BSc (Hons) Artificial Intelligence and Data Science

Module: CM 2604
Individual Coursework
Module Leader : Sahan Priyanayana

RGU Student ID	**:**	**2331416**
IIT Student ID	**:**	**20230177**
Student Name	**:**	**M.A.Wasif Asi**

December 2024

Contents

List of figures

List of figures

1.0 Introduction

This report presents an assessment for predicting whether a client will subscribe to a term deposit, based on the Bank Marketing dataset from the UCI Machine Learning Repository. The task involves implementing two machine learning models—a Neural Network (NN) and a Random Forest Classifier (RF)—to perform binary classification on this dataset. The primary objective is to evaluate and compare the performance of these models using well-defined metrics and methodologies.

File understanding

5 csv files and 5 ipynb files are there in the github repository.

- bank-full.csv - The data set used in the project
- Bank-full_tabled.csv – Cleaned data drived from bank-ful.csv
- processed_data_Encoding.csv - Dataset processed up to encoding
- processed_data_Scaling.csv - Dataset processed up to scaling
- processed_data_PCA.csv - Dataset processed up to Dimensionality reduction(PCA)

- Preprocessing.ipynb - Data preparing
- RF_PCA.ipynb - Default and tunning Random forest Model
- RF_Smote_PCA.ipynb - Smote used Random Forest Model
- Tensorflow_NN.ipynb – Neural Network Model with tunning
- Tensorflow_NN_smote_PCA.ipynb – Somte used Neural Network Model with tunning

2.0 Corpus Preparation

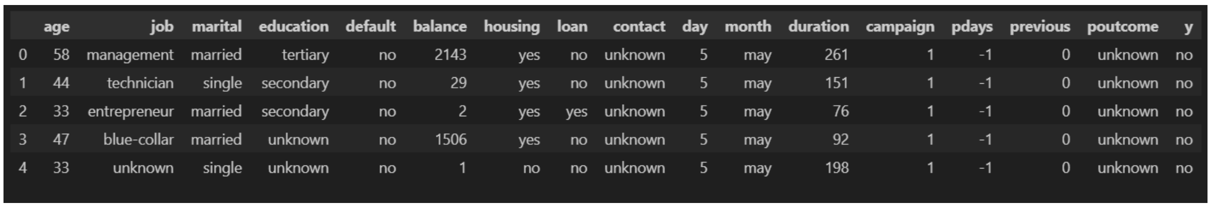
The Bank Marketing dataset consists of client data collected during marketing campaigns conducted by a Portuguese banking institution the dataset named as “bank-full.csv” . The dataset includes 17 features such as client age, job type, marital status, education, contact information, and campaign-related attributes. The target variable, denoted as y, is binary, indicating whether the client subscribed to a term deposit (‘yes’ or ‘no’).

All the dataset loading , checking for the data, missing values handling, duplication handling, Exploratory Data analysis (EDA), and all the feature engineering are conducted in this phase. This notebook file is saved as ‘Preprocessing.ipynb’ .

2.1 Data Preparation and cleaning

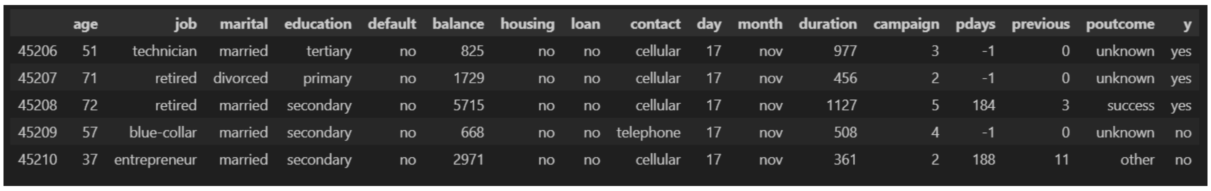
2.1.1.Data Loading

The dataset named “bak-ful.csv” was loaded using Python’s pandas library with separating the data by ‘;’symbol. If I didn’t load by separating ‘;’ symbol then all the data will be in one column. Initial inspection involved displaying the first five rows and last five rows.



	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

Figure 1 first 5 data

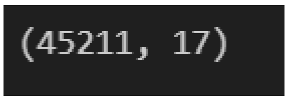


	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
45206	51	technician	married	tertiary	no	825	no	no	cellular	17	nov	977	3	-1	0	unknown	yes
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17	nov	456	2	-1	0	unknown	yes
45208	72	retired	married	secondary	no	5715	no	no	cellular	17	nov	1127	5	184	3	success	yes
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	508	4	-1	0	unknown	no
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	nov	361	2	188	11	other	no

Figure 2 Last 5 data

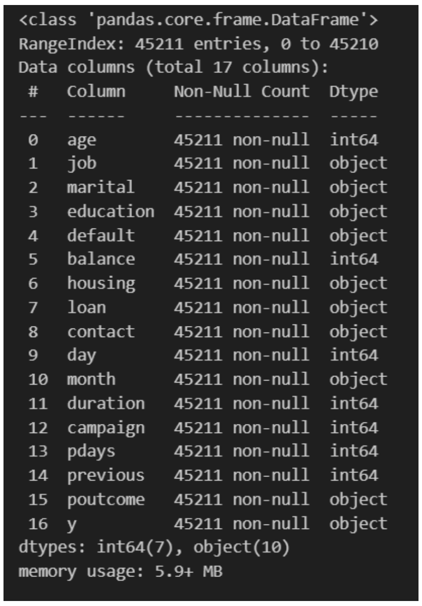
2.1.2 Sanity check

Here I am just analyzing and checking how many rows and columns are there, summarizing the data types, checking for the duplicated values and Identifying the unique for each column.



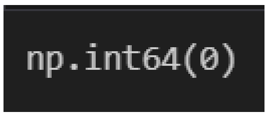
```
(45211, 17)
```

Figure 3 Shape of the dataset



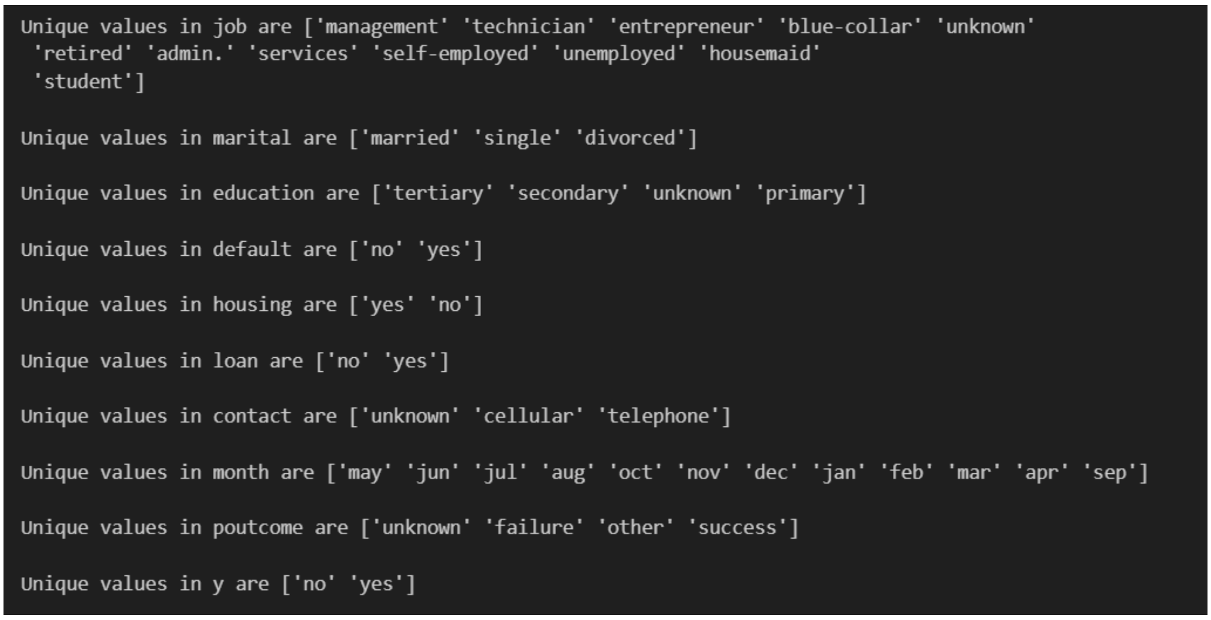
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

Figure 4 Data types



```
np.int64(0)
```

Figure 5 Duplication Values

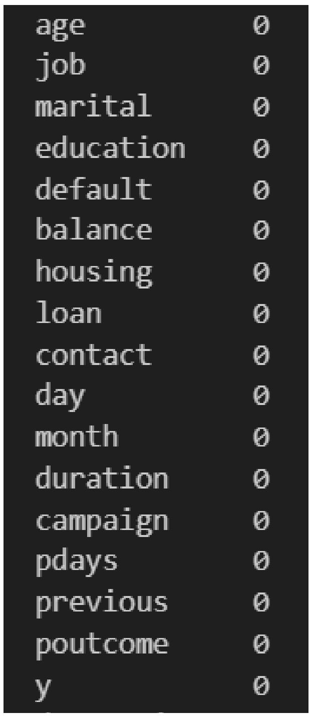


```
Unique values in job are ['management' 'technician' 'entrepreneur' 'blue-collar' 'unknown'
 'retired' 'admin.' 'services' 'self-employed' 'unemployed' 'housemaid'
 'student']

Unique values in marital are ['married' 'single' 'divorced']

Unique values in education are ['tertiary' 'secondary' 'unknown' 'primary']

Unique values in default are ['no' 'yes']

Unique values in housing are ['yes' 'no']

Unique values in loan are ['no' 'yes']

Unique values in contact are ['unknown' 'cellular' 'telephone']

Unique values in month are ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'jan' 'feb' 'mar' 'apr' 'sep']

Unique values in poutcome are ['unknown' 'failure' 'other' 'success']

Unique values in y are ['no' 'yes']
```

Figure 6 Unique Values of Columns

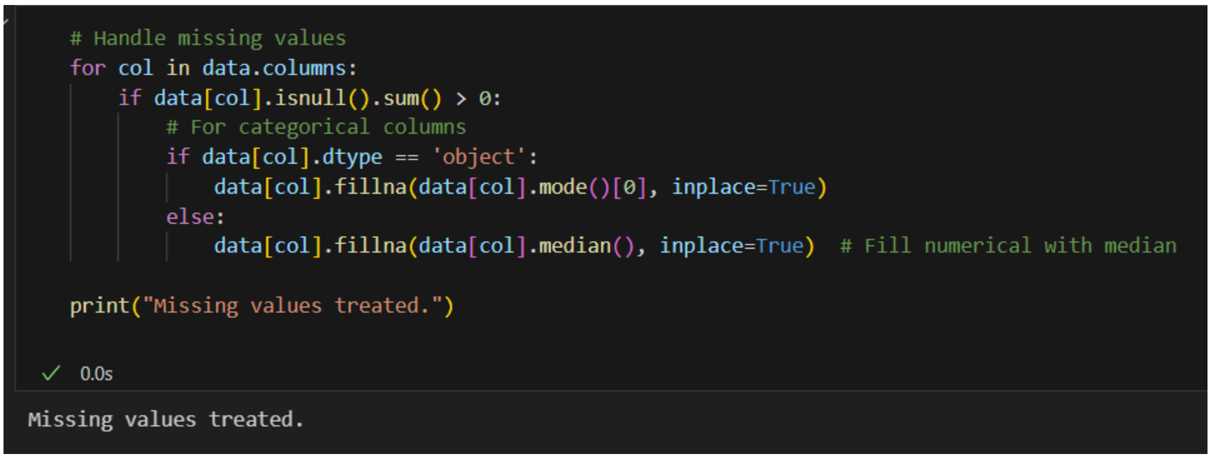
2.1.3 Missing Values Treatment

Up to here only two data types are there. One is Numerical and another one is object. Here I am checking for the missing values and if it is there it will place the mode for the object data type and place the mode for numerical values. In my datasets there was no missing values but putting it as a advances feature and for safety purpose.



```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
```

Figure 7 Missing value count



```
# Handle missing values
for col in data.columns:
    if data[col].isnull().sum() > 0:
        # For categorical columns
        if data[col].dtype == 'object':
            data[col].fillna(data[col].mode()[0], inplace=True)
        else:
            data[col].fillna(data[col].median(), inplace=True)  # Fill numerical with median

print("Missing values treated.")
```

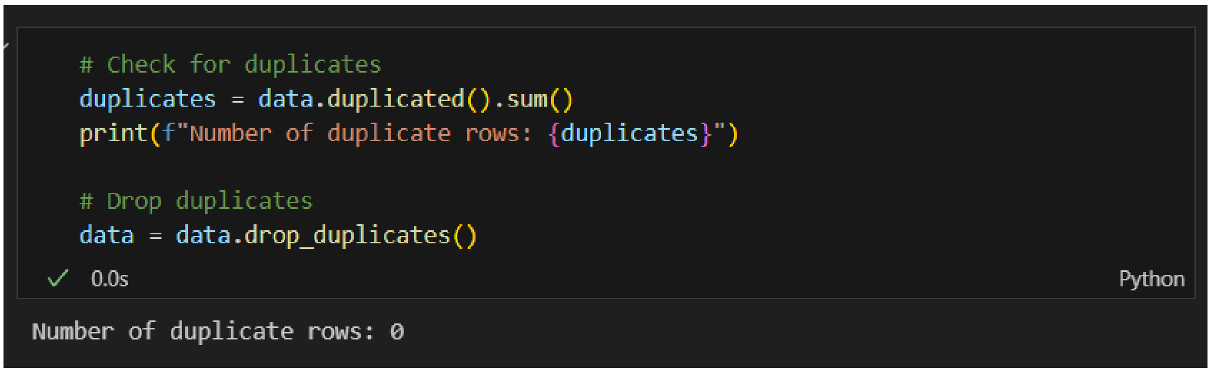
✓ 0.0s

```
Missing values treated.
```

Figure 8 Missing value treatment

2.1.4 Duplication value treatment

Same as the missing value treatment here checking for the duplicated values. If it is there then the data will be dropped from the data set.



```
# Check for duplicates
duplicates = data.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")

# Drop duplicates
data = data.drop_duplicates()
```

✓ 0.0s Python

```
Number of duplicate rows: 0
```

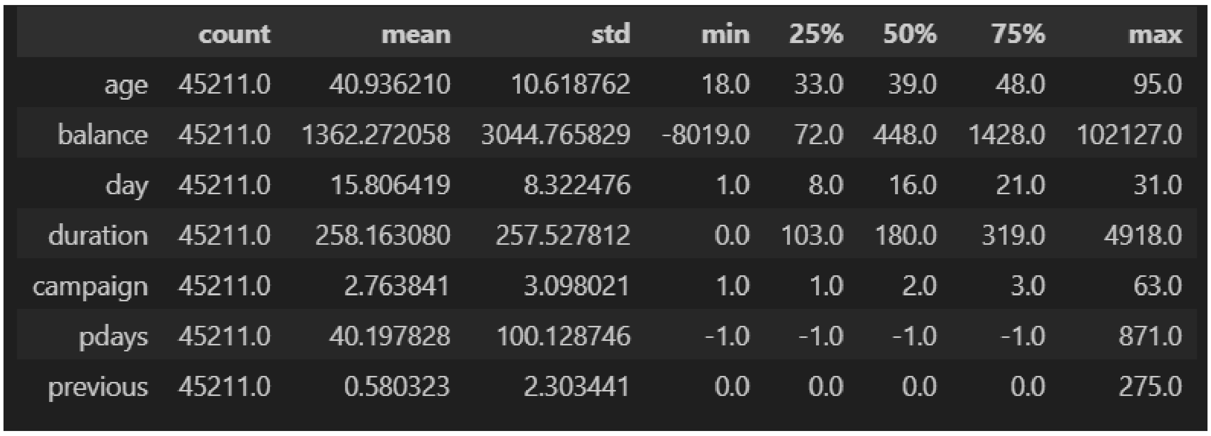
Figure 9 Dropping duplication Values

2.2 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is to understand the structure, quality, and relationships within a dataset. It helps identify patterns, detect anomalies, uncover data quality issues (e.g. outliers), and explore relationships between features and the target variable.

2.2.1 Statistical Summary for Numeric features

This step gives statistics like mean, median, standard deviation, and min/max values for numerical features. It helps in detecting outliers, extreme values, and understanding distributions.

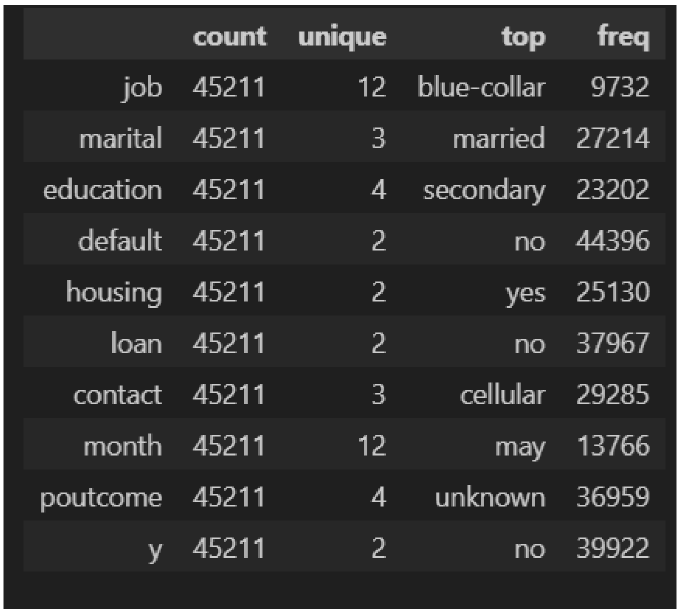


	count	mean	std	min	25%	50%	75%	max
age	45211.0	40.936210	10.618762	18.0	33.0	39.0	48.0	95.0
balance	45211.0	1362.272058	3044.765829	-8019.0	72.0	448.0	1428.0	102127.0
day	45211.0	15.806419	8.322476	1.0	8.0	16.0	21.0	31.0
duration	45211.0	258.163080	257.527812	0.0	103.0	180.0	319.0	4918.0
campaign	45211.0	2.763841	3.098021	1.0	1.0	2.0	3.0	63.0
pdays	45211.0	40.197828	100.128746	-1.0	-1.0	-1.0	-1.0	871.0
previous	45211.0	0.580323	2.303441	0.0	0.0	0.0	0.0	275.0

Figure 10 Statistical Summary for Numeric features

2.2.2 Summary statistics for categorical features

This will show the number of occurrences of each category within each categorical feature. It helps identify imbalances in the data and understand the spread of different categories.



	count	unique	top	freq
job	45211	12	blue-collar	9732
marital	45211	3	married	27214
education	45211	4	secondary	23202
default	45211	2	no	44396
housing	45211	2	yes	25130
loan	45211	2	no	37967
contact	45211	3	cellular	29285
month	45211	12	may	13766
poutcome	45211	4	unknown	36959
y	45211	2	no	39922

Figure 11 Summary statistics for categorical features

2.2.3 Visualizing distribution for Numerical Features

Histograms help visualize the distribution of numerical features. Can identify skewed distributions and outliers.

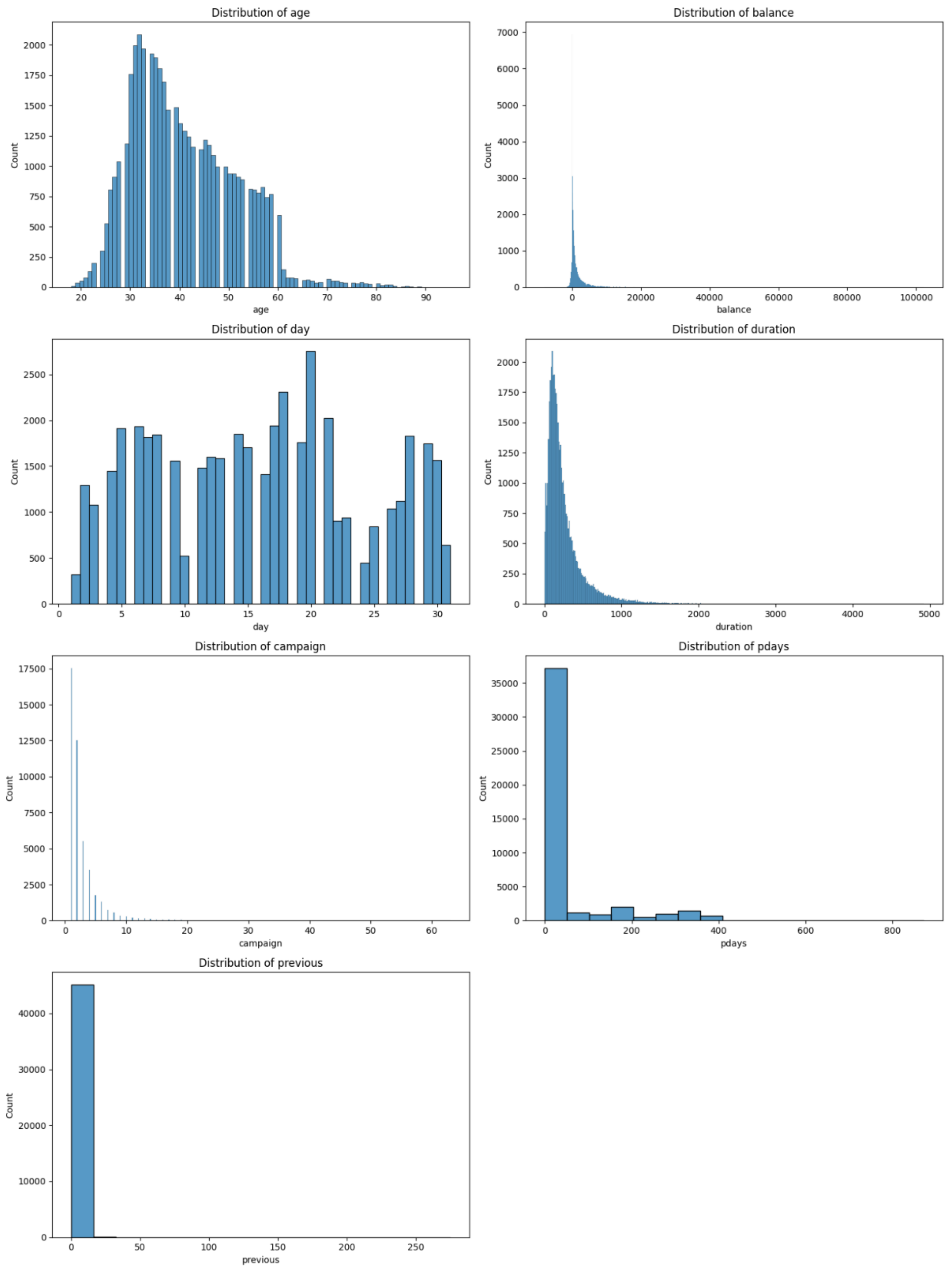


Figure 12 Visualizing distribution for Numerical Features

2.2.4 Distribution using Boxplot

Boxplots allow to visually identify Distribution and outliers in numerical features. Outliers may need to be handled by capping, transformation, or removal depending on their significance.

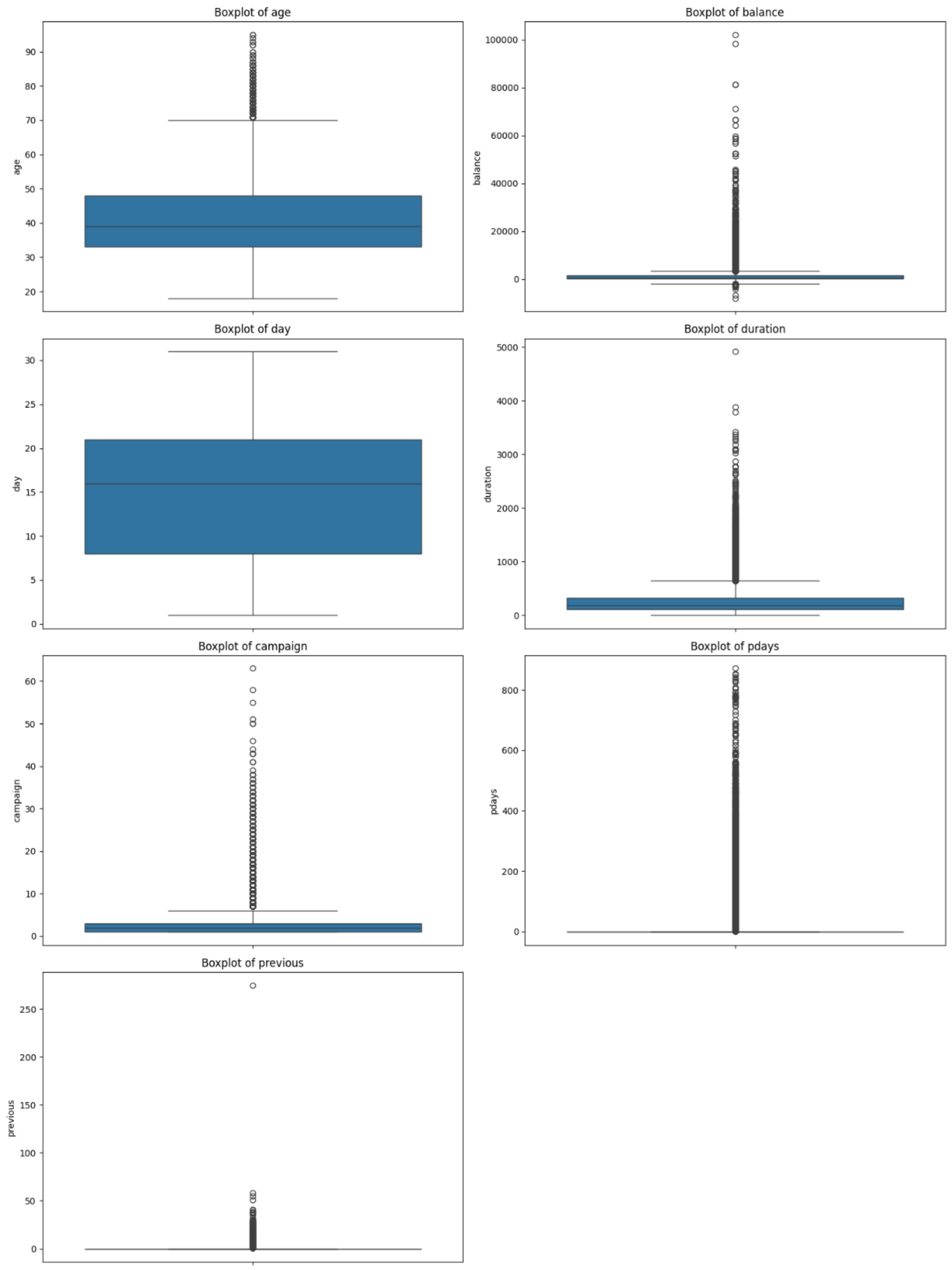


Figure 13 Distribution using Boxplot

2.2.5 Bivariate Analysis for Numerical Features

Boxplots help compare the distribution of a numerical feature across the categories of the target variable (y), allowing to see if there’s a difference between subscribers and non-subscribers

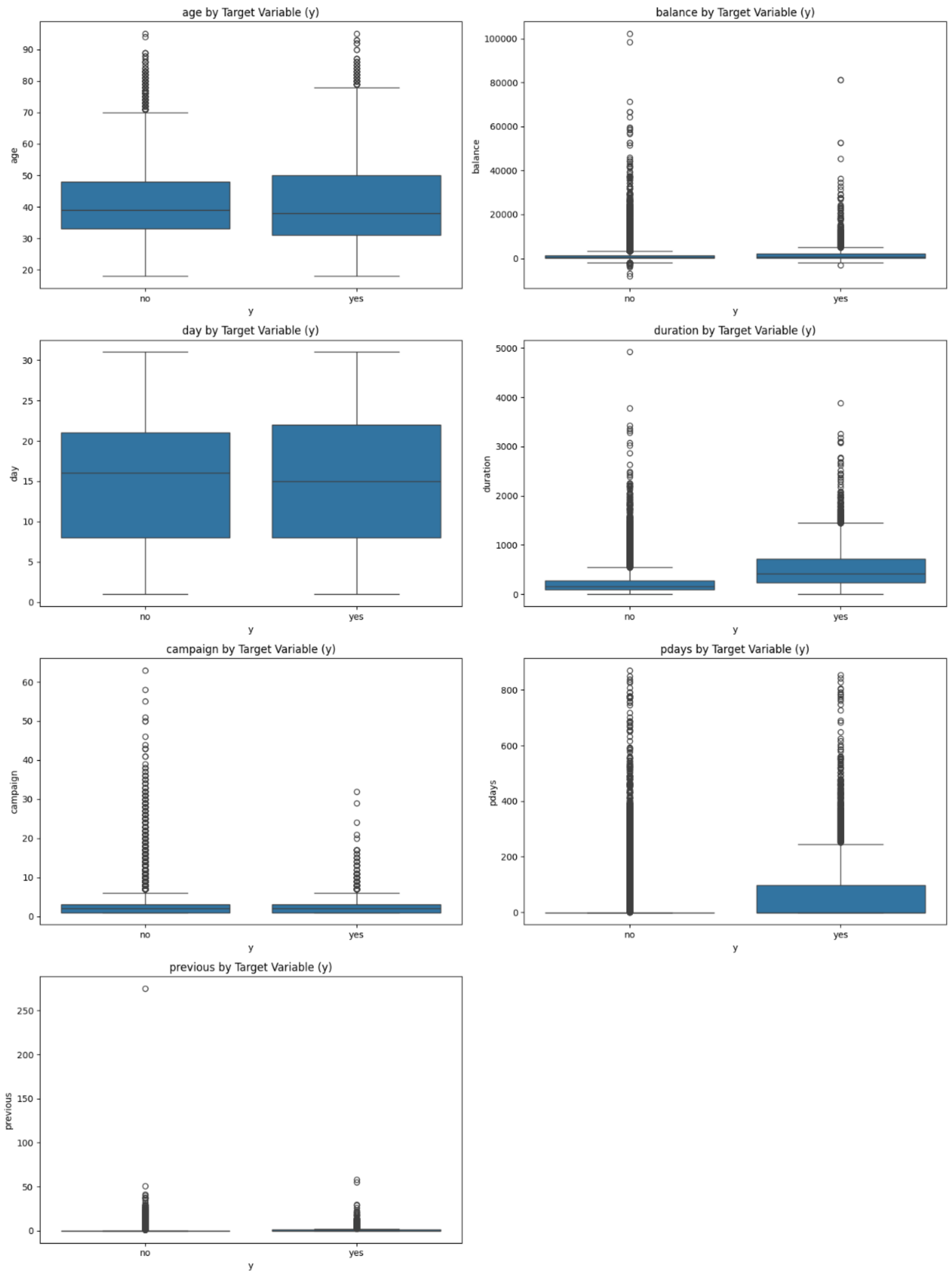


Figure 14 Bivariate Analysis for Numerical Features

2.2.6 Count plot for Categorical features

A count plot helps you quickly understand how many observations exist in each category of a feature.

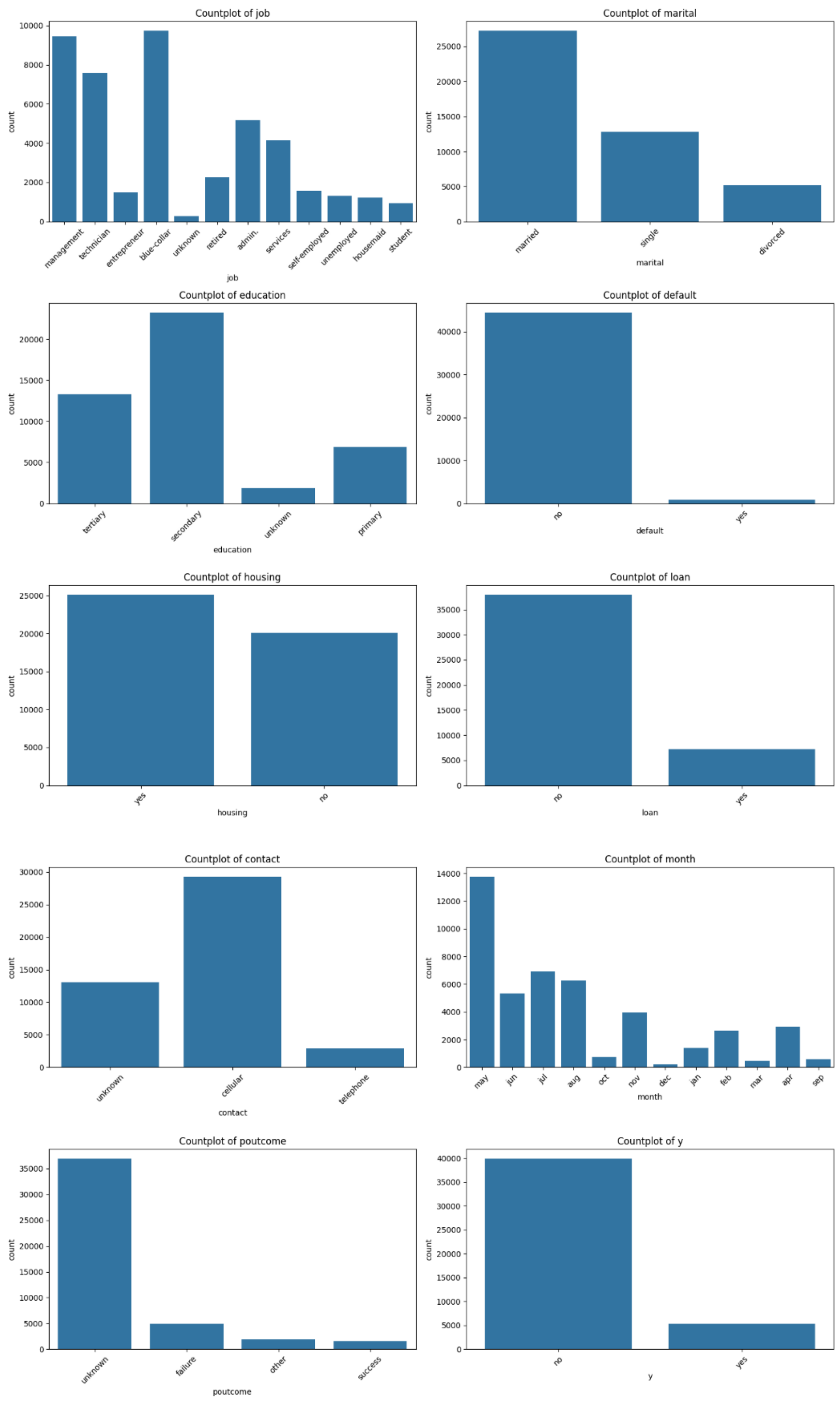


Figure 15 Count plot for Categorical features

1.2.6 Target variable analyzing Using Count plot

This help to understand the distribution of the Target variable.

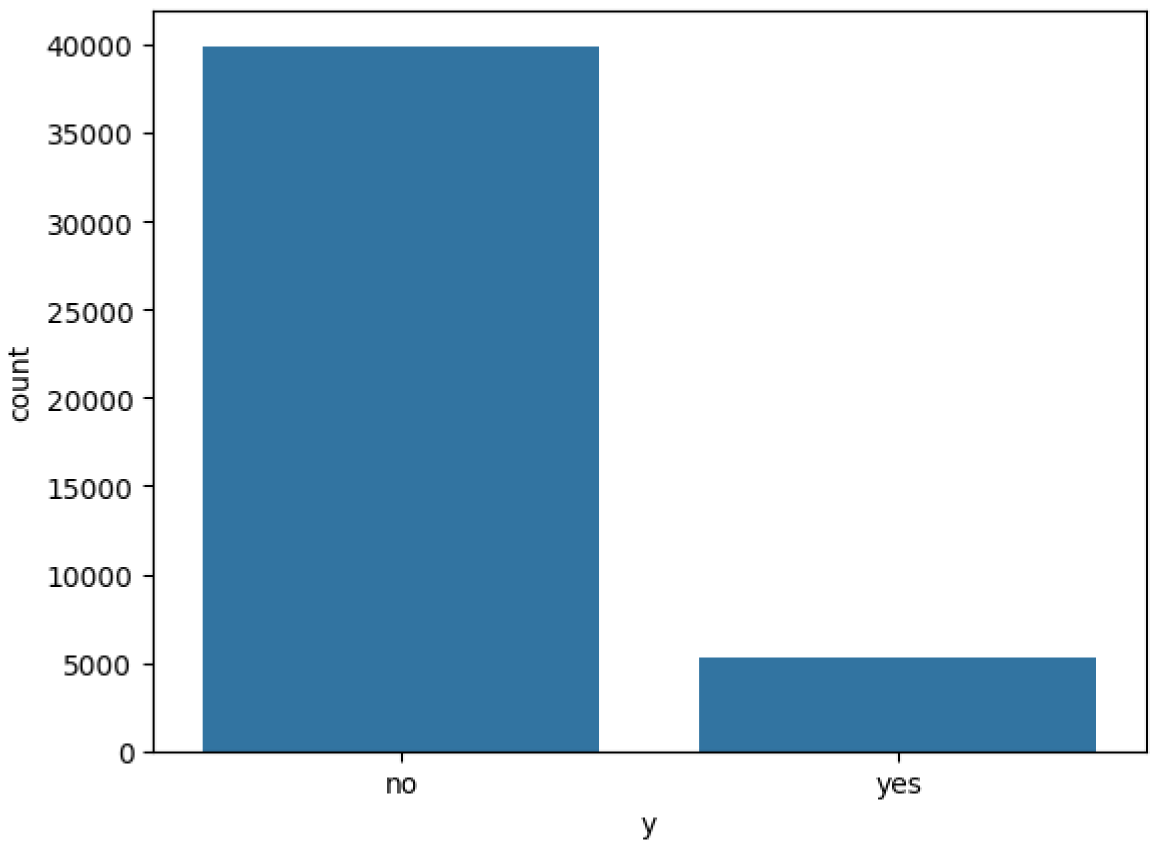


Figure 16 Target variable analyzing Using Count plot

2.2.7 Bivariate Analysis for Categorical features.

Count plots help visualize the distribution of categorical features across the categories of the target variable (y), allowing to see how different categories of a feature are associated with subscribers and non-subscribers.

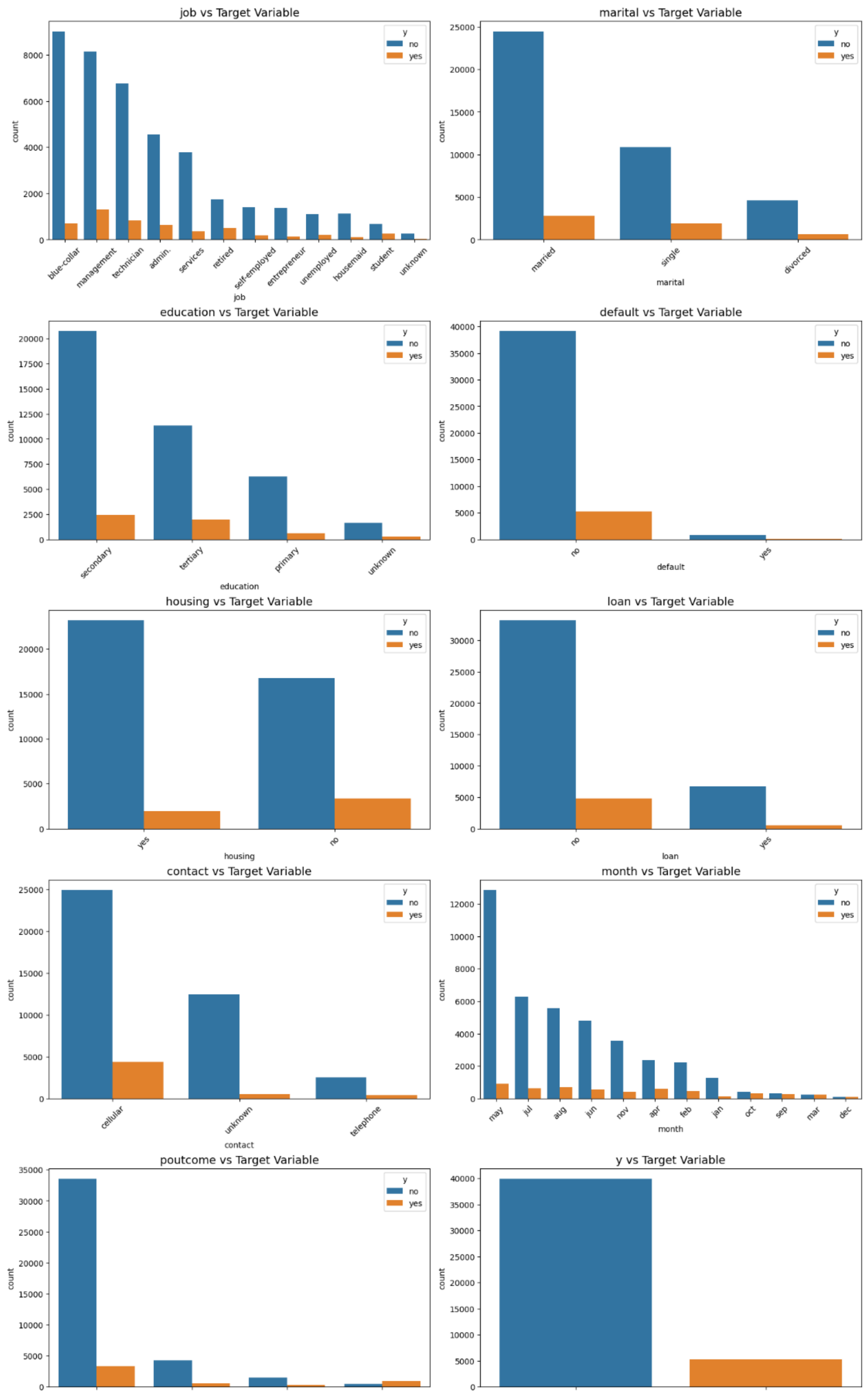
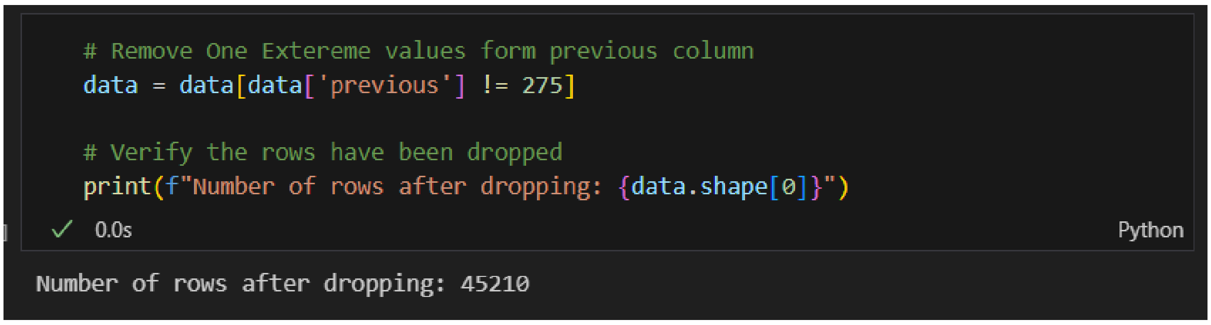


Figure 17 Bivariate Analysis for Categorical features.

2.3 Outliers Handling

Form the data that got from the EDA and manual searching in Excel there is one extreme value in previous column. The value is 275. Others are almost under 60. So here at first dropping the row from the dataset.



```python
# Remove One Extereme values form previous column
data = data[data['previous'] != 275]

# Verify the rows have been dropped
print(f"Number of rows after dropping: {data.shape[0]}")
```

```
Number of rows after dropping: 45210
```

Figure 18 Dropping Extreme value

After that checking the outliers using boxplot. Figure 13 from the EDA section shows the same Boxplot . the only different here is the previous feature.

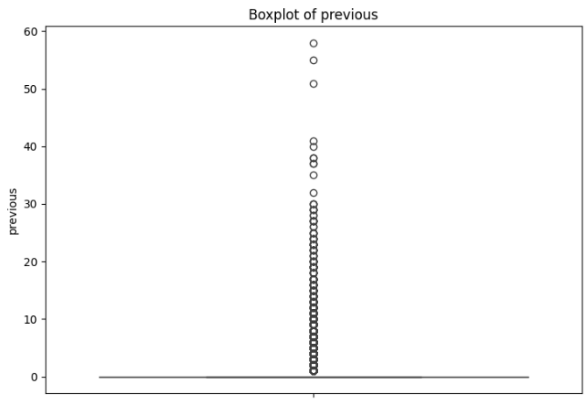


Figure 19 Boxplot After Dropping the Extreme value

After that handling outliers using capping for age and previous.

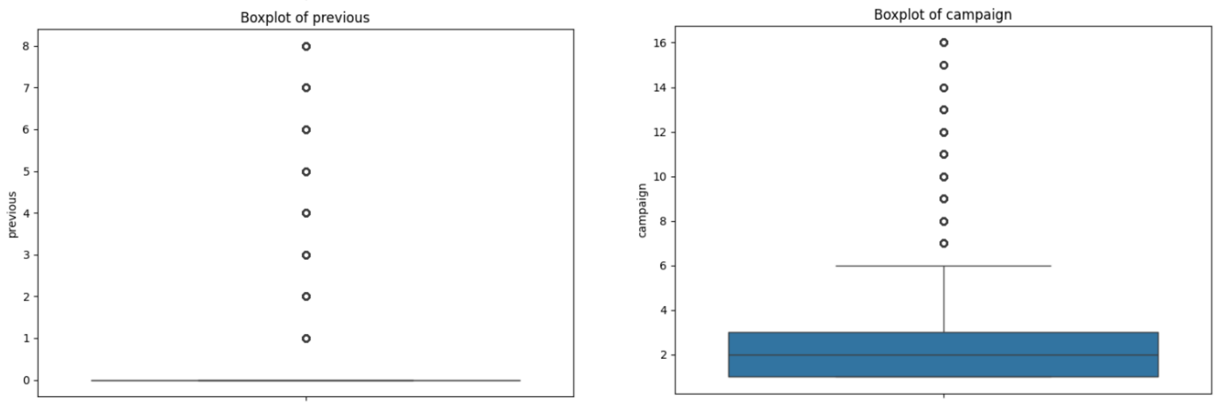
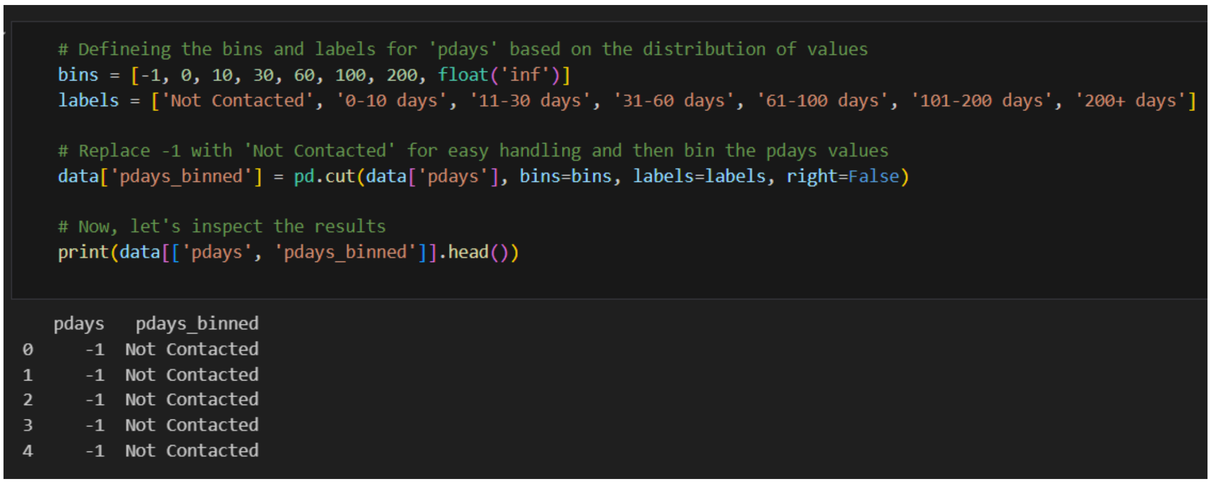


Figure 20 After Handling Outliers

Here age and the balance can have extreme values in the real-world scenario . So, there was no outliers handling used.

2.4 Feature Transformation

Here the ‘pday’ column contain values like ‘-1’. That means the ‘not previously contacted’ and most of the data are ‘-1’. So, cant drop or cap that data. For that using a bin and creating a new feature could be useful.

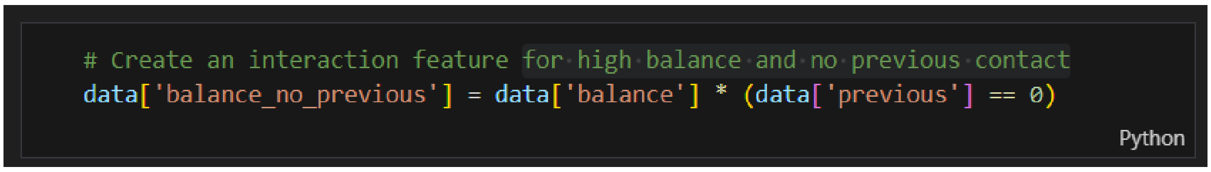


```
# Defineing the bins and labels for 'pdays' based on the distribution of values
bins = [-1, 0, 10, 30, 60, 100, 200, float('inf')]
labels = ['Not Contacted', '0-10 days', '11-30 days', '31-60 days', '61-100 days', '101-200 days', '200+ days']

# Replace -1 with 'Not Contacted' for easy handling and then bin the pdays values
data['pdays_binned'] = pd.cut(data['pdays'], bins=bins, labels=labels, right=False)

# Now, let's inspect the results
print(data[['pdays', 'pdays_binned']].head())
```

```
   pdays   pdays_binned
0     -1  Not Contacted
1     -1  Not Contacted
2     -1  Not Contacted
3     -1  Not Contacted
4     -1  Not Contacted
```

Figure 21 Transforming pdays

After that creating a new feature called ‘balance_no_previous’ to identify for high balance and no previous contact.

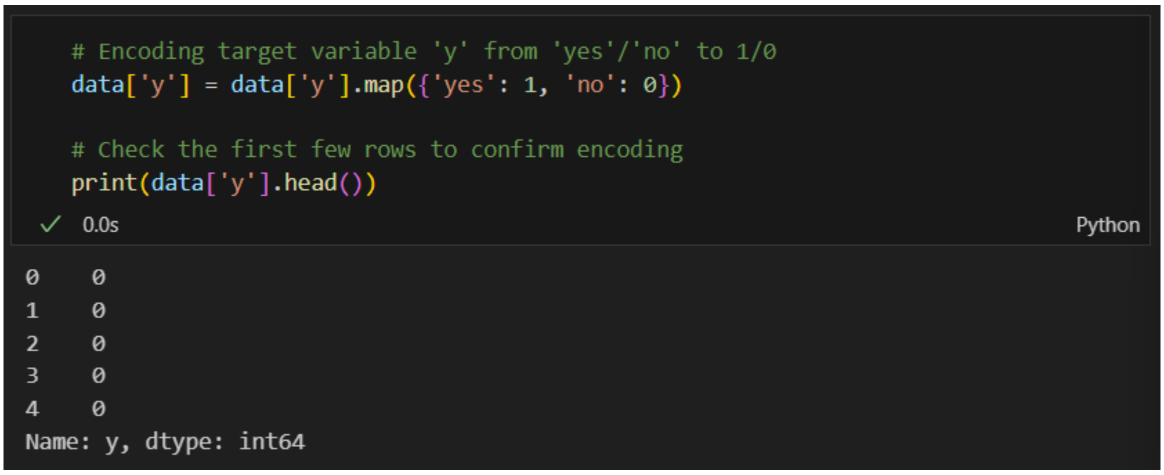


```
# Create an interaction feature for high balance and no previous contact
data['balance_no_previous'] = data['balance'] * (data['previous'] == 0)
```

Figure 22 New feature balance_no_previous’

2.5 Feature Selecting

For the feature selection using correlation to understand. For that ‘y’ has to be encoded first.



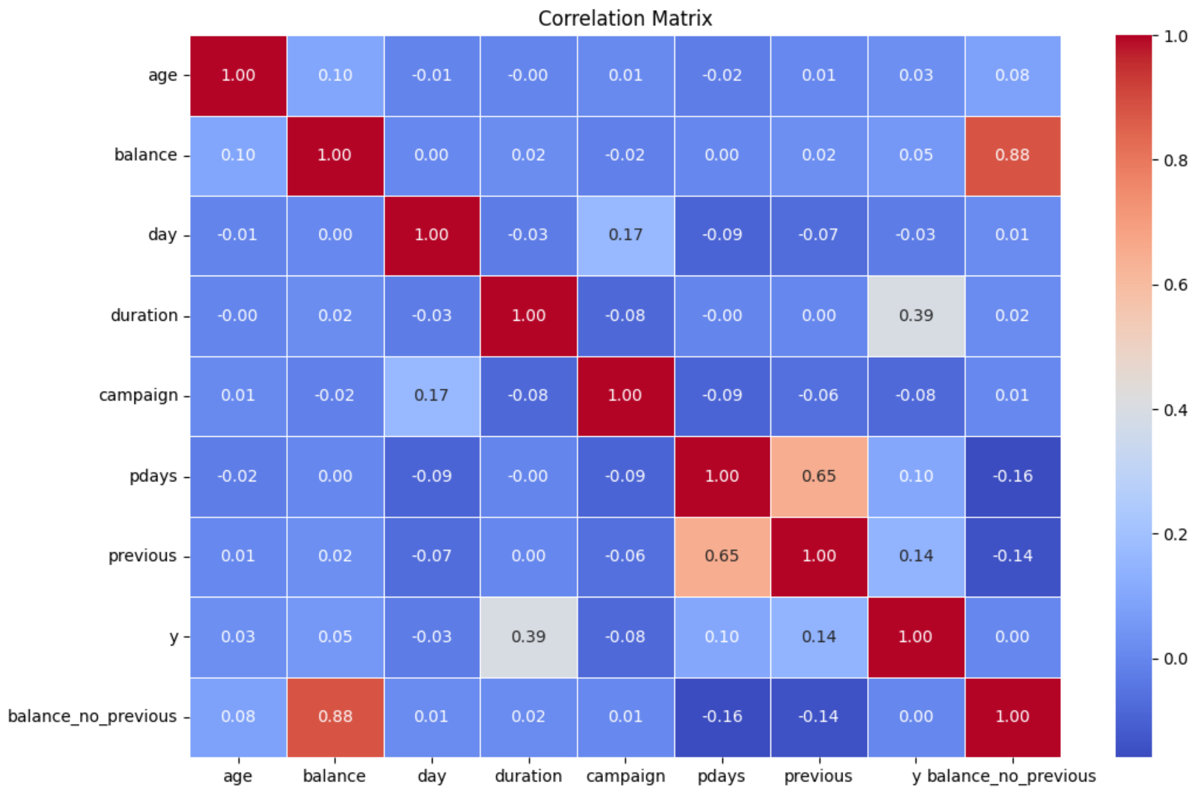
```
# Encoding target variable 'y' from 'yes'/'no' to 1/0
data['y'] = data['y'].map({'yes': 1, 'no': 0})

# Check the first few rows to confirm encoding
print(data['y'].head())
```

✓ 0.0s Python

```
0    0
1    0
2    0
3    0
4    0
Name: y, dtype: int64
```

Figure 23 Encoding Feature y

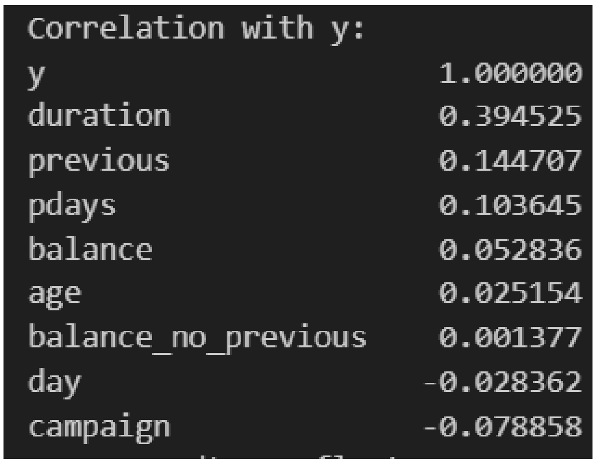
After that checking the correlation with heatmap.



	age	balance	day	duration	campaign	pdays	previous	y	balance_no_previous
age	1.00	0.10	-0.01	-0.00	0.01	-0.02	0.01	0.03	0.08
balance	0.10	1.00	0.00	0.02	-0.02	0.00	0.02	0.05	0.88
day	-0.01	0.00	1.00	-0.03	0.17	-0.09	-0.07	-0.03	0.01
duration	-0.00	0.02	-0.03	1.00	-0.08	-0.00	0.00	0.39	0.02
campaign	0.01	-0.02	0.17	-0.08	1.00	-0.09	-0.06	-0.08	0.01
pdays	-0.02	0.00	-0.09	-0.00	-0.09	1.00	0.65	0.10	-0.16
previous	0.01	0.02	-0.07	0.00	-0.06	0.65	1.00	0.14	-0.14
y	0.03	0.05	-0.03	0.39	-0.08	0.10	0.14	1.00	0.00
balance_no_previous	0.08	0.88	0.01	0.02	0.01	-0.16	-0.14	0.00	1.00

Figure 24 Correlation matrix

After that checking the correlation with y and other features



```
Correlation with y:
y                      1.000000
duration               0.394525
previous               0.144707
pdays                  0.103645
balance                0.052836
age                    0.025154
balance_no_previous    0.001377
day                   -0.028362
campaign              -0.078858
```

Figure 25 Correlation with y

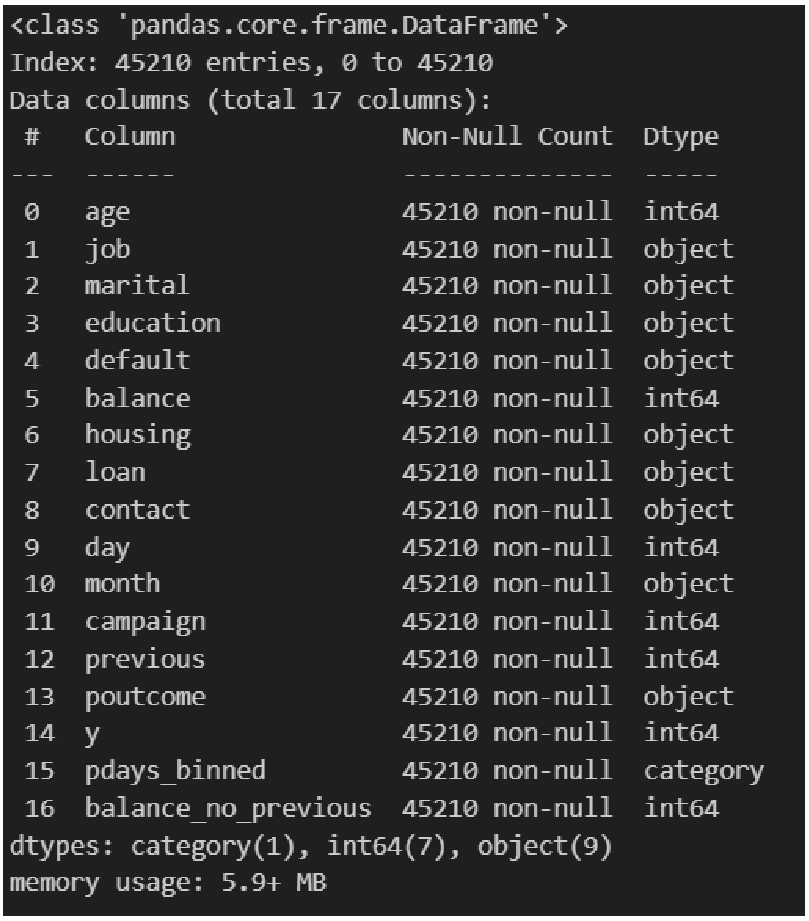
Here we can’t use Duration and it was mentioned in the website. Because of that and avid the data leakage dropping the ‘duration’ feature.

Also dropping the ‘pdays’ because of already created new feature for that.

Here all the other feature correlations are low. So, not selection any specific features. Selecting all the features.

2.6 Encoding the data

Here encoding used convert all the data into numerical values. At first check the data types and according to that doing the encodings here.



```
<class 'pandas.core.frame.DataFrame'>
Index: 45210 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  45210 non-null  int64
 1   job                  45210 non-null  object
 2   marital              45210 non-null  object
 3   education            45210 non-null  object
 4   default              45210 non-null  object
 5   balance              45210 non-null  int64
 6   housing              45210 non-null  object
 7   loan                 45210 non-null  object
 8   contact              45210 non-null  object
 9   day                  45210 non-null  int64
 10  month                45210 non-null  object
 11  campaign             45210 non-null  int64
 12  previous             45210 non-null  int64
 13  poutcome             45210 non-null  object
 14  y                    45210 non-null  int64
 15  pdays_binned         45210 non-null  category
 16  balance_no_previous  45210 non-null  int64
dtypes: category(1), int64(7), object(9)
memory usage: 5.9+ MB
```

Figure 26 After Encoding all the data

Using mapping for binary features like default ,housing, and loan .

Using One-Hot Encoding for categorical features like job, marital, education, contact, month, poutcome , and pdays_binned. Here using one-hot encoding because all the feature are nominal not ordinal .

After that aging checking fo the data types and set all the data types to ‘int64’ and saving the data as csv file named “processed_data_Encoding.csv”.

2.7 Normalizing the features.

Scaling ensures that numerical features such as age, balance, campaign, previous and balance_no_previous, are on a similar scale. This is done using StandardScaler, which standardizes features by removing the mean and scaling to unit variance. Then after using Scaler saving the dataset as csv file named “processed_data_Scaling.csv” .

2.8 Dimensionality reduction

Principal Component Analysis (PCA) was applied to reduce the number of features while retaining most of the variance in the data. Same like the Normalization using PCA for only the numerical vales and then saving the dataset as csv file named “processed_data_PCA.csv” .

3.0 Solution Methodology

In this project, we set out to predict whether a client would subscribe to a term deposit based on various features from the bank marketing dataset. To tackle this classification problem, we selected two powerful machine learning models that could handle complex patterns and the imbalance in the data:

3.1 Model Selection

3.1.1 Neural Network:

We opted for a deep learning model using the Keras library with TensorFlow as the backend. The architecture of this model includes:

- Input Layer: The number of neurons in this layer corresponds to the number of features in the dataset.
- Hidden Layers: Multiple hidden layers with ReLU (Rectified Linear Unit) activation functions were added to allow the model to capture the intricate relationships and patterns in the data.
- Output Layer: The output layer uses a sigmoid activation function, which is ideal for binary classification problems, as it outputs probabilities that the client will subscribe to a term deposit or not.

The neural network was trained over 50 epochs to adjust its weights, learning the underlying patterns in the data.

3.1.2 Random Forest Classifier:

We also used Random Forest model to improve prediction accuracy. This model aggregates the results of multiple decision trees, each trained on different subsets of the data. We implemented it using the Scikit-learn library and tuned several hyperparameters to enhance performance:

- n_estimators: We set this parameter to 100-300, meaning the model would use 100-300 decision trees.
- max_depth: This controlled the maximum depth of each tree, preventing them from growing too complex and overfitting the data.
- min_samples_split and min_samples_leaf: These parameters were fine-tuned to avoid

overfitting by requiring a minimum number of samples to split a node and to ensure leaf nodes have sufficient data.

3.2 Model Training and Tuning

Both models were trained on the preprocessed dataset(with and without SMOTE), which had been scaled, encoded, and transformed. Given the class imbalance, we applied SMOTE (Synthetic Minority Over-sampling Technique) to create synthetic samples for the minority class, ensuring that the models could learn from a more balanced dataset.

The training process for each model involved the following:

- **Neural Network**: The number of batch size was key parameters to adjust during training. A validation split of 20% was used to monitor performance and avoid overfitting.
- **Random Forest**: We performed hyperparameter tuning using grid search and cross-validation to find the best combination of parameters, including the number of trees (n_estimators), the maximum depth of the trees (max_depth), and the minimum number of samples required to split nodes and form leaves. The aim was to balance model complexity with the ability to generalize to new data.

4.0 Model Evaluation Criteria

To assess how well our Neural Network and Random Forest models performed, we used several key metrics:

1. **Accuracy**: This is the percentage of correct predictions the model made. While higher accuracy is generally better, it may not always be reliable in cases where the data is imbalanced, like ours.
2. **Classification Report**: This report gives a detailed look at the model's performance by showing:
    - **Precision**: How many of the positive predictions were actually correct.
    - **Recall**: How many of the actual positives were correctly identified by the model.
    - **F1-Score**: A balance between Precision and Recall, useful when there’s a need to account for both false positives and false negatives.
3. **ROC Curve and AUC**: The ROC curve shows how well the model can distinguish between the two classes (subscribed vs. not subscribed). The **AUC** (Area Under the Curve) score tells us how good the model is at making those distinctions, with a higher AUC indicating better performance.

5.0 Experimental Results

The following tables summarize the performance of the different Random Forest (RF) and Neural Network (NN) models. Each model's performance is evaluated based on accuracy, precision, recall, F1-score, and AUC.

5.1 Random Forest Models

Evolution Metrics	RF default model	RF Model Tunned	RF model with Smote and best hyperparameter
Training Accuracy	1.0	0.9294	0.9453
Testing Accuracy	0.8947	0.8761	0.9164
Precision (Class 0)	0.90	0.93	0.90
Precision (Class 1)	0.72	0.49	0.94
Recall (Class 0)	0.99	0.93	0.94
Recall (Class 1)	0.21	0.51	0.90
F1-Score (Class 0)	0.94	0.93	0.92
F1-Score(Class 1)	0.33	0.50	0.92
AUC	0.7929	0.7950	0.9670

Table 1 Random Forest Models comparition

Conclusion : The SMOTE-enhanced RF model with hyperparameter tuning outperforms the default and tuned models, particularly in precision, recall, and AUC for predicting Class 1.

5.2 Neural Network Models

Evolution Metrics	NN default model	NN model tunned	NN model with SMOTE	NN model with SMOTE and Tunned
Training Accuracy		0.9012		0.9326
Testing Accuracy	0.8871	0.8930	0.9138	0.9162
Precision (Class 0)	0.91	0.90	0.88	0.88
Precision (Class 1)	0.57	0.66	0.95	0.95
Recall (Class 0)	0.97	0.98	0.96	0.96
Recall (Class 1)	0.28	0.23	0.87	0.88
F1-Score (Class 0)	0.94	0.94	0.92	0.92
F1-Score(Class 1)	0.37	0.34	0.91	0.91
AUC	0.77	0.80	0.96	0.96

Table 2 Neural Network Models comparition

Conclusion : The NN model with SMOTE and tuning performs the best, achieving the highest testing accuracy, precision, recall, F1-score, and AUC, especially for Class 1 predictions.

5.3 Model Comparison with best models(NN and RF)

Evolution Metrics	**Best RF model**	**Best NN Model**
Training Accuracy	0.9453	0.9326
Testing Accuracy	0.9164	0.9162
Precision (Class 0)	0.90	0.88
Precision (Class 1)	0.94	0.95
Recall (Class 0)	0.94	0.96
Recall (Class 1)	0.90	0.88
F1-Score (Class 0)	0.92	0.92
F1-Score(Class 1)	0.92	0.91
AUC	0.96	0.96

Table 3 Comparition between best RF and NN models

Conclusion : Both the Random Forest and Neural Network models perform similarly, with high accuracy and AUC. The RF model slightly excels in Class 0 metrics, while the NN model performs better in Class 1 precision.

5.5 Summary

Both the best Random Forest and best Neural Network models perform similarly, with testing accuracy, F1-score, and AUC being almost identical. The primary difference lies in the recall for Class 0, where the NN model slightly outperforms the RF model.

6.0 Discussion

6.1 Limitations

1. **Imbalanced Data:** Despite using techniques like SMOTE, the dataset's class imbalance may still affect model performance, particularly for the minority class.
2. **Model Interpretability:** Both Random Forest and Neural Networks are complex models, making it difficult to understand how specific features influence predictions.
3. **Risk of Overfitting:** The Random Forest model showed high training accuracy, which raises concerns about overfitting and generalization to unseen data.
4. **Computational Cost:** Hyperparameter tuning for Neural Networks can be time-consuming and resource-intensive.

6.2 Enhancements

1. **Better Handling of Imbalanced Data:** Trying other techniques like adjusting the decision threshold could improve performance, especially for the minority class.
2. **Advanced Feature Engineering:** Exploring more complex feature transformations or domain-specific features could help capture hidden patterns.
3. **Model Interpretability:** Using tools like SHAP could make the models more interpretable and trustworthy.
4. **Ensemble Methods:** Combining the strengths of both models through ensemble methods could enhance accuracy.
5. **Cross-Validation:** Using k-fold cross-validation would give a better understanding of how the models perform on different data subsets.
6. **External Data:** Adding external data could provide more context and potentially improve model predictions.

7.0 GitHub Link

The GitHub link of the project is given below.

https://github.com/WasifAsi/Mechine-Learning-Course-work.git

8.0 Appendix

8.1 Data processing

```
# Data Preprocessing
# Import neccesary libarys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Reading the DataSet
# Load the dataset
data = pd.read_csv('bank-full.csv' , sep=';')

# View the first few rows
data.head()

#view the last few rows
data.tail()

# Data Cleanning
## Sanity check data
# Check the shape of the data
# cloums and rows
data.shape

# Check column information
data.info()

#check for duplicates
data.duplicated().sum()

#identify the unique values and garbage values
for i in data.select_dtypes(include='object').columns:
    print(f'\nUnique values in {i} are {data[i].unique()}')

## Missing Value Treatment

# Check for missing values
print(data.isnull().sum())

# Handle missing values
for col in data.columns:
    if data[col].isnull().sum() > 0:
        # For categorical columns
        if data[col].dtype == 'object':
            data[col].fillna(data[col].mode()[0], inplace=True)
        else:
            data[col].fillna(data[col].median(), inplace=True)  # Fill numerical with median

print("Missing values treated.")
```

```
data.isnull().sum()
```

## **Duplicate value treatment**

```
# Check for duplicates
duplicates = data.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")

# Drop duplicates
data = data.drop_duplicates()

# Saving the cleanned data
data.to_csv('bank-full_tabled.csv', index=False)
```

# **Exploratory Data Analysis (EDA)**

```
# Summary statistics for Numeric features
data.describe().T


# Summary statistics for categorical features
data.describe(include='object').T
```

## **Vizualization for all the numerical columns**

```
# Define the numerical columns
numerical_columns = data.select_dtypes(include='number').columns

# Determine the number of rows and columns for subplots
n_cols = 2  # Number of columns
n_rows = (len(numerical_columns) + n_cols - 1) // n_cols  # Calculate rows dynamically
```

### **Distribution Histogram**

```
# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))  # Adjust size for better
visibility

# Flatten axes array for easier indexing
axes = axes.flatten()

# Plot each histogram
for i, col in enumerate(numerical_columns):
    sns.histplot(data=data, x=col, ax=axes[i])
    axes[i].set_title(f"Distribution of {col}")

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Adjust layout
plt.tight_layout()
plt.show()
```

### **Distribution using Boxplot**

```
# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))  # Adjust size for better
visibility

# Flatten axes array for easier indexing
axes = axes.flatten()

# Plot each boxplot
for i, col in enumerate(numerical_columns):
    sns.boxplot(data=data, y=col, ax=axes[i])  # Use 'y' for vertical boxplots
    axes[i].set_title(f"Boxplot of {col}")

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Adjust layout
plt.tight_layout()
plt.show()
```

### **Bivariate Analysis**

```
# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))  # Adjust size for better
visibility

# Flatten axes array for easier indexing
axes = axes.flatten()

for i, col in enumerate(numerical_columns):
    sns.boxplot(x='y', y=col, data=data, ax=axes[i])
    axes[i].set_title(f"{col} by Target Variable (y)")


# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Show the figure for numerical columns
plt.tight_layout()
plt.show()
```

## **Vizualization for all the categorical columns**

```
# Define the categorical columns
categorical_columns = data.select_dtypes(include='object').columns

# Determine the number of rows and columns for subplots
n_cols = 2  # Number of columns
n_rows = (len(categorical_columns) + n_cols - 1) // n_cols  # Calculate rows dynamically
```

### **Countplot**

```
# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))
```

```
# Flatten axes array for easier indexing
axes = axes.flatten()

# Plot each barplot
for i, col in enumerate(categorical_columns):
    sns.countplot(data=data, x=col, ax=axes[i])
    axes[i].set_title(f"Countplot of {col}")
    axes[i].tick_params(axis='x', rotation=45)

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Adjust layout
plt.tight_layout()
plt.show()
```

### Target Variable Analysis

```
print(data['y'].value_counts(normalize=True))
sns.countplot(x='y', data=data)
plt.show()
```

### Bivariate Analysis

```
# Plot for Categorical Columns
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows), constrained_layout=True)  #
Adjusted figsize

axes = axes.flatten()

for i, col in enumerate(categorical_columns):
    sns.countplot(x=col, hue='y', data=data, order=data[col].value_counts().index, ax=axes[i])
    axes[i].set_title(f"{col} vs Target Variable", fontsize=14)
    axes[i].tick_params(axis='x', rotation=45, labelsize=10)
    axes[i].tick_params(axis='y', labelsize=10)

# Show the figure for categorical columns
plt.show()
```

# Outlier Treatment

```
# Remove One Extereme values form previous column
data = data[data['previous'] != 275]

# Verify the rows have been dropped
print(f"Number of rows after dropping: {data.shape[0]}")
```

### Checking the outlayers using boxplot

```
# Define the numerical columns
numerical_columns = data.select_dtypes(include='number').columns

# Determine the number of rows and columns for subplots
n_cols = 2  # Number of columns
n_rows = (len(numerical_columns) + n_cols - 1) // n_cols  # Calculate rows dynamically
```

```
# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))  # Adjust size for better
visibility

# Flatten axes array for easier indexing
axes = axes.flatten()

# Plot each boxplot
for i, col in enumerate(numerical_columns):
    sns.boxplot(data=data, y=col, ax=axes[i])  # Use 'y' for vertical boxplots
    axes[i].set_title(f"Boxplot of {col}")

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Adjust layout
plt.tight_layout()
plt.show()
```

### **Handling the outlayers**

```
# Cap 'campaign' at 99th percentile
cap_campaign = data['campaign'].quantile(0.99)
data['campaign'] = data['campaign'].clip(upper=cap_campaign)

# Cap 'previous' at 99th percentile
cap_previous = data['previous'].quantile(0.99)
data['previous'] = data['previous'].clip(upper=cap_previous)
```

### **Again Checking the outlayers using boxplot**

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

# Define the numerical columns
numerical_columns = data.select_dtypes(include='number').columns

# Determine the number of rows and columns for subplots
n_cols = 2  # Number of columns
n_rows = (len(numerical_columns) + n_cols - 1) // n_cols  # Calculate rows dynamically

# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows))  # Adjust size for better
visibility

# Flatten axes array for easier indexing
axes = axes.flatten()

# Plot each boxplot
for i, col in enumerate(numerical_columns):
    sns.boxplot(data=data, y=col, ax=axes[i])  # Use 'y' for vertical boxplots
    axes[i].set_title(f"Boxplot of {col}")

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')
```

```
# Adjust layout
plt.tight_layout()
plt.show()
```

# Feature Engineering
## Feature transformation
##### new changed pdays

```
# Defineing the bins and labels for 'pdays' based on the distribution of values
bins = [-1, 0, 10, 30, 60, 100, 200, float('inf')]
labels = ['Not Contacted', '0-10 days', '11-30 days', '31-60 days', '61-100 days', '101-200
days', '200+ days']

# Replace -1 with 'Not Contacted' for easy handling and then bin the pdays values
data['pdays_binned'] = pd.cut(data['pdays'], bins=bins, labels=labels, right=False)

# Now, let's inspect the results
print(data[['pdays', 'pdays_binned']].head())

# Create an interaction feature for high balance and no previous contact
data['balance_no_previous'] = data['balance'] * (data['previous'] == 0)
```

## Feature selecting

```
# Encoding target variable 'y' from 'yes'/'no' to 1/0
data['y'] = data['y'].map({'yes': 1, 'no': 0})

# Check the first few rows to confirm encoding
print(data['y'].head())

data.to_csv('ycheck.csv', index=False)
```

### correlation

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

# Select only numerical columns
numerical_data = data.select_dtypes(include=['number'])

# Calculate the correlation matrix for numerical columns
correlation_matrix = numerical_data.corr()

# Create a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()


correlation_with_y = correlation_matrix['y'].sort_values(ascending=False)

print("Correlation with y:")
print(correlation_with_y)
```

```
#### Drop unnecessary features
# Drop 'duration' feature to avoid leakage
data = data.drop(['duration'], axis=1)

# Drop the original 'pdays' feature
data = data.drop(columns=['pdays'])

## Encoding the data
data.info()


# Binary Features
binary_columns = ['default', 'housing', 'loan']

for col in binary_columns:
    data[col] = data[col].map({'yes': 1, 'no': 0})


# Categorical Features
categorical_columns = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']

data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)


# Apply One-Hot Encoding to the binned 'pdays' feature
data = pd.get_dummies(data, columns=['pdays_binned'])

# Check the result
print(data[['pdays_binned_Not Contacted', 'pdays_binned_0-10 days', 'pdays_binned_11-30
days']].head())

data.info()


data = data.astype('int64')


data.to_csv('processed_data_Encoding.csv', index=False)
print("Dataset with Scaling saved as 'processed_data_Encoding.csv'")


#  Normalize/Scale Numerical Features
### StandardScaler
from sklearn.preprocessing import StandardScaler

numerical_cols = ['age', 'balance', 'campaign', 'previous','balance_no_previous']

# scaling only to the specified numerical columns
scaler = StandardScaler()
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

# # Save the processed DataFrame to a CSV file
data.to_csv("processed_data_Scaling.csv", index=False)
print("Dataset with Scaling saved as 'processed_data_Scaling.csv'")
```

```
# Dimensionality reduction

### PCA
from sklearn.decomposition import PCA

pca = PCA(n_components=0.95)  # Retain 95% of the variance
principal_components = pca.fit_transform(data[numerical_cols])

# Create a DataFrame with the principal components
pca_columns = [f"PC{i+1}" for i in range(principal_components.shape[1])]
pca_df = pd.DataFrame(principal_components, columns=pca_columns)

data = pd.concat([data, pca_df], axis=1)

# Checking explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f"Explained variance by each component: {explained_variance}")
print(f"Total variance explained: {explained_variance.sum()}")

data = data.dropna()

data.head()

# Save the DataFrame with PCA components to a CSV file
data.to_csv("processed_data_PCA.csv", index=False)
print("Dataset with PCA components saved as 'processed_data_PCA.csv'")
```

8.2 Best Neural Network Model source code

```
# Librays
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.decomposition import PCA
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, roc_curve, auc
import matplotlib.pyplot as plt

from kerastuner import HyperModel
from kerastuner.tuners import RandomSearch
from tensorflow.keras import backend as K

# Loading the data
data = pd.read_csv('processed_data_Encoding.csv')

X = data.drop('y', axis=1)
y = data['y']

# Apply SMOTE to the training data (this only applies to the training set)
smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X, y)

from sklearn.preprocessing import StandardScaler

numerical_cols = ['age', 'balance', 'campaign', 'previous','balance_no_previous']
```

```
# scaling only to the specified numerical columns
scaler = StandardScaler()
X[numerical_cols] = scaler.fit_transform(X[numerical_cols])

from sklearn.decomposition import PCA

pca = PCA(n_components=0.95)  # Retain 95% of the variance
principal_components = pca.fit_transform(X[numerical_cols])

# Create a DataFrame with the principal components
pca_columns = [f"PC{i+1}" for i in range(principal_components.shape[1])]
pca_df = pd.DataFrame(principal_components, columns=pca_columns)

X = pd.concat([X, pca_df], axis=1)

# Split into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

# Model Trainning

```
# Build the model
model = Sequential()

# Input layer: Define the number of input features (adjust based on your dataset)
model.add(Dense(128, activation='relu', input_dim=X_train.shape[1]))  # X_train should be
your feature matrix
model.add(Dropout(0.3))  # Regularization using dropout

# Hidden layers
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.3))
```

```
# Output layer: Sigmoid for binary classification
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
        loss='binary_crossentropy',
        metrics=['accuracy'])

# Print model summary
model.summary()

# Training the model (adjust the epochs and batch size based on your dataset)
history = model.fit(X_train, y_train, epochs=50, batch_size=32,  validation_split=0.2)

# Evaluate the model on test data (X_test and y_test)
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy}")

from sklearn.metrics import classification_report

# Predict the labels for the test set
y_pred = (model.predict(X_test) > 0.5).astype("int32")  # Convert probabilities to binary
labels

# Print classification report
print(classification_report(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Predict probabilities for the test set
y_prob = model.predict(X_test)

# Compute ROC curve and AUC
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# Print the AUC score
print(f"AUC: {roc_auc:.2f}")

# Tunning
# Clear any previous TensorFlow session
K.clear_session()

# Define the HyperModel class
class MyHyperModel(HyperModel):
    def build(self, hp):
        model = Sequential()

        # First hidden layer with hyperparameters for number of units and dropout rate
        model.add(Dense(units=hp.Int('units_1', min_value=64, max_value=256, step=32),
                        activation='relu', input_dim=X_train.shape[1]))
        model.add(Dropout(rate=hp.Float('dropout_1', min_value=0.2, max_value=0.5,
step=0.1)))

        # Second hidden layer with hyperparameters for number of units and dropout rate
        model.add(Dense(units=hp.Int('units_2', min_value=64, max_value=256, step=32),
                        activation='relu'))
    model.add(Dropout(rate=hp.Float('dropout_2', min_value=0.2, max_value=0.5,
step=0.1)))

    # Output layer
    model.add(Dense(1, activation='sigmoid'))

    # Hyperparameter for learning rate
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', min_value=1e-
5, max_value=1e-2, sampling='LOG')),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model

# Initialize the Keras Tuner RandomSearch
tuner = RandomSearch(
  MyHyperModel(),
  objective='val_accuracy',  # Optimize for validation accuracy
  max_trials=10,             # Number of different models to try
  executions_per_trial=3,    # Number of times to train each model
  directory='smote_tuner_results', # Directory to store the results
  project_name='banking_model_tuning'
)

# Perform the hyperparameter search
tuner.search(X_train, y_train,
             epochs=50,
             batch_size=32,
             validation_split=0.2)

# Get the best hyperparameters
best_hyperparameters = tuner.get_best_hyperparameters(1)[0]
print(f"Best hyperparameters: {best_hyperparameters.values}")
```

```
# Manually rebuild the best model
best_model = Sequential()
best_model.add(Dense(units=best_hyperparameters.get('units_1'),
                    activation='relu',
                    input_dim=X_train.shape[1]))
best_model.add(Dropout(rate=best_hyperparameters.get('dropout_1')))
best_model.add(Dense(units=best_hyperparameters.get('units_2'),
                    activation='relu'))
best_model.add(Dropout(rate=best_hyperparameters.get('dropout_2')))
best_model.add(Dense(1, activation='sigmoid'))

best_model.compile(optimizer=Adam(learning_rate=best_hyperparameters.get('learning_rate
')),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

# Train the manually rebuilt best model
history = best_model.fit(X_train, y_train,
                        epochs=50,
                        batch_size=32,
                       validation_split=0.2)

# Evaluate the best model on the test data
test_loss, test_accuracy = best_model.evaluate(X_test, y_test)
print(f"Test accuracy of the best model: {test_accuracy}")

# Get the history of the best model's training
history = best_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Get training accuracy from the history object
print("Training accuracy:", history.history['accuracy'][-1])  # Accuracy of the last epoch

# Evaluate the model on the test data (X_test and y_test)
test_loss, test_accuracy = best_model.evaluate(X_test, y_test)
```

```
print(f"Test accuracy: {test_accuracy}")

# Predict the labels for the test set
y_pred = (best_model.predict(X_test) > 0.5).astype("int32")  # Convert probabilities to
binary labels

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Predict probabilities for the test set (for ROC curve)
y_prob = best_model.predict(X_test)

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# Print the AUC score
print(f"AUC: {roc_auc:.2f}")

# You can also print the best hyperparameters found
best_hyperparameters = tuner.get_best_hyperparameters(1)[0]
print(f"Best hyperparameters: {best_hyperparameters.values}")
```

8.2 Best Random Forest Model source code

```
# Import Library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

# Load Data and split
# Load the dataset
data = pd.read_csv('processed_data_Encoding.csv')

# Separate features and target
X = data.drop('y', axis=1)  # Features
y = data['y']  # Target

# Apply SMOTE
smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X,y)

# Specify numerical columns for scaling and PCA
numerical_cols = ['age', 'balance', 'campaign', 'previous','balance_no_previous']

# scaling only to the specified numerical columns
scaler = StandardScaler()
X[numerical_cols] = scaler.fit_transform(X[numerical_cols])
```

```
pca = PCA(n_components=0.95)  # Retain 95% of the variance
principal_components = pca.fit_transform(X[numerical_cols])

# Create a DataFrame with the principal components
pca_columns = [f"PC{i+1}" for i in range(principal_components.shape[1])]
pca_df = pd.DataFrame(principal_components, columns=pca_columns)

X = pd.concat([X, pca_df], axis=1)

# Split into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

# Model Trainning

```
# Initialize the model
rf_model = RandomForestClassifier(
    n_estimators=300,
    max_depth=20,
    min_samples_split=10,
    min_samples_leaf=4,
    class_weight='balanced',
    random_state=42
)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions
rf_y_pred_train = rf_model.predict(X_train)
rf_y_pred_test = rf_model.predict(X_test)

# Evaluate
print("Random Forest - Training Accuracy:", accuracy_score(y_train, rf_y_pred_train))
print("Random Forest - Test Accuracy:", accuracy_score(y_test, rf_y_pred_test))
```

```
# Classification Report
print("Random Forest - Classification Report:")
print(classification_report(y_test, rf_y_pred_test))

# Get predicted probabilities for the positive class
y_pred_prob = rf_model.predict_proba(X_test)[:, 1]

# Calculate ROC-AUC score
roc_auc = roc_auc_score(y_test, y_pred_prob)
print("ROC-AUC Score:", roc_auc)

# Get ROC curve data
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line (no discrimination)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```