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**

**Report**

**Module Leader: Mr. Sahan**

**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 6](#_Toc185782000)

[2. Neural Network Model 6](#_Toc185782001)

[2.1. Data Preprocessing 6](#_Toc185782002)

[2.2. Checking NULL values and duplicates. 7](#_Toc185782003)

[2.3. Data format analysis. 8](#_Toc185782004)

[2.4. Getting the values of the features 9](#_Toc185782005)

[2.5. Checking the Unique values for each column 10](#_Toc185782006)

[2.6. Analyzing feature contact 10](#_Toc185782007)

[2.7. Analyzing poutcome feature 11](#_Toc185782008)

[2.8. Encoding Y variable and analyzing numerical columns. 11](#_Toc185782009)

[2.9. Removing unnecessary columns. 13](#_Toc185782010)

[2.10. Plotting on boxplot to identify extreme values and outliers 14](#_Toc185782011)

[2.10.1. Plotting duration column. 14](#_Toc185782012)

[2.10.2. Plotting age column. 15](#_Toc185782013)

[2.10.3. Plotting balance column. 15](#_Toc185782014)

[2.10.4. Plotting previous column. 16](#_Toc185782015)

[2.11. Applying Label encoding. 16](#_Toc185782016)

[2.12. Performing Label Encoding for education feature. 18](#_Toc185782017)

[2.13. Encoding Job feature. 18](#_Toc185782018)

[2.14. Encoding contact features. 19](#_Toc185782019)

[2.15. Encoding poutcome and marital features. 19](#_Toc185782020)

[2.16. Training the model. 21](#_Toc185782021)

[2.17. Model Evaluation. 23](#_Toc185782022)

[2.18. Parameter tuning for the Model. 25](#_Toc185782023)

[2.19. Conclusion for Neural Network. 26](#_Toc185782024)

[3. Random Forest. 27](#_Toc185782025)

[3.1. Data Preprocessing. 27](#_Toc185782026)

[3.2. Model training. 27](#_Toc185782027)

[3.3. Model Evaluation. 28](#_Toc185782028)

[3.4. Hyperparameter tuning for RF. 29](#_Toc185782029)

[4. Git hub. 31](#_Toc185782030)

[5. Appendix. 31](#_Toc185782031)

[5.1. Source codes for Neural Netowork 31](#_Toc185782032)

[5.2. Source code for RandomForest. 42](#_Toc185782033)

[6. References. 50](#_Toc185782034)

# Introduction

The Machine Learning course work is processed on 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"

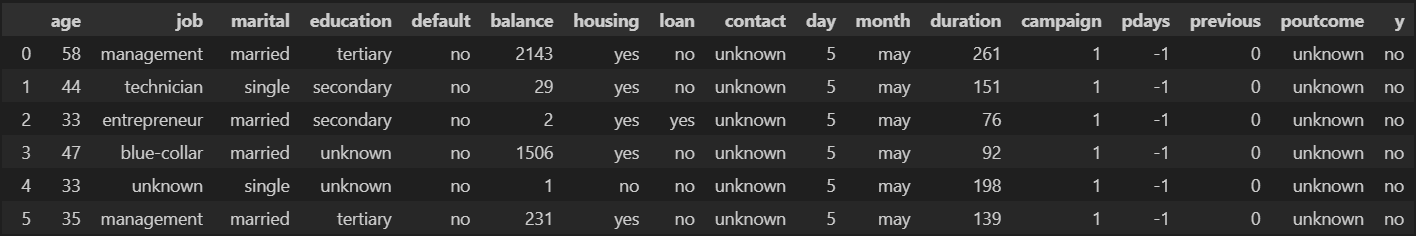
47;"blue-collar";"married";"unknown";"no";1506;"yes";"no";**"unknown";**5;"may";92;1;-1;0;"unknown";"no"

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.



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

A screenshot of a computer

Description automatically generated

## 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

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.

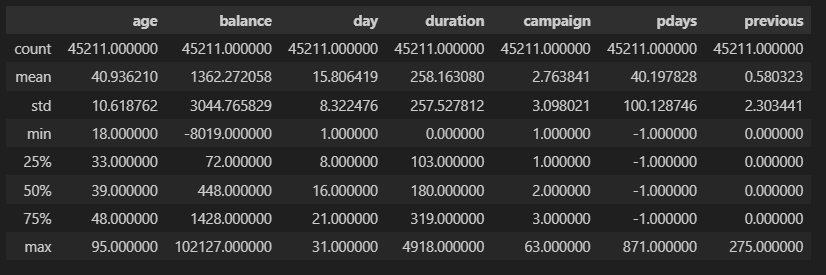
The usual format of checking for format is performed here, with .info() code the Dtypes including null counts are displayed.

df.info()



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

df.describe()



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 doesn’t need data transformation currently.

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

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

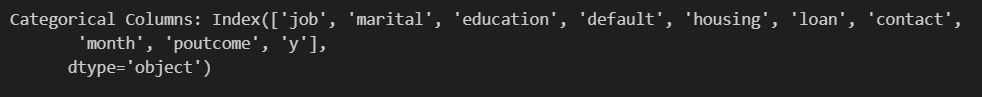


and the output given out put had all the numerical values columns. Next to get all the categorical valued features, the code below is run.

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

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

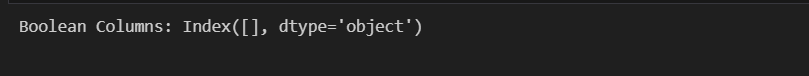
The output gave all the columns which has categorical values.



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 wasn’t any Boolean valued features available.



## 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

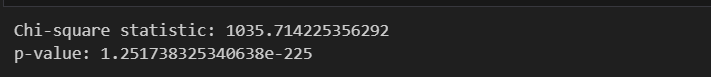
crosstab = pd.crosstab(df['contact'], df['y'])

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.



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

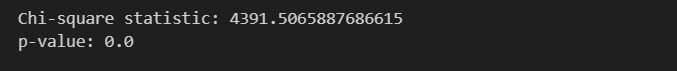
df.drop('contact', axis=1, inplace=True)

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.



## 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))

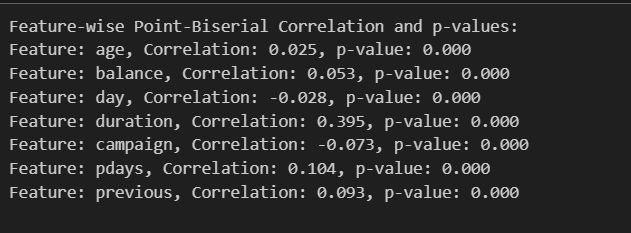
# Step 4: Print results

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

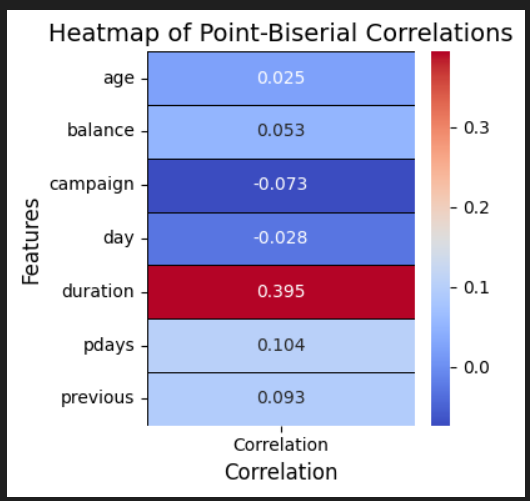
for feature, corr, p\_value in correlation\_results:

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

The output we got is below.



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



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

## 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'])

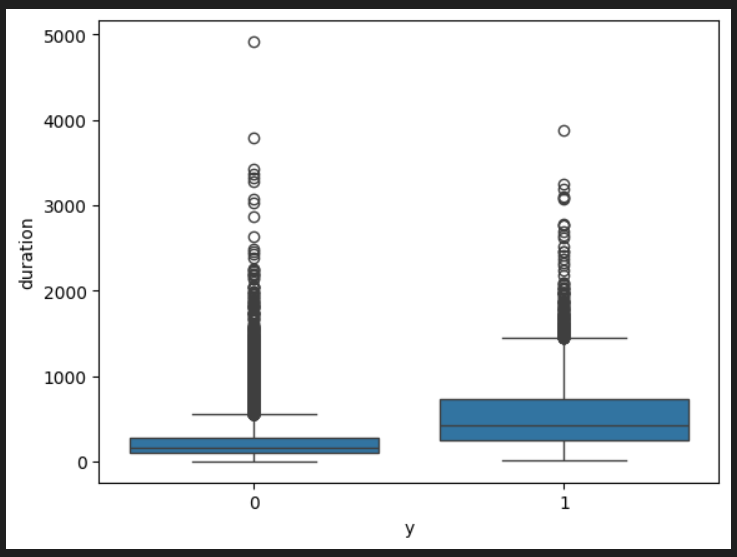
Box plotting is performed between numerical values and target values to analyse the extreme values or incorrect values. For every column similar format of this code is performed for plotting.

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.



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

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 with numbers and lines

Description automatically generated with medium confidence

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 with numbers

Description automatically generated with medium confidence

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

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

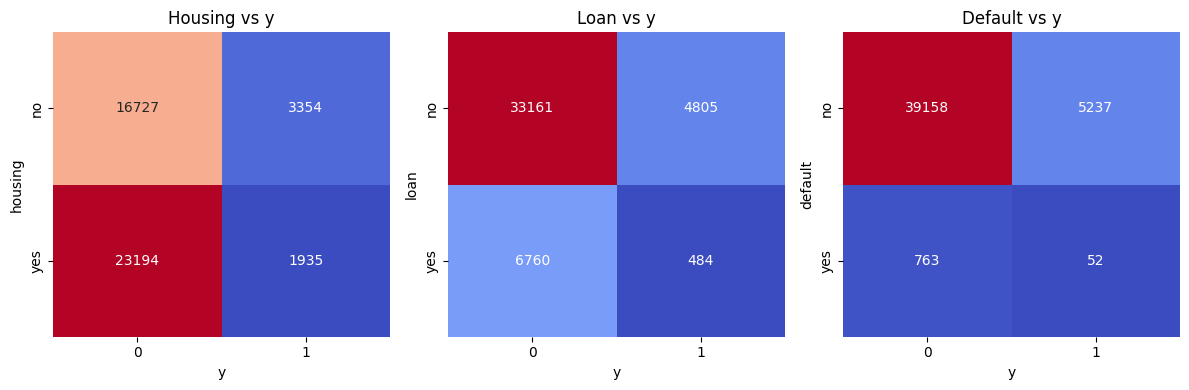
df = df.astype({col: 'int' for col in df.select\_dtypes(include='bool').columns})

# 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.



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

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.



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()

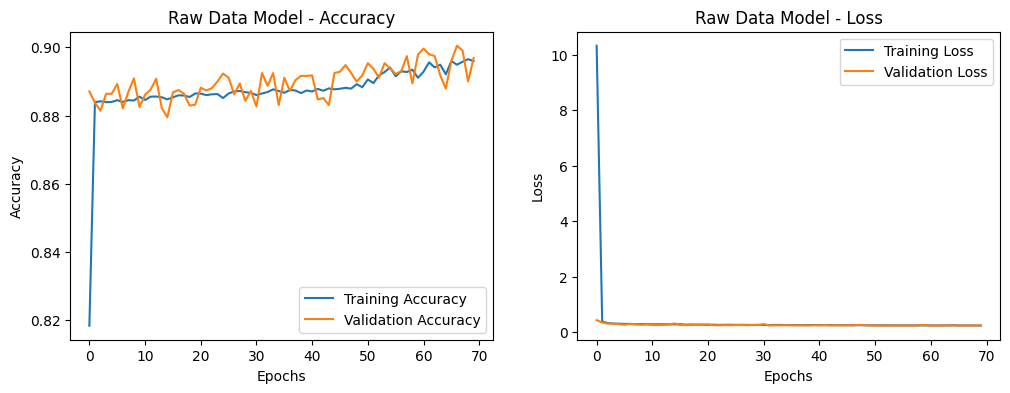
history\_raw = model\_raw.fit(X\_train, y\_train, epochs=70, batch\_size=32, validation\_split=0.2, verbose=1)

# 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.



This shows that raw data works pretty well for the model. Secondly the model which trained with the standardized data is displayed below.

A graph of a model

Description automatically generated with medium confidence

The standardized data model is also performing better. Validation loss is also gradually decreasing which is a good sign. But its 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):")

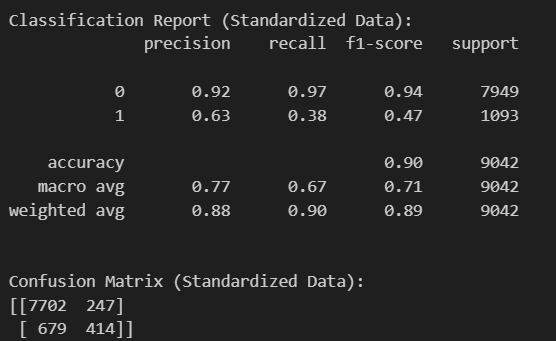
print(confusion\_matrix(y\_test, y\_pred\_scaled))

The result we got is displayed below. The first one is for raw data.

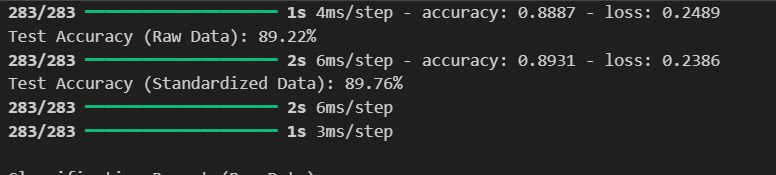
A screenshot of a computer

Description automatically generated

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



As you can see. The class imbalance is slightly reduced. Even though it has to be handled. It might not predict class 1 properly with this metrics



We can see that it has got a descent accuracy. But we cannot finalize it with the class imbalance. So further improvements have to be made.

## Hyperprameter tuning for the Model.

To enhance the model, Hyperparameterr tuning is performed.

# 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(15, activation='relu'),  # Second 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

and the change in epoch.

# Train the model on raw data (without standardization)

model\_raw = build\_model()

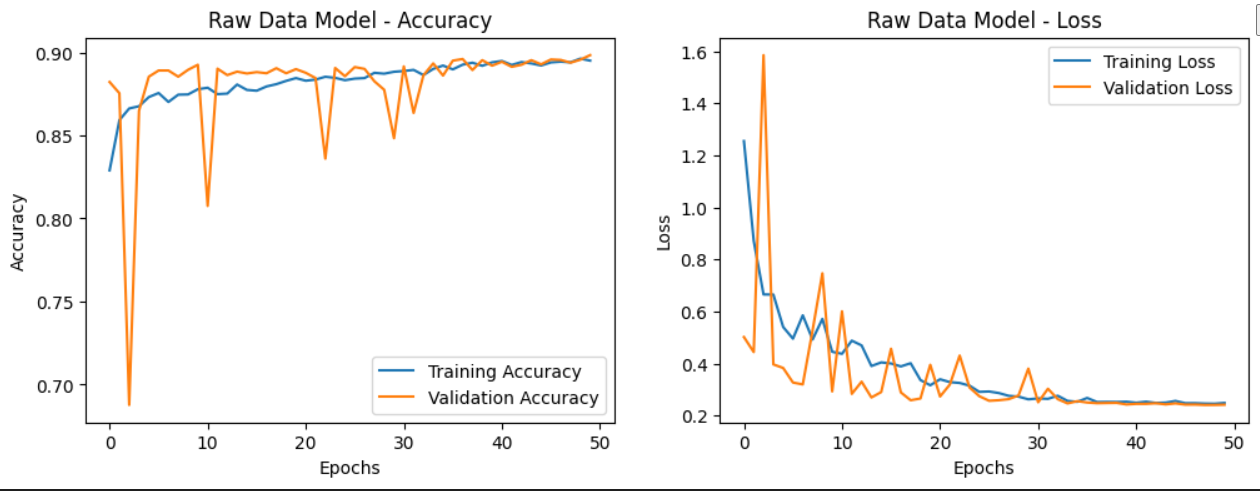
history\_raw = model\_raw.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2, verbose=1)

# Train the model on standardized data (with standardization)

model\_scaled = build\_model()

history\_scaled = model\_scaled.fit(X\_train\_scaled, y\_train, epochs=50, batch\_size=32, validation\_split=0.2, verbose=1)

This is the result we got for the parameter tuned model.



The above plotting is for raw dataset. We can see that it doesn’t have a proper flow. At some points it takes high value as well as low values. The result for the standardized data looks like this.

A graph of a model

Description automatically generated with medium confidence

Even though the accuracy graph shows major changes it’s actually minor changes. We can see that the accuracy is changing between 90 and 89. Also we can see the validation loss is gradually decreasing. So ideally the parameter tuned model is performing best compared to the model we trained before. And finally, the model is evaluated using scikit learn with the same piece of code. This was the result we got.

The first one is for raw data and the second one is for standardized data.  
  
A screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generated

We can see that comparatively this parameter tuned model is performing well as the result showed here.

## Conclusion for Neural Network.

As we can see the results. The Parameter tuned model is best for prediction. Further analysis can be made but as far as this enhancement is sufficient. Further improvements can be made by adjusting parameters, increasing hidden layers and other changes.

# 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))

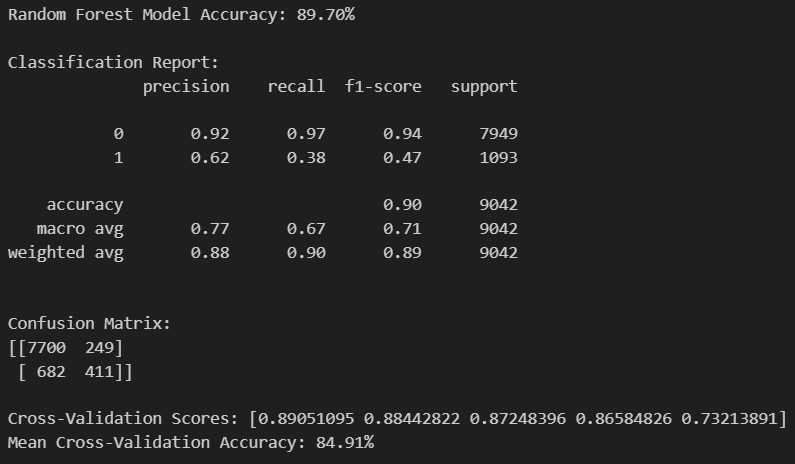
# Optional: Perform cross-validation

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.



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.

## 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.

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)

Then the best combination of parameters is displayed and the result of the model accuracy with f1score, precision, support and recall are displayed. Here’s the result.

A screenshot of a computer

Description automatically generated

Following that the cross-validation score is calculated for the best combination parameter model. The code below shows the part of the coding and result.

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

# Perform cross-validation on the best model found by RandomizedSearchCV

cv\_scores = cross\_val\_score(best\_rf\_model, X\_train, y\_train, cv=5, scoring='accuracy', n\_jobs=-1)

# Print the cross-validation scores and the average score

print(f"Cross-validation scores: {cv\_scores}")

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



Here we can see that it trained with 5 different subsets of dataset. For each set it performed excellent. It has great accuracy for the best model. So, this can be used for prediction. Further enhancement can be made by adding a few more parameters and rerunning. It might get a further enhanced model.

# Git hub.

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

# Appendix.

## Source codes for Neural Netowork

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. crosstab = pd.crosstab(df['poutcome'], df['y'])
33. chi2, p, dof, expected = chi2\_contingency(crosstab)
34. print("Chi-square statistic:", chi2)
35. print("p-value:", p)
36. from scipy.stats import pointbiserialr
37. # Step 1: Convert 'y' to numeric (binary)
38. df['y'] = df['y'].map({'no': 0, 'yes': 1})
39. # Step 2: Define numerical features
40. numerical\_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
41. # Step 3: Calculate Point-Biserial Correlation for each feature
42. correlation\_results = []
43. for col in numerical\_features:
44. corr, p\_value = pointbiserialr(df[col], df['y'])
45. correlation\_results.append((col, corr, p\_value))
46. # Step 4: Print results
47. print("Feature-wise Point-Biserial Correlation and p-values:")
48. for feature, corr, p\_value in correlation\_results:
49. print(f"Feature: {feature}, Correlation: {corr:.3f}, p-value: {p\_value:.3f}")
50. import seaborn as sns
51. # Convert correlation results to a DataFrame
52. correlation\_df = pd.DataFrame(correlation\_results, columns=['Feature', 'Correlation', 'p\_value'])
53. # Create a pivot for heatmap visualization (correlation coefficients)
54. heatmap\_data = correlation\_df.pivot\_table(values='Correlation', index=['Feature'])
55. # Plot heatmap
56. plt.figure(figsize=(4, 4))
57. sns.heatmap(heatmap\_data, annot=True, cmap='coolwarm', cbar=True, fmt='.3f', linewidths=0.6, linecolor='black')
58. plt.title('Heatmap of Point-Biserial Correlations', fontsize=14)
59. plt.xlabel('Correlation', fontsize=12)
60. plt.ylabel('Features', fontsize=12)
61. plt.tight\_layout()
62. # Display the heatmap
63. plt.show()
64. import seaborn as sns
65. import matplotlib.pyplot as plt
66. # Count plot for the 'campaign' variable (discrete) and binary target 'y'
67. sns.countplot(x='campaign', hue='y', data=df, palette='Set2')
68. plt.title('Campaign vs Default Outcome')
69. plt.xlabel('Campaign')
70. plt.ylabel('Count')
71. plt.show()
72. import pandas as pd
73. # Assuming df is your DataFrame with 'campaign' and 'y' columns
74. campaign\_counts = df.groupby('campaign')['y'].value\_counts().unstack(fill\_value=0)
75. # Display the result
76. print(campaign\_counts)
77. df = df.drop(columns=['day', 'month'])
78. sns.boxplot(x='y', y='duration', data=df)
79. sns.boxplot(x='y', y='age', data=df)
80. sns.boxplot(x='y', y='balance', data=df)
81. sns.boxplot(x='y', y='previous', data=df)
82. # Identify the record with the extreme value in 'previous'
83. outlier\_row = df[df['previous'] > 250]
84. # Display the details of the record
85. print("Outlier row details:")
86. print(outlier\_row)
87. # Drop the specific row
88. df = df.drop(outlier\_row.index)
89. # Verify the row is removed
90. print(f"Updated dataset shape: {df.shape}")
91. df.head()
92. # Create cross-tabulation for each feature with 'y'
93. cross\_tab\_housing = pd.crosstab(df['housing'], df['y'])
94. cross\_tab\_loan = pd.crosstab(df['loan'], df['y'])
95. cross\_tab\_default = pd.crosstab(df['default'], df['y'])
96. # Plot heatmaps
97. plt.figure(figsize=(12, 4))
98. # Heatmap for housing feature
99. plt.subplot(1, 3, 1)
100. sns.heatmap(cross\_tab\_housing, annot=True, fmt='d', cmap='coolwarm', cbar=False)
101. plt.title('Housing vs y')
102. # Heatmap for loan feature
103. plt.subplot(1, 3, 2)
104. sns.heatmap(cross\_tab\_loan, annot=True, fmt='d', cmap='coolwarm', cbar=False)
105. plt.title('Loan vs y')
106. # Heatmap for default feature
107. plt.subplot(1, 3, 3)
108. sns.heatmap(cross\_tab\_default, annot=True, fmt='d', cmap='coolwarm', cbar=False)
109. plt.title('Default vs y')
110. plt.tight\_layout()
111. plt.show()
112. # List of columns to apply Label Encoding to (yes/no columns)
113. yes\_no\_columns = ['default', 'housing', 'loan']
114. # Apply Label Encoding to each of the columns in the list
115. df[yes\_no\_columns] = df[yes\_no\_columns].replace({'yes': 1, 'no': 0})
116. df.head()
117. # Label Encoding
118. education\_mapping = {'primary': 0, 'secondary': 1, 'tertiary': 2, 'unknown': 3}
119. df['education\_encoded'] = df['education'].map(education\_mapping)
120. df = df.drop(columns=['education'])
121. df.head()
122. df['job'].unique()
123. # Grouped bar plot for job and y
124. plt.figure(figsize=(10, 6))
125. # Create a crosstab of job vs y
126. job\_y\_crosstab = pd.crosstab(df['job'], df['y'])
127. # Plot the grouped bar chart
128. job\_y\_crosstab.plot(kind='bar', figsize=(12, 6), color=['skyblue', 'salmon'])
129. plt.title("Distribution of Target Variable (y) by Job", fontsize=16)
130. plt.xlabel("Job", fontsize=14)
131. plt.ylabel("Count", fontsize=14)
132. plt.xticks(rotation=45, fontsize=12)
133. plt.legend(["No", "Yes"], title="Target (y)", fontsize=12)
134. plt.tight\_layout()
135. plt.grid(axis='y', alpha=0.3)
136. plt.show()
137. # Apply One-Hot Encoding to the 'job' column
138. df\_encoded = pd.get\_dummies(df['job'], prefix='job')
139. # Convert True/False to 1/0
140. df\_encoded = df\_encoded.astype(int)
141. # Optionally, concatenate the encoded columns with the original dataframe
142. df = pd.concat([df, df\_encoded], axis=1)
143. # Drop the original 'job' column
144. df.drop('job', axis=1, inplace=True)
145. df.head()
146. # Apply One-Hot Encoding to the 'contact' column
147. df\_encoded = pd.get\_dummies(df['contact'], prefix='contact')
148. # Convert True/False to 1/0
149. df\_encoded = df\_encoded.astype(int)
150. # concatenate the encoded columns with the original dataframe
151. df = pd.concat([df, df\_encoded], axis=1)
152. # Drop the original 'contact' column
153. df.drop('contact', axis=1, inplace=True)
154. df.head()
155. from scipy.stats import chi2\_contingency
156. # If p value is < 0.05, the feature is influencing the targeted variable
157. crosstab = pd.crosstab(df['poutcome'], df['y'])
158. chi2, p, dof, expected = chi2\_contingency(crosstab)
159. print("Chi-square statistic:", chi2)
160. print("p-value:", p)
161. from sklearn.preprocessing import LabelEncoder
162. # Initialize the LabelEncoder
163. label\_encoder = LabelEncoder()
164. # Apply Label Encoding to the 'poutcome' column
165. df['poutcome\_encoded'] = label\_encoder.fit\_transform(df['poutcome'])
166. df.drop('poutcome', axis=1, inplace=True)
167. # Display the resulting DataFrame
168. print(df)
169. from scipy.stats import chi2\_contingency
170. # If p value is < 0.05, the feature is influencing the targeted variable
171. crosstab = pd.crosstab(df['marital'], df['y'])
172. chi2, p, dof, expected = chi2\_contingency(crosstab)
173. print("Chi-square statistic:", chi2)
174. print("p-value:", p)
175. # Apply One-Hot Encoding to the 'marital' column
176. df\_encoded = pd.get\_dummies(df['marital'], prefix='marital')
177. # Convert boolean columns to integers (1 for True, 0 for False)
178. df\_encoded = df\_encoded.astype(int)
179. # Concatenate the encoded columns with the original dataframe
180. df = pd.concat([df, df\_encoded], axis=1)
181. # Drop the original 'marital' column
182. df.drop('marital', axis=1, inplace=True)
183. # Display the resulting DataFrame
184. print(df)
185. df.info()
186. import tensorflow as tf
187. from sklearn.model\_selection import train\_test\_split
188. from sklearn.preprocessing import StandardScaler
189. import matplotlib.pyplot as plt
190. # Features (all columns except 'y')
191. X = df.drop('y', axis=1).values
192. y = df['y'].values
193. # Split the dataset into training and testing sets (80% train, 20% test)
194. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
195. # Standardize the data (standardization)
196. scaler = StandardScaler()
197. X\_train\_scaled = scaler.fit\_transform(X\_train)
198. X\_test\_scaled = scaler.transform(X\_test)
199. # Model architecture (both models will be the same)
200. def build\_model():
201. model = tf.keras.Sequential([
202. tf.keras.layers.Dense(10, activation='relu', input\_shape=(X\_train.shape[1],)),  # Hidden layer with 10 neurons
203. tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer for binary classification
204. ])
205. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])
206. return model
207. # Train the model on raw data (without standardization)
208. model\_raw = build\_model()
209. history\_raw = model\_raw.fit(X\_train, y\_train, epochs=70, batch\_size=32, validation\_split=0.2, verbose=1)
210. # Train the model on standardized data (with standardization)
211. model\_scaled = build\_model()
212. history\_scaled = model\_scaled.fit(X\_train\_scaled, y\_train, epochs=70, batch\_size=32, validation\_split=0.2, verbose=1)
213. # Plotting the graphs
214. def plot\_history(history, title):
215. # Plot Accuracy
216. plt.figure(figsize=(12, 4))
217. plt.subplot(1, 2, 1)
218. plt.plot(history.history['accuracy'], label='Training Accuracy')
219. plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')
220. plt.title(f'{title} - Accuracy')
221. plt.xlabel('Epochs')
222. plt.ylabel('Accuracy')
223. plt.legend()
224. # Plot Loss
225. plt.subplot(1, 2, 2)
226. plt.plot(history.history['loss'], label='Training Loss')
227. plt.plot(history.history['val\_loss'], label='Validation Loss')
228. plt.title(f'{title} - Loss')
229. plt.xlabel('Epochs')
230. plt.ylabel('Loss')
231. plt.legend()
232. plt.show()
233. # Plot for the raw data model
234. plot\_history(history\_raw, 'Raw Data Model')
235. # Plot for the standardized data model
236. plot\_history(history\_scaled, 'Standardized Data Model')
237. from sklearn.metrics import classification\_report, confusion\_matrix
238. # Evaluate the model on raw data
239. test\_loss\_raw, test\_accuracy\_raw = model\_raw.evaluate(X\_test, y\_test)
240. print(f"Test Accuracy (Raw Data): {test\_accuracy\_raw \* 100:.2f}%")
241. # Evaluate the model on standardized data
242. test\_loss\_scaled, test\_accuracy\_scaled = model\_scaled.evaluate(X\_test\_scaled, y\_test)
243. print(f"Test Accuracy (Standardized Data): {test\_accuracy\_scaled \* 100:.2f}%")
244. # Make predictions and evaluate using sklearn (both raw and standardized data)
245. y\_pred\_raw = (model\_raw.predict(X\_test) > 0.5).astype("int32")
246. y\_pred\_scaled = (model\_scaled.predict(X\_test\_scaled) > 0.5).astype("int32")
247. # Evaluation metrics for raw data
248. print("\nClassification Report (Raw Data):")
249. print(classification\_report(y\_test, y\_pred\_raw))
250. print("\nConfusion Matrix (Raw Data):")
251. print(confusion\_matrix(y\_test, y\_pred\_raw))
252. # Evaluation metrics for standardized data
253. print("\nClassification Report (Standardized Data):")
254. print(classification\_report(y\_test, y\_pred\_scaled))
255. print("\nConfusion Matrix (Standardized Data):")
256. print(confusion\_matrix(y\_test, y\_pred\_scaled))
257. import tensorflow as tf
258. from sklearn.model\_selection import train\_test\_split
259. from sklearn.preprocessing import StandardScaler
260. import matplotlib.pyplot as plt
261. # Features (all columns except 'y')
262. X = df.drop('y', axis=1).values
263. y = df['y'].values
264. # Split the dataset into training and testing sets (80% train, 20% test)
265. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
266. # Standardize the data (standardization)
267. scaler = StandardScaler()
268. X\_train\_scaled = scaler.fit\_transform(X\_train)
269. X\_test\_scaled = scaler.transform(X\_test)
270. # Model architecture (both models will be the same)
271. def build\_model():
272. model = tf.keras.Sequential([
273. tf.keras.layers.Dense(10, activation='relu', input\_shape=(X\_train.shape[1],)),  # Hidden layer with 10 neurons
274. tf.keras.layers.Dense(15, activation='relu'),  # Second Hidden layer with 10 neurons
275. tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer for binary classification
276. ])
277. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])
278. return model
279. # Train the model on raw data (without standardization)
280. model\_raw = build\_model()
281. history\_raw = model\_raw.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2, verbose=1)
282. # Train the model on standardized data (with standardization)
283. model\_scaled = build\_model()
284. history\_scaled = model\_scaled.fit(X\_train\_scaled, y\_train, epochs=50, batch\_size=32, validation\_split=0.2, verbose=1)
285. # Plotting the graphs
286. def plot\_history(history, title):
287. # Plot Accuracy
288. plt.figure(figsize=(12, 4))
289. plt.subplot(1, 2, 1)
290. plt.plot(history.history['accuracy'], label='Training Accuracy')
291. plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')
292. plt.title(f'{title} - Accuracy')
293. plt.xlabel('Epochs')
294. plt.ylabel('Accuracy')
295. plt.legend()
296. # Plot Loss
297. plt.subplot(1, 2, 2)
298. plt.plot(history.history['loss'], label='Training Loss')
299. plt.plot(history.history['val\_loss'], label='Validation Loss')
300. plt.title(f'{title} - Loss')
301. plt.xlabel('Epochs')
302. plt.ylabel('Loss')
303. plt.legend()
304. plt.show()
305. # Plot for the raw data model
306. plot\_history(history\_raw, 'Raw Data Model')
307. # Plot for the standardized data model
308. plot\_history(history\_scaled, 'Standardized Data Model')
309. from sklearn.metrics import  classification\_report, confusion\_matrix
310. # Evaluate the model on raw data
311. test\_loss\_raw, test\_accuracy\_raw = model\_raw.evaluate(X\_test, y\_test)
312. print(f"Test Accuracy (Raw Data): {test\_accuracy\_raw \* 100:.2f}%")
313. # Evaluate the model on standardized data
314. test\_loss\_scaled, test\_accuracy\_scaled = model\_scaled.evaluate(X\_test\_scaled, y\_test)
315. print(f"Test Accuracy (Standardized Data): {test\_accuracy\_scaled \* 100:.2f}%")
316. # Make predictions and evaluate using sklearn (both raw and standardized data)
317. y\_pred\_raw = (model\_raw.predict(X\_test) > 0.5).astype("int32")
318. y\_pred\_scaled = (model\_scaled.predict(X\_test\_scaled) > 0.5).astype("int32")
319. # Evaluation metrics for raw data
320. print("\nClassification Report (Raw Data):")
321. print(classification\_report(y\_test, y\_pred\_raw))
322. print("\nConfusion Matrix (Raw Data):")
323. print(confusion\_matrix(y\_test, y\_pred\_raw))
324. # Evaluation metrics for standardized data
325. print("\nClassification Report (Standardized Data):")
326. print(classification\_report(y\_test, y\_pred\_scaled))
327. print("\nConfusion Matrix (Standardized Data):")
328. print(confusion\_matrix(y\_test, y\_pred\_scaled))

## 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. pd.crosstab(df['contact'], df['y'], normalize='index')
27. from scipy.stats import chi2\_contingency
28. # If p value is < 0.05, the feature is influencing the targeted variable
29. crosstab = pd.crosstab(df['contact'], df['y'])
30. chi2, p, dof, expected = chi2\_contingency(crosstab)
31. print("Chi-square statistic:", chi2)
32. print("p-value:", p)
33. crosstab = pd.crosstab(df['poutcome'], df['y'])
34. chi2, p, dof, expected = chi2\_contingency(crosstab)
35. print("Chi-square statistic:", chi2)
36. print("p-value:", p)
37. sns.boxplot(x='y', y='pdays', data=df)
38. from scipy.stats import pointbiserialr
39. # Step 1: Convert 'y' to numeric (binary)
40. df['y'] = df['y'].map({'no': 0, 'yes': 1})
41. # Step 2: Define numerical features
42. numerical\_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
43. # Step 3: Calculate Point-Biserial Correlation for each feature
44. correlation\_results = []
45. for col in numerical\_features:
46. corr, p\_value = pointbiserialr(df[col], df['y'])
47. correlation\_results.append((col, corr, p\_value))
48. # Step 4: Print results
49. print("Feature-wise Point-Biserial Correlation and p-values:")
50. for feature, corr, p\_value in correlation\_results:
51. print(f"Feature: {feature}, Correlation: {corr:.3f}, p-value: {p\_value:.3f}")
52. import seaborn as sns
53. # Convert correlation results to a DataFrame
54. correlation\_df = pd.DataFrame(correlation\_results, columns=['Feature', 'Correlation', 'p\_value'])
55. # Create a pivot for heatmap visualization (correlation coefficients)
56. heatmap\_data = correlation\_df.pivot\_table(values='Correlation', index=['Feature'])
57. # Plot heatmap
58. plt.figure(figsize=(4, 4))
59. sns.heatmap(heatmap\_data, annot=True, cmap='coolwarm', cbar=True, fmt='.3f', linewidths=0.6, linecolor='black')
60. plt.title('Heatmap of Point-Biserial Correlations', fontsize=14)
61. plt.xlabel('Correlation', fontsize=12)
62. plt.ylabel('Features', fontsize=12)
63. plt.tight\_layout()
64. # Display the heatmap
65. plt.show()
66. # Histogram of 'campaign' for each category of 'y'
67. plt.figure(figsize=(8, 6))
68. sns.histplot(data=df, x='campaign', hue='y', kde=True, palette='Set2', bins=30)
69. plt.title('Campaign vs Default Outcome (Histogram)')
70. plt.xlabel('Campaign')
71. plt.ylabel('Count')
72. plt.show()
73. df = df.drop(columns=['day', 'month'])
74. sns.boxplot(x='y', y='duration', data=df)
75. sns.boxplot(x='y', y='age', data=df)
76. sns.boxplot(x='y', y='balance', data=df)
77. sns.boxplot(x='y', y='previous', data=df)
78. # Identify the record with the extreme value in 'previous'
79. outlier\_row = df[df['previous'] > 250]
80. # Display the details of the record
81. print("Outlier row details:")
82. print(outlier\_row)
83. # Drop the specific row
84. df = df.drop(outlier\_row.index)
85. # Verify the row is removed
86. print(f"Updated dataset shape: {df.shape}")
87. df.head()
88. # Create cross-tabulation for each feature with 'y'
89. cross\_tab\_housing = pd.crosstab(df['housing'], df['y'])
90. cross\_tab\_loan = pd.crosstab(df['loan'], df['y'])
91. cross\_tab\_default = pd.crosstab(df['default'], df['y'])
92. # Plot heatmaps
93. plt.figure(figsize=(12, 4))
94. # Heatmap for housing feature
95. plt.subplot(1, 3, 1)
96. sns.heatmap(cross\_tab\_housing, annot=True, fmt='d', cmap='coolwarm', cbar=False)
97. plt.title('Housing vs y')
98. # Heatmap for loan feature
99. plt.subplot(1, 3, 2)
100. sns.heatmap(cross\_tab\_loan, annot=True, fmt='d', cmap='coolwarm', cbar=False)
101. plt.title('Loan vs y')
102. # Heatmap for default feature
103. plt.subplot(1, 3, 3)
104. sns.heatmap(cross\_tab\_default, annot=True, fmt='d', cmap='coolwarm', cbar=False)
105. plt.title('Default vs y')
106. plt.tight\_layout()
107. plt.show()
108. # List of columns to apply Label Encoding to (yes/no columns)
109. yes\_no\_columns = ['default', 'housing', 'loan']
110. # Apply Label Encoding to each of the columns in the list
111. df[yes\_no\_columns] = df[yes\_no\_columns].replace({'yes': 1, 'no': 0})
112. df.head()
113. # Label Encoding
114. education\_mapping = {'primary': 0, 'secondary': 1, 'tertiary': 2, 'unknown': 3}
115. df['education\_encoded'] = df['education'].map(education\_mapping)
116. df = df.drop(columns=['education'])
117. df.head()
118. df['job'].unique()
119. # Grouped bar plot for job and y
120. plt.figure(figsize=(10, 6))
121. # Create a crosstab of job vs y
122. job\_y\_crosstab = pd.crosstab(df['job'], df['y'])
123. # Plot the grouped bar chart
124. job\_y\_crosstab.plot(kind='bar', figsize=(12, 6), color=['skyblue', 'salmon'])
125. plt.title("Distribution of Target Variable (y) by Job", fontsize=16)
126. plt.xlabel("Job", fontsize=14)
127. plt.ylabel("Count", fontsize=14)
128. plt.xticks(rotation=45, fontsize=12)
129. plt.legend(["No", "Yes"], title="Target (y)", fontsize=12)
130. plt.tight\_layout()
131. plt.grid(axis='y', alpha=0.3)
132. plt.show()
133. # Apply One-Hot Encoding to the 'job' column
134. df\_encoded = pd.get\_dummies(df['job'], prefix='job')
135. # Convert True/False to 1/0
136. df\_encoded = df\_encoded.astype(int)
137. # Optionally, concatenate the encoded columns with the original dataframe
138. df = pd.concat([df, df\_encoded], axis=1)
139. # Drop the original 'job' column (optional)
140. df.drop('job', axis=1, inplace=True)
141. df.head()
142. # Apply One-Hot Encoding to the 'contact' column
143. df\_encoded = pd.get\_dummies(df['contact'], prefix='contact')
144. # Convert True/False to 1/0
145. df\_encoded = df\_encoded.astype(int)
146. # Optionally, concatenate the encoded columns with the original dataframe
147. df = pd.concat([df, df\_encoded], axis=1)
148. # Drop the original 'contact' column
149. df.drop('contact', axis=1, inplace=True)
150. df.head()
151. print(df)
152. from sklearn.preprocessing import LabelEncoder
153. # Initialize the LabelEncoder
154. label\_encoder = LabelEncoder()
155. # Apply Label Encoding to the 'poutcome' column
156. df['poutcome\_encoded'] = label\_encoder.fit\_transform(df['poutcome'])
157. df.drop('poutcome', axis=1, inplace=True)
158. # Display the resulting DataFrame
159. print(df)
160. # Apply One-Hot Encoding to the 'marital' column
161. df\_encoded = pd.get\_dummies(df['marital'], prefix='marital')
162. # Convert boolean columns to integers (1 for True, 0 for False)
163. df\_encoded = df\_encoded.astype(int)
164. # Concatenate the encoded columns with the original dataframe
165. df = pd.concat([df, df\_encoded], axis=1)
166. # Drop the original 'marital' column
167. df.drop('marital', axis=1, inplace=True)
168. # Display the resulting DataFrame
169. print(df)
170. df.info()
171. from sklearn.ensemble import RandomForestClassifier
172. from sklearn.model\_selection import train\_test\_split, cross\_val\_score
173. from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix
174. # Features (all columns except 'y')
175. X = df.drop('y', axis=1)
176. # Target variable
177. y = df['y']
178. # Split the dataset into training and testing sets (80% train, 20% test)
179. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
180. # Initialize the Random Forest model
181. rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)
182. # Train the Random Forest model on the training data
183. rf\_model.fit(X\_train, y\_train)
184. # Make predictions on the test set
185. y\_pred = rf\_model.predict(X\_test)
186. # Evaluate the model
187. accuracy = accuracy\_score(y\_test, y\_pred)
188. print(f"Random Forest Model Accuracy: {accuracy \* 100:.2f}%")
189. # Detailed Classification Report
190. print("\nClassification Report:")
191. print(classification\_report(y\_test, y\_pred))
192. # Confusion Matrix
193. print("\nConfusion Matrix:")
194. print(confusion\_matrix(y\_test, y\_pred))
195. # Optional: Perform cross-validation
196. cv\_scores = cross\_val\_score(rf\_model, X, y, cv=5, scoring='accuracy')
197. print(f"\nCross-Validation Scores: {cv\_scores}")
198. print(f"Mean Cross-Validation Accuracy: {cv\_scores.mean() \* 100:.2f}%")
199. from sklearn.model\_selection import RandomizedSearchCV
200. from sklearn.ensemble import RandomForestClassifier
201. # Define the parameter distribution for random sampling
202. param\_distributions = {
203. 'n\_estimators': [100, 200, 300],
204. 'max\_depth': [10, 20, 30, None],
205. 'min\_samples\_split': [2, 5, 10],
206. 'min\_samples\_leaf': [1, 2, 4]
207. }
208. # Initialize the Random Forest model
209. rf\_model = RandomForestClassifier(random\_state=42)
210. # Perform RandomizedSearchCV
211. random\_search = RandomizedSearchCV(
212. estimator=rf\_model,
213. param\_distributions=param\_distributions,
214. n\_iter=30,  # Number of random combinations to try
215. cv=5,       # 5-fold cross-validation
216. scoring='accuracy',  # Scoring metric
217. random\_state=42,
218. n\_jobs=-1    # Use all available CPU cores
219. )
220. # Fit RandomizedSearchCV on the training data
221. random\_search.fit(X\_train, y\_train)
222. # Get the best model
223. best\_rf\_model = random\_search.best\_estimator\_
224. # Print the best hyperparameters
225. print("Best Hyperparameters:", random\_search.best\_params\_)
226. # Evaluate the best model on the test set
227. y\_pred = best\_rf\_model.predict(X\_test)
228. from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix
229. accuracy = accuracy\_score(y\_test, y\_pred)
230. print(f"Model Accuracy: {accuracy \* 100:.2f}%")
231. print("\nClassification Report:")
232. print(classification\_report(y\_test, y\_pred))
233. print("\nConfusion Matrix:")
234. print(confusion\_matrix(y\_test, y\_pred))
235. from sklearn.model\_selection import cross\_val\_score, cross\_validate
236. # Perform cross-validation on the best model found by RandomizedSearchCV
237. cv\_scores = cross\_val\_score(best\_rf\_model, X\_train, y\_train, cv=5, scoring='accuracy', n\_jobs=-1)
238. # Print the cross-validation scores and the average score
239. print(f"Cross-validation scores: {cv\_scores}")
240. print(f"Mean Cross-validation Accuracy: {cv\_scores.mean() \* 100:.2f}%")

# 6. References.