C:\Users\janani\Desktop\RGU.PNGA close up of a sign

Description automatically generated

**BSc (Hons) Artificial Intelligence and Data Science**

**Module: CM2601 Machine Learning**

**Machine Learning Report**

**Module Leader: Mr. Sahan Priyanaya**

**RGU Student ID : 2330923**

**IIT Student ID : 20230976**

**Student Name : Abdullah Nazly**

Acknowledgment .

I want to extend my sincere appreciation to all those who contributed and provided steadfast support throughout this coursework. A special acknowledgment goes to Mr. Sahan Priyanaya, the module leader, for his guidance that led to the successful completion of this coursework.

Additionally, I express extra gratitude to Mr. Pushpika Prasad Liyanaarachchi assistant lecturer, for assisting throughout the tutorials and addressing doubts, which facilitated the smooth execution of the coursework. I genuinely appreciate these individuals for their assistance in the successful completion of the coursework.

Executive Summary

This course work aims to build two machine learning models using python. Implementation is focused on understanding the key concepts of data preprocessing and training a model. A bank dataset is used to train the model here. Two models were used. One is a neural network model, and the other is a Random Forest model. For both models, the same way of data preprocessing is done. Though both got good predicting accuracy.

Contents

[1. Introduction. 1](#_Toc186298262)

[2. Neural Network Model 1](#_Toc186298263)

[2.1. Data Preprocessing 1](#_Toc186298264)

[2.2. Checking NULL values and duplicates. 2](#_Toc186298265)

[2.3. Data format analysis. 3](#_Toc186298266)

[2.4. Getting the values of the features 4](#_Toc186298267)

[2.5. Checking the Unique values for each column 5](#_Toc186298268)

[2.6. Analyzing feature contact 5](#_Toc186298269)

[2.7. One hot encoding contact feature. 6](#_Toc186298270)

[2.8. Analyzing poutcome feature. 6](#_Toc186298271)

[2.9. Label Encoding poutcome. 7](#_Toc186298272)

[2.10. Encoding Y variable and analyzing numerical columns. 7](#_Toc186298273)

[2.11. Removing unnecessary columns. 9](#_Toc186298274)

[2.12. Plotting on boxplot to identify extreme values and outliers 10](#_Toc186298275)

[2.10.1. Plotting duration column. 10](#_Toc186298276)

[2.10.2. Plotting age column. 11](#_Toc186298277)

[2.10.3. Plotting balance column. 11](#_Toc186298278)

[2.12.4. Plotting previous column. 12](#_Toc186298279)

[2.12.5. Analysing pdays. 13](#_Toc186298280)

[2.13. Analyzing other features. 13](#_Toc186298281)

[2.14. Performing Label Encoding for education feature. 15](#_Toc186298282)

[2.15. Encoding Job feature. 15](#_Toc186298283)

[2.16. Encoding marital feature. 16](#_Toc186298284)

[2.17. Training the model. 17](#_Toc186298285)

[2.18. Model Evaluation. 19](#_Toc186298286)

[2.19. Adding class weight to balance the class. 21](#_Toc186298287)

[2.20. Hyperparameter tuning for the Model. 22](#_Toc186298288)

[2.21. Manually adjusting parameters. 25](#_Toc186298289)

[2.22. Handling class imbalance. 26](#_Toc186298290)

[2.23. ROC curve. 26](#_Toc186298291)

[2.24. Conclusion for Neural Network. 27](#_Toc186298292)

[3. Random Forest. 28](#_Toc186298293)

[3.1. Data Preprocessing. 28](#_Toc186298294)

[3.2. Model training. 28](#_Toc186298295)

[3.3. Model Evaluation. 29](#_Toc186298296)

[3.4. Hyperparameter tuning for RF. 30](#_Toc186298297)

[3.5. Manually adjusting the parameters. 33](#_Toc186298298)

[4. Git hub. 36](#_Toc186298299)

[5. Appendix. 36](#_Toc186298300)

[5.1. Source codes for Neural Network 36](#_Toc186298301)

[5.2. Source code for RandomForest. 52](#_Toc186298302)

[6. References. 63](#_Toc186298303)

Table of figures

[Figure 1. head 2](#_Toc186292811)

[Figure 2. head incorrect format 2](#_Toc186292812)

[Figure 3. null value 2](#_Toc186292813)

[Figure 4. data type 3](#_Toc186292814)

[Figure 5. statistics 3](#_Toc186292815)

[Figure 6. numerics 4](#_Toc186292816)

[Figure 7. categoricals 4](#_Toc186292817)

[Figure 8. Boolean 4](#_Toc186292818)

[Figure 9. p value contact 6](#_Toc186292819)

[Figure 10. p value poutcome 6](#_Toc186292820)

[Figure 11. correlation for numerics 8](#_Toc186292821)

[Figure 12. heatmap numerics 8](#_Toc186292822)

[Figure 13. bar plot camapign 9](#_Toc186292823)

[Figure 14. box plot duration 10](#_Toc186292824)

[Figure 15. box plot age 11](#_Toc186292825)

[Figure 16. box plot balance 11](#_Toc186292826)

[Figure 17. box plot previous 12](#_Toc186292827)

[Figure 18. boxplot pdays 13](#_Toc186292828)

[Figure 19.heat map for binary 14](#_Toc186292829)

[Figure 20. bar plot job 15](#_Toc186292830)

[Figure 21. p value marital 16](#_Toc186292831)

[Figure 22. neural network raw 18](#_Toc186292832)

[Figure 23. neural network standardize 19](#_Toc186292833)

[Figure 24. report raw 20](#_Toc186292834)

[Figure 25. report standardize 20](#_Toc186292835)

[Figure 26. accuracy first model 20](#_Toc186292836)

[Figure 27. class weight model 21](#_Toc186292837)

[Figure 28. report for weigted model 22](#_Toc186292838)

[Figure 29. parameter tuned model 24](#_Toc186292839)

[Figure 30. report best param 24](#_Toc186292840)

[Figure 31. manual adjustedd result 25](#_Toc186292841)

[Figure 32. manual adjusted report 26](#_Toc186292842)

[Figure 33. ROC for neural network 27](#_Toc186292843)

[Figure 34. report random forest 29](#_Toc186292844)

[Figure 35. best rf model report 32](#_Toc186292845)

[Figure 36. best rf cross validation 33](#_Toc186292846)

[Figure 37. manually adjusted report 34](#_Toc186292847)

[Figure 38. manually adjusted cross validation 34](#_Toc186292848)

[Figure 39. ROC random forest 35](#_Toc186292849)

# 

# Introduction.

The Machine Learning course work is processed on a simple approach. Starting with data preprocessing and going up to training the model. First the data is processed accordingly to train the neural network model. First the null values checked, the duplicates checked, and problems were solved. Then unique values were checked to make decisions on encoding the categorical variables. Outliers and extreme values were checked for numerical values and no problems were found. Did EDA (Exploitary Data Analysis) for feature selection. For some features we must consider are influencing the target variable or not. No standardizations were made to test whether this processed data is enough to train the model. As the test the model showed a great performance to both models. The conclusion was made to keep the data as it is without further standardization and improvements. Finally, the models went through a hyper parameter tuning and evaluated.

# Neural Network Model

## Data Preprocessing

Bank detail dataset is used here to train the model. As always the dataset has to be preprocessed. The process starts with importing the library to manipulate dataset which is pandas. This dataset is a little different in csv format. Usually, it will be comma separated file. But in this dataset values are separated by semicolons.

"age";"job";"marital";"education";"default";"balance";"housing";"loan";**"contact";**"day";"month";"duration";"campaign";"pdays";"previous";"poutcome";"y"

58;"management";"married";"tertiary";"no";2143;"yes";"no";**"unknown";**5;"may";261;1;-1;0;"unknown";"no"

44;"technician";"single";"secondary";"no";29;"yes";"no";**"unknown";**5;"may";151;1;-1;0;"unknown";"no"

33;"entrepreneur";"married";"secondary";"no";2;"yes";"yes";**"unknown";**5;"may";76;1;-1;0;"unknown";"no"

47;"blue-collar";"married";"unknown";"no";1506;"yes";"no";**"unknown";**5;"may";92;1;-1;0;"unknown";"no"

So as a result. When we try to read it with normal csv format, it won’t be able to read properly. Therefor another parameter is passed to divide the semicolons and read the csv file properly

df = pd.read\_csv('./Dataset/bank-full.csv', delimiter=';')

df.head(20)

Then we get the head output as usual.

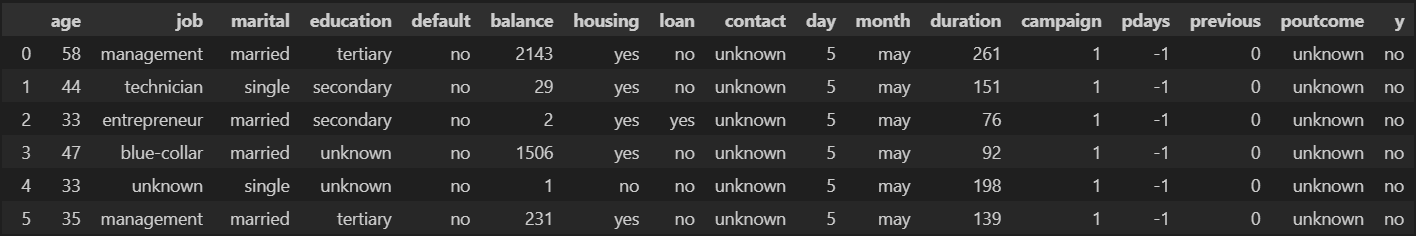


Figure 1. head

If we didn’t use the delimiter parameter, the output would be like this.

A screenshot of a computer

Description automatically generated

Figure 2. head incorrect format

## Checking NULL values and duplicates.

The next step is to check whether there’s any null values or duplicate values in the dataset. This is an important step to be considered before training the model. Checking the null values is performed using the below code which it gets the null values for every column in the dataset.

print(df.isnull().sum())

The output I got gave me the result that it doesn’t have any null value.

A black screen with white text

Description automatically generated

Figure 3. null value

Next identifying the duplicate value is performed in the code. In this code also. The duplication is checked for all the columns at once by using this code.

print(df.duplicated().sum())

The result was zero, which means there’s no duplicate value either. As a result, the part of data preprocessing becomes easier. We don’t have to handle missing values or duplicate values. We can directly go for the other areas of data preprocessing.

## Data format analysis.

The usual format of checking for format is performed here, with .info() code the Dtypes including null counts are displayed.

df.info()



Figure 4. data type

Then we get a statistic of the features to analyze the mean values, min-max values and other values as well.

df.describe()

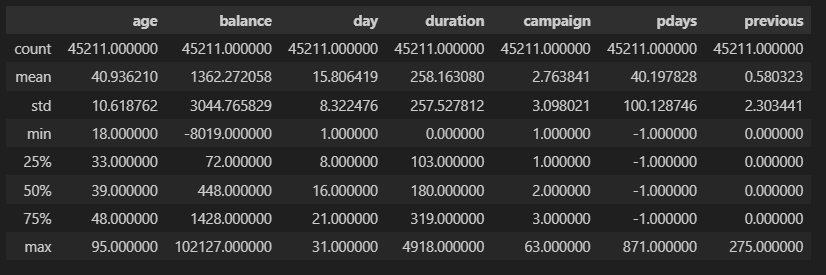


Figure 5. statistics

By this we can understand that They are acceptable values. Theres no extreme values in these numerical features.

## Getting the values of the features

First the code below is run to get all the numerical features which don’t need data transformation currently.

numeric\_columns = df.select\_dtypes(include=['int64', 'float64']).columns

print("Numeric Columns:", numeric\_columns)



Figure 6. numerics

and the output given out put had all the numerical values columns. Next to get all the categorical valued features, the code below is run.

categorical\_columns = df.select\_dtypes(include=['object']).columns

print("Categorical Columns:", categorical\_columns)

The output gave all the columns which has categorical values.

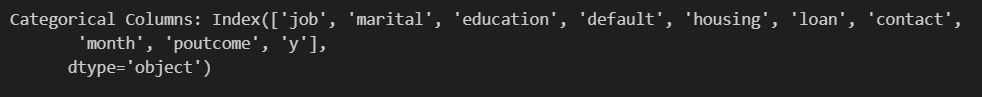


Figure 7. categoricals

Finally, checking whether if any Boolean valued features are available using the below code  
  
boolean\_columns = df.select\_dtypes(include=['bool']).columns

print("Boolean Columns:", boolean\_columns)

but there weren’t any Boolean valued features available.

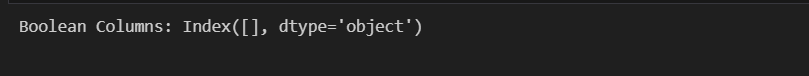


Figure 8. Boolean

## Checking the Unique values for each column

The unique values of each column are printed to make decision on encoding. Only categorical values should be considered under encoding, so categorical valued features are chosen here.

df['job'].unique()

df['marital'].unique()

df['education'].unique()

df['default'].unique()

df['housing'].unique()

df['contact'].unique()

df['month'].unique()

df['poutcome'].unique()

df['y'].unique()

df['campaign'].unique()

Codes were ran multiple times to identify the unique values of each columns to take the decision for encoding.

## Analyzing feature contact

When we consider the feature contact it seems like y variable doesn’t actually influenced by it. To make sure some EDA is done to the feature. We used chi-square test to evaluate whether there is a statistical relationship between two categorical variables. It will help to identify whether the feature is potentially important for the predictive model. This is done by the codes below.

from scipy.stats import chi2\_contingency

# If p value is < 0.05, the feature is influencing the targeted variable

crosstab = pd.crosstab(df['contact'], df['y'])

chi2, p, dof, expected = chi2\_contingency(crosstab)

print("Chi-square statistic:", chi2)

print("p-value:", p)

scipy library is imported to do statistical tests for the categorical variables. This is the output that was given to the above code.

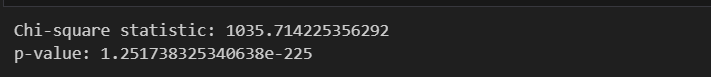


Figure 9. p value contact

As we can see, the p-value is extremely low, and chi-square value is a bit higher. Therefore, the contact feature makes a huge influence in the prediction model.

## One hot encoding contact feature.

As we can see it is a categorical variable, it must be encoded. Since it has no relation between it’s values, one hot encoding is the best option. The code below is used to encode the contact feature.

# Apply One-Hot Encoding to the 'contact' column

df\_encoded = pd.get\_dummies(df['contact'], prefix='contact')

# Convert True/False to 1/0

df\_encoded = df\_encoded.astype(int)

# concatenate the encoded columns with the original dataframe

df = pd.concat([df, df\_encoded], axis=1)

# Drop the original 'contact' column

df.drop('contact', axis=1, inplace=True)

df.head()

the feature is encoded nicely.

## Analyzing poutcome feature.

The same way of chi-square test is used to evaluate the relationship statistics between target variable and poutcome. This resulted in this feature also deeply influences the target variable as the p-value is exactly equal to zero and chi-square value has a higher value.

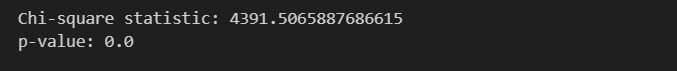


Figure 10. p value poutcome

## Label Encoding poutcome.

This feature has unique values like success, failure, and other values. So, we can identify a relationship between each other. For example, success > failure. So the best option is to label encoding here. The code below is used to label code the poutcome feature.

from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder

label\_encoder = LabelEncoder()

# Apply Label Encoding to the 'poutcome' column

df['poutcome\_encoded'] = label\_encoder.fit\_transform(df['poutcome'])

df.drop('poutcome', axis=1, inplace=True)

# Display the resulting DataFrame

print(df)

## Encoding Y variable and analyzing numerical columns.

In this step we evaluate the relationship between numerical features and the binary target variable y by using Point-Biserial Correlation. By this we can determine the significance of association between these features. First Label encoding is done for the y variable. Then numerical columns are selected. Then the point-biserial correlation is performed for each feature. Then the result is printed. These are the codes which are used to calculate the correlation.

from scipy.stats import pointbiserialr

# Step 1: Convert 'y' to numeric (binary)

df['y'] = df['y'].map({'no': 0, 'yes': 1})

# Step 2: Define numerical features

numerical\_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']

# Step 3: Calculate Point-Biserial Correlation for each feature

correlation\_results = []

for col in numerical\_features:

    corr, p\_value = pointbiserialr(df[col], df['y'])

    correlation\_results.append((col, corr, p\_value))

# Step 4: Print results

print("Feature-wise Point-Biserial Correlation and p-values:")

for feature, corr, p\_value in correlation\_results:

    print(f"Feature: {feature}, Correlation: {corr:.3f}, p-value: {p\_value:.3f}")

The output we got is below.

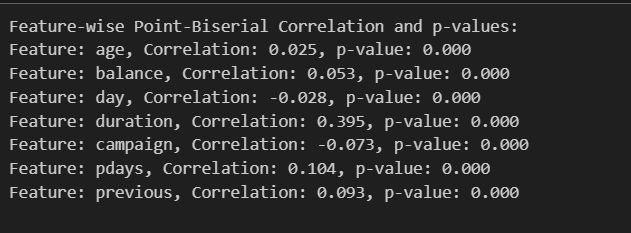


Figure 11. correlation for numerics

Then for a better visualization the output is plotted in a heatmap. In the heatmap the correlation of each feature with y is plotted.

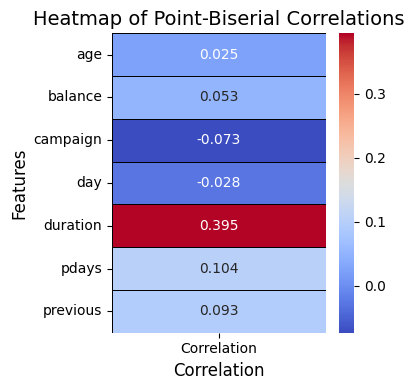


Figure 12. heatmap numerics

As we can see, the day column has a weak negative correlation. So, it might be less meaningful to the model. Age column also has a less correlation value, even though when considering real life scenarios, it may influence prediction. Below output is a Countplot to analyze the relationship between campaign feature and y variable. So, considering the Countplot we assume that there might be a tiny influence on the prediction. Therefore, the campaign feature is kept as it is.

A graph with green and orange bars

Description automatically generated

Figure 13. bar plot camapign

## Removing unnecessary columns.

So according to analysis, the month column also can be removed. Because both day and month have similar characteristics. As the day has less influence, the month is also considered to be removed. Columns were dropped using this code

df = df.drop(columns=['day', 'month'])

Box plotting is performed between numerical values and target values to analyse the extreme values or incorrect values. For every column similar format of this code is performed for plotting.

sns.boxplot(x='y', y='duration', data=df)

## Plotting on boxplot to identify extreme values and outliers

### 2.10.1. Plotting duration column.

The result we got looks like this.

A graph of a graph of a graph

Description automatically generated with medium confidence

Figure 14. box plot duration

As we can see for the duration there’s no negative values which can be incorrect format. So, this feature is good to go.

### 2.10.2. Plotting age column.

A diagram of a graph

Description automatically generated with medium confidence

Figure 15. box plot age

As we can see there are some extreme values in the above plotting. Though these values are acceptable. These can happen in real life scenario as well. So, this column is also good to go.

### 2.10.3. Plotting balance column.

A graph of a graph of a graph

Description automatically generated with medium confidence

Figure 16. box plot balance

As we can see from the plotting that there are some extreme values. But those can rarely happen. There are negative values as well. These could give a false assumption that there cannot be negative values for balance. But we can assume that the client could be under credit rather than having balance. Therefore, there is no need to do cleanup or transformation here.

### Plotting previous column.

A graph of a graph of a person

Description automatically generated with medium confidence

Figure 17. box plot previous

As we can see from the plotting there is one particular data point which is unusually extreme. It’s better to remove that record. It is performed by the code below.

# Identify the record with the extreme value in 'previous'

outlier\_row = df[df['previous'] > 250]

# Display the details of the record

print("Outlier row details:")

print(outlier\_row)

# Drop the specific row

df = df.drop(outlier\_row.index)

# Verify the row is removed

print(f"Updated dataset shape: {df.shape}")

### Analysing pdays.

First the box plotting is done to identify extreme values.

A graph of a graph of a graph

Description automatically generated with medium confidence

Figure 18. boxplot pdays

Theres negative value -1 in this feature which represents the client has not contacted previously. This must be handled. If not, the model may get false information. first buckets are created to put a range for the amount of calls. By this we can take the -1 as not contacted. It is done by the code below.

# Create buckets or categories for 'pdays'

df['pdays\_category'] = pd.cut(

    df['pdays'],

    bins=[-2, 0, 100, 300, 900],

    labels=['Not Contacted', 'Recently Contacted', 'Contacted Long Ago', 'Very Long Ago']

)

Then one hot encoding is done by the code below for each label as it is the most suitable for this feature.

# One-hot encode the categories

df = pd.get\_dummies(df, columns=['pdays\_category'], prefix='pdays\_cat')

# Ensure all boolean-like columns are integers

df = df.astype({col: 'int' for col in df.select\_dtypes(include='bool').columns})

# Display the transformed dataset

print(df)

## Analyzing other features.

Categorical features like default, housing and loan have only yes and no as their values which is binary datatype. So doing label encoding will reduce the dimension of the dataset. To analyse the importance of these features a simple cross-tabulation heatmap is plotted for each feature with y variable. The result looked like this.

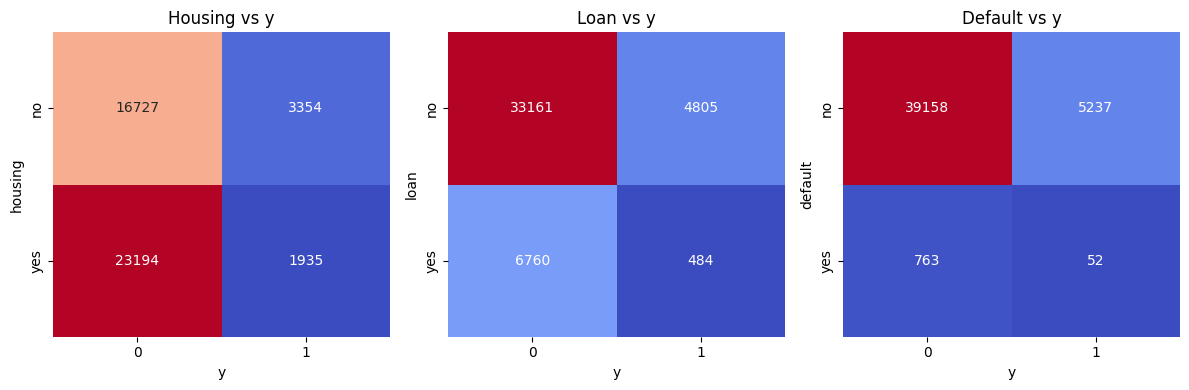


Figure 19.heat map for binary

By this diagram we can understand that each feature has an influence on y variable. When a client has a housing loan there’s more chance that client can subscribe to a term deposit. Likewise, the other two features also have that influence.

Here we didn’t use any libraries. This can be achieved easily as they only have yes and no. The code below is used to encode the columns.

# List of columns to apply Label Encoding to (yes/no columns)

yes\_no\_columns = ['default', 'housing', 'loan']

# Apply Label Encoding to each of the columns in the list

df[yes\_no\_columns] = df[yes\_no\_columns].replace({'yes': 1, 'no': 0})

df.head()

## Performing Label Encoding for education feature.

Education feature’s values have a relationship with each other. For example, primary < secondary < tertiary < unknown. for this kind of relationship, it is better to use Label encoding. The label encoding is performed by the code below.

# Label Encoding

education\_mapping = {'primary': 0, 'secondary': 1, 'tertiary': 2, 'unknown': 3}

df['education\_encoded'] = df['education'].map(education\_mapping)

df = df.drop(columns=['education'])

df.head()

## Encoding Job feature.

Job is the main determining feature for the prediction. It is the source of income. The most suitable encoding type for this feature is one hot encoding as it does not have any relationship between its categorical values. The below plotting shows how the y is distributed for the job feature.

A graph of a number of people

Description automatically generated

Figure 20. bar plot job

The one hot encoding is performed for the job column by the code below.

# Apply One-Hot Encoding to the 'job' column

df\_encoded = pd.get\_dummies(df['job'], prefix='job')

# Convert True/False to 1/0

df\_encoded = df\_encoded.astype(int)

# Optionally, concatenate the encoded columns with the original dataframe

df = pd.concat([df, df\_encoded], axis=1)

# Drop the original 'job' column

df.drop('job', axis=1, inplace=True)

df.head()

## Encoding marital feature.

Next the same chi-square test is done for marital feature. It gave an output like this.



Figure 21. p value marital

We can see that there’s a strong connection between marital and y variable as the p-value is extremely low. As per my opinion, the one hot encoding is the better opinion. Because value might have connection. But we cannot decide which can be greater or lower.

# Apply One-Hot Encoding to the 'marital' column

df\_encoded = pd.get\_dummies(df['marital'], prefix='marital')

# Convert boolean columns to integers (1 for True, 0 for False)

df\_encoded = df\_encoded.astype(int)

# Concatenate the encoded columns with the original dataframe

df = pd.concat([df, df\_encoded], axis=1)

# Drop the original 'marital' column

df.drop('marital', axis=1, inplace=True)

# Display the resulting DataFrame

print(df)

## Training the model.

A simple fully connected neural network is used for training the model. This is a simple feedforward neural network for binary classification from tensorflow. The model is trained two times. One for raw data and one for scaled data. Neural networks work efficiently with scaled data. Both ways of training are evaluated to check which one performs best. First, important libraries were imported. After that the y values is separated for the prediction. Then the dataset is split into a training set and a testing set. The code below is used to perform these actions.

import tensorflow as tf

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

from sklearn.preprocessing import StandardScaler

import pandas as pd

# Features (all columns except 'y')

X = df.drop('y', axis=1).values

y = df['y'].values

# Split the dataset into training and testing sets (80% train, 20% test)

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

After that, the training data of x and test data of x is standardized for one model. Standardscaler is used from the scikit learn. Then the model is built with keras. For now, it only has one hidden layer with Relu function and 10 neurons. The reason for using Relu function is because it is computationally efficient and avoids saturation problems. Then the output layer with sigmoid function as it is for binary classification.

# Standardize the data (standardization)

scaler = StandardScaler()

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

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

# Model architecture (both models will be the same)

def build\_model():

    model = tf.keras.Sequential([

        tf.keras.layers.Dense(10, activation='relu', input\_shape=(X\_train.shape[1],)),  # Hidden layer with 10 neurons

        tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer for binary classification

    ])

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

    return model

Then two more models were trained. One for standardized dataset and other for raw dataset and the model was evaluated and plotted.

# Train the model on raw data (without standardization)

model\_raw = build\_model()

history\_raw = model\_raw.fit(X\_train, y\_train, epochs=70, batch\_size=32, validation\_split=0.2, verbose=1)

# Train the model on standardized data (with standardization)

model\_scaled = build\_model()

history\_scaled = model\_scaled.fit(X\_train\_scaled, y\_train, epochs=70, batch\_size=32, validation\_split=0.2, verbose=1)

Then the plotting codes were coded. The result for training the model with raw data is displayed below.

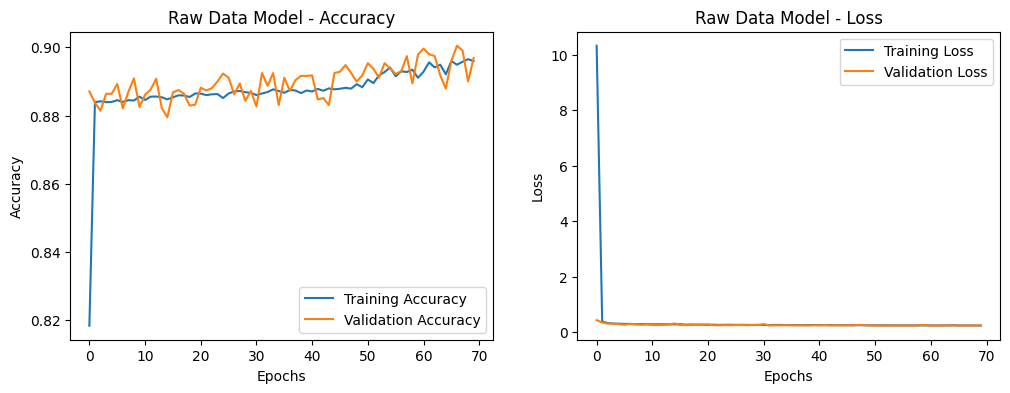


Figure 22. neural network raw

This shows that raw data works pretty well for the model. The model is learning effectively. Though there might be overfitting as it the validation accuracy is rising more than the training accuracy.in the loss, there’s a high initial loss, therefore it has to be considered. Secondly the model which trained with the standardized data is displayed below.

A graph of a model

Description automatically generated with medium confidence

Figure 23. neural network standardize

The standardized data model is also performing better. Both validation and accuracy are stabilizing faster compared to raw data. This indicates better learning. Validation loss is also gradually decreasing which is a good sign. But it’s not possible to decide as it has only one hidden layer. The model might not learn properly.

### Model Evaluation.

Finally, the above model is evaluated with the test accuracy for raw data and test accuracy for standardized data. Scikit learn is used for the evaluation metrices.

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

# Evaluate the model on raw data

test\_loss\_raw, test\_accuracy\_raw = model\_raw.evaluate(X\_test, y\_test)

print(f"Test Accuracy (Raw Data): {test\_accuracy\_raw \* 100:.2f}%")

# Evaluate the model on standardized data

test\_loss\_scaled, test\_accuracy\_scaled = model\_scaled.evaluate(X\_test\_scaled, y\_test)

print(f"Test Accuracy (Standardized Data): {test\_accuracy\_scaled \* 100:.2f}%")

Then the predictions were made and evaluated by the code below.

# Make predictions and evaluate using sklearn (both raw and standardized data)

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

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

# Evaluation metrics for raw data

print("\nClassification Report (Raw Data):")

print(classification\_report(y\_test, y\_pred\_raw))

print("\nConfusion Matrix (Raw Data):")

print(confusion\_matrix(y\_test, y\_pred\_raw))

# Evaluation metrics for standardized data

print("\nClassification Report (Standardized Data):")

print(classification\_report(y\_test, y\_pred\_scaled))

print("\nConfusion Matrix (Standardized Data):")

print(confusion\_matrix(y\_test, y\_pred\_scaled))The result we got is displayed below. The first one is for raw data.

A screenshot of a computer

Description automatically generated

Figure 24. report raw

As you can see, there’s class imbalance for the raw data model. The recall is very low. And the f1-score as well.

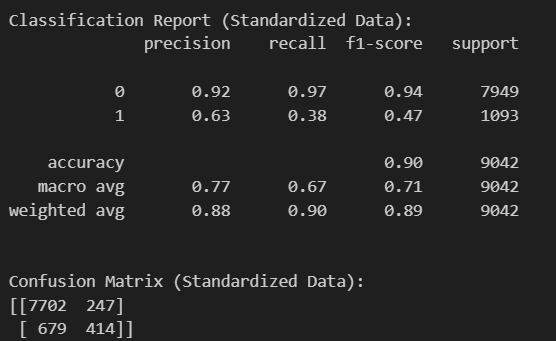


Figure 25. report standardized

As you can see. The class imbalance is slightly reduced for the standardized data. Even though it must be handled. It might not predict class 1 properly with this metrics

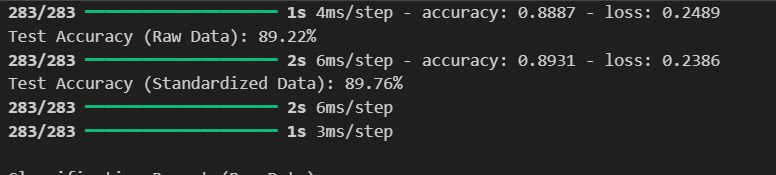


Figure 26. accuracy first model

We can see that it has got a descent accuracy. Both models have validations close to training metrics. So we can see that there’s less chances for overfitting. But we cannot finalize it with the class imbalance. So further improvements must be made.

## Adding class weight to balance the class.

As we can see there is a class imbalance, class weights were added for the model to handle the class imbalance. So we can get better scores for other metrics as well. First class weights were assigned by these codes.

# Compute class weights

from sklearn.utils.class\_weight import compute\_class\_weight

import numpy as np

class\_weights = compute\_class\_weight(

    class\_weight="balanced",

    classes=np.unique(y\_train),

    y=y\_train

)

class\_weights = dict(enumerate(class\_weights))

It is applied for y variable. Then the model is built again with scaled data. The same parameters were used to build the model again.

# Train the model with class weights

model\_scaled\_class\_weights = build\_model()

history\_class\_weights = model\_scaled\_class\_weights.fit(

    X\_train\_scaled, y\_train,

    epochs=70,

    batch\_size=32,

    validation\_split=0.2,

    class\_weight=class\_weights,  # Add class weights here

    verbose=1

)

# Plot for the model trained with class weights

plot\_history(history\_class\_weights, 'Standardized Data Model with Class Weights')

The result is displayed below.

A graph of weight loss

Description automatically generated with medium confidence

Figure 27. class weight model

We can see that the accuracy is pretty low as it is 80 percent. Also, we can see a significant increase in the validation loss that suggests there might be an overfit. So again, those things have to be considered. A classification report is also being made to check the changes in the other metrics.

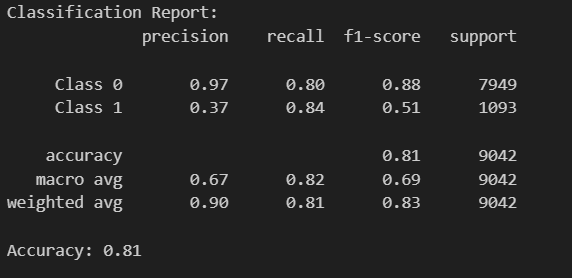


Figure 28. report for weigted model

We can see that the recall has come to a better-balanced position. Though precision is pretty low. So going for further improvements.

## Hyperparameter tuning for the Model.

To enhance the model, Hyperparameter tuning is performed. First libraries were imported.

import tensorflow as tf

import keras\_tuner as kt

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import StandardScaler

from sklearn.utils.class\_weight import compute\_class\_weight

again, the standardized data is used here as neural network works better on standardized data. Here the number of layers, neurons, and learning rates were optimized using keras tuner. Each tunable part are coded according to the tuner. The code below is the code used to build the model.

# Define the model-building function for the tuner

def build\_model(hp):

    model = tf.keras.Sequential()

    # Tuning the number of layers and neurons

    model.add(tf.keras.layers.Dense(

        hp.Int('units1', min\_value=8, max\_value=64, step=8),

        activation='relu',

        input\_shape=(X\_train.shape[1],)

    ))

    # Optionally add more layers

    for i in range(hp.Int('num\_layers', 1, 3)):  # 1 to 3 hidden layers

        model.add(tf.keras.layers.Dense(

            hp.Int(f'units\_{i+2}', min\_value=8, max\_value=64, step=8),

            activation='relu'

        ))

    model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

    # Compile the model with a tunable learning rate

    model.compile(

        optimizer=tf.keras.optimizers.Adam(

            learning\_rate=hp.Float('learning\_rate', min\_value=1e-5, max\_value=1e-2, sampling='LOG')

        ),

        loss='binary\_crossentropy',

        metrics=['accuracy']

    )

    return model

and the below is the code for the tuner.

tuner = kt.Hyperband(

    build\_model,  # The model-building function

    objective='val\_accuracy',  # The metric to optimize

    max\_epochs=60,  # Max epochs to train each model

    factor=3,  # Factor by which the number of trials is reduced

    directory='tuner\_results',  # Directory to save results

    project\_name='hyperparameter\_tuning',  # Name for the project

)

Hyperband optimizes validation accuracy. It trains the model upto 60 epochs. Then tuner.search executes it with different training models with different training data and retrieves the best parameters.

tuner.search(X\_train\_scaled, y\_train, epochs=60, validation\_split=0.2, verbose=1)

# Get the best hyperparameters

best\_hyperparameters = tuner.get\_best\_hyperparameters(num\_trials=1)[0]

print("Best hyperparameters: ", best\_hyperparameters.values)

class weights also considered for the imbalance data as we faced earlier. It is tuned with different code set

# Compute class weights

class\_weights = compute\_class\_weight(

    class\_weight='balanced',

    classes=np.unique(y\_train),

    y=y\_train

)

class\_weights = dict(enumerate(class\_weights))

print("Class Weights:", class\_weights)

The best model which is retrieved is saved and trained separately to get the results. The trained history with each trail with the parameters are saved for efficiency. It does not require more time to train again as it retrieves it from the information saved. And finally, the graph of the training is plotted.

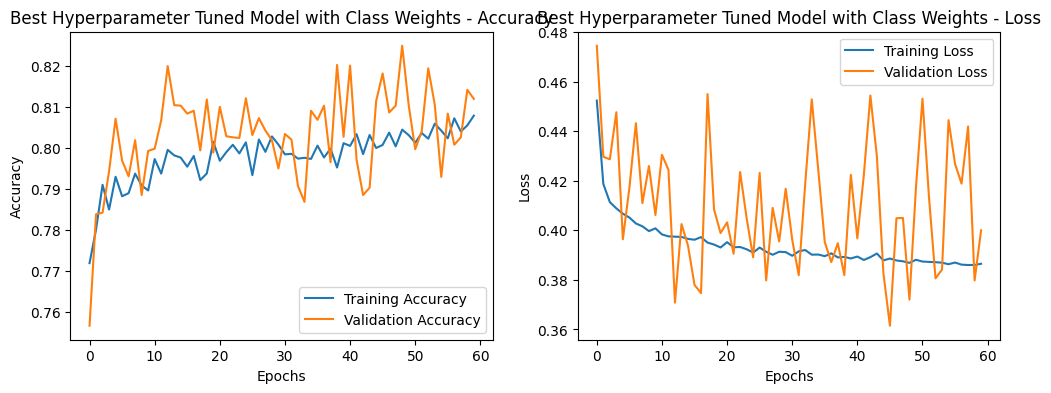


Figure 29. parameter tuned model

The best parameter was this.

Best hyperparameters: {'units1': 8, 'num\_layers': 1, 'units\_2': 24, 'learning\_rate': 0.008583846101691946, 'units\_3': 48, 'units\_4': 40, 'tuner/epochs': 60, 'tuner/initial\_epoch': 0, 'tuner/bracket': 0, 'tuner/round': 0}

We can see that it has 3 hidden layers. With 8, 24, 48 neurons respectively. Learning rate also pretty much low. Even though it has got 3 hidden layers it considers only first 2 layers as the given num\_layers is 1. So, the 3rd layer is ignored. From the graph we can see that that accuracy is optimized to 80 percent. It is fairly a low accuracy. Validation loss also has fairly a larger value. When we have a look at the classification report.

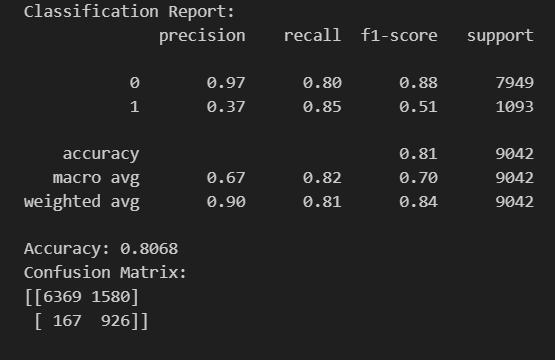


Figure 30. report best param

We can see that the class imbalance is handled a bit. Still, there’s an imbalance in precision. The accuracy is also pretty low compared to the other model. There for further improvement are considered.

## Manually adjusting parameters.

The next step that was taken is adjusting the parameters and repeat training. By adjusting class weights for each and every time and the results were monitored. The class weights were handled by SMOTE and ADASYN. Both of them used different times to check whether they performed best. Many attempts of changing weights, thresholds, increasing and decreasing hidden layers, adjusting neurons, adjusting learning rate, adding class weights, changing epoch value and so much other stuff. Kernel regularization is made in each layer and batch normalization is also made between the layers. After hours of training, I managed to get a graph in this pattern.

A graph of a model and a smote

Description automatically generated with medium confidence

Figure 31. manual adjustedd result

Some how managed to align the training and validation together. Overall, this model performed well. The parameter is mentioned below.

The model had these parameters.

def build\_model():

    optimizer = tf.keras.optimizers.Adam(learning\_rate=0.0001)

    model = tf.keras.Sequential([

        tf.keras.layers.Dense(64, activation='relu', input\_shape=(X\_train.shape[1],), kernel\_regularizer=regularizers.l2(0.01)),

        tf.keras.layers.BatchNormalization(),  # Add BatchNormalization layer

        tf.keras.layers.Dense(64, activation='relu',kernel\_regularizer=regularizers.l2(0.01)),

        tf.keras.layers.BatchNormalization(),  # Add BatchNormalization layer

        tf.keras.layers.Dense(1, activation='sigmoid')

    ])

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

    return model

class weights had this parameter to handle class imbalance.

class\_weights = {0: 1, 1: 1.15}

and other model parameters.

history\_smote = model\_smote.fit(

    X\_train\_smote, y\_train\_smote,

    epochs=40,

    batch\_size=64,

    validation\_split=0.2,

    verbose=1,

    class\_weight=class\_weights

)

## Handling class imbalance.

To get a good imbalance. Model is predicted with different thresholds. It was performed by the code below.

thresholds = [0.71, 0.73, 0.75, 0.78]

for t in thresholds:

    y\_test\_pred = (model\_smote.predict(X\_test\_scaled) > t).astype(int)

    print(f"Threshold: {t}")

    print(classification\_report(y\_test, y\_test\_pred, target\_names=['Class 0', 'Class 1']))

among these threshold, threshold = 0.71 was the best one. The classification report for it is displayed below.

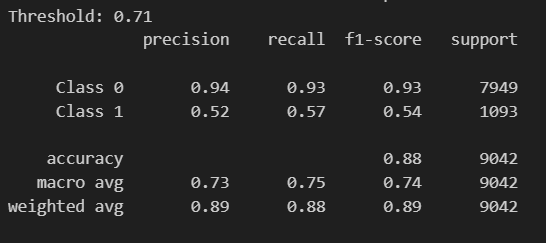


Figure 32. manual adjusted report

It has a class imbalance. But this is the best value that could be retrieved by this model. Further it can be achieved by more experiments.

## ROC curve.

Finally, the ROC curve is plotted for the above model. It was plotted by the code below.  
  
from sklearn.metrics import roc\_curve, roc\_auc\_score

import matplotlib.pyplot as plt

# Calculate ROC curve and AUC

y\_pred\_probs = model\_smote.predict(X\_test\_scaled)

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

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

# Plot ROC Curve

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

plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc\_auc:.2f})")

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

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend()

plt.show()

The threshold is applied, and this was the result.

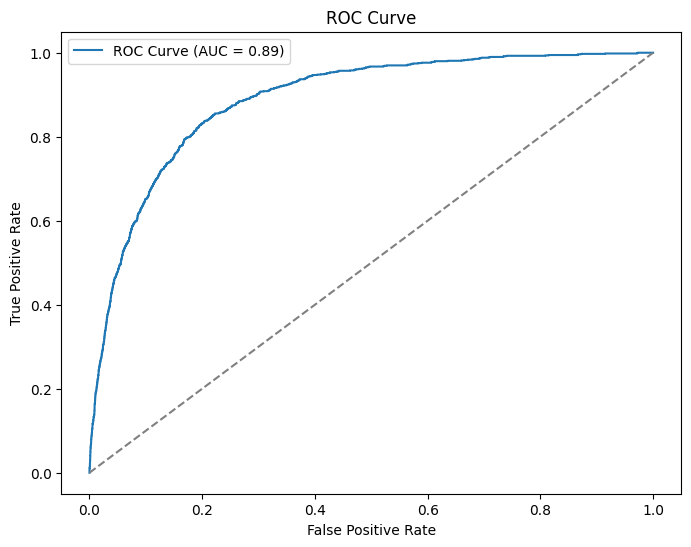


Figure 33. ROC for neural network

We can see that the model is performing well. The curve is closer to the left top corner. This indicates better performance.

## Conclusion for Neural Network.

As we can see from the results, after many attempts of manually adjusted parameter model worked the best. Both training and validation aligned almost the same. Training accuracy increased from 0.70 to 0.87 and validation loss decreased from 1.6 to 0.35 indicating that it can predict well.   
We can see that in the validation accuracy it went from 0.75 to 0.87 which indicates that it can perform well on unseen data. Validation loss from 1.29 to 0.36 shows the good performance of the model. So, we can come to the conclusion that this model is performing well for unseen data.

# Random Forest.

## Data Preprocessing.

The same way of data preprocessing which was used in the neural network is applied here. Therefore, there’s no change in the way of data cleaning data transformation and other analysis. Directly training section will be explained for the random forest classifier.

## Model training.

First the libraires were imported. Randomforest classifier is imported from scikit learn. The dataset is split for training set and testing set then rf model is Initialized and trained. 100 decision trees were used for training. Below code performs these tasks.

from sklearn.ensemble import RandomForestClassifier

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

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

# Features (all columns except 'y')

X = df.drop('y', axis=1)

# Target variable

y = df['y']

# Split the dataset into training and testing sets (80% train, 20% test)

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

# Initialize the Random Forest model

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the Random Forest model on the training data

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

# Make predictions on the test set

y\_pred = rf\_model.predict(X\_test)

## Model Evaluation.

To evaluate the model, simply usual metrices were used. Accuracy and the classification report with f1score, support and precision etc. more than that cross validation are performed here. Commonly used k-fold-cross-validation is used with k equal to 5. We can see if it overfits or not. Also, it can utilize the full dataset to be trained. The code below is the way it is implemented.

# Evaluate the model

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

print(f"Random Forest Model Accuracy: {accuracy \* 100:.2f}%")

# Detailed Classification Report

print("\nClassification Report:")

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

# Confusion Matrix

print("\nConfusion Matrix:")

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

# Optional: Perform cross-validation

cv\_scores = cross\_val\_score(rf\_model, X, y, cv=5, scoring='accuracy')

print(f"\nCross-Validation Scores: {cv\_scores}")

print(f"Mean Cross-Validation Accuracy: {cv\_scores.mean() \* 100:.2f}%")

The result we got for the above model is displayed below.

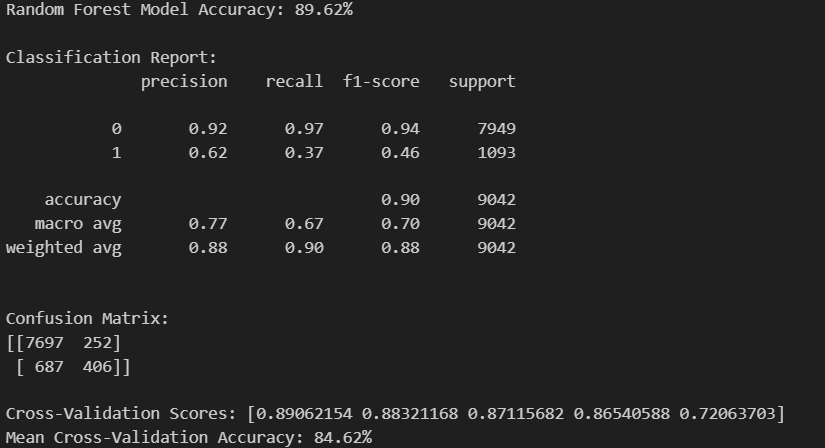


Figure 34. report random forest

We can see that the above model is not performing well, as for different subsets it gives different accuracy even though the accuracy it gave is 89%. When we look at the cross-validation accuracy it ranges from 73 to 89, which can be a worse case. So further improvements must be made. There’s class imbalance as well. That thing also must be considered.

## Hyperparameter tuning for RF.

Now the model is trained with hyperparameter. RandomizedSearchCV is used to the randomforest for the hyperparameter tuning. Define arrays of parameters for the random combination. The code below shows the parameter distribution. SMOTE is applied to handle class imbalances.

# Apply SMOTE to balance the dataset

smote = SMOTE(random\_state=42)

X\_train\_smote, y\_train\_smote = smote.fit\_resample(X\_train, y\_train)

param\_distributions = { 'n\_estimators': [100, 200, 300],

'max\_depth': [10, 20, 30, None],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4] }

After that a model is initialized and then the randomsearch with random combination is performed by using the codes below. Then randomized search is trained.

# Initialize the Random Forest model

rf\_model = RandomForestClassifier(random\_state=42)

# Perform RandomizedSearchCV

random\_search = RandomizedSearchCV(

    estimator=rf\_model,

    param\_distributions=param\_distributions,

    n\_iter=30,  # Number of random combinations to try

    cv=5,       # 5-fold cross-validation

    scoring='accuracy',  # Scoring metric

    random\_state=42,

    n\_jobs=-1    # Use all available CPU cores

)

# Fit RandomizedSearchCV on the training data

random\_search.fit(X\_train, y\_train)

selected parameters are defined as below.

# Define the parameter distribution for random sampling

param\_distributions = {

    'n\_estimators': [100, 200, 300],

    'max\_depth': [10, 20, 30, None],

    'min\_samples\_split': [2, 5, 10],

    'min\_samples\_leaf': [1, 2, 4],

    'class\_weight': ['balanced', 'balanced\_subsample', None]

}

After all the randomized search the best model is selected which has the best accuracy. Then the classification report is displayed for that.

# Get the best model

best\_rf\_model = random\_search.best\_estimator\_

# Print the best hyperparameters

print("Best Hyperparameters:", random\_search.best\_params\_)

# Evaluate the best model on the test set

y\_pred = best\_rf\_model.predict(X\_test)

# Metrics and evaluation

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

print(f"Model Accuracy: {accuracy \* 100:.2f}%")

print("\nClassification Report:")

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

print("\nConfusion Matrix:")

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

The best rf model that detected was displayed with it’s classification report with it.

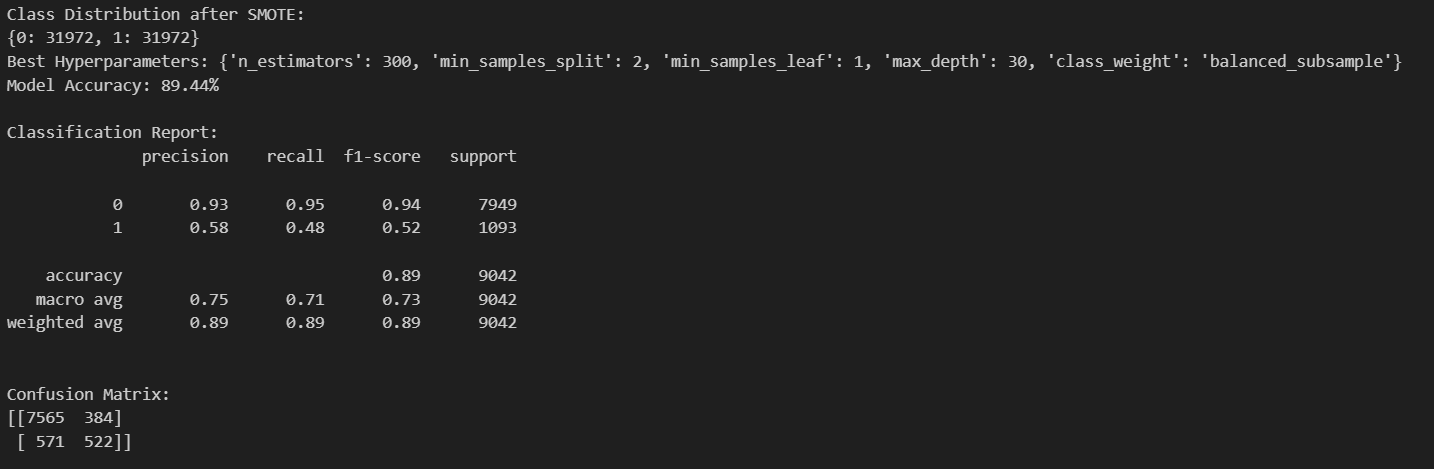


Figure 35. best rf model report

By the report we can see that it has a decent accuracy with 89 percent accuracy. But recall has comparatively less value. So still the class imbalance is not handled properly by the best model from the hyperparameter also. Then cross validation is performed for the best model by the code below.

from sklearn.model\_selection import cross\_val\_score

# Perform cross-validation on the best model

cv\_scores = cross\_val\_score(

    best\_rf\_model,    # Best model selected by RandomizedSearchCV

    X\_train\_smote,    # The SMOTE-augmented training data

    y\_train\_smote,    # The target labels

    cv=5,             # 5-fold cross-validation

    scoring='accuracy',  # Use accuracy as the scoring metric

    n\_jobs=-1         # Use all available CPU cores

)

# Print the cross-validation scores

print("Cross-validation scores for the best model:")

print(cv\_scores)

# Print the mean and standard deviation of the cross-validation scores

print(f"\nMean Accuracy: {cv\_scores.mean() \* 100:.2f}%")

print(f"Standard Deviation: {cv\_scores.std() \* 100:.2f}%")

the output it gave for the cross validation is displayed below.

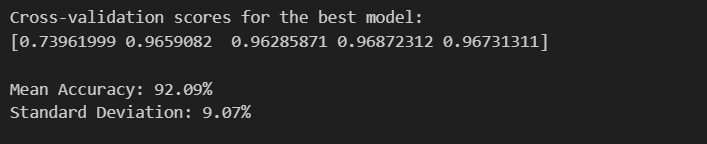


Figure 36. best rf cross validation

Here we can see that it trained with 5 different subsets of dataset. Then the mean accuracy is calculated with the standard deviation. We can see that it is clearly overfitting. For one data fold it goes down to 0.73 but for others it is more than 0.90. This model cannot be used for prediction. Further adjustments and improvements must be made.

## Manually adjusting the parameters.

The next task was to adjust the parameters of the random forest model multiple times to get a good accuracy with class balance. The parameters were defined in this manner. Changes will be made within this code.  
  
# Adjust parameters

manual\_params = {

    "n\_estimators": 200,

    "max\_depth": 25,

    "min\_samples\_split": 5,

    "min\_samples\_leaf": 2,

    "class\_weight": "balanced"

}

The model is trained with a def function so that the parameters can be passed easily when it changes each time. Classification report with accuracy is displayed for each run. The function is defined as below.

# Function to manually adjust hyperparameters and evaluate the model

def evaluate\_rf\_model(X\_train, y\_train, X\_test, y\_test, params):

    # Create the Random Forest model with the given parameters

    rf\_model = RandomForestClassifier(\*\*params, random\_state=42)

    # Fit the model to the training data

    rf\_model.fit(X\_train, y\_train)

    # Make predictions on the test data

    y\_pred = rf\_model.predict(X\_test)

    # Compute accuracy, classification report, and confusion matrix

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

    print(f"Random Forest Model Accuracy: {accuracy \* 100:.2f}%\n")

    print("Classification Report:")

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

    print("Confusion Matrix:")

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

Then the function is called, and the model is trained.

# Call the function with the parameters to evaluate

evaluate\_rf\_model(X\_train, y\_train, X\_test, y\_test, manual\_params)

After many attempts at adjusting parameters and rerunning the codes the best accuracy with a better class balance that could be retrieved is displayed below.

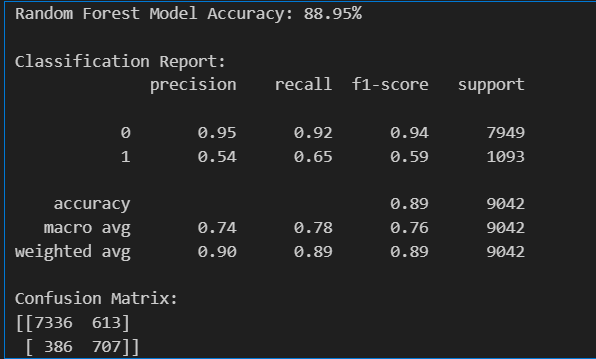


Figure 37. manually adjusted report

Comparatively this is the best model with good accuracy and better class balance than other models. So, this can be used for prediction. The cross validation is performed for this model to see the performance for different subsets of dataset. This was the result we got for this model is displayed below.

A screen shot of a black screen

Description automatically generated

Figure 38. manually adjusted cross validation

So we can see that this model performs well even on the cross validation. The ROC is plotted to identify how well it can be performed. The ROC is plotted for the above model to see the performance using the code below.

import matplotlib.pyplot as plt

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

# Fit the model and get predictions with probabilities

rf\_model = RandomForestClassifier(\*\*manual\_params, random\_state=42)

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

# Get predicted probabilities for the positive class (class 1)

y\_prob = rf\_model.predict\_proba(X\_test)[:, 1]

# Compute the ROC curve and AUC

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

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

# Plot the ROC curve

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

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

plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='Random Guess')

plt.xlabel('False Positive Rate (FPR)')

plt.ylabel('True Positive Rate (TPR)')

plt.title('ROC Curve for Random Forest Classifier')

plt.legend(loc='lower right')

plt.grid()

plt.show()

the output of the code is displayed.

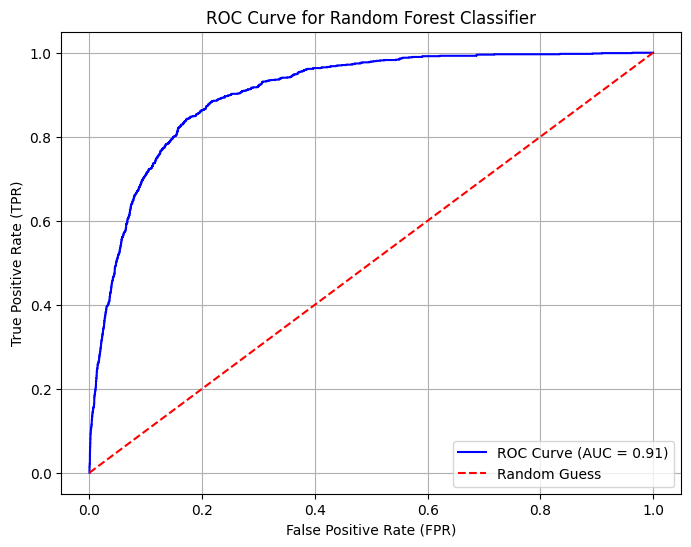


Figure 39. ROC random forest

We can see that there’s a great performance in the model. It has AUC value of 0.91. We can see that the class is well balanced. So, this model can be trusted for predictions. Further evaluations can be made of the model by adjusting a few more parameters or adding some other improvements. But for now, overall, this model is perfect for predictions.

# Git hub.

Git hub URL for the codes - <https://github.com/Abdullah-Nazly/ML_CW.git>

# Appendix.

## Source codes for Neural Network

1. import pandas as pd
2. import matplotlib.pyplot as plt
3. import seaborn as sns
4. df = pd.read\_csv('./Dataset/bank-full.csv', delimiter=';')
5. df.head(20)
6. print(df.isnull().sum())
7. print(df.duplicated().sum())  # Count the number of duplicate rows
8. df.info()
9. df.describe()
10. numeric\_columns = df.select\_dtypes(include=['int64', 'float64']).columns
11. print("Numeric Columns:", numeric\_columns)
12. categorical\_columns = df.select\_dtypes(include=['object']).columns
13. print("Categorical Columns:", categorical\_columns)
14. boolean\_columns = df.select\_dtypes(include=['bool']).columns
15. print("Boolean Columns:", boolean\_columns)
16. df['job'].unique()
17. df['marital'].unique()
18. df['education'].unique()
19. df['default'].unique()
20. df['housing'].unique()
21. df['contact'].unique()
22. df['month'].unique()
23. df['poutcome'].unique()
24. df['y'].unique()
25. df['campaign'].unique()
26. from scipy.stats import chi2\_contingency
27. # If p value is < 0.05, the feature is influencing the targeted variable
28. crosstab = pd.crosstab(df['contact'], df['y'])
29. chi2, p, dof, expected = chi2\_contingency(crosstab)
30. print("Chi-square statistic:", chi2)
31. print("p-value:", p)
32. # Apply One-Hot Encoding to the 'contact' column
33. df\_encoded = pd.get\_dummies(df['contact'], prefix='contact')
34. # Convert True/False to 1/0
35. df\_encoded = df\_encoded.astype(int)
36. # concatenate the encoded columns with the original dataframe
37. df = pd.concat([df, df\_encoded], axis=1)
38. # Drop the original 'contact' column
39. df.drop('contact', axis=1, inplace=True)
40. df.head()
41. crosstab = pd.crosstab(df['poutcome'], df['y'])
42. chi2, p, dof, expected = chi2\_contingency(crosstab)
43. print("Chi-square statistic:", chi2)
44. print("p-value:", p)
45. from scipy.stats import pointbiserialr
46. from sklearn.preprocessing import LabelEncoder
47. # Initialize the LabelEncoder
48. label\_encoder = LabelEncoder()
49. # Apply Label Encoding to the 'poutcome' column
50. df['poutcome\_encoded'] = label\_encoder.fit\_transform(df['poutcome'])
51. df.drop('poutcome', axis=1, inplace=True)
52. # Display the resulting DataFrame
53. print(df)
54. from scipy.stats import pointbiserialr
55. # Step 1: Convert 'y' to numeric (binary)
56. df['y'] = df['y'].map({'no': 0, 'yes': 1})
57. # Step 2: Define numerical features
58. numerical\_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
59. # Step 3: Calculate Point-Biserial Correlation for each feature
60. correlation\_results = []
61. for col in numerical\_features:
62. corr, p\_value = pointbiserialr(df[col], df['y'])
63. correlation\_results.append((col, corr, p\_value))
64. # Step 4: Print results
65. print("Feature-wise Point-Biserial Correlation and p-values:")
66. for feature, corr, p\_value in correlation\_results:
67. print(f"Feature: {feature}, Correlation: {corr:.3f}, p-value: {p\_value:.3f}")
68. import seaborn as sns
69. # Convert correlation results to a DataFrame
70. correlation\_df = pd.DataFrame(correlation\_results, columns=['Feature', 'Correlation', 'p\_value'])
71. # Create a pivot for heatmap visualization (correlation coefficients)
72. heatmap\_data = correlation\_df.pivot\_table(values='Correlation', index=['Feature'])
73. # Plot heatmap
74. plt.figure(figsize=(4, 4))
75. sns.heatmap(heatmap\_data, annot=True, cmap='coolwarm', cbar=True, fmt='.3f', linewidths=0.6, linecolor='black')
76. plt.title('Heatmap of Point-Biserial Correlations', fontsize=14)
77. plt.xlabel('Correlation', fontsize=12)
78. plt.ylabel('Features', fontsize=12)
79. plt.tight\_layout()
80. # Display the heatmap
81. plt.show()
82. import seaborn as sns
83. import matplotlib.pyplot as plt
84. # Count plot for the 'campaign' variable (discrete) and binary target 'y'
85. sns.countplot(x='campaign', hue='y', data=df, palette='Set2')
86. plt.title('Campaign vs Default Outcome')
87. plt.xlabel('Campaign')
88. plt.ylabel('Count')
89. plt.show()
90. # Assuming df is your DataFrame with 'campaign' and 'y' columns
91. campaign\_counts = df.groupby('campaign')['y'].value\_counts().unstack(fill\_value=0)
92. # Display the result
93. print(campaign\_counts)
94. df = df.drop(columns=['day', 'month'])
95. sns.boxplot(x='y', y='duration', data=df)
96. sns.boxplot(x='y', y='age', data=df)
97. sns.boxplot(x='y', y='balance', data=df)
98. sns.boxplot(x='y', y='previous', data=df)
99. # Identify the record with the extreme value in 'previous'
100. outlier\_row = df[df['previous'] > 250]
101. # Display the details of the record
102. print("Outlier row details:")
103. print(outlier\_row)
104. # Drop the specific row
105. df = df.drop(outlier\_row.index)
106. # Verify the row is removed
107. print(f"Updated dataset shape: {df.shape}")
108. df.head()
109. sns.boxplot(x='y', y='pdays', data=df)
110. from sklearn.preprocessing import StandardScaler
111. # Create buckets or categories for 'pdays'
112. df['pdays\_category'] = pd.cut(
113. df['pdays'],
114. bins=[-2, 0, 100, 300, 900],
115. labels=['Not Contacted', 'Recently Contacted', 'Contacted Long Ago', 'Very Long Ago']
116. )
117. # One-hot encode the categories
118. df = pd.get\_dummies(df, columns=['pdays\_category'], prefix='pdays\_cat')
119. # Ensure all boolean-like columns are integers
120. df = df.astype({col: 'int' for col in df.select\_dtypes(include='bool').columns})
121. # Display the transformed dataset
122. print(df)
123. # Create cross-tabulation for each feature with 'y'
124. cross\_tab\_housing = pd.crosstab(df['housing'], df['y'])
125. cross\_tab\_loan = pd.crosstab(df['loan'], df['y'])
126. cross\_tab\_default = pd.crosstab(df['default'], df['y'])
127. # Plot heatmaps
128. plt.figure(figsize=(12, 4))
129. # Heatmap for housing feature
130. plt.subplot(1, 3, 1)
131. sns.heatmap(cross\_tab\_housing, annot=True, fmt='d', cmap='coolwarm', cbar=False)
132. plt.title('Housing vs y')
133. # Heatmap for loan feature
134. plt.subplot(1, 3, 2)
135. sns.heatmap(cross\_tab\_loan, annot=True, fmt='d', cmap='coolwarm', cbar=False)
136. plt.title('Loan vs y')
137. # Heatmap for default feature
138. plt.subplot(1, 3, 3)
139. sns.heatmap(cross\_tab\_default, annot=True, fmt='d', cmap='coolwarm', cbar=False)
140. plt.title('Default vs y')
141. plt.tight\_layout()
142. plt.show()
143. # List of columns to apply Label Encoding to (yes/no columns)
144. yes\_no\_columns = ['default', 'housing', 'loan']
145. # Apply Label Encoding to each of the columns in the list
146. df[yes\_no\_columns] = df[yes\_no\_columns].replace({'yes': 1, 'no': 0})
147. df.head()
148. # Label Encoding
149. education\_mapping = {'primary': 0, 'secondary': 1, 'tertiary': 2, 'unknown': 3}
150. df['education\_encoded'] = df['education'].map(education\_mapping)
151. df = df.drop(columns=['education'])
152. df.head()
153. df['job'].unique()
154. # Grouped bar plot for job and y
155. plt.figure(figsize=(10, 6))
156. # Create a crosstab of job vs y
157. job\_y\_crosstab = pd.crosstab(df['job'], df['y'])
158. # Plot the grouped bar chart
159. job\_y\_crosstab.plot(kind='bar', figsize=(12, 6), color=['skyblue', 'salmon'])
160. plt.title("Distribution of Target Variable (y) by Job", fontsize=16)
161. plt.xlabel("Job", fontsize=14)
162. plt.ylabel("Count", fontsize=14)
163. plt.xticks(rotation=45, fontsize=12)
164. plt.legend(["No", "Yes"], title="Target (y)", fontsize=12)
165. plt.tight\_layout()
166. plt.grid(axis='y', alpha=0.3)
167. plt.show()
168. # Apply One-Hot Encoding to the 'job' column
169. df\_encoded = pd.get\_dummies(df['job'], prefix='job')
170. # Convert True/False to 1/0
171. df\_encoded = df\_encoded.astype(int)
172. # Optionally, concatenate the encoded columns with the original dataframe
173. df = pd.concat([df, df\_encoded], axis=1)
174. # Drop the original 'job' column
175. df.drop('job', axis=1, inplace=True)
176. df.head()
177. from scipy.stats import chi2\_contingency
178. # If p value is < 0.05, the feature is influencing the targeted variable
179. crosstab = pd.crosstab(df['marital'], df['y'])
180. chi2, p, dof, expected = chi2\_contingency(crosstab)
181. print("Chi-square statistic:", chi2)
182. print("p-value:", p)
183. # Apply One-Hot Encoding to the 'marital' column
184. df\_encoded = pd.get\_dummies(df['marital'], prefix='marital')
185. # Convert boolean columns to integers (1 for True, 0 for False)
186. df\_encoded = df\_encoded.astype(int)
187. # Concatenate the encoded columns with the original dataframe
188. df = pd.concat([df, df\_encoded], axis=1)
189. # Drop the original 'marital' column
190. df.drop('marital', axis=1, inplace=True)
191. # Display the resulting DataFrame
192. print(df)
193. df.info()
194. import tensorflow as tf
195. from sklearn.model\_selection import train\_test\_split
196. from sklearn.preprocessing import StandardScaler
197. import matplotlib.pyplot as plt
198. # Features (all columns except 'y')
199. X = df.drop('y', axis=1).values
200. y = df['y'].values
201. # Split the dataset into training and testing sets (80% train, 20% test)
202. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
203. # Standardize the data (standardization)
204. scaler = StandardScaler()
205. X\_train\_scaled = scaler.fit\_transform(X\_train)
206. X\_test\_scaled = scaler.transform(X\_test)
207. # Model architecture (both models will be the same)
208. def build\_model():
209. model = tf.keras.Sequential([
210. tf.keras.layers.Dense(10, activation='relu', input\_shape=(X\_train.shape[1],)),  # Hidden layer with 10 neurons
211. tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer for binary classification
212. ])
213. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])
214. return model
215. # Train the model on raw data (without standardization)
216. model\_raw = build\_model()
217. history\_raw = model\_raw.fit(X\_train, y\_train, epochs=70, batch\_size=32, validation\_split=0.2, verbose=1)
218. # Train the model on standardized data (with standardization)
219. model\_scaled = build\_model()
220. history\_scaled = model\_scaled.fit(X\_train\_scaled, y\_train, epochs=70, batch\_size=32, validation\_split=0.2, verbose=1)
221. # Plotting the graphs
222. def plot\_history(history, title):
223. # Plot Accuracy
224. plt.figure(figsize=(12, 4))
225. plt.subplot(1, 2, 1)
226. plt.plot(history.history['accuracy'], label='Training Accuracy')
227. plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')
228. plt.title(f'{title} - Accuracy')
229. plt.xlabel('Epochs')
230. plt.ylabel('Accuracy')
231. plt.legend()
232. # Plot Loss
233. plt.subplot(1, 2, 2)
234. plt.plot(history.history['loss'], label='Training Loss')
235. plt.plot(history.history['val\_loss'], label='Validation Loss')
236. plt.title(f'{title} - Loss')
237. plt.xlabel('Epochs')
238. plt.ylabel('Loss')
239. plt.legend()
240. plt.show()
241. # Plot for the raw data model
242. plot\_history(history\_raw, 'Raw Data Model')
243. # Plot for the standardized data model
244. plot\_history(history\_scaled, 'Standardized Data Model')
245. from sklearn.metrics import classification\_report, confusion\_matrix
246. # Evaluate the model on raw data
247. test\_loss\_raw, test\_accuracy\_raw = model\_raw.evaluate(X\_test, y\_test)
248. print(f"Test Accuracy (Raw Data): {test\_accuracy\_raw \* 100:.2f}%")
249. # Evaluate the model on standardized data
250. test\_loss\_scaled, test\_accuracy\_scaled = model\_scaled.evaluate(X\_test\_scaled, y\_test)
251. print(f"Test Accuracy (Standardized Data): {test\_accuracy\_scaled \* 100:.2f}%")
252. # Make predictions and evaluate using sklearn (both raw and standardized data)
253. y\_pred\_raw = (model\_raw.predict(X\_test) > 0.5).astype("int32")
254. y\_pred\_scaled = (model\_scaled.predict(X\_test\_scaled) > 0.5).astype("int32")
255. # Evaluation metrics for raw data
256. print("\nClassification Report (Raw Data):")
257. print(classification\_report(y\_test, y\_pred\_raw))
258. print("\nConfusion Matrix (Raw Data):")
259. print(confusion\_matrix(y\_test, y\_pred\_raw))
260. # Evaluation metrics for standardized data
261. print("\nClassification Report (Standardized Data):")
262. print(classification\_report(y\_test, y\_pred\_scaled))
263. print("\nConfusion Matrix (Standardized Data):")
264. print(confusion\_matrix(y\_test, y\_pred\_scaled))
265. # Compute class weights
266. from sklearn.utils.class\_weight import compute\_class\_weight
267. import numpy as np
268. class\_weights = compute\_class\_weight(
269. class\_weight="balanced",
270. classes=np.unique(y\_train),
271. y=y\_train
272. )
273. class\_weights = dict(enumerate(class\_weights))
274. print("Class Weights:", class\_weights)
275. # Train the model with class weights
276. model\_scaled\_class\_weights = build\_model()
277. history\_class\_weights = model\_scaled\_class\_weights.fit(
278. X\_train\_scaled, y\_train,
279. epochs=70,
280. batch\_size=32,
281. validation\_split=0.2,
282. class\_weight=class\_weights,  # Add class weights here
283. verbose=1
284. )
285. # Plot for the model trained with class weights
286. plot\_history(history\_class\_weights, 'Standardized Data Model with Class Weights')
287. from sklearn.metrics import classification\_report, accuracy\_score
288. # Make predictions on the test set
289. y\_pred = model\_scaled\_class\_weights.predict(X\_test\_scaled)
290. y\_pred\_classes = (y\_pred > 0.5).astype(int)  # Convert probabilities to class labels (0 or 1)
291. # Calculate accuracy
292. accuracy = accuracy\_score(y\_test, y\_pred\_classes)
293. # Generate the classification report
294. report = classification\_report(y\_test, y\_pred\_classes, target\_names=['Class 0', 'Class 1'])
295. print("Classification Report:")
296. print(report)
297. print(f"Accuracy: {accuracy:.2f}")
298. import tensorflow as tf
299. import keras\_tuner as kt
300. import matplotlib.pyplot as plt
301. from sklearn.model\_selection import train\_test\_split
302. from sklearn.preprocessing import StandardScaler
303. from sklearn.utils.class\_weight import compute\_class\_weight
304. # Features (all columns except 'y')
305. X = df.drop('y', axis=1).values
306. y = df['y'].values
307. # Split the dataset into training and testing sets (80% train, 20% test)
308. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
309. # Standardize the data (standardization)
310. scaler = StandardScaler()
311. X\_train\_scaled = scaler.fit\_transform(X\_train)
312. X\_test\_scaled = scaler.transform(X\_test)
313. # Define the model-building function for the tuner
314. def build\_model(hp):
315. model = tf.keras.Sequential()
316. # Tuning the number of layers and neurons
317. model.add(tf.keras.layers.Dense(
318. hp.Int('units1', min\_value=8, max\_value=64, step=8),
319. activation='relu',
320. input\_shape=(X\_train.shape[1],)
321. ))
323. # Optionally add more layers
324. for i in range(hp.Int('num\_layers', 1, 3)):  # 1 to 3 hidden layers
325. model.add(tf.keras.layers.Dense(
326. hp.Int(f'units\_{i+2}', min\_value=8, max\_value=64, step=8),
327. activation='relu'
328. ))
330. model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
332. # Compile the model with a tunable learning rate
333. model.compile(
334. optimizer=tf.keras.optimizers.Adam(
335. learning\_rate=hp.Float('learning\_rate', min\_value=1e-5, max\_value=1e-2, sampling='LOG')
336. ),
337. loss='binary\_crossentropy',
338. metrics=['accuracy']
339. )
341. return model
342. # Define the tuner
343. tuner = kt.Hyperband(
344. build\_model,  # The model-building function
345. objective='val\_accuracy',  # The metric to optimize
346. max\_epochs=60,  # Max epochs to train each model
347. factor=3,  # Factor by which the number of trials is reduced
348. directory='tuner\_results',  # Directory to save results
349. project\_name='hyperparameter\_tuning',  # Name for the project
350. )
351. # Run the tuner
352. tuner.search(X\_train\_scaled, y\_train, epochs=60, validation\_split=0.2, verbose=1)
353. # Get the best hyperparameters
354. best\_hyperparameters = tuner.get\_best\_hyperparameters(num\_trials=1)[0]
355. print("Best hyperparameters: ", best\_hyperparameters.values)
356. # Build the best model with the best hyperparameters
357. best\_model = tuner.hypermodel.build(best\_hyperparameters)
358. # Compute class weights
359. class\_weights = compute\_class\_weight(
360. class\_weight='balanced',
361. classes=np.unique(y\_train),
362. y=y\_train
363. )
364. class\_weights = dict(enumerate(class\_weights))
365. print("Class Weights:", class\_weights)
366. # Modify the training step to use class weights
367. history = best\_model.fit(
368. X\_train\_scaled, y\_train,
369. epochs=60,
370. validation\_split=0.2,
371. batch\_size=32,
372. verbose=1,
373. class\_weight=class\_weights  # Pass class weights here
374. )
375. # Evaluate on the test set
376. test\_loss, test\_accuracy = best\_model.evaluate(X\_test\_scaled, y\_test)
377. print(f"Test Loss: {test\_loss}")
378. print(f"Test Accuracy: {test\_accuracy}")
379. # Plot the history of the best model
380. plot\_history(history, 'Best Hyperparameter Tuned Model with Class Weights')
381. # Plotting the training and validation metrics
382. def plot\_history(history, title):
383. # Plot Accuracy
384. plt.figure(figsize=(12, 4))
385. plt.subplot(1, 2, 1)
386. plt.plot(history.history['accuracy'], label='Training Accuracy')
387. plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')
388. plt.title(f'{title} - Accuracy')
389. plt.xlabel('Epochs')
390. plt.ylabel('Accuracy')
391. plt.legend()
392. # Plot Loss
393. plt.subplot(1, 2, 2)
394. plt.plot(history.history['loss'], label='Training Loss')
395. plt.plot(history.history['val\_loss'], label='Validation Loss')
396. plt.title(f'{title} - Loss')
397. plt.xlabel('Epochs')
398. plt.ylabel('Loss')
399. plt.legend()
400. plt.tight\_layout()
401. plt.show()
402. # Make predictions on the test set
403. y\_pred = best\_model.predict(X\_test\_scaled)
404. y\_pred = (y\_pred > 0.5).astype(int)  # Convert probabilities to binary (0 or 1)
405. # Print the classification report
406. from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix
407. print("Classification Report:")
408. print(classification\_report(y\_test, y\_pred))
409. # Calculate the accuracy
410. accuracy = accuracy\_score(y\_test, y\_pred)
411. print(f"Accuracy: {accuracy:.4f}")
412. # Confusion Matrix
413. cm = confusion\_matrix(y\_test, y\_pred)
414. print("Confusion Matrix:")
415. print(cm)
416. from imblearn.over\_sampling import SMOTE
417. import tensorflow as tf
418. from sklearn.model\_selection import train\_test\_split
419. from sklearn.preprocessing import StandardScaler
420. import matplotlib.pyplot as plt
421. from tensorflow.keras import regularizers
422. # Features (all columns except 'y')
423. X = df.drop('y', axis=1).values
424. y = df['y'].values
425. # Split the dataset into training and testing sets (80% train, 20% test)
426. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
427. # Standardize the data (standardization)
428. scaler = StandardScaler()
429. X\_train\_scaled = scaler.fit\_transform(X\_train)
430. X\_test\_scaled = scaler.transform(X\_test)
431. smote = SMOTE(random\_state=42)
432. X\_train\_smote, y\_train\_smote = smote.fit\_resample(X\_train\_scaled, y\_train)
433. def build\_model():
434. optimizer = tf.keras.optimizers.Adam(learning\_rate=0.0001)
435. model = tf.keras.Sequential([
436. tf.keras.layers.Dense(64, activation='relu', input\_shape=(X\_train.shape[1],), kernel\_regularizer=regularizers.l2(0.01)),
437. tf.keras.layers.BatchNormalization(),  # Add BatchNormalization layer
438. tf.keras.layers.Dense(64, activation='relu',kernel\_regularizer=regularizers.l2(0.01)),
439. tf.keras.layers.BatchNormalization(),  # Add BatchNormalization layer
440. tf.keras.layers.Dense(1, activation='sigmoid')
441. ])
442. model.compile(optimizer=optimizer, loss='binary\_crossentropy', metrics=['accuracy'])
443. return model
444. # Train the model on SMOTE-balanced data
445. model\_smote = build\_model()
446. class\_weights = {0: 1, 1: 1.15}
447. history\_smote = model\_smote.fit(
448. X\_train\_smote, y\_train\_smote,
449. epochs=40,
450. batch\_size=64,
451. validation\_split=0.2,
452. verbose=1,
453. class\_weight=class\_weights
454. )
455. # Function to plot training and validation metrics
456. def plot\_history(history, title):
457. # Plot Accuracy
458. plt.figure(figsize=(12, 4))
459. plt.subplot(1, 2, 1)
460. plt.plot(history.history['accuracy'], label='Training Accuracy')
461. plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')
462. plt.title(f'{title} - Accuracy')
463. plt.xlabel('Epochs')
464. plt.ylabel('Accuracy')
465. plt.legend()
466. # Plot Loss
467. plt.subplot(1, 2, 2)
468. plt.plot(history.history['loss'], label='Training Loss')
469. plt.plot(history.history['val\_loss'], label='Validation Loss')
470. plt.title(f'{title} - Loss')
471. plt.xlabel('Epochs')
472. plt.ylabel('Loss')
473. plt.legend()
474. plt.show()
475. # Plot for the SMOTE-balanced data model
476. plot\_history(history\_smote, 'SMOTE Balanced Data Model')
477. thresholds = [0.71, 0.73, 0.75, 0.78]
478. for t in thresholds:
479. y\_test\_pred = (model\_smote.predict(X\_test\_scaled) > t).astype(int)
480. print(f"Threshold: {t}")
481. print(classification\_report(y\_test, y\_test\_pred, target\_names=['Class 0', 'Class 1']))
482. from sklearn.metrics import roc\_curve, roc\_auc\_score
483. import matplotlib.pyplot as plt
484. # Calculate ROC curve and AUC
485. y\_pred\_probs = model\_smote.predict(X\_test\_scaled)
486. fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_probs)
487. roc\_auc = roc\_auc\_score(y\_test, y\_pred\_probs)
488. # Plot ROC Curve
489. plt.figure(figsize=(8, 6))
490. plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc\_auc:.2f})")
491. plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
492. plt.xlabel("False Positive Rate")
493. plt.ylabel("True Positive Rate")
494. plt.title("ROC Curve")
495. plt.legend()
496. plt.show()

## 5.2. Source code for RandomForest.

1. import pandas as pd
2. import matplotlib.pyplot as plt
3. import seaborn as sns
4. df = pd.read\_csv('./Dataset/bank-full.csv', delimiter=';')
5. df.head(20)
6. print(df.isnull().sum())
7. print(df.duplicated().sum())  # Count the number of duplicate rows
8. df.info()
9. df.describe()
10. numeric\_columns = df.select\_dtypes(include=['int64', 'float64']).columns
11. print("Numeric Columns:", numeric\_columns)
12. categorical\_columns = df.select\_dtypes(include=['object']).columns
13. print("Categorical Columns:", categorical\_columns)
14. boolean\_columns = df.select\_dtypes(include=['bool']).columns
15. print("Boolean Columns:", boolean\_columns)
16. df['job'].unique()
17. df['marital'].unique()
18. df['education'].unique()
19. df['default'].unique()
20. df['housing'].unique()
21. df['contact'].unique()
22. df['month'].unique()
23. df['poutcome'].unique()
24. df['y'].unique()
25. df['campaign'].unique()
26. from scipy.stats import chi2\_contingency
27. # If p value is < 0.05, the feature is influencing the targeted variable
28. crosstab = pd.crosstab(df['contact'], df['y'])
29. chi2, p, dof, expected = chi2\_contingency(crosstab)
30. print("Chi-square statistic:", chi2)
31. print("p-value:", p)
32. # Apply One-Hot Encoding to the 'contact' column
33. df\_encoded = pd.get\_dummies(df['contact'], prefix='contact')
34. # Convert True/False to 1/0
35. df\_encoded = df\_encoded.astype(int)
36. # concatenate the encoded columns with the original dataframe
37. df = pd.concat([df, df\_encoded], axis=1)
38. # Drop the original 'contact' column
39. df.drop('contact', axis=1, inplace=True)
40. df.head()
41. crosstab = pd.crosstab(df['poutcome'], df['y'])
42. chi2, p, dof, expected = chi2\_contingency(crosstab)
43. print("Chi-square statistic:", chi2)
44. print("p-value:", p)
45. from sklearn.preprocessing import LabelEncoder
46. # Initialize the LabelEncoder
47. label\_encoder = LabelEncoder()
48. # Apply Label Encoding to the 'poutcome' column
49. df['poutcome\_encoded'] = label\_encoder.fit\_transform(df['poutcome'])
50. df.drop('poutcome', axis=1, inplace=True)
51. # Display the resulting DataFrame
52. print(df)
53. from scipy.stats import pointbiserialr
54. # Step 1: Convert 'y' to numeric (binary)
55. df['y'] = df['y'].map({'no': 0, 'yes': 1})
56. # Step 2: Define numerical features
57. numerical\_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
58. # Step 3: Calculate Point-Biserial Correlation for each feature
59. correlation\_results = []
60. for col in numerical\_features:
61. corr, p\_value = pointbiserialr(df[col], df['y'])
62. correlation\_results.append((col, corr, p\_value))
63. # Step 4: Print results
64. print("Feature-wise Point-Biserial Correlation and p-values:")
65. for feature, corr, p\_value in correlation\_results:
66. print(f"Feature: {feature}, Correlation: {corr:.3f}, p-value: {p\_value:.3f}")
67. import seaborn as sns
68. # Convert correlation results to a DataFrame
69. correlation\_df = pd.DataFrame(correlation\_results, columns=['Feature', 'Correlation', 'p\_value'])
70. # Create a pivot for heatmap visualization (correlation coefficients)
71. heatmap\_data = correlation\_df.pivot\_table(values='Correlation', index=['Feature'])
72. # Plot heatmap
73. plt.figure(figsize=(4, 4))
74. sns.heatmap(heatmap\_data, annot=True, cmap='coolwarm', cbar=True, fmt='.3f', linewidths=0.6, linecolor='black')
75. plt.title('Heatmap of Point-Biserial Correlations', fontsize=14)
76. plt.xlabel('Correlation', fontsize=12)
77. plt.ylabel('Features', fontsize=12)
78. plt.tight\_layout()
79. # Display the heatmap
80. plt.show()
81. import seaborn as sns
82. import matplotlib.pyplot as plt
83. # Count plot for the 'campaign' variable (discrete) and binary target 'y'
84. sns.countplot(x='campaign', hue='y', data=df, palette='Set2')
85. plt.title('Campaign vs Default Outcome')
86. plt.xlabel('Campaign')
87. plt.ylabel('Count')
88. plt.show()
89. import pandas as pd
90. campaign\_counts = df.groupby('campaign')['y'].value\_counts().unstack(fill\_value=0)
91. # Display the result
92. print(campaign\_counts)
93. df = df.drop(columns=['day', 'month'])
94. sns.boxplot(x='y', y='duration', data=df)
95. sns.boxplot(x='y', y='age', data=df)
96. sns.boxplot(x='y', y='balance', data=df)
97. sns.boxplot(x='y', y='previous', data=df)
98. # Identify the record with the extreme value in 'previous'
99. outlier\_row = df[df['previous'] > 250]
100. # Display the details of the record
101. print("Outlier row details:")
102. print(outlier\_row)
103. # Drop the specific row
104. df = df.drop(outlier\_row.index)
105. # Verify the row is removed
106. print(f"Updated dataset shape: {df.shape}")
107. df.head()
108. sns.boxplot( x='y', y= 'pdays', data=df )
109. from sklearn.preprocessing import StandardScaler
110. # Create buckets or categories for 'pdays'
111. df['pdays\_category'] = pd.cut(
112. df['pdays'],
113. bins=[-2, 0, 100, 300, 900],
114. labels=['Not Contacted', 'Recently Contacted', 'Contacted Long Ago', 'Very Long Ago']
115. )
116. # One-hot encode the categories
117. df = pd.get\_dummies(df, columns=['pdays\_category'], prefix='pdays\_cat')
118. # Ensure all boolean-like columns are integers
119. df = df.astype({col: 'int' for col in df.select\_dtypes(include='bool').columns})
120. # Display the transformed dataset
121. print(df)
122. # Create cross-tabulation for each feature with 'y'
123. cross\_tab\_housing = pd.crosstab(df['housing'], df['y'])
124. cross\_tab\_loan = pd.crosstab(df['loan'], df['y'])
125. cross\_tab\_default = pd.crosstab(df['default'], df['y'])
126. # Plot heatmaps
127. plt.figure(figsize=(12, 4))
128. # Heatmap for housing feature
129. plt.subplot(1, 3, 1)
130. sns.heatmap(cross\_tab\_housing, annot=True, fmt='d', cmap='coolwarm', cbar=False)
131. plt.title('Housing vs y')
132. # Heatmap for loan feature
133. plt.subplot(1, 3, 2)
134. sns.heatmap(cross\_tab\_loan, annot=True, fmt='d', cmap='coolwarm', cbar=False)
135. plt.title('Loan vs y')
136. # Heatmap for default feature
137. plt.subplot(1, 3, 3)
138. sns.heatmap(cross\_tab\_default, annot=True, fmt='d', cmap='coolwarm', cbar=False)
139. plt.title('Default vs y')
140. plt.tight\_layout()
141. plt.show()
142. # List of columns to apply Label Encoding to (yes/no columns)
143. yes\_no\_columns = ['default', 'housing', 'loan']
144. # Apply Label Encoding to each of the columns in the list
145. df[yes\_no\_columns] = df[yes\_no\_columns].replace({'yes': 1, 'no': 0})
146. df.head()
147. # Label Encoding
148. education\_mapping = {'primary': 0, 'secondary': 1, 'tertiary': 2, 'unknown': 3}
149. df['education\_encoded'] = df['education'].map(education\_mapping)
150. df = df.drop(columns=['education'])
151. df.head()
152. df['job'].unique()
153. # Grouped bar plot for job and y
154. plt.figure(figsize=(10, 6))
155. # Create a crosstab of job vs y
156. job\_y\_crosstab = pd.crosstab(df['job'], df['y'])
157. # Plot the grouped bar chart
158. job\_y\_crosstab.plot(kind='bar', figsize=(12, 6), color=['skyblue', 'salmon'])
159. plt.title("Distribution of Target Variable (y) by Job", fontsize=16)
160. plt.xlabel("Job", fontsize=14)
161. plt.ylabel("Count", fontsize=14)
162. plt.xticks(rotation=45, fontsize=12)
163. plt.legend(["No", "Yes"], title="Target (y)", fontsize=12)
164. plt.tight\_layout()
165. plt.grid(axis='y', alpha=0.3)
166. plt.show()
167. # Apply One-Hot Encoding to the 'job' column
168. df\_encoded = pd.get\_dummies(df['job'], prefix='job')
169. # Convert True/False to 1/0
170. df\_encoded = df\_encoded.astype(int)
171. # Optionally, concatenate the encoded columns with the original dataframe
172. df = pd.concat([df, df\_encoded], axis=1)
173. # Drop the original 'job' column (optional)
174. df.drop('job', axis=1, inplace=True)
175. df.head()
176. # Apply One-Hot Encoding to the 'marital' column
177. df\_encoded = pd.get\_dummies(df['marital'], prefix='marital')
178. # Convert boolean columns to integers (1 for True, 0 for False)
179. df\_encoded = df\_encoded.astype(int)
180. # Concatenate the encoded columns with the original dataframe
181. df = pd.concat([df, df\_encoded], axis=1)
182. # Drop the original 'marital' column
183. df.drop('marital', axis=1, inplace=True)
184. # Display the resulting DataFrame
185. print(df)
186. df.info()
187. from sklearn.ensemble import RandomForestClassifier
188. from sklearn.model\_selection import train\_test\_split, cross\_val\_score
189. from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix
190. # Features (all columns except 'y')
191. X = df.drop('y', axis=1)
192. # Target variable
193. y = df['y']
194. # Split the dataset into training and testing sets (80% train, 20% test)
195. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
196. # Initialize the Random Forest model
197. rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)
198. # Train the Random Forest model on the training data
199. rf\_model.fit(X\_train, y\_train)
200. # Make predictions on the test set
201. y\_pred = rf\_model.predict(X\_test)
202. # Evaluate the model
203. accuracy = accuracy\_score(y\_test, y\_pred)
204. print(f"Random Forest Model Accuracy: {accuracy \* 100:.2f}%")
205. # Detailed Classification Report
206. print("\nClassification Report:")
207. print(classification\_report(y\_test, y\_pred))
208. # Confusion Matrix
209. print("\nConfusion Matrix:")
210. print(confusion\_matrix(y\_test, y\_pred))
211. # Optional: Perform cross-validation
212. cv\_scores = cross\_val\_score(rf\_model, X, y, cv=5, scoring='accuracy')
213. print(f"\nCross-Validation Scores: {cv\_scores}")
214. print(f"Mean Cross-Validation Accuracy: {cv\_scores.mean() \* 100:.2f}%")
215. from imblearn.over\_sampling import SMOTE
216. from sklearn.model\_selection import RandomizedSearchCV
217. from sklearn.ensemble import RandomForestClassifier
218. from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix
219. import numpy as np
220. # Apply SMOTE to balance the dataset
221. smote = SMOTE(random\_state=42)
222. X\_train\_smote, y\_train\_smote = smote.fit\_resample(X\_train, y\_train)
223. # Print class distribution after SMOTE
224. print("Class Distribution after SMOTE:")
225. unique, counts = np.unique(y\_train\_smote, return\_counts=True)
226. print(dict(zip(unique, counts)))
227. # Define the parameter distribution for random sampling
228. param\_distributions = {
229. 'n\_estimators': [100, 200, 300],
230. 'max\_depth': [10, 20, 30, None],
231. 'min\_samples\_split': [2, 5, 10],
232. 'min\_samples\_leaf': [1, 2, 4],
233. 'class\_weight': ['balanced', 'balanced\_subsample', None]  # No custom weights since data is now balanced
234. }
235. # Initialize the Random Forest model
236. rf\_model = RandomForestClassifier(random\_state=42)
237. # Perform RandomizedSearchCV
238. random\_search = RandomizedSearchCV(
239. estimator=rf\_model,
240. param\_distributions=param\_distributions,
241. n\_iter=30,  # Number of random combinations to try
242. cv=5,       # 5-fold cross-validation
243. scoring='accuracy',  # Scoring metric
244. random\_state=42,
245. n\_jobs=-1    # Use all available CPU cores
246. )
247. # Fit RandomizedSearchCV on the SMOTE-augmented training data
248. random\_search.fit(X\_train\_smote, y\_train\_smote)
249. # Get the best model
250. best\_rf\_model = random\_search.best\_estimator\_
251. # Print the best hyperparameters
252. print("Best Hyperparameters:", random\_search.best\_params\_)
253. # Evaluate the best model on the test set
254. y\_pred = best\_rf\_model.predict(X\_test)
255. # Metrics and evaluation
256. accuracy = accuracy\_score(y\_test, y\_pred)
257. print(f"Model Accuracy: {accuracy \* 100:.2f}%")
258. print("\nClassification Report:")
259. print(classification\_report(y\_test, y\_pred))
260. print("\nConfusion Matrix:")
261. print(confusion\_matrix(y\_test, y\_pred))
262. from sklearn.model\_selection import cross\_val\_score
263. # Perform cross-validation on the best model
264. cv\_scores = cross\_val\_score(
265. best\_rf\_model,    # Best model selected by RandomizedSearchCV
266. X\_train\_smote,    # The SMOTE-augmented training data
267. y\_train\_smote,    # The target labels
268. cv=5,             # 5-fold cross-validation
269. scoring='accuracy',  # Use accuracy as the scoring metric
270. n\_jobs=-1         # Use all available CPU cores
271. )
272. # Print the cross-validation scores
273. print("Cross-validation scores for the best model:")
274. print(cv\_scores)
275. # Print the mean and standard deviation of the cross-validation scores
276. print(f"\nMean Accuracy: {cv\_scores.mean() \* 100:.2f}%")
277. print(f"Standard Deviation: {cv\_scores.std() \* 100:.2f}%")
278. from sklearn.ensemble import RandomForestClassifier
279. from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score
280. # Function to manually adjust hyperparameters and evaluate the model
281. def evaluate\_rf\_model(X\_train, y\_train, X\_test, y\_test, params):
282. # Create the Random Forest model with the given parameters
283. rf\_model = RandomForestClassifier(\*\*params, random\_state=42)
285. # Fit the model to the training data
286. rf\_model.fit(X\_train, y\_train)
288. # Make predictions on the test data
289. y\_pred = rf\_model.predict(X\_test)
291. # Compute accuracy, classification report, and confusion matrix
292. accuracy = accuracy\_score(y\_test, y\_pred)
293. print(f"Random Forest Model Accuracy: {accuracy \* 100:.2f}%\n")
295. print("Classification Report:")
296. print(classification\_report(y\_test, y\_pred))
298. print("Confusion Matrix:")
299. print(confusion\_matrix(y\_test, y\_pred))
300. # Adjust parameters
301. manual\_params = {
302. "n\_estimators": 200,
303. "max\_depth": 25,
304. "min\_samples\_split": 5,
305. "min\_samples\_leaf": 2,
306. "class\_weight": "balanced"
307. }
308. # Call the function with the parameters to evaluate
309. evaluate\_rf\_model(X\_train, y\_train, X\_test, y\_test, manual\_params)
310. from sklearn.ensemble import RandomForestClassifier
311. from sklearn.model\_selection import cross\_val\_score
312. import numpy as np
313. # Perform cross-validation
314. print("Performing 5-Fold Cross-Validation...")
315. cv\_scores = cross\_val\_score(rf\_model, X\_train, y\_train, cv=5, scoring='accuracy')
316. # Display results
317. print("Cross-Validation Scores (Per Fold):")
318. for i, score in enumerate(cv\_scores):
319. print(f"Fold {i+1}: Accuracy = {score \* 100:.2f}%")
320. # Calculate and display the mean and standard deviation of cross-validation accuracy
321. mean\_accuracy = np.mean(cv\_scores) \* 100
322. std\_accuracy = np.std(cv\_scores) \* 100
323. print(f"\nMean Cross-Validation Accuracy: {mean\_accuracy:.2f}%")
324. print(f"Standard Deviation of Accuracy: {std\_accuracy:.2f}%")
325. import matplotlib.pyplot as plt
326. from sklearn.metrics import roc\_curve, roc\_auc\_score
327. # Fit the model and get predictions with probabilities
328. rf\_model = RandomForestClassifier(\*\*manual\_params, random\_state=42)
329. rf\_model.fit(X\_train, y\_train)
330. # Get predicted probabilities for the positive class (class 1)
331. y\_prob = rf\_model.predict\_proba(X\_test)[:, 1]
332. # Compute the ROC curve and AUC
333. fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)
334. roc\_auc = roc\_auc\_score(y\_test, y\_prob)
335. # Plot the ROC curve
336. plt.figure(figsize=(8, 6))
337. plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc\_auc:.2f})')
338. plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='Random Guess')
339. plt.xlabel('False Positive Rate (FPR)')
340. plt.ylabel('True Positive Rate (TPR)')
341. plt.title('ROC Curve for Random Forest Classifier')
342. plt.legend(loc='lower right')
343. plt.grid()
344. plt.show()

# 6. References.

GeeksforGeeks, 2024. *Neural Networks – A Beginner’s Guide*. [online] Available at: <https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/> [Accessed 17 December 2024].

YouTube, 2024. *Neural Networks – A Beginner’s Guide*. [video] Available at: <https://youtu.be/v6VJ2RO66Ag?si=OKzPFFoR8huOMl1T> [Accessed 18 December 2024].

YouTube, 2024. *Deep Learning Tutorial - Full Course for Beginners*. [video] Available at: <https://youtu.be/jmmW0F0biz0?si=PXdfjI-Dyjfi7APu> [Accessed 17 December 2024].