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 her 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 5](#_Toc185758308)

[2. Neural Network Model 5](#_Toc185758309)

[2.1. Data Preprocessing 5](#_Toc185758310)

[2.2. Checking NULL values and duplicates. 6](#_Toc185758311)

[2.3. Data format analysis. 7](#_Toc185758312)

[2.4. Getting the values of the features 8](#_Toc185758313)

[2.5. Checking the Unique values for each column 9](#_Toc185758314)

[2.6. Analyzing feature contact 9](#_Toc185758315)

[2.7. Analyzing poutcome feature 10](#_Toc185758316)

[2.8. Encoding Y variable and analyzing numerical columns. 10](#_Toc185758317)

[2.9. Removing unnecessary columns. 12](#_Toc185758318)

[2.10. Plotting on boxplot to identify extreme values and outliers 13](#_Toc185758319)

[2.10.1. Plotting duration column. 13](#_Toc185758320)

[2.10.2. Plotting age column. 13](#_Toc185758321)

[2.10.3. Plotting balance column. 14](#_Toc185758322)

[2.10.4. Plotting previous column. 14](#_Toc185758323)

[2.11. Applying Label encoding. 15](#_Toc185758324)

[2.12. Performing Label Encoding for education feature. 16](#_Toc185758325)

[2.13. Encoding Job feature. 16](#_Toc185758326)

[2.14. Encoding contact feature. 17](#_Toc185758327)

[2.15. Encoding poutcome and marital features. 17](#_Toc185758328)

[2.16. Training the model. 19](#_Toc185758329)

# 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 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"

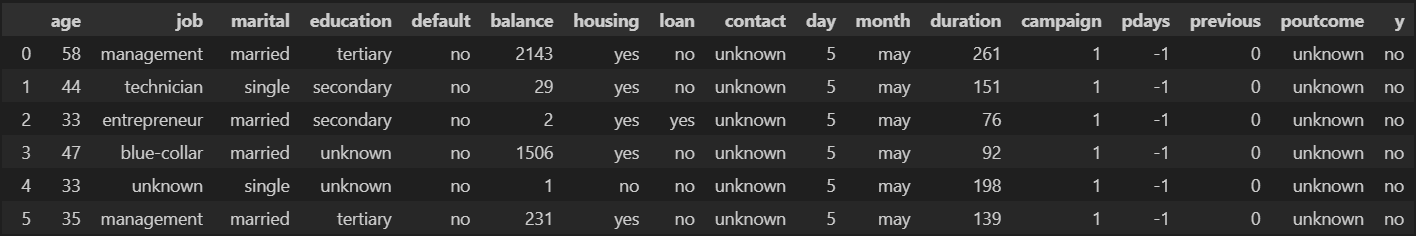
47;"blue-collar";"married";"unknown";"no";1506;"yes";"no";**"unknown";**5;"may";92;1;-1;0;"unknown";"no"

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 values 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.

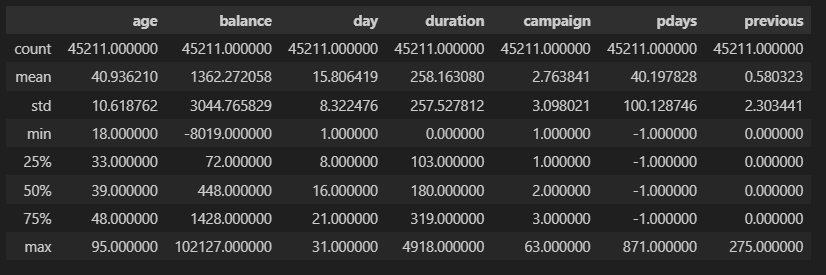
The usual format of checking for format is performed here, with .info() code the Dtypes including null counts are displayed.

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)

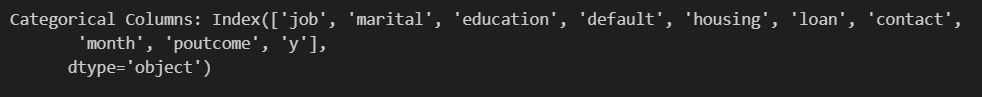


and the output given out put had all the numerical values columns. Next to get all the categorical valued features, the code below is run.

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

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

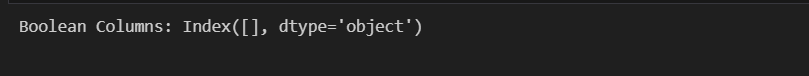
The output gave all the columns which has categorical values.



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

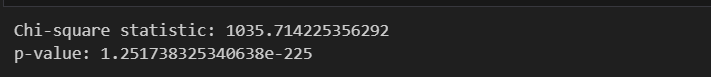
crosstab = pd.crosstab(df['contact'], df['y'])

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

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

print("p-value:", p)

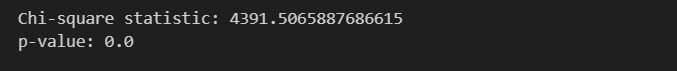
scipy library is imported to do statistical tests for the categorical variables. This is the output that was given to the above code.



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.

## 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 that 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.



## 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))

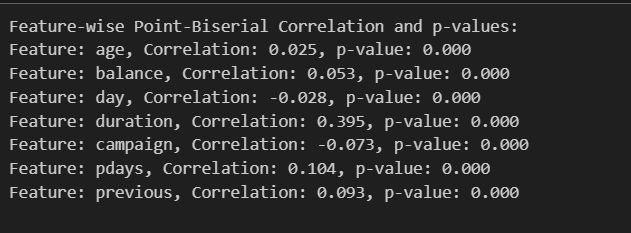
# Step 4: Print results

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

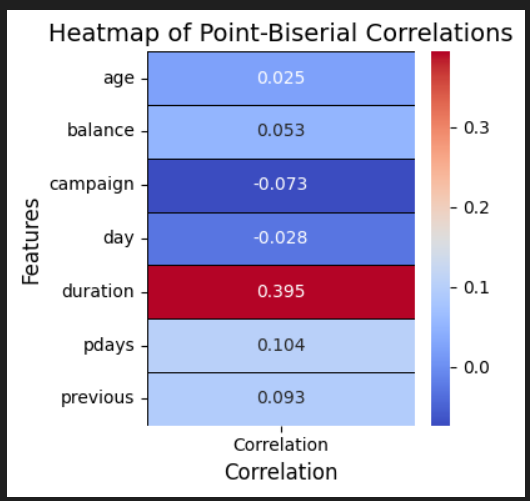
for feature, corr, p\_value in correlation\_results:

    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 compaign feature and y variable. So considering the Countplot we assume that there might be a tiny influence on the prediction. Therefore, the compaign 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'])

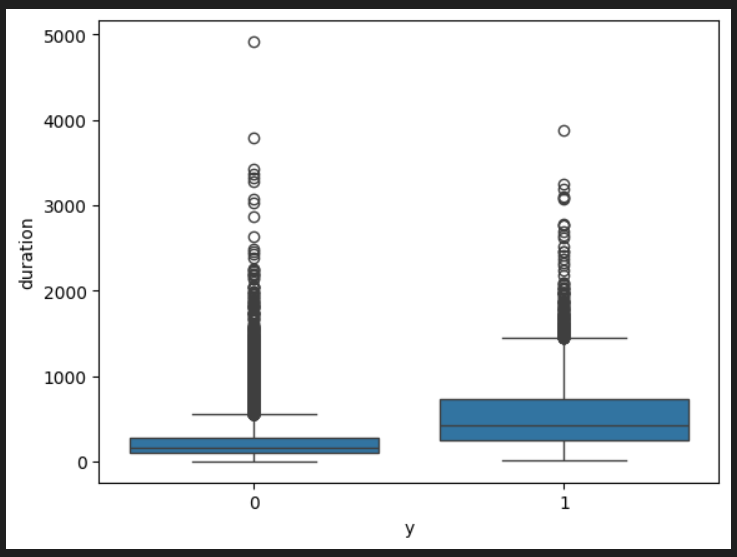
## Plotting on boxplot to identify extreme values and outliers

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)

### 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 in 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 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

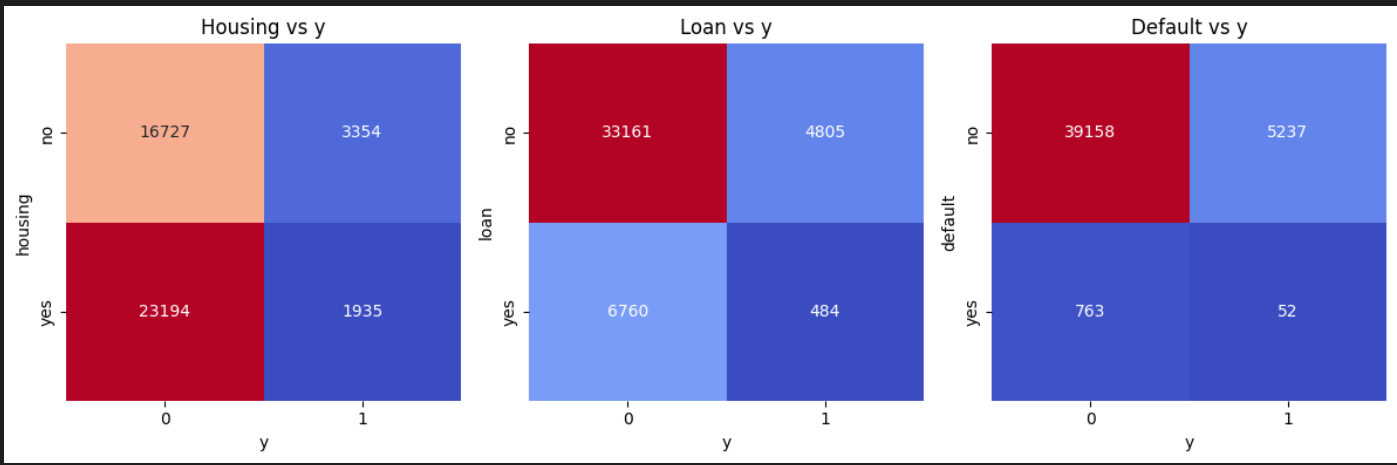
df = df.drop(outlier\_row.index)

# Verify the row is removed

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

## Applying Label encoding.

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 we can understand that each feature has an influence on y variable. When client has a housing loan theres more chance that client can subscribe a term deposit. Like wise other two features also has 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}

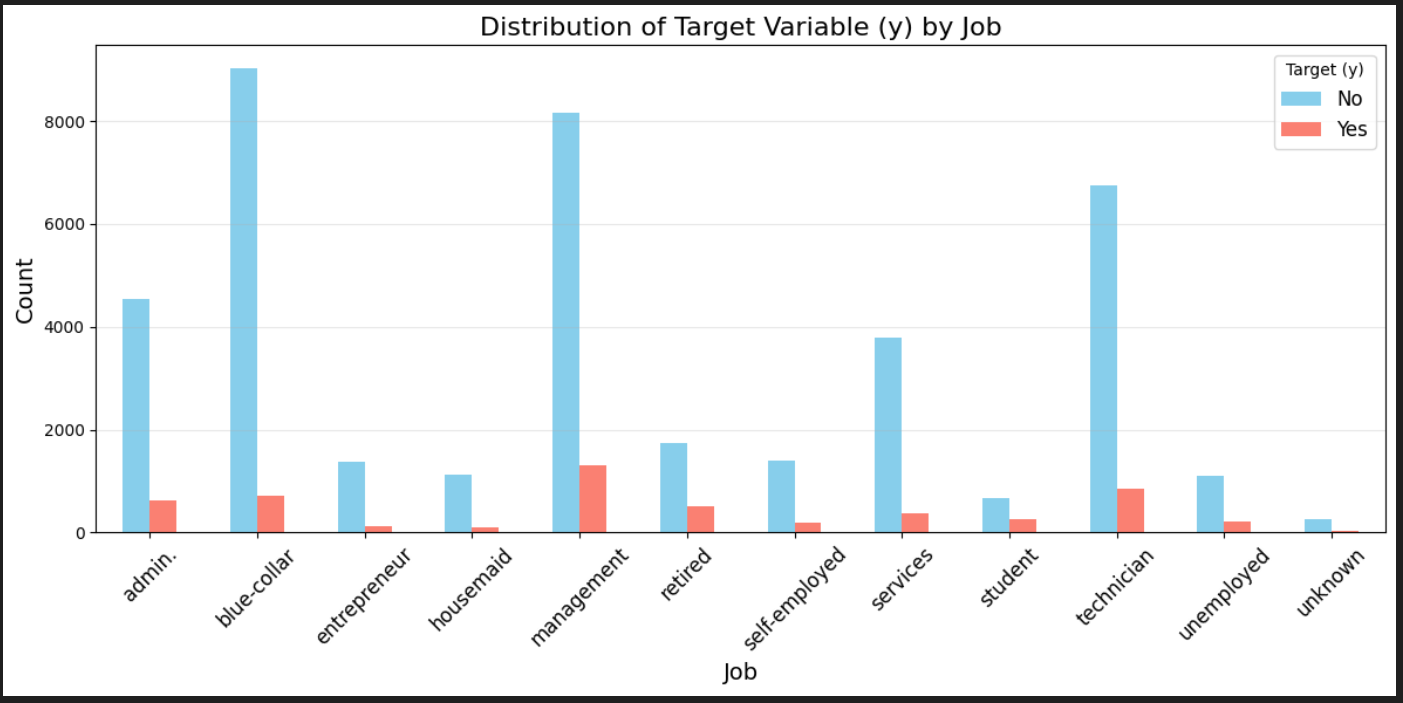
df['education\_encoded'] = df['education'].map(education\_mapping)

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 does not have any relationship between its categorical values. The below plotting shows how the y is distributed for the job feature.



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 contact feature.

As we analyzed earlier the contact feature is one of the most important feature. So, it was also considered to undergo one hot encoding. The code below is performed for 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

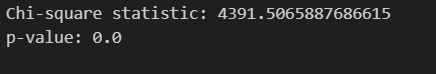
df.drop('contact', axis=1, inplace=True)

df.head()

## Encoding poutcome and marital features.

Firstly chi-square test is performed to analyse the statistical relationship between putcome and y. the same chunk of code is used to get the value which is from scipy library.

The output was.



This means there’s a strong relationship between these two features. Therefore, it must be used and encoded. For encode this feature, label encode is used so that the dimensionality of the dataset can be reduced.

Here, I have imported the scikit learn library to get the label encoder. The code below shows how the encoding is done.

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)

Next the same chi-square test is done for the marital feature and it also gave a similar output showing that it is also an important feature for the prediction model. For this feature, onehoet encoding is used for a better prediction. It was done by the below code.

# 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. The model is trained two times. One for raw data and one for scaled data. Neural networks work efficiently 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 splited to training set and training set. The below code 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 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 build 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 datset and the model was evaluated and plotted.

# Train the model on raw data (without standardization)

model\_raw = build\_model()

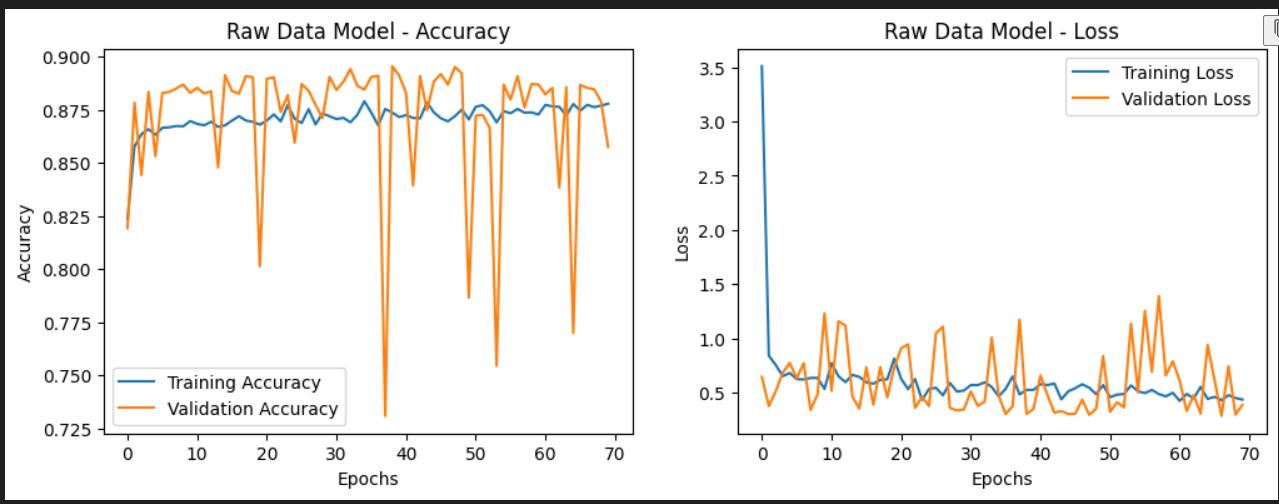
history\_raw = model\_raw.fit(X\_train, y\_train, epochs=70, batch\_size=32, validation\_split=0.2, verbose=1)

# 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 the training the model wit raw data is displayed below.



This shows that raw data is not working properly 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

This graph shows that the model is worked better with the standardized data. we can see that validation loss is gradually decreased which is a good sign. And the accuracy also increased gradually. But for the raw dataset you can see that its completely a gibberish pattern. The validation loss is both increased and decreased at certain.

### 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 below code.

# 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 the raw data.

A screenshot of a computer

Description automatically generated

And this one is for standardized data.

A screenshot of a computer screen

Description automatically generated

The accuracy for both models is displayed.

A screen shot of a computer

Description automatically generated

With the above results we can see that the model is performing quite well in standardized data rather than raw data.

## Parameter tuning for the Model.

To enhance the model, some parameters were modified to increase model accuracy and reduce validation loss. In this place, another hidden layer with 15 neuron is added. Epoch is reduced from 70 to 50 to reduce the validation loss. Here are the updated parts in the codes.

# 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)

# 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.