_value = input(f"Enter the value for {chosen\_column}: ")

if chosen\_column not in available\_columns:

    print(f"Invalid column name '{chosen\_column}'. Please enter one of the following: ")

    for column in available\_columns:

        print(f"- {column}")

    exit()

try:

    chosen\_value = float(chosen\_value)

except ValueError:

    if chosen\_column in ['gender', 'work\_type', 'Residence\_type', 'smoking\_status']:

        # Check if the input value is a valid categorical value

        if chosen\_value not in data[chosen\_column].unique():

            print(f"Invalid value '{chosen\_value}' for column '{chosen\_column}'.")

            exit()

    else:

        print(f"Invalid value '{chosen\_value}' for column '{chosen\_column}'.")

        exit()

if chosen\_value in data[chosen\_column].unique():

    class\_prior\_probabilities = data['stroke'].value\_counts(normalize=True)

    frequency\_table = data.groupby([chosen\_column, 'stroke']).size().unstack().fillna(0)

    posterior\_probabilities = frequency\_table.apply(lambda x: x / x.sum(), axis=1)

    predicted\_probabilities = {}

    for outcome in data['stroke'].unique():

        probability = class\_prior\_probabilities[outcome] \* posterior\_probabilities.loc[chosen\_value, outcome]

        predicted\_probabilities[outcome] = probability

    map\_estimate = max(predicted\_probabilities, key=predicted\_probabilities.get)

    sorted\_predictions = {k: v for k, v in sorted(predicted\_probabilities.items(), key=lambda item: item[1], reverse=True)}

    print("Class Prior Probabilities:")

    print(class\_prior\_probabilities)

    print("\n")

    print(f"Frequency Table for {chosen\_column} and stroke:")

    print(frequency\_table)

    print("\n")

    print("Posterior Probabilities:")

    print(posterior\_probabilities)

    print("\n")

    print(f"Predicted Probabilities for stroke ({chosen\_column}='{chosen\_value}'):")

    print(predicted\_probabilities)

    print("\n")

    print(f"MAP Estimate for stroke: {map\_estimate}")

    print("\n")

    print(f"Sorted Predictions:")

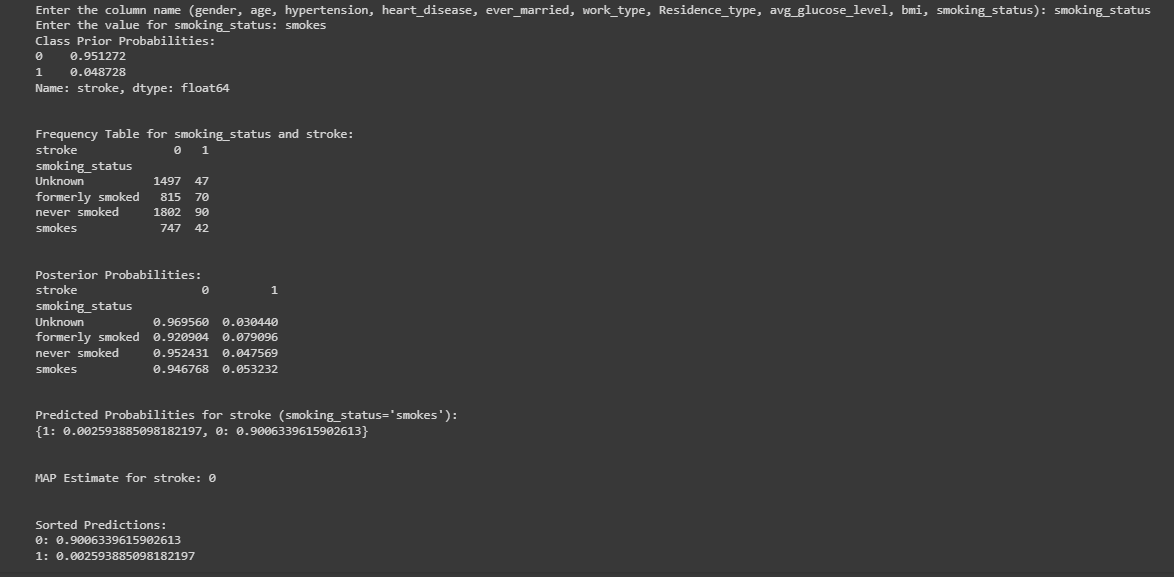
    for k, v in sorted\_predictions.items():

        print(f"{k}: {v}")

else:

    print(f"The value '{chosen\_value}' does not exist in the '{chosen\_column}' column.")

**Output:**



**XGBOOST**

#importing the XGBoost Classifier module

from xgboost  import XGBClassifier

# Create the classifier object

xgb = XGBClassifier()

# Training the classifier

xgb.fit(X\_train,y\_train)

#predicting result using the test dataset

y\_pred\_xgb = xgb.predict(X\_test)

y\_pred\_prob\_xgb = xgb.predict\_proba(X\_test)[:, 1]

# Printing the accuracy and roc-auc score of the model

print('Accuracy:', accuracy\_score(y\_test, y\_pred\_xgb))

print('ROC AUC Score:', roc\_auc\_score(y\_test, y\_pred\_prob\_xgb))

# plots of roc\_auc

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob\_xgb)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2, color= 'teal')

plt.plot([0,1], [0,1], 'r--' )

plt.title('ROC Curve of XGBOOST')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

**XGBOOST:**

from xgboost  import XGBClassifier

from sklearn.metrics import roc\_auc\_score,roc\_curve,accuracy\_score

# Create the classifier object

xgb = XGBClassifier()

# Training the classifier

xgb.fit(X\_train,y\_train)

#predicting result using the test dataset

y\_pred\_xgb = xgb.predict(X\_test)

y\_pred\_prob\_xgb = xgb.predict\_proba(X\_test)[:, 1]

# Printing the accuracy and roc-auc score of the model

print('Accuracy:', accuracy\_score(y\_test, y\_pred\_xgb))

print('ROC AUC Score:', roc\_auc\_score(y\_test, y\_pred\_prob\_xgb))

# plots of roc\_auc

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob\_xgb)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2, color= 'teal')

plt.plot([0,1], [0,1], 'r--' )

plt.title('ROC Curve of XGBOOST')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# from sklearn.metrics import plot\_confusion\_matrix,precision\_recall\_fscore\_support

# plot\_confusion\_matrix(xgb,X\_test,y\_test)

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score

print("Accuracy\_score:",accuracy\_score(y\_test,y\_pred\_xgb))

print("Precision\_score:",precision\_score(y\_test,y\_pred\_xgb))

print("Recall\_score:",recall\_score(y\_test,y\_pred\_xgb))

print("f1\_score:",f1\_score(y\_test,y\_pred\_xgb))

print('ROC AUC Score:', roc\_auc\_score(y\_test, y\_pred\_prob\_xgb))

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

clf\_xgb = Perceptron(tol=1e-3, random\_state=42)

clf\_xgb.fit(X\_train, y\_train)

predictions = clf\_xgb.predict(X\_test)

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(xgb, X\_train, y\_train, cv=10, scoring='neg\_mean\_squared\_error')

train\_scores\_mean = -train\_scores.mean(axis = 1)

test\_scores\_mean = -test\_scores.mean(axis = 1)

plt.figure()

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r", label="Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g", label="Cross-validation score")

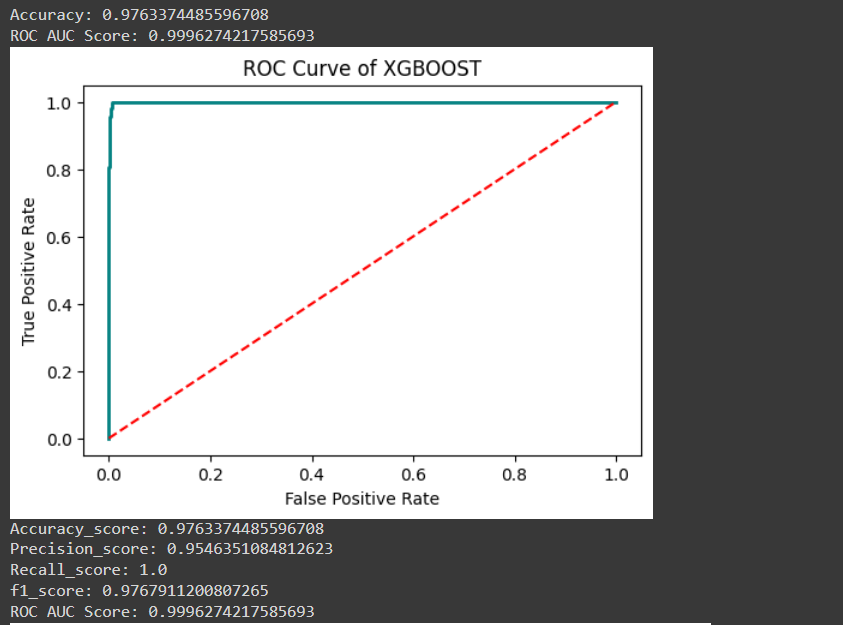
plt.xlabel("Training examples")

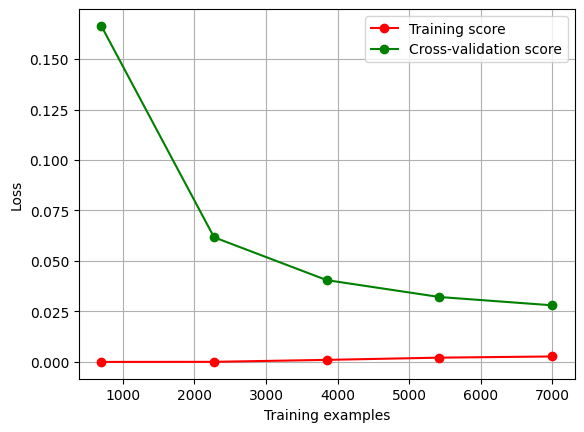
plt.ylabel("Loss")

plt.legend(loc="best")

plt.grid()

plt.show()





SVM Model:

Approach 1 : 1 hot encoding dataset

from sklearn import svm

clf\_svm = svm.SVC(kernel='rbf')  # Linear Kernel

clf\_svm.fit(X\_train, y\_train)

# Predict the response for the test dataset

y\_pred = clf\_svm.predict(X\_test)

# Model Accuracy: how often is the classifier correct?

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

# Model Precision: what percentage of positive tuples are labeled as such?

print("Precision:", precision\_score(y\_test, y\_pred))

# Model Recall: what percentage of positive tuples are labelled as such?

print("Recall:", recall\_score(y\_test, y\_pred))

# Model F1 Score: the weighted average of Precision and Recall

print("F1 Score:", f1\_score(y\_test, y\_pred))

print('ROC AUC Score:', roc\_auc\_score(y\_test, y\_pred))

# plots of roc\_auc

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2, color= 'teal')

plt.plot([0,1], [0,1], 'r--' )

plt.title('ROC Curve of SVM -rbf')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

print("\n")

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(clf\_svm, X\_train, y\_train, cv=10, scoring='neg\_mean\_squared\_error')

train\_scores\_mean = -train\_scores.mean(axis = 1)

test\_scores\_mean = -test\_scores.mean(axis = 1)

plt.figure()

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r", label="Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g", label="Cross-validation score")

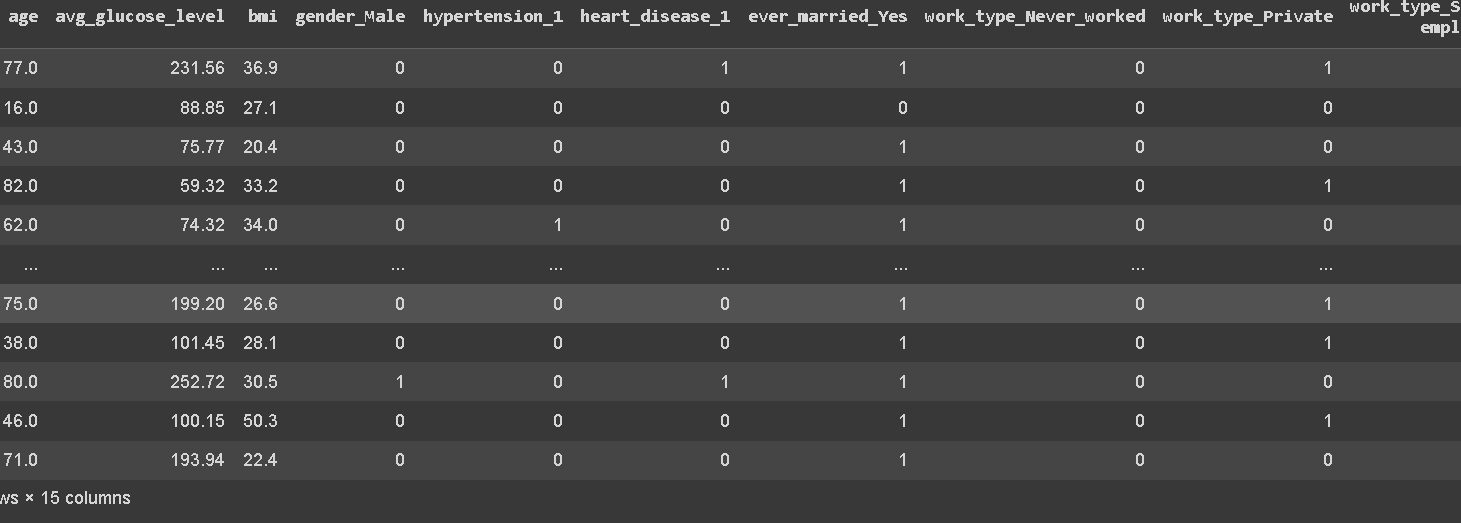
plt.xlabel("Training examples")

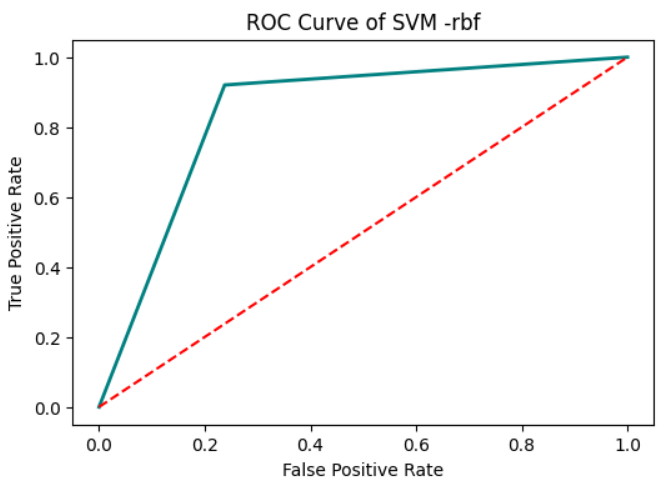
plt.ylabel("Loss")

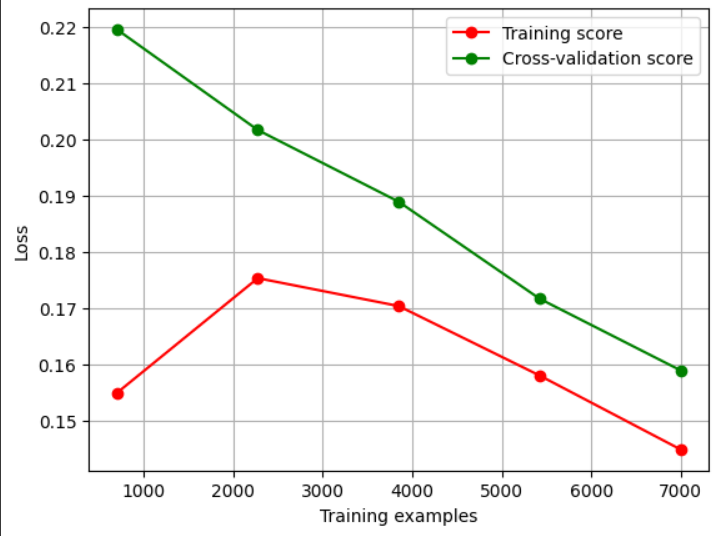
plt.legend(loc="best")

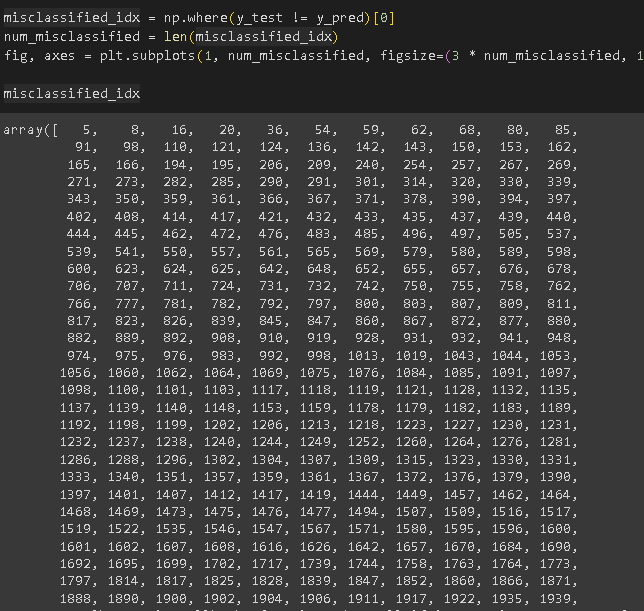
plt.grid()

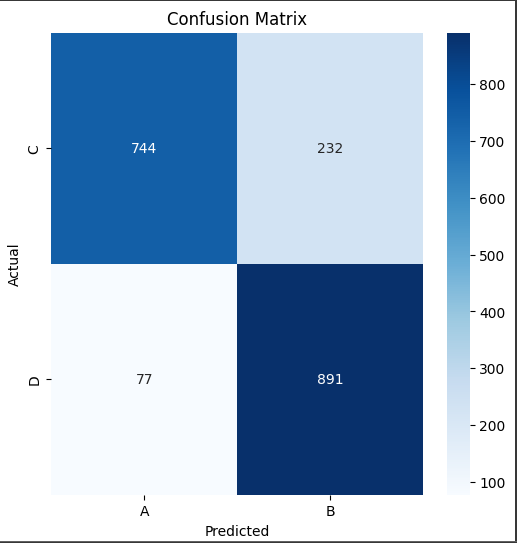
plt.show()

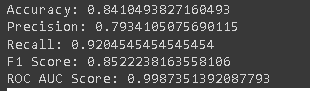












Approach 2 : Label encoder dataset

## SUPPORT VECTOR MACHINES

### DATA PREPROCESSING

#### LIBRARIES USED

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import metrics

#### DATA COLLECTION

df=pd.read\_csv('healthcare-dataset-stroke-data.csv')

z=df.columns

df

#### DATA CLEANING

df1=df.copy(deep=True)

df1=df1.drop(['id'],axis=1)

y=df1['stroke']

df1=df1.drop(['stroke'],axis=1)

df1.head()

y

print("number of null values in each column")

print(df1.isna().sum())

df1=df1.fillna(df1['bmi'].mean())

print("number of null values in each column")

print(df1.isna().sum())

df1.info()

#### LABEL ENCODING

from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder

label\_encoder = LabelEncoder()

# Assuming 'col1' and 'col2' are the columns you want to encode

df1['gender'] = label\_encoder.fit\_transform(df1['gender'])

df1['ever\_married'] = label\_encoder.fit\_transform(df1['ever\_married'])

df1['Residence\_type'] = label\_encoder.fit\_transform(df1['Residence\_type'])

df1['smoking\_status'] = label\_encoder.fit\_transform(df1['smoking\_status'])

df1['work\_type'] = label\_encoder.fit\_transform(df1['work\_type'])

df1.head()

# Since our Dataset is highly undersampled (based on target instances) we are going to perform a over sampling method to have equal representation of both the target classes

# Using random oversampling - importing the library

from imblearn.over\_sampling import RandomOverSampler

# Performing a minority oversampling

oversample = RandomOverSampler(sampling\_strategy='minority')

X=df1

# Obtaining the oversampled dataframes - testing and training

X ,y = oversample.fit\_resample(X, y)

no standard scaling because if we want to scale means it would be difficult to give custom inputs

### DATA VISUALISATION //NEEYE PANNIKO MENDOKUSAI DA

### DATA TRAIN TEST SPLIT

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=62)

print("Training data shape:", X\_train.shape, y\_train.shape)

print("Testing data shape:", X\_test.shape, y\_test.shape)

### MODEL TRANING

from sklearn import svm

from sklearn.metrics import accuracy\_score,roc\_auc\_score,roc\_curve

#### LINEAR

# Linear SVM

linear\_svm = svm.SVC(kernel='linear')

linear\_svm.fit(X\_train, y\_train)

linear\_predictions = linear\_svm.predict(X\_test)

linear\_accuracy = accuracy\_score(y\_test, linear\_predictions)

print("Linear SVM Accuracy:", linear\_accuracy)

#### POLYNOMIAL

# Polynomial SVM

polynomial\_svm = svm.SVC(kernel='poly', degree=3)

polynomial\_svm.fit(X\_train, y\_train)

polynomial\_predictions = polynomial\_svm.predict(X\_test)

polynomial\_accuracy = accuracy\_score(y\_test, polynomial\_predictions)

print("Polynomial SVM Accuracy:", polynomial\_accuracy)

#### Radial Basis Function

# RBF SVM

rbf\_svm = svm.SVC(kernel='rbf')

rbf\_svm.fit(X\_train, y\_train)

rbf\_predictions = rbf\_svm.predict(X\_test)

rbf\_accuracy = accuracy\_score(y\_test, rbf\_predictions)

print("RBF SVM Accuracy:", rbf\_accuracy)

#### Sigmoid SVM:

# Sigmoid SVM

sigmoid\_svm = svm.SVC(kernel='sigmoid', gamma='auto')

sigmoid\_svm.fit(X\_train, y\_train)

sigmoid\_predictions = sigmoid\_svm.predict(X\_test)

sigmoid\_accuracy = accuracy\_score(y\_test, sigmoid\_predictions)

print("Sigmoid SVM Accuracy:", sigmoid\_accuracy)

#### Custom Kernels need not to be explored in the following proj presentation because linear itself is sufficient for classifiying all these

# li=df1.columns

# columns1=[]

# for i in range(len(li)-1):

# t=int(input(f"enter {li[i]} "))

# columns1.append(t)

# n=np.array(columns1)

# y\_testcase = linear\_svm.predict([n])

# y\_testcase

#### POST TRAINING ANALYSIS

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

cm = confusion\_matrix(y\_test, linear\_predictions)

plt.figure(figsize=(6, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["A "," B"], yticklabels=["C","D"])

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix')

plt.show()

### HYPERPARAMETER FINETUNING

# from sklearn.model\_selection import GridSearchCV

# # Define a grid of hyperparameter values to try

# param\_grid = {

# 'C': [0.1, 1, 10, 100],

# 'tol': [1e-3, 1e-4, 1e-5],

# }

# # Create a grid search object

# grid\_search = GridSearchCV(linear\_svm, param\_grid, cv=5)

# # Fit the grid search object to the training data

# grid\_search.fit(X\_train, y\_train)

# # Get the best hyperparameters

# best\_params = grid\_search.best\_params\_

# # Train the linear SVM model with the best hyperparameters

# best\_linear\_svm = svm.SVC(kernel='linear', \*\*best\_params)

# best\_linear\_svm.fit(X\_train, y\_train)

#### SAVING THE MODEL IN A FILE

import joblib

# Save the SVM model to a file

joblib.dump(linear\_svm, 'svm\_model.joblib')

#### LOADING IT IN A FILE

import joblib

# Load the SVM model from a file

linear\_svm = joblib.load('svm\_model.joblib')

from sklearn.feature\_selection import RFE

# Create an RFE object

rfe = RFE(estimator=linear\_svm, n\_features\_to\_select=10)

# Fit the RFE object to the training data

rfe.fit(X\_train, y\_train)

# feature importance scores

feature\_importance\_scores = rfe.support\_

#### FEATURE IMPORTANCE EXTRACTION

from sklearn.inspection import permutation\_importance

# Calculate the permutation importance scores

permutation\_importance\_scores = permutation\_importance(linear\_svm, X\_train, y\_train)

permutation\_importance\_scores

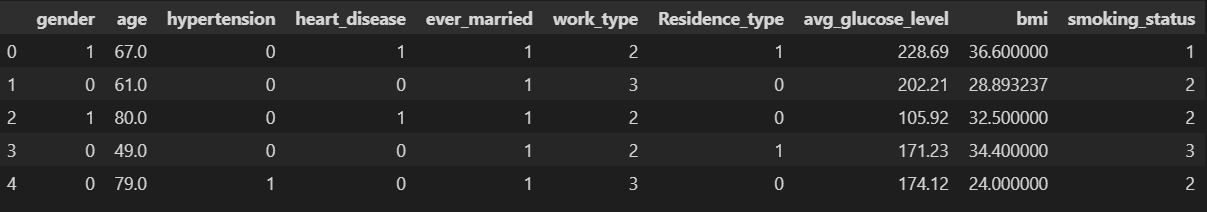
#### POST CLASSIFICATION MISCLASSIFIED POINTS

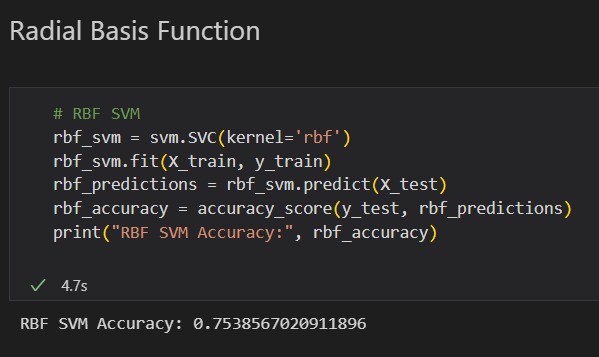
misclassified\_idx = np.where(y\_test != linear\_predictions)[0]

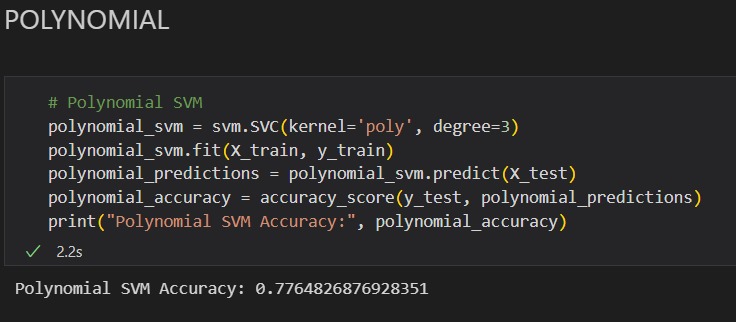
num\_misclassified = len(misclassified\_idx)

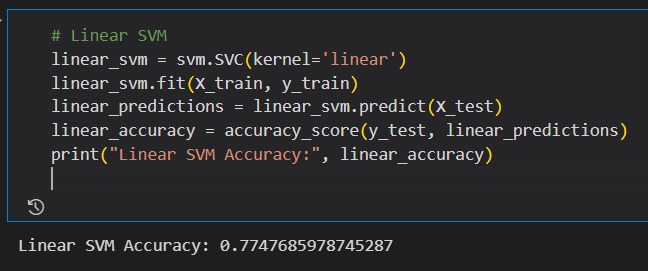
fig, axes = plt.subplots(1, num\_misclassified, figsize=(3 \* num\_misclassified, 12))

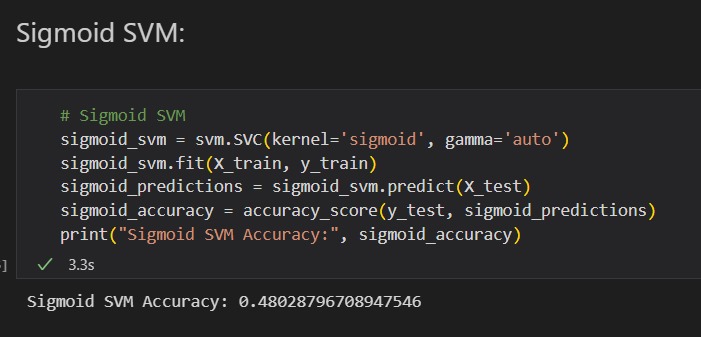
misclassified\_idx











**XGBClassifier from xgboost**

from xgboost  import XGBClassifier

from sklearn.metrics import roc\_auc\_score,roc\_curve,accuracy\_score

from sklearn.preprocessing import StandardScaler

# Create the classifier object

xgb = XGBClassifier()

# Training the classifier

xgb.fit(X\_train,y\_train)

#predicting result using the test dataset

y\_pred\_xgb = xgb.predict(X\_test)

y\_pred\_prob\_xgb = xgb.predict\_proba(X\_test)[:, 1]

# Printing the accuracy and roc-auc score of the model

print('Accuracy:', accuracy\_score(y\_test, y\_pred\_xgb))

print('ROC AUC Score:', roc\_auc\_score(y\_test, y\_pred\_prob\_xgb))

# plots of roc\_auc

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob\_xgb)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2, color= 'teal')

plt.plot([0,1], [0,1], 'r--' )

plt.title('ROC Curve of XGBOOST')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# from sklearn.metrics import plot\_confusion\_matrix,precision\_recall\_fscore\_support

# plot\_confusion\_matrix(xgb,X\_test,y\_test)

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score

print("Accuracy\_score:",accuracy\_score(y\_test,y\_pred\_xgb))

print("Precision\_score:",precision\_score(y\_test,y\_pred\_xgb))

print("Recall\_score:",recall\_score(y\_test,y\_pred\_xgb))

print("f1\_score:",f1\_score(y\_test,y\_pred\_xgb))

print('ROC AUC Score:', roc\_auc\_score(y\_test, y\_pred\_prob\_xgb))

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# clf\_xgb = xgb(tol=1e-3, random\_state=42)

xgb.fit(X\_train, y\_train)

predictions = xgb.predict(X\_test)

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(xgb, X\_train, y\_train, cv=10, scoring='neg\_mean\_squared\_error')

train\_scores\_mean = -train\_scores.mean(axis = 1)

test\_scores\_mean = -test\_scores.mean(axis = 1)

plt.figure()

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r", label="Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g", label="Cross-validation score")

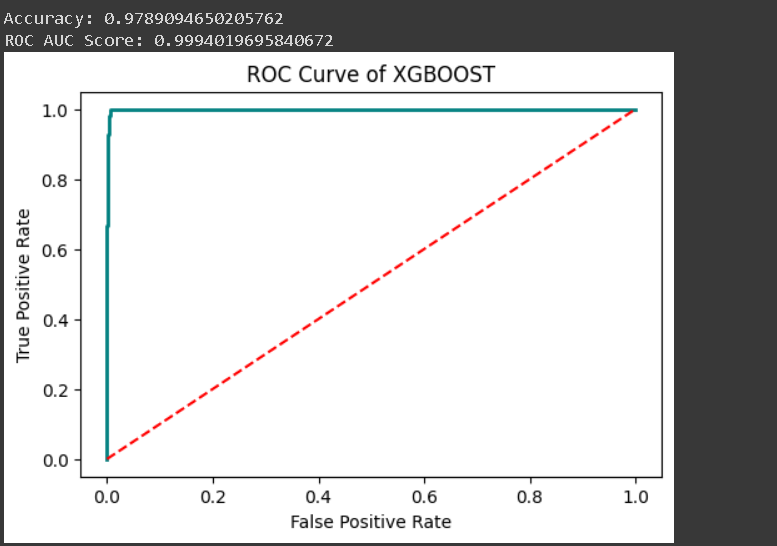
plt.xlabel("Training examples")

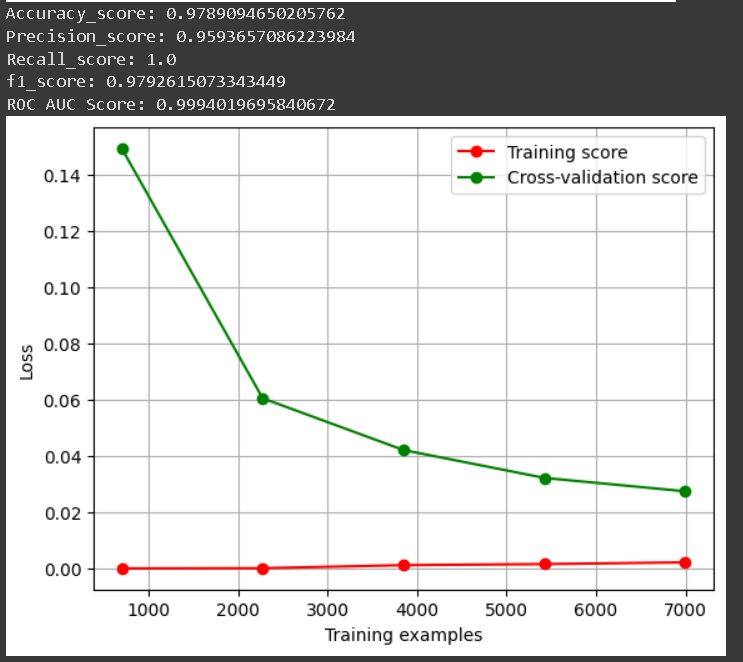
plt.ylabel("Loss")

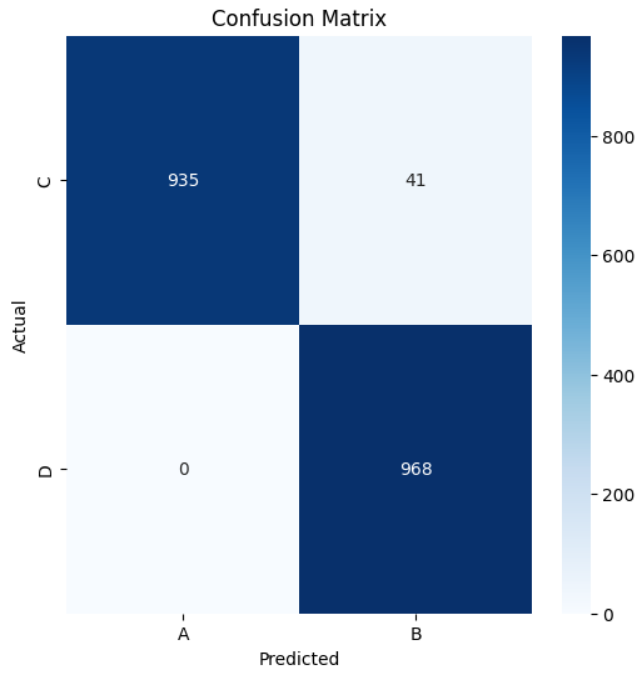
plt.legend(loc="best")

plt.grid()

plt.show()

****

****



**DBSCAN**

from sklearn.cluster import DBSCAN

from sklearn.preprocessing import RobustScaler

rb=RobustScaler()

from sklearn.metrics import adjusted\_mutual\_info\_score

from sklearn.metrics import normalized\_mutual\_info\_score

x1=rb.fit\_transform(X\_train)

x2=rb.fit\_transform(X\_test)

eps = [1.01,1.02,1.05,1.1,1.15,1.2,1.25,1.3,1.35]

min\_samples = [7,8,9,10,11,12,13,14,15]

ks=list(range(1,10))

inds = [f"eps={e}" for e in eps]

cols = [f"min\_samples={m}" for m in min\_samples]

dbdata = pd.DataFrame(np.zeros((9,9)),columns=cols,index=inds)

for i in range(len(eps)):

    for j in range(len(min\_samples)):

        dbscan = DBSCAN(eps=eps[i], min\_samples=min\_samples[j])

        dbscan.fit(x1)

        dbdata.iloc[i,j] = np.unique(dbscan.labels\_).size

plt.figure(figsize=(20,9))

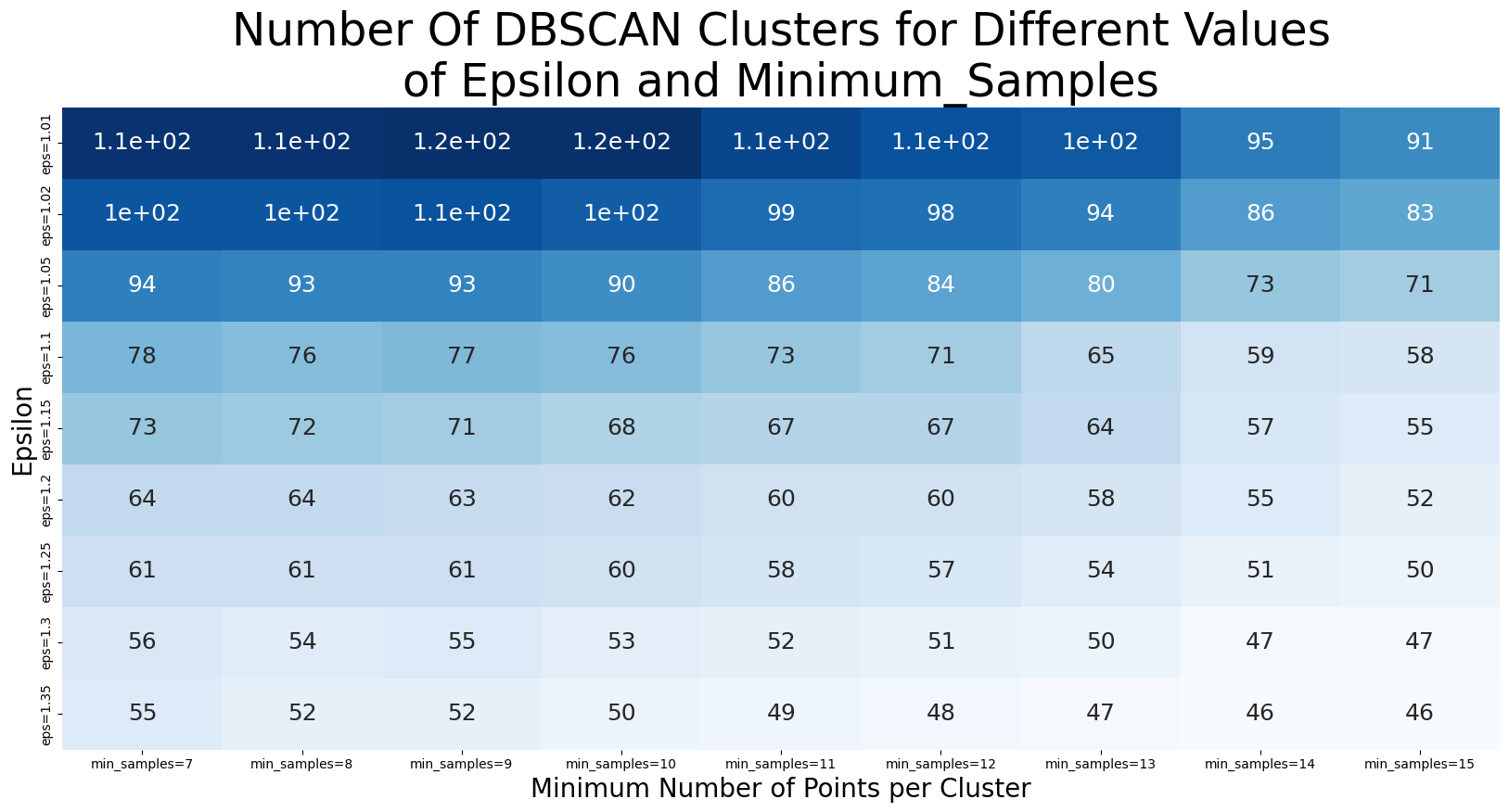
sns.heatmap(dbdata, cmap='Blues', annot=True, annot\_kws={'fontsize':18},cbar=False)

plt.title("Number Of DBSCAN Clusters for Different Values\nof Epsilon and Minimum\_Samples", fontsize=35)

plt.xlabel("Minimum Number of Points per Cluster",fontsize=20)

plt.ylabel("Epsilon",fontsize=20)

plt.show()

****

inds = [f"  eps={e}" for e in eps]

cols = [f"min\_samples={m}" for m in min\_samples]

ami = pd.DataFrame(np.zeros((9,9)),columns=cols,index=inds)

nmi = pd.DataFrame(np.zeros((9,9)),columns=cols,index=inds)

for i in range(len(eps)):

    for j in range(len(min\_samples)):

        dbscan = DBSCAN(eps=eps[i], min\_samples=min\_samples[j])

        labels=dbscan.fit\_predict(x1)

        nmi.iloc[i,j] = normalized\_mutual\_info\_score(labels,y\_train)

        ami.iloc[i,j] = adjusted\_mutual\_info\_score(labels,y\_train)

plt.figure(figsize=(20,10))

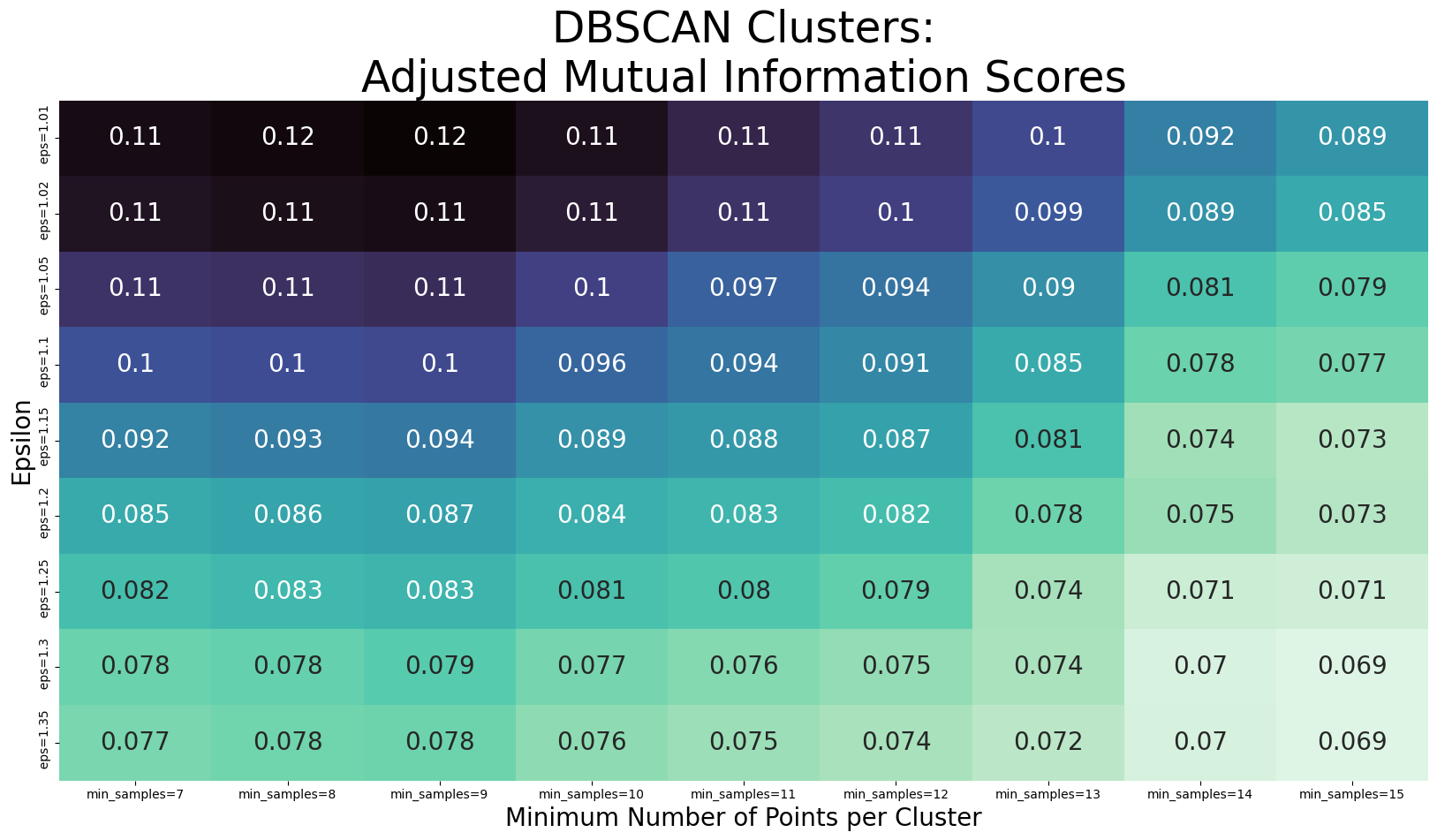
sns.heatmap(ami, cmap='mako\_r', annot=True, annot\_kws={'fontsize':20},cbar=False)

plt.title("DBSCAN Clusters:\nAdjusted Mutual Information Scores", fontsize=35)

plt.xlabel("Minimum Number of Points per Cluster",fontsize=20)

plt.ylabel("Epsilon",fontsize=20)

plt.show()

****

plt.figure(figsize=(20,10))

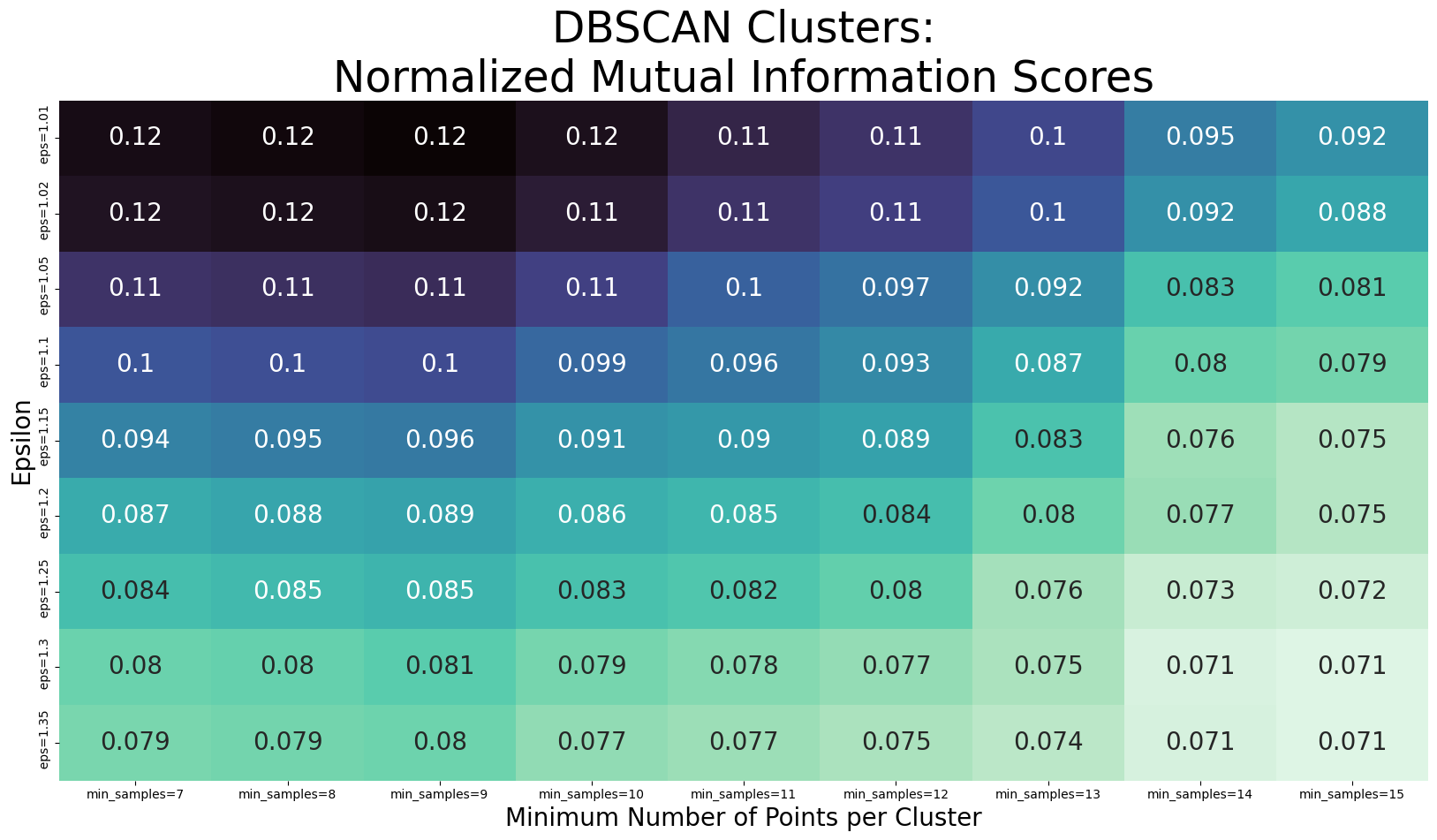
sns.heatmap(nmi, cmap='mako\_r', annot=True, annot\_kws={'fontsize':20},cbar=False)

plt.title("DBSCAN Clusters:\nNormalized Mutual Information Scores", fontsize=35)

plt.xlabel("Minimum Number of Points per Cluster",fontsize=20)

plt.ylabel("Epsilon",fontsize=20)

plt.show()

****

fig, axes = plt.subplots(nrows=len(eps), ncols=len(min\_samples),figsize=(20,15))

fig.suptitle("DBSCAN Cluster Sizes", fontsize=40)

for i in range(len(eps)):

    for j in range(len(min\_samples)):

        dbscan = DBSCAN(eps=eps[i], min\_samples=min\_samples[j])

        labels=dbscan.fit\_predict(x1)

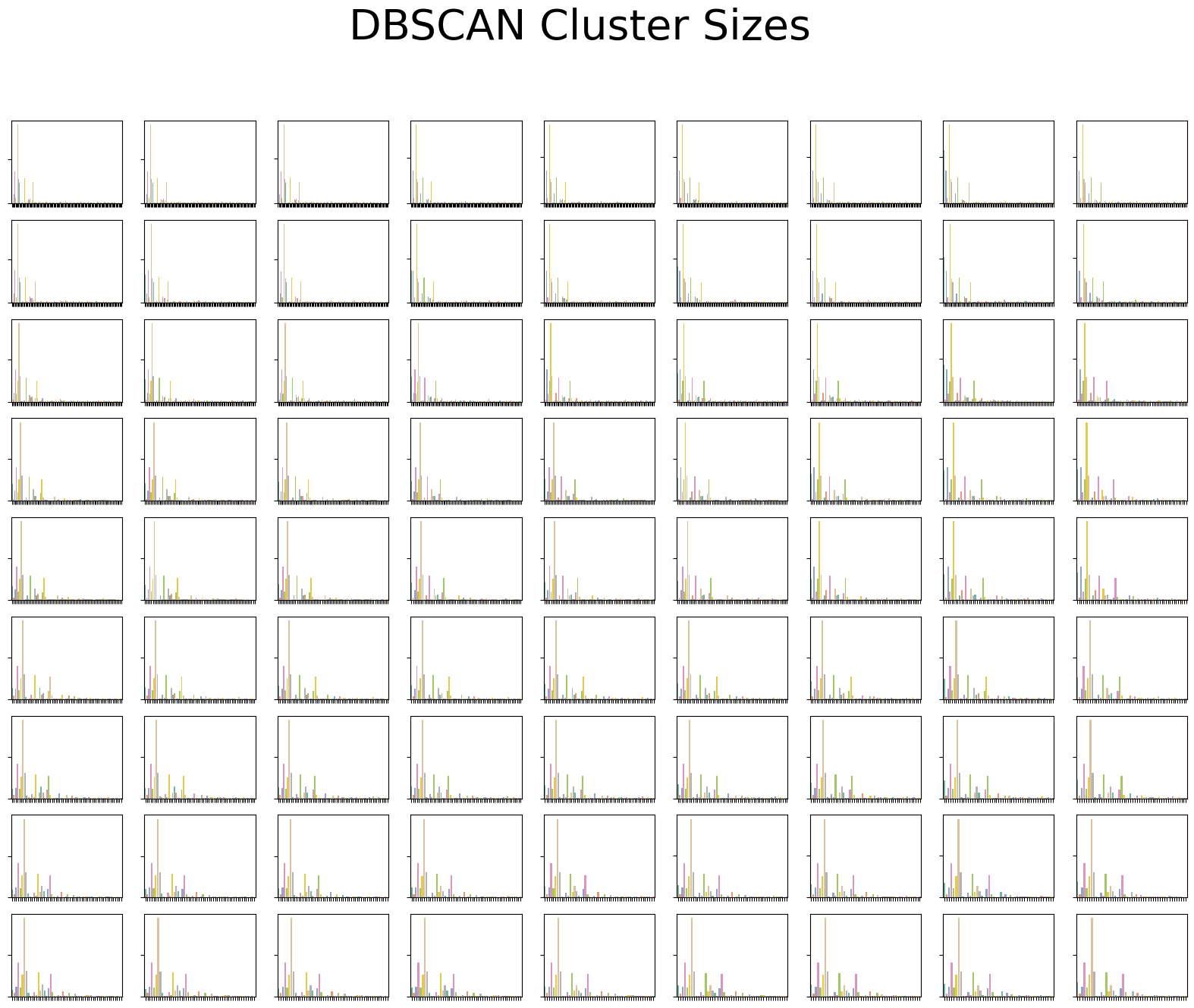
        sns.countplot(x=labels, palette='Set2',ax=axes[i,j])

        axes[i,j].set\_xticklabels([])

        axes[i,j].set\_ylabel(None)

        axes[i,j].set\_yticklabels([])

plt.show()

****

from sklearn.metrics import silhouette\_score

silhouette\_avg = silhouette\_score(x1, labels)

print(f"Silhouette Score: {silhouette\_avg}")



Verifying dbscan cluster with pca standard scaled datapoints

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_train)

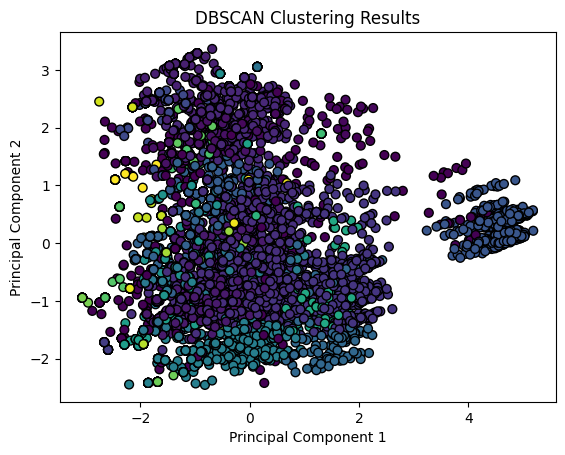
plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=labels, cmap='viridis', edgecolor='k', s=40)

plt.title("DBSCAN Clustering Results")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.show()



from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(x1)

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=labels, cmap='viridis', edgecolor='k', s=40)

plt.title("DBSCAN Clustering Results")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.show()

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(x1)

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=labels, cmap='viridis', edgecolor='k', s=40)

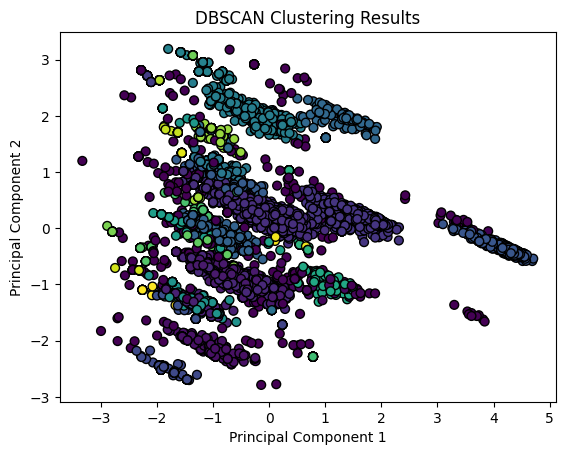
plt.title("DBSCAN Clustering Results")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.show()

Verifying dbscan cluster with pca robust scaled datapoints



Results:

DBSCAN couldn't find any densely-packed clusters. It tends to lump most of the data points into one (or a few) big cluster, and several tiny clusters.

I can't use cluster labels as a variable for the following reasons:

The tiny clusters would cause the predictive model to overfit.

Most points would be in the bigger lump (I don't think the word "cluster" fits it) which isn't informative as it contains pretty much all points.

Sklearn doesn't have a predict method for the DBSCAN class XD They could make one that assigns points to clusters by looking at nearest neighbours but oh well. One could try coding this from scratch (which might be a little tedious to do) but for now there's no need to, considering the 2 points above.