Predicting Term Deposit Subscription

Centennial College

BA723 – Business Analytics Capstone

Professor Bilal Hasanzadah

Bautista, Glen George

Sanchez Galindo, Maria Paula

August 12, 2022

Table of Contents

Executive Summary

0.1. Executive Introduction

In this project, we will examine the possibility of clients signing up for a term deposit as a result of a Portuguese bank's marketing effort. For their marketing plan to be effective, the bank should understand the characteristics of term deposit clients. Bank marketing relies on client data. These data sources are too vast for a human analyst to extract relevant data. These efforts benefit greatly from data mining. The goal is to boost campaign effectiveness by identifying important success elements via different predictive modelling techniques.

0.2. Executive Objective

We will look at how bank customers respond to marketing. We'll examine the customer's financial records and behavior to identify which factors influence their decision to open a term deposit. We'll conduct statistical analysis and evaluate the various models to establish the best accurate model for predicting term deposit subscriptions.

0.3. Executive Model Description

The modelling approach in this study was divided into two groups, with the same set of techniques used in both (Random Forest Classifier, Decision Tree Classifier, Logistic Regression Classifier, and Gradient Boost Classifier), with the distinction that in the second group, we did a hyperparameter tuning using grid search. After evaluating the different models, we concluded that the Gradient Boost Classifier performed the best using the criteria accuracy.

0.4. Executive Recommendations

For future bank marketing campaigns, we recommend to consider setting age into intervals to see which group subscribes most to term deposits. Moreover, always consider hyperparameter tuning, address outliers and handle missing values. Some variables may contain 'unknown' data that could distort the analysis; this should be evaluated by doing variables investigation.

Introduction

1.0. Background

Banks offer a variety of products and services based on their capacities. More services will be offered by a bank if its capabilities and quality are greater. Direct product introduction is prevalent in numerous industries, including the banking industry. When directly offering products, banks might do market research by utilizing the information technology sector to assist in decision making. By evaluating bank marketing data, it is possible to select the style of marketing to implement.

In this project, we will assess the likelihood of clients signing up for a term deposit as a result of a marketing campaign run by a Portuguese banking organization. A financial institution must discover the elements that impact a customer's decision to subscribe to a term deposit in order to determine the efficacy of its marketing campaign. To be effective with their marketing strategy, the banking institution would benefit from understanding and grasping the characteristics of clients who opened a term deposit.

2.0. Problem Statement

All bank marketing strategies rely on client data. These data sources are too large for a human analyst to extract useful information for decision-making. Data mining models help these initiatives immensely. The goal is to increase campaign effectiveness by identifying key success factors using a few algorithms (e.g. Logistic Regression, Random Forests, Decision Trees and others). With experimental findings, we'll show the models' accuracy, sensitivity, precision, recall, etc. With better metrics, we can measure the models' success in anticipating the optimum campaign contact

with clients for subscribing deposit. The marketing campaign aimed to increase term deposit subscriptions. Data set variable 'y' indicates if they accomplished this. The bank is evaluating future marketing improvements.

Banks run several marketing efforts to promote their products and increase the number of people who sign up for their special offers. The effectiveness of marketing campaigns should be addressed in order to reduce labor costs and enhance revenues through the employment of increasingly productive direct marketing strategies that do not interfere with customer relationships. This initiative may shed light on the bank's marketing effectiveness in terms of increasing client trust. Furthermore, by examining a bank's marketing data, it is possible to determine the type of marketing to perform.

3.0. Objectives and Measurement

We will examine how clients respond to the marketing effort of their financial institution. Regarding the customer's financial records and behavior, we will carefully determine which characteristics, in this case the factors, significantly influence their decision to subscribe to a term deposit. After taking into account all of the variables, we will conduct statistical analysis and evaluate the various models in order to determine the most accurate model for predicting term deposit subscriptions.

4.0. Assumptions and Dependencies

Before beginning the investigation, only a few assumptions were made. First, a customer's higher educational level will have a significant impact on his or her propensity to take a term deposit with the bank following the campaign. Second, a

customer's marital status may influence whether or not he or she will subscribe to a term deposit because it will affect his or her spending and income. Third, by focusing the bank's advertising effort on this client profile, the bank's advertising costs might be reduced.

In this project, the necessity to investigate the data and select the final variables before applying the prediction techniques would be regarded a dependency.

Data Sources

5.0. Data Set Introduction

The dataset contains all records of final outcomes indicating whether or not campaign initiatives were successful in the binary format of "yes" or "no" for a Portuguese financial institution. A successful campaign, on the other hand, reveals that the target clientele have finally opened a term deposit account with the Portuguese Bank at the end of the campaign period. The purpose is to identify successful campaign elements for the Portuguese bank that will assist it in increasing its direct marketing effectiveness by targeting only the right clients.

The data used in this analysis is legitimately related to the direct advertising operations of Portuguese financial institutions and originates from the UCH Machine Learning Repository. As illustrated in Figure 1, the promotional activities were centered on phone interactions.

Data Set Characteristics:	Multivariate	**Number of Instances:**	45211	**Area:**	Business
Attribute Characteristics:	Real	**Number of Attributes:**	17	**Date Donated**	2012-02-14
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	1805918

Figure 1. Bank Marketing Data Set Summary.

The dataset has 45,211 instances of calls to clients (rows) and 17 variables (columns) including the target variable ‘y’. The data is relevant to a Portuguese banking institution's direct marketing initiatives. The marketing campaigns relied on phone calls. More than one contact with the same consumer was frequently required in order to determine if the product (bank term deposit) would be (or would not be) subscribed to.

6.0. Exclusions

There are no exclusions considered in this study as all of the 17 variables were considered before doing the data exploration.

7.0. Data Dictionary

There were 45,211 client calls (rows) and 17 variables (columns): 10 object-types (including the Target Variable) and 7 numeric-types. In certain circumstances, the same client was reached many times, yet each call was treated independently even if the client was the same. Table 1 below shows the descriptions of the different variables and the values of each of the attributes.

Table 1. Bank Marketing Data Variables

Attributes	Type	Description	Values of attributes
age	numeric	age of client	values between 18 and 95
job	categorical	type of job	'management', 'technician', 'entrepreneur', 'blue-collar', 'unknown', 'retired', 'admin.', 'services', 'self-employed', 'unemployed', 'housemaid', 'student'
marital	categorical	marital status note: 'divorced' means divorced or widowed	'divorced', 'married', 'single'
education	categorical	degree of education	'primary', 'secondary', 'tertiary', 'unknown'
default	binary	has credit in default?	'no', 'yes'
balance	numeric	account balance	values between -8019 and 102127
housing	binary	has housing loan?	'no', 'yes'
loan	binary	has personal loan?	'no', 'yes'
contact	categorical	contact communication type	'cellular', 'telephone, 'unknown'
day	numeric	day in month	values between 1 and 31
month	categorical	last contact month of year	'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'
duration	numeric	last contact duration, in seconds	values between 0 and 4918
campaign	numeric	number of contacts performed during this campaign and for this client (included last contact)	values between 1 and 63
p-days	numeric	number of days that passed by after the client was last contacted from a previous campaign note: 999 means client was not previously contacted	values between -1 and 871
previous	numeric	number of contacts performed before this campaign and for this client	values between 0 and 275
p-outcome	categorical	outcome of the previous marketing campaign	'failure', 'other', 'success', 'unknown'
y	binary	has the client subscribed a term deposit?	'no', 'yes'

Data Exploration

8.0. Data Exploration Techniques

In this study, we used Python programming language to study the data. With the use of this programming language, we were able to see the structure of the data, get information about the different variables, and acquire initial insights about the different features of the variables.

8.1. Importing the Necessary Libraries

```
#Load the packages
import pandas as pd
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from google.colab import files
from IPython.display import Image

%matplotlib inline
```

Figure 2. Libraries imported for Data Exploration.

8.2. Importing the File and Viewing the Data

```
#Load the data
bank = pd.read_csv( 'bank-full-finaldata.csv')

#View the data
bank.head()
```

	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 3. File upload and data viewing.

8.3. Data Structure Investigation

```
#Show columns of the data
bank.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

```
#Show size of the dataset
bank.shape
```

```
(45211, 17)
```

```
#Count how many times each data type is present in the dataset
pd.value_counts(bank.dtypes)
```

```
object    10
int64      7
dtype: int64
```

Figure 4a. Data Structure Investigation: columns, data shape, and data types.

```
#Show dataframe information
banks6.info()
```

```
<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 4b. Data Structure Investigation: columns, data shape, and data types.

8.4. Variables Investigation

a. age

The variable age is the age of clients, which is in numeric type. Ages range from 18 to 95 years old. Below is the bar plot, histogram and density of this variable.

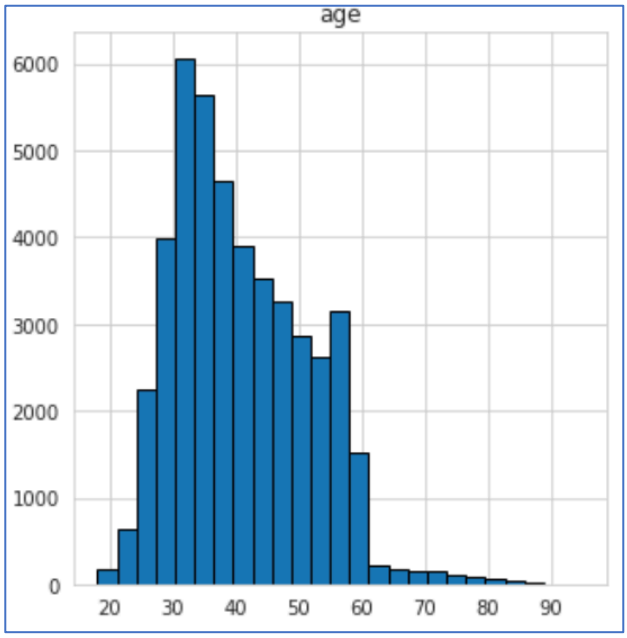


Figure 5a. Bar plot of the variable 'age'.

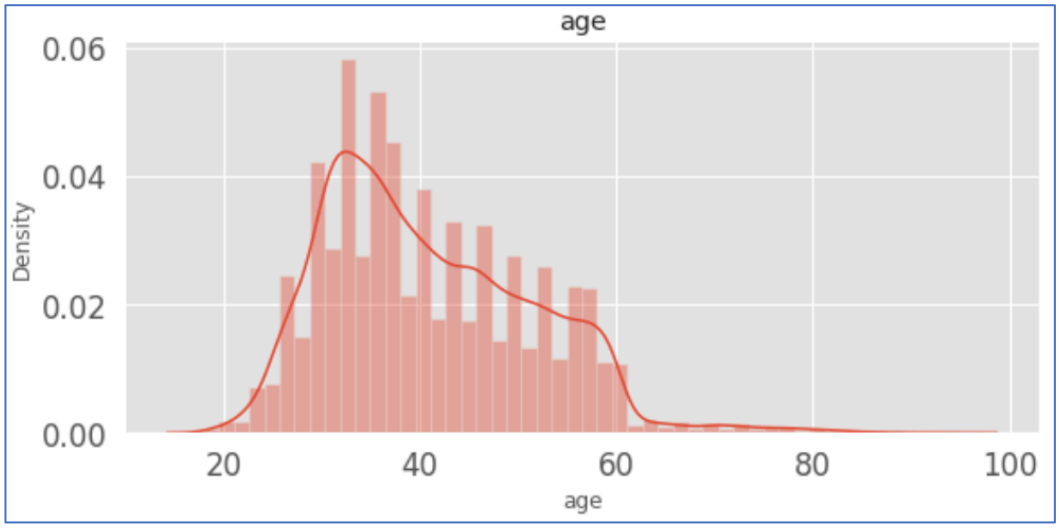


Figure 5b. Histogram and density of the variable 'age'.

b. job

The variable job, which is categorical in nature and contains categorical qualities. According to the census results, blue-collar jobs have the most employees, while managerial jobs have the second most employees. The bar-plot of job kinds below reveals that management has the largest term deposit subscription, followed by technician, and blue-collar has the lowest.

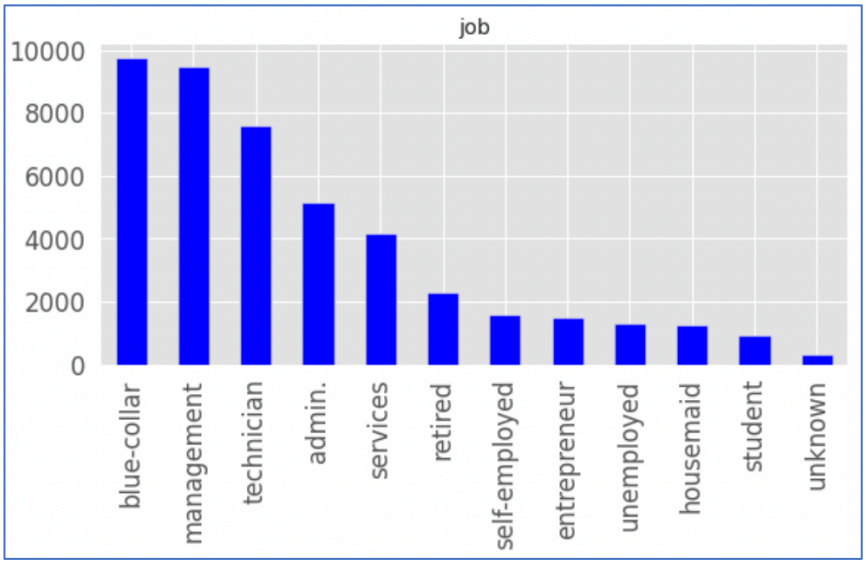


Figure 6a. Bar plot of the variable 'job'.

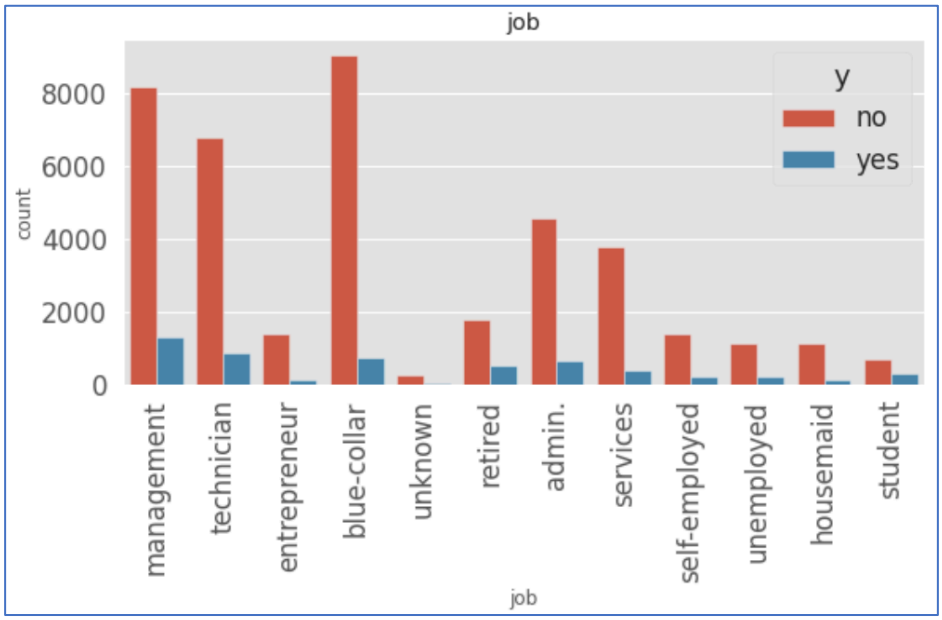


Figure 6b. Bar plot of the variable 'job' according to the results of campaign.

c. marital

The third variable is marital status, which is categorical in nature and has the following values: 'divorced', 'married', and 'single'. The 'married' have the highest subscription to a term deposit as well as the highest unsubscription to a term deposit, according to Figure 7b below, because 'married' individuals are the largest category of analyzed clients. The 'single' has the second greatest term deposit subscription and the second largest term deposit unsubscription. The 'married' have the highest number of term deposits subscribed and unsubscribed, followed by the 'single', while the 'divorced' have the lowest number of term deposits subscribed and unsubscribed.

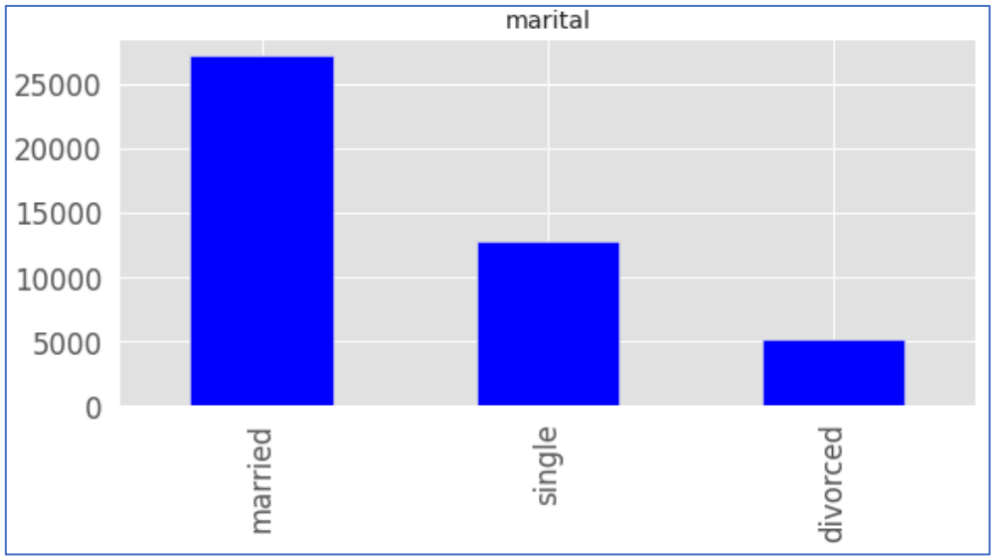


Figure 7a. Bar plot of the variable 'marital'.

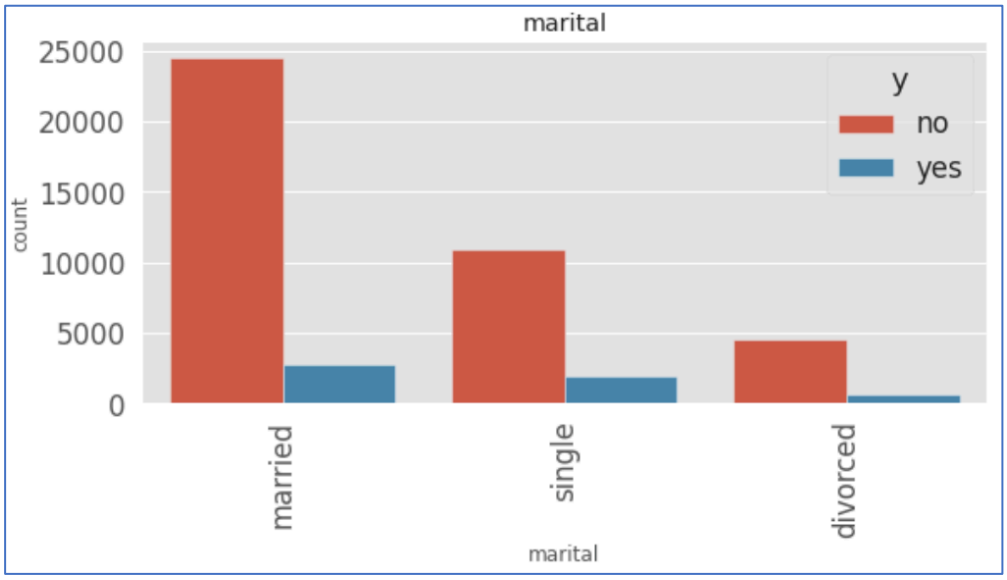


Figure 7b. Bar plot of the variable ‘marital’ according to the results of campaign.

d. education

The variable 'education' is categorical and has the following attribute values: 'primary’, ‘secondary’, ‘tertiary’, and 'unknown’. Figure 8b shows that secondary education has the most subscribers and unsubscribers to the bank term deposit, followed by tertiary education, primary education, and the unknown, which has the fewest subscribers and unsubscribers to the bank term deposit.

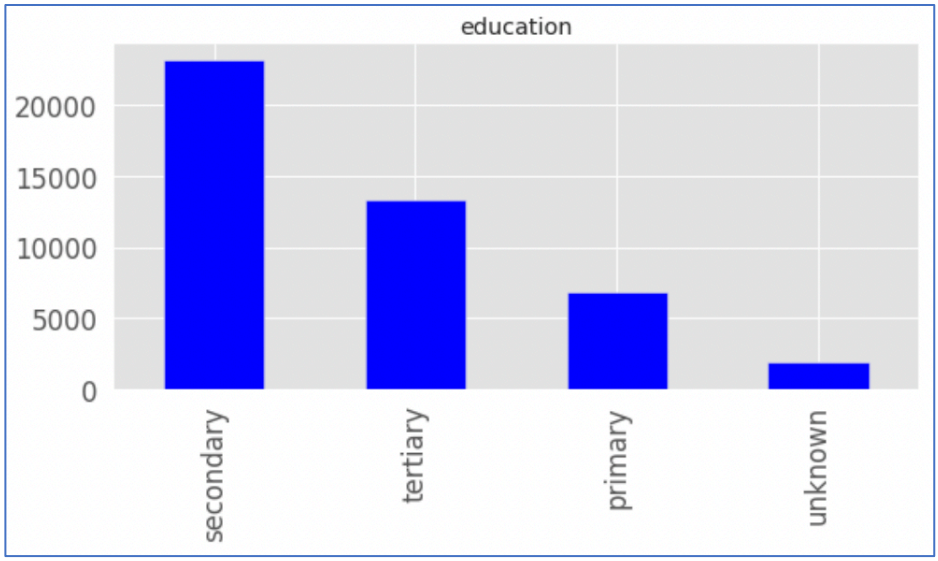


Figure 8a. Bar plot of the variable ‘education’.

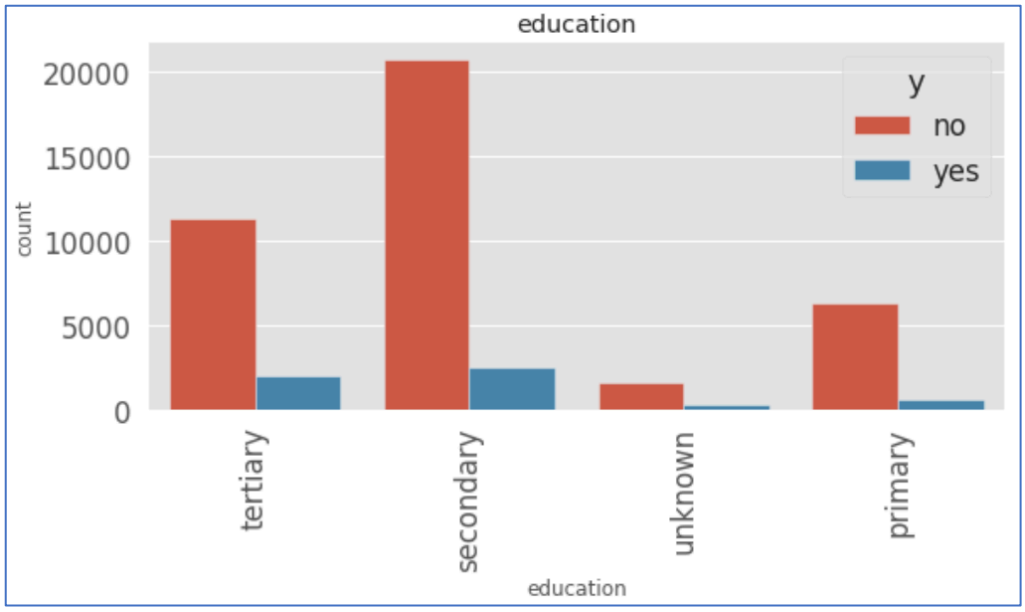


Figure 8b. Bar plot of the variable ‘education’ based on the results of campaign.

e. default

The variable 'default' is categorical and has the following attribute values: 'no’, ‘yes’, and 'unknown’. A lot of clients have no credit card in default, and those who have no credit card in default mostly did not subscribe to a term deposit.

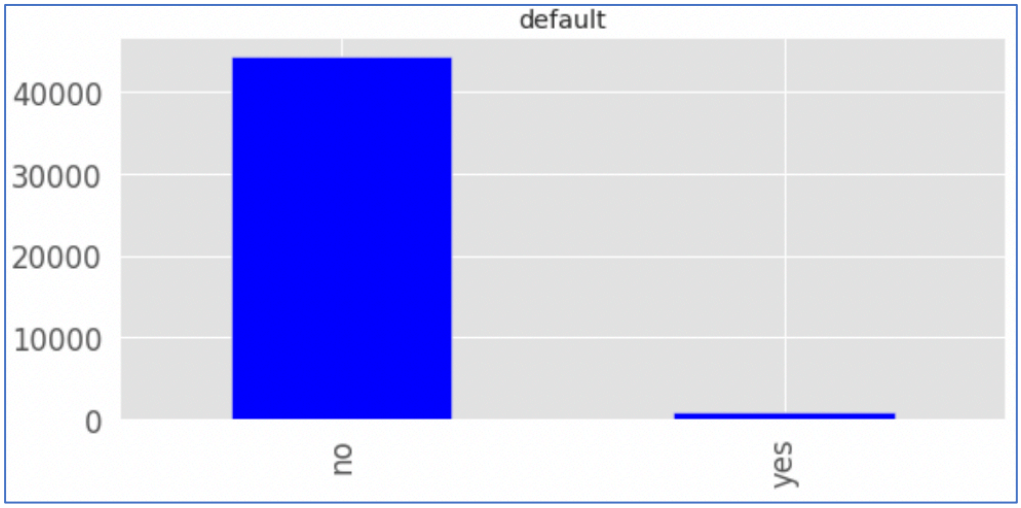


Figure 9a. Bar plot of the variable ‘default’.

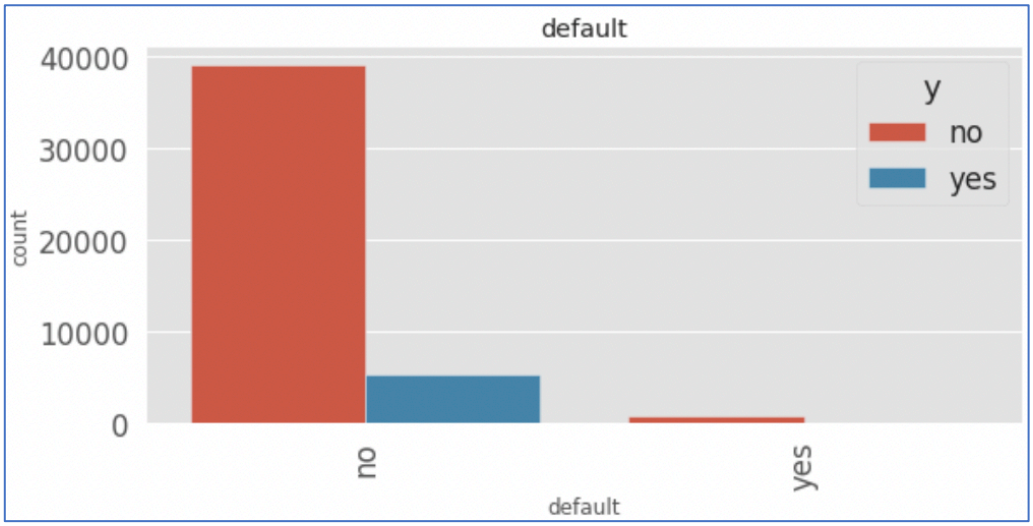


Figure 9b. Bar plot of the variable 'default' based on the results of campaign.

f. balance

The variable balance is numerical in nature, with values ranging from -8019 to 102127.

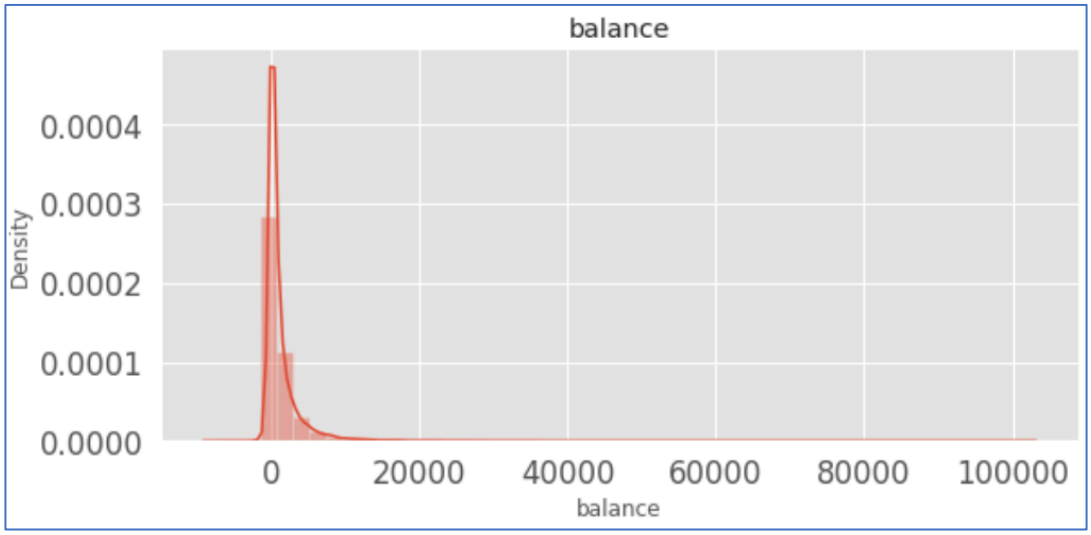


Figure 10. Histogram and density of the variable 'balance'.

g. housing

The variable 'housing' is categorical and has the following attribute values: 'no', 'yes', and 'unknown'.

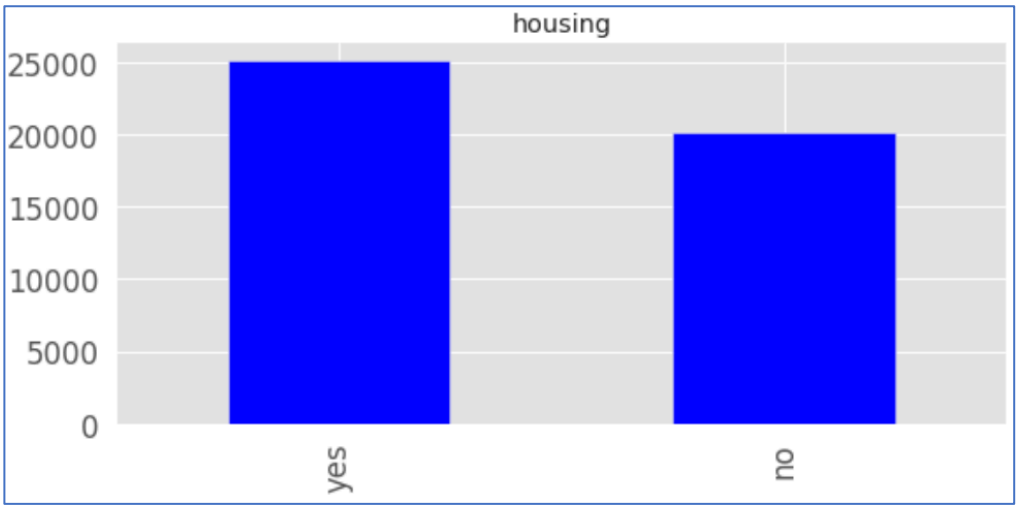


Figure 11a. Bar plot of the variable 'housing'.

Figure 11b. Bar plot of the variable 'housing' based on the results of campaign.

h. loan

The variable 'loan' is categorical and has the following attribute values: 'no', ‘yes’, and 'unknown’.

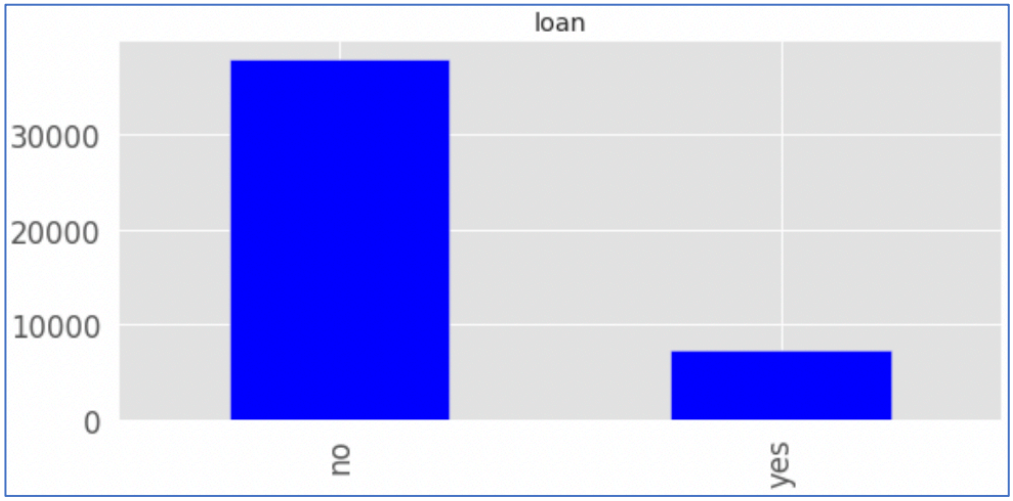


Figure 12a. Bar plot of the variable ‘loan’.

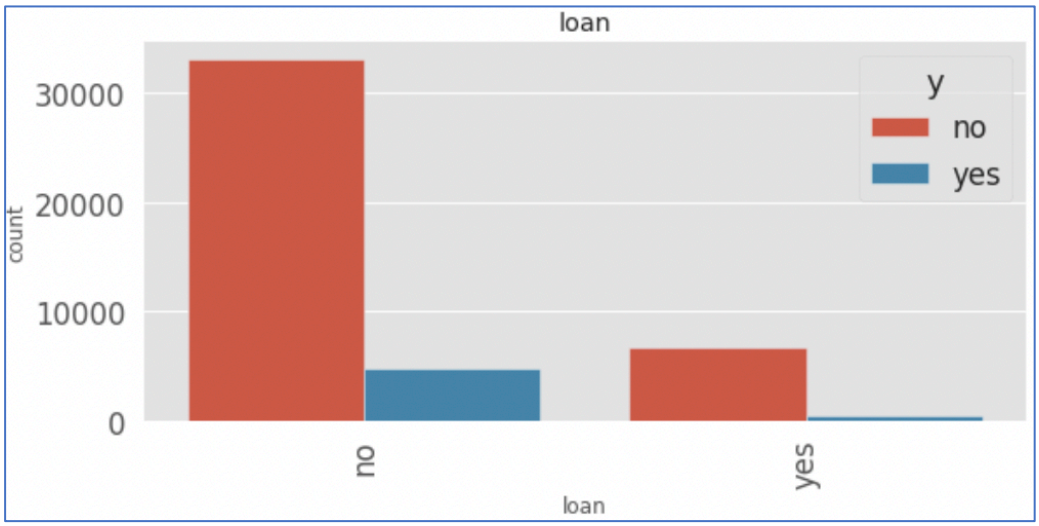


Figure 12b. Bar plot of the variable ‘loan’ based on the results of campaign.

i. contact

The variable 'contact' is categorical in nature and has attribute values of ‘cellular’, ‘telephone’, and ‘unknown’.

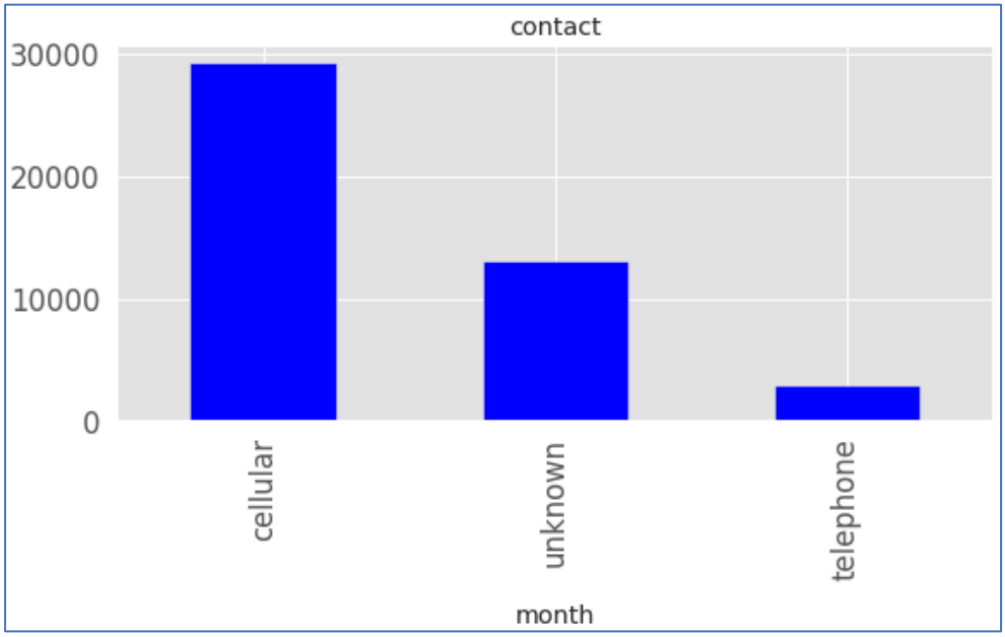


Figure 13a. Bar plot of the variable ‘contact’.

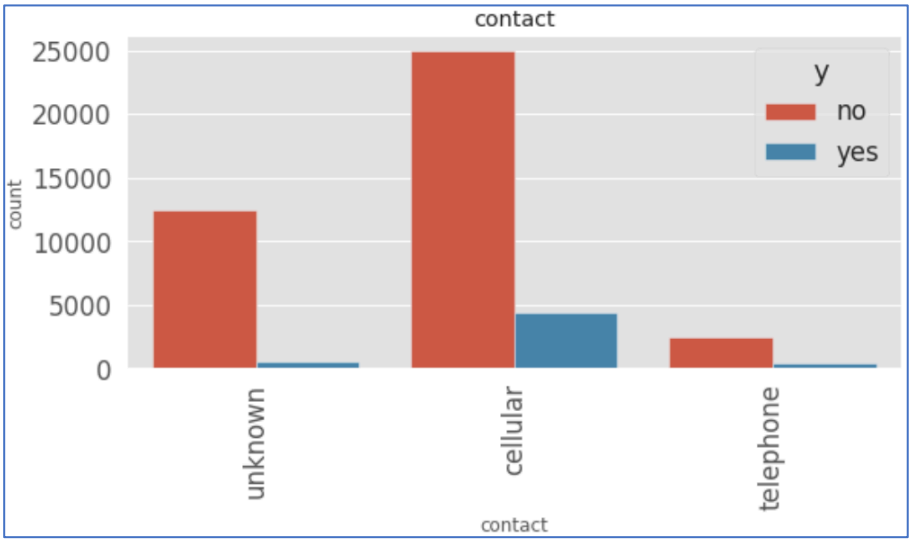


Figure 13b. Bar plot of the variable ‘contact’ based on the results of campaign.

j. day

The variable 'day' is numerical in nature and has attribute values ranging from 1 to 31.

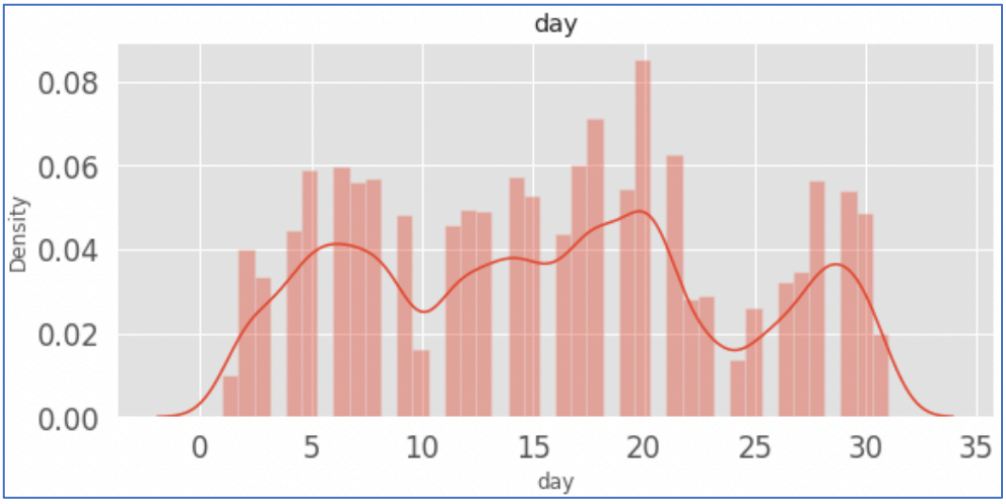


Figure 14. Histogram and density of the variable ‘day’.

k. month

The month attribute ranges from January through December. Figure 15b shows when a Portuguese bank ran direct marketing efforts and how many clients signed up for a term deposit each month. Red bars reflect the number of clients who did not subscribe to a bank term deposit, while blue bars show those who did. May had the most term deposit subscriptions and unsubscriptions, followed by July and August. Figure 15b illustrates a pictorial bar plot of days in a month with the number of clients that registered to the Portuguese bank's term deposit campaign.

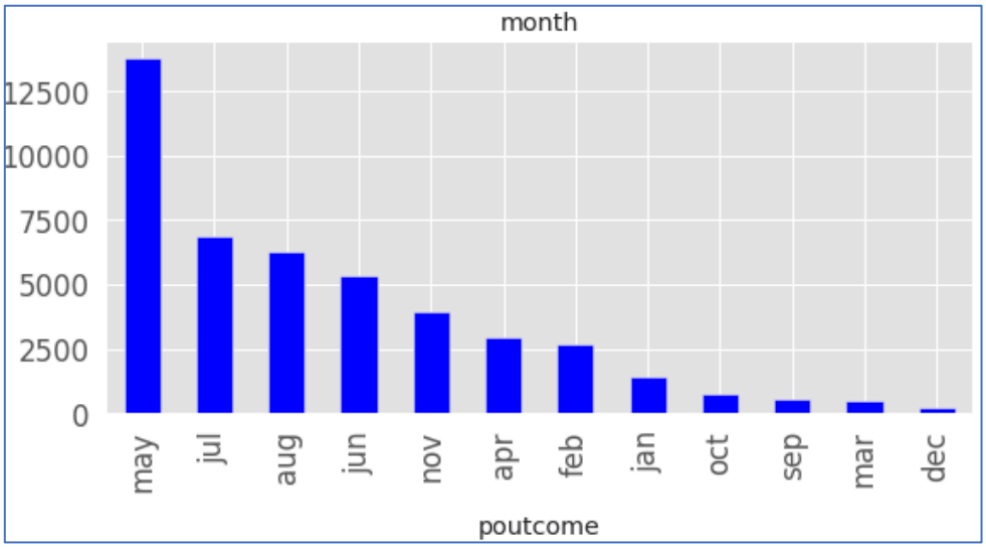


Figure 15a. Bar plot of the variable 'month'.

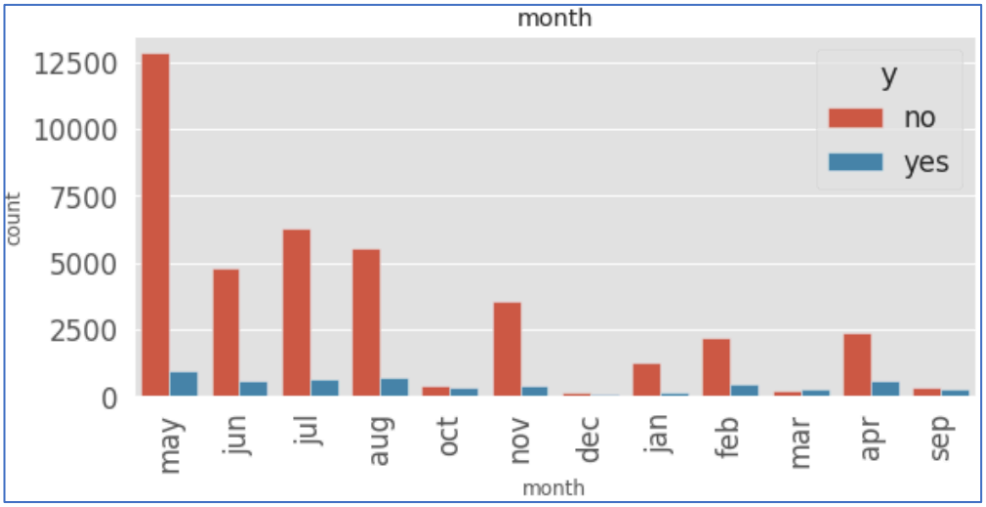


Figure 15b. Bar plot of the variable 'month' based on the results of campaign.

I. duration

The variable 'duration' is numerical in nature and has attribute values ranging from 0 to 4918. This variable impacts the target; if the duration is zero, no last contact was made within that time, hence y = 'no'. The duration of a call is not known before it is made, but it is known at the conclusion. This characteristic is provided for record keeping, clarity, and accuracy and should be removed for a realistic predictive model.

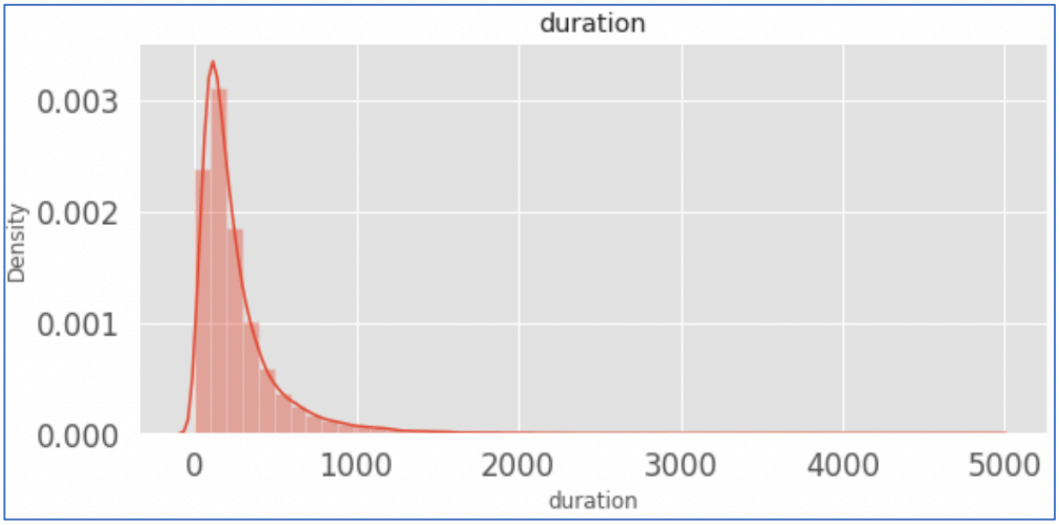


Figure 16. Histogram and density of the variable ‘day’.

m. campaign

The ‘campaign’ variable is numeric in nature, with values ranging from 1 to 63.

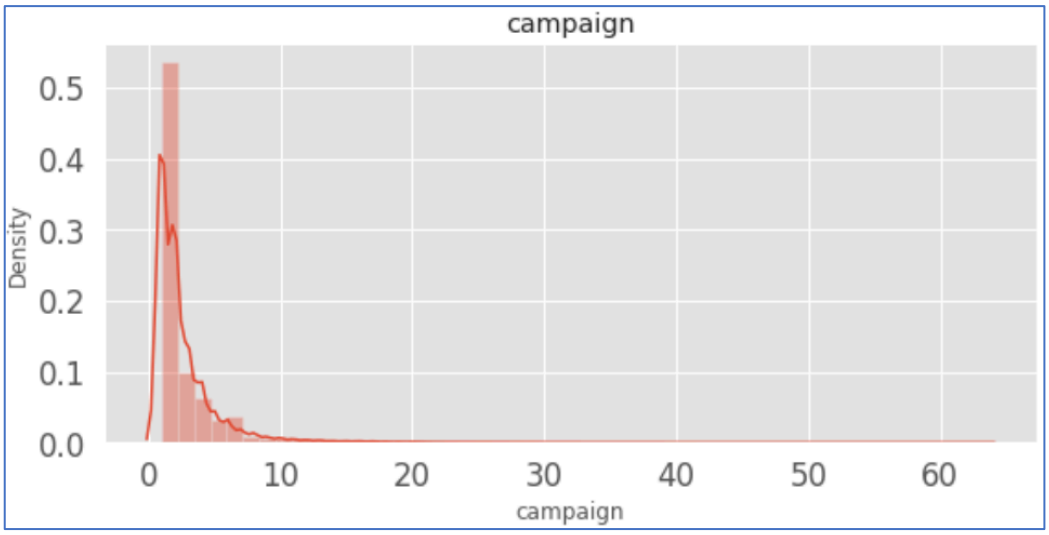


Figure 17. Histogram and density of the variable ‘campaign’.

n. p-days

The ‘p-days’ variable is numeric in nature, with values ranging from -1 to 87.

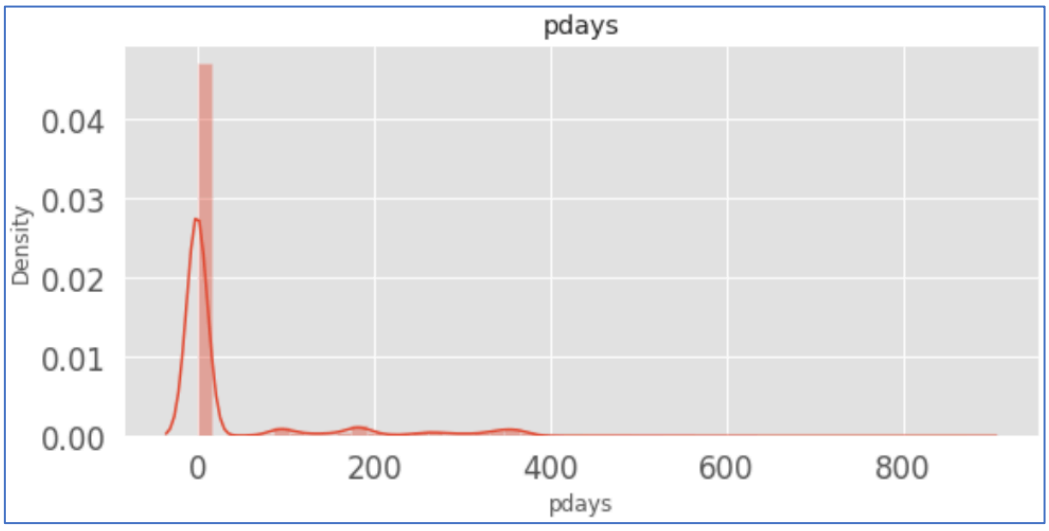


Figure 18. Histogram and density of the variable ‘p-days’.

o. previous

The ‘previous’ variable is numeric in nature, with values ranging from 0 to 275.

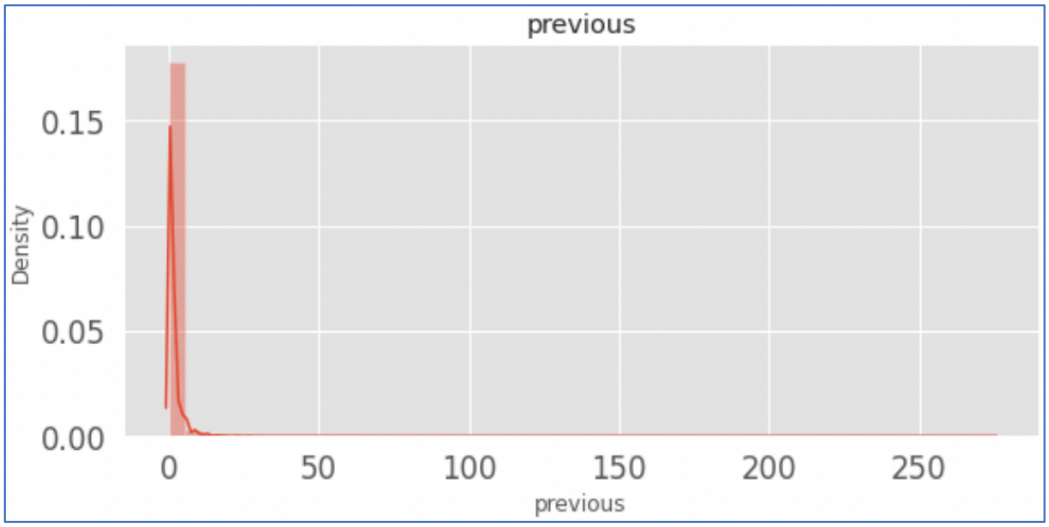


Figure 19. Histogram and density of the variable ‘previous’.

p. p-outcome

The category variable 'p-outcome' has the attributes 'failure', 'other', 'success', and 'unknown'. Figure 20b demonstrates the success, failure, unknown, and other results of Portuguese bank marketing programs. The plot shows that 'failure' in previous campaigns was due to a large number of clients refusing to subscribe to the bank term deposit/policy, while 'success' was due to a large number of clients subscribing to the bank term deposit policy, the 'other'. The previous marketing campaigns' 'unknown' outcome was estimated based on the number of clients who did not subscribe to the bank's term deposit policy.

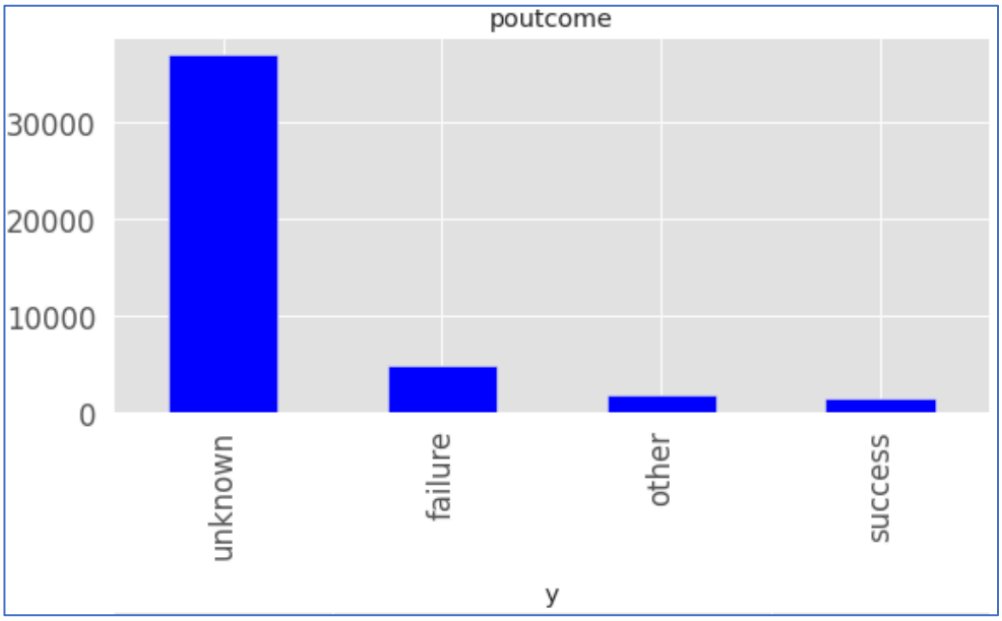


Figure 20a. Bar plot of the variable ‘month’.

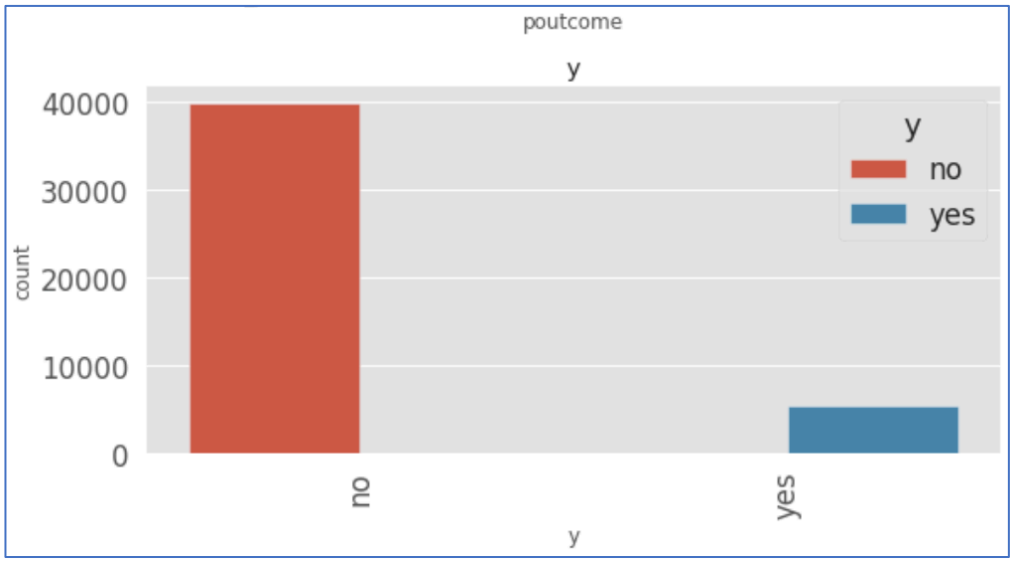


Figure 20b. Bar plot of the variable ‘p-outcome’ based on the results of campaign.

q. y

The variable ‘y’ is binary in nature and answers the question “has the client subscribed a term deposit?”. Out of the 45,211 instances, 39,922 clients did not subscribed for a term deposit while only 5,289 clients subscribed for a term deposit.

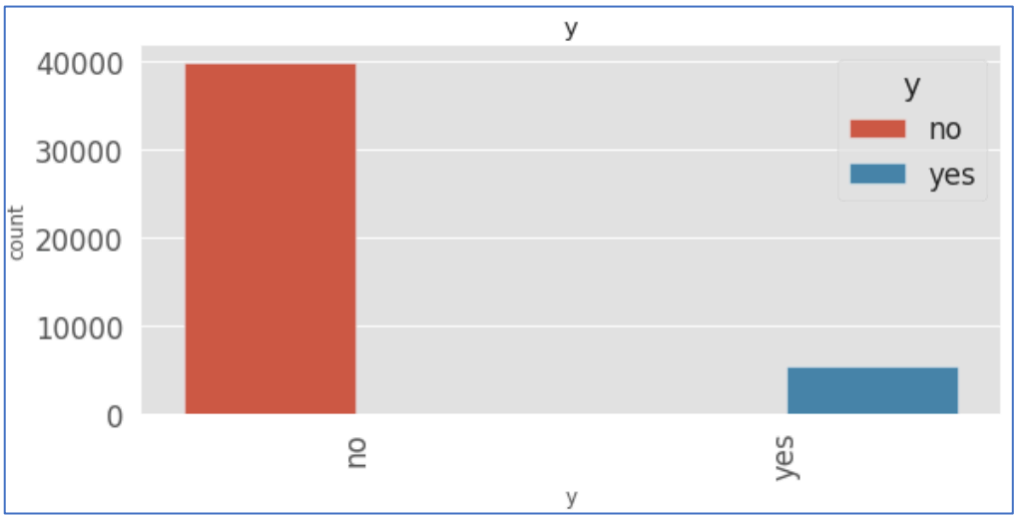


Figure 21. Bar plot of the variable ‘y’ based on the results of campaign.

9.0. Data Cleansing

Data cleaning was done and performed to some variables. For the variable ‘job’, of the data is ‘admin’. Thus, we needed to clean up this data and rename the data, removing the dot. For the variable ‘contact’, we replaced the ‘unknown’ data into ‘no’.

```
#Checking the different types of job
banks6['job'].head(15)
```

```
0       management
1       technician
2     entrepreneur
3      blue-collar
4      blue-collar
5       management
6       management
7     entrepreneur
8          retired
9       technician
10          admin.
11          admin.
12      technician
13      technician
14        services
Name: job, dtype: object
```

```
#Renaming 'admin.' into 'admin' (removed the dot) to make the data look cleaner
banks6['job'] = banks6['job'].replace('admin.', 'admin')
```

```
banks6['job'].head(15)
```

```
0       management
1       technician
2     entrepreneur
3      blue-collar
4      blue-collar
5       management
6       management
7     entrepreneur
8          retired
9       technician
10          admin
11          admin
12      technician
13      technician
14        services
Name: job, dtype: object
```

Figure 22a. Data cleansing of the variable ‘job’.

```
#Replaced 'unknown' data into 'no
banks6['contact'] = banks6['contact'].replace('unknown', 'no')
```

Figure 22b. Data cleansing of the variable 'contact'.

```
#Changing yes to 1 and no to 0
banks6['y'] = banks6['y'].map({'yes': 1, 'no': 0})

#How many people(client) have subscribed to a term deposit?
banks6.y.value_counts()

0    33126
1     4225
```

Figure 23. Data cleansing of the target variable 'y'.

10.0. Summary

With the use of Python programming, we first imported the necessary libraries and packages in order to run codes and to be successful with our initial data exploration. Once libraries and packages are loaded, we uploaded the Bank Marketing Data Set file from UCI Machine Learning Repository and viewed it's data.

To completely know the structure of the data file, we did a structure investigation wherein we ran codes to know how many columns and rows exist, the number of variables and their data types. Information such as how many of the variables are categorical and numerical were also obtained.

Since we did not know what each of the variables contain, we performed a variables investigation (univariate analysis of continuous columns, univariate analysis of numerical columns, and bivariate analysis of categorical columns) to understand each of the data. In this part, we produced bar plots, histograms and density of each

of the variables. Frequencies of each of the data for each variable were known and hints for further cleaning of the data were obtained.

From the initial investigation of the data set and variables, we performed a data cleansing for some of the variables. Unnecessary characters were removed, the ‘unknown’ value of the variable ‘contact’ was converted to ‘no’, and the value of the target variable ‘y’ was transformed to ‘0’ (‘no’) and ‘1’ (‘yes’).

Data Preparation and Feature Engineering

11.0. Data Preparation Needs

Before performing the different predictive modeling techniques, we must assure that cleansing of the data is performed, outliers are identified and resolved, and missing values are handled. First, we identified outliers by creating a boxplot of the data frame, and then further looking deeper into the boxplot of each of the variables. Four variables were found to have outliers: balance, age, campaign, and duration. After identifying the outliers, we converted the outlier values to nan, making it missing value, in order for us to easily remove the outliers by dropping the rows with missing values.

After dealing with the data set outliers, we handles missing values in different ways. Initially, there were no missing values and regarded the data as complete. After doing the variables investigation, we found out that values such as ‘unknown’ exist, and that we considered some of them as missing values, depending on the percentage of the data ‘unknown’ for each variable. For the variables job, education, and contact, we imputed missing values (‘unknown’ data) by replacing the ‘unknown’ data to the mode of the data of the variables. For the variable p-outcome, since there are 36,959 ‘unknown’ data, we considered it as unnecessary column and dropped the entire column.

11.1. Dealing with Outliers

```
#Checking for outliers in dataset
plt.figure(figsize=(14,10))
banks6.boxplot()
plt.title("Boxplot of the dataframe", fontsize = 15)
print()
```

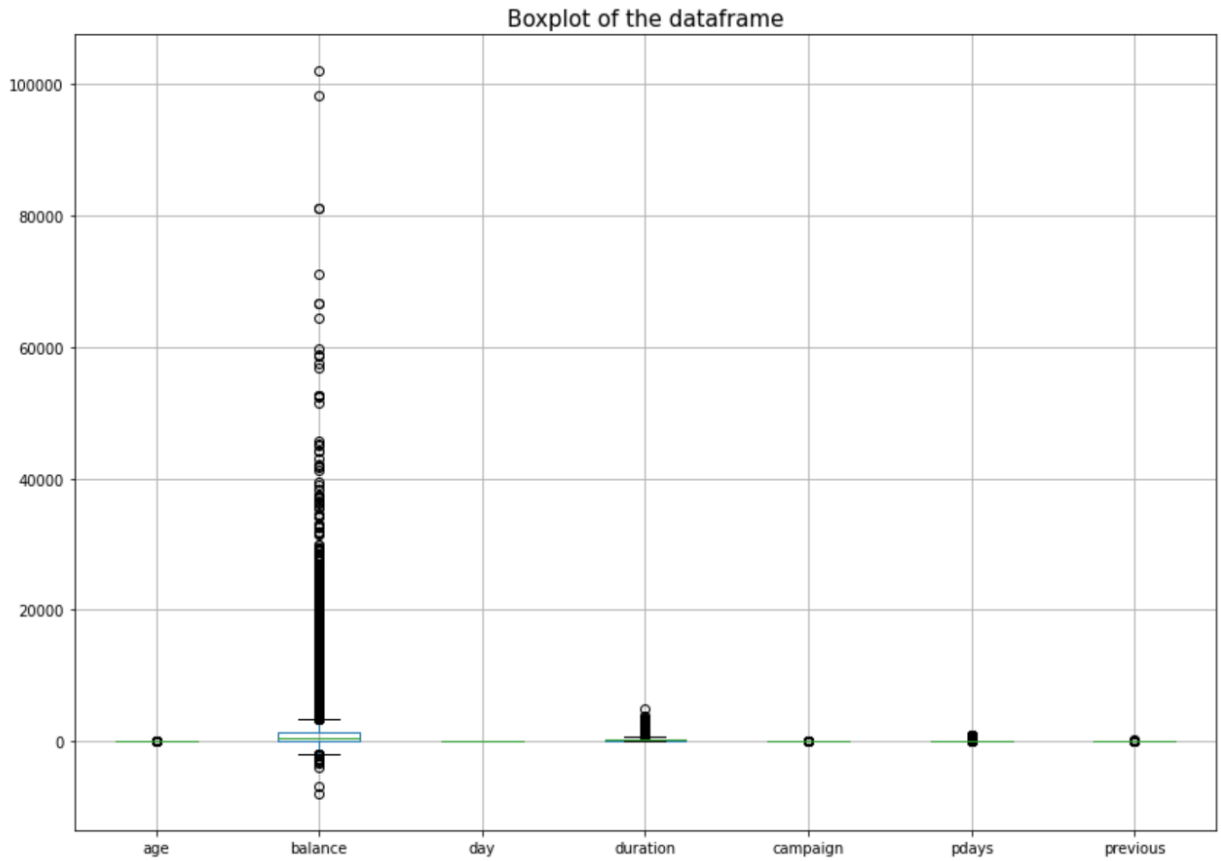
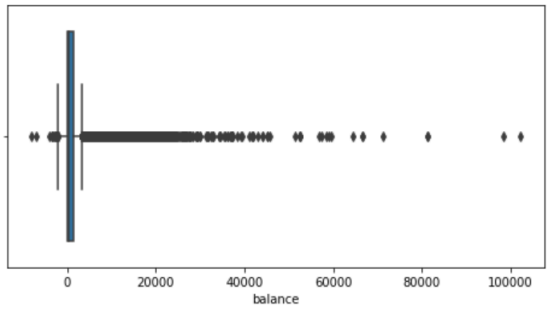


Figure 24. Box plot of the data frame.

As we can see from the figure above, variables 'balance' and 'duration seem to have outliers. However, the figure does not really explain whether outliers exist or not. To confirm the existence of outliers, a more detailed and deep understanding of the variables are needed. We produced box plots to most of the variables and identified the outliers using Python. Below are the code for producing the box plots of some of the variables.

i. balance

```
[ ]  #Looking inside balance variable boxplot
     plt.figure(figsize=(8, 4))
     sns.boxplot(x=banks6['balance'])
     plt.show()
```

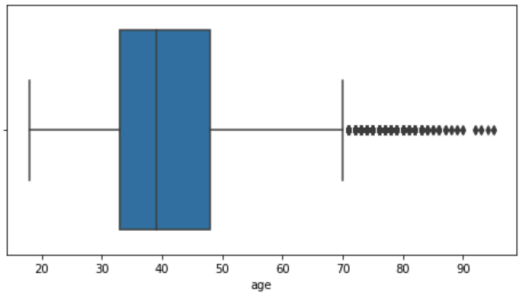



```
[ ]  for x in ['balance']:
         q75,q25 = np.percentile(banks6.loc[:,x],[75,25])
         intr_qr = q75-q25

         max = q75+(1.5*intr_qr)
         min = q25-(1.5*intr_qr)

         banks6.loc[banks6[x] < min,x] = np.nan
         banks6.loc[banks6[x] > max,x] = np.nan
```

Figure 25. Python code for the box plot of the variable ‘balance’.

ii. age

```
[ ]  #Looking inside age variable boxplot
     plt.figure(figsize=(8, 4))
     sns.boxplot(x=banks6['age'])
     plt.show()
```

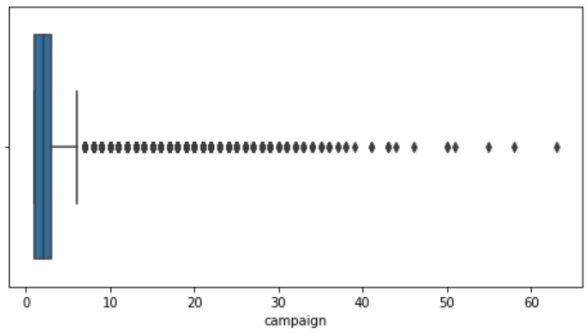



```
[ ]  for y in ['age']:
         q75_age,q25_age = np.percentile(banks6.loc[:,y],[75,25])
         intr_qr_age = q75_age-q25_age

         max_age = q75_age+(1.5*intr_qr_age)
         min_age = q25_age-(1.5*intr_qr_age)

         banks6.loc[banks6[y] < min_age,y] = np.nan
         banks6.loc[banks6[y] > max_age,y] = np.nan
```

Figure 26. Python code for the box plot of the variable ‘age’.

iii. campaign

```
#Looking inside campaign variable boxplot
plt.figure(figsize=(8, 4))
sns.boxplot(x=banks6['campaign'])
plt.show()
```




```
[ ] for z in ['campaign']:
        q75_c,q25_c = np.percentile(banks6.loc[:,z],[75,25])
        intr_qr_c = q75_c-q25_c

        max_c = q75_c+(1.5*intr_qr_c)
        min_c = q25_c-(1.5*intr_qr_c)

        banks6.loc[banks6[z] < min_c,z] = np.nan
        banks6.loc[banks6[z] > max_c,z] = np.nan
```

Figure 27. Python code for the box plot of the variable ‘campaign’.

After observing the box plots produced, we used Python to code and store the 25th and 75th percentiles into a variable. The interquartile range was also identified and is calculated as the 75th percentile minus the 25th percentile. For the minimum range and maximum range, formula can be seen above. Once the minimum and maximum values are known, they will be converted as ‘missing values’ or ‘null’. Figure 28 below shows the code for producing the summary of variables with the count of ‘null’ values. These null values will then be dropped in order for our data set to have no missing values after dealing with the outliers. Figure 29 shows the code for dropping the null values and the confirmation of non-existence of missing values.

```
#Show the number of null values per variable
banks6.isnull().sum()
```

```
age          487
job            0
marital        0
education      0
default        0
balance     4729
housing        0
loan           0
contact        0
day            0
month          0
duration       0
campaign    3064
pdays          0
previous       0
poutcome       0
y              0
dtype: int64
```

Figure 28. Python code for the number of null values of each variable.

```
#Dropping the rows will null values
banks6 = banks6.dropna(axis = 0)
```

```
#Rechecking if there are still null values
banks6.isnull().sum()
```

```
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
dtype: int64
```

Figure 29. Python code for the dropping the null values.

11.2. Handling Missing Values

During the initial data exploration, we found out that there were no missing values. After conducting the variables investigation, we figured out that some of the variables contain ‘unknown’ data (job, education, contact, and p-outcome). These ‘unknown’ data were regarded as ‘missing’ values as they do not entail a specific meaning. The variables ‘job, ‘education’, ‘contact’, and p-outcome’ contain 288, 1857, 13020, and 36959, respectively. Figure 30 shows the summary of ‘unknown’ data form each of the variables.

```
#Investigating 'unknown' values
print("# Missing value 'age' variable: {0}".format(len(bank.loc[bank['age'] == "unknown"])))
print("# Missing value 'job' variable: {0}".format(len(bank.loc[bank['job'] == "unknown"])))
print("# Missing value 'marital' variable: {0}".format(len(bank.loc[bank['marital'] == "unknown"])))
print("# Missing value 'education' variable: {0}".format(len(bank.loc[bank['education'] == "unknown"])))
print("# Missing value 'default' variable: {0}".format(len(bank.loc[bank['default'] == "unknown"])))
print("# Missing value 'balance' variable: {0}".format(len(bank.loc[bank['balance'] == "unknown"])))
print("# Missing value 'housing' variable: {0}".format(len(bank.loc[bank['housing'] == "unknown"])))
print("# Missing value 'loan' variable: {0}".format(len(bank.loc[bank['loan'] == "unknown"])))
print("# Missing value 'contact' variable: {0}".format(len(bank.loc[bank['contact'] == "unknown"])))
print("# Missing value 'day' variable: {0}".format(len(bank.loc[bank['day'] == "unknown"])))
print("# Missing value 'month' variable: {0}".format(len(bank.loc[bank['month'] == "unknown"])))
print("# Missing value 'duration' variable: {0}".format(len(bank.loc[bank['duration'] == "unknown"])))
print("# Missing value 'campaign' variable: {0}".format(len(bank.loc[bank['campaign'] == "unknown"])))
print("# Missing value 'pdays' variable: {0}".format(len(bank.loc[bank['pdays'] == "unknown"])))
print("# Missing value 'previous' variable: {0}".format(len(bank.loc[bank['previous'] == "unknown"])))
print("# Missing value 'poutcome' variable: {0}".format(len(bank.loc[bank['poutcome'] == "unknown"])))

# Missing value 'age' variable: 0
# Missing value 'job' variable: 288
# Missing value 'marital' variable: 0
# Missing value 'education' variable: 1857
# Missing value 'default' variable: 0
# Missing value 'balance' variable: 0
# Missing value 'housing' variable: 0
# Missing value 'loan' variable: 0
# Missing value 'contact' variable: 13020
# Missing value 'day' variable: 0
# Missing value 'month' variable: 0
# Missing value 'duration' variable: 0
# Missing value 'campaign' variable: 0
# Missing value 'pdays' variable: 0
# Missing value 'previous' variable: 0
# Missing value 'poutcome' variable: 36959
```

Figure 30. Python code for determining the number of ‘unknown’ data.

Imputation of missing values was executed to handle ‘unknown’ data. For the first variable imputed, ‘job’ has few ‘unknown’ data of 0.567% of the ‘job’ data. For imputing the missing values, mode of each variable was identified and replaced the ‘unknown’ values. For the case of the variable ‘job’, ‘unknown’ was replaced to ‘blue-collar’. For the variable ‘education’, ‘unknown’ data represents 3.927% of the data and those ‘unknown’ were replaced by the mode ‘secondary’.

i. job

```
[ ] #Percentages of each value under the variable 'job'
    banks6['job'].value_counts(normalize=True) * 100
```

```
blue-collar      22.334074
management       20.160103
technician       16.941983
admin.           11.860459
services          9.595459
retired           4.050762
self-employed     3.421595
entrepreneur      3.290407
unemployed        2.931648
housemaid         2.685336
student           2.160585
unknown           0.567589
Name: job, dtype: float64
```

```
[ ] #Imputing 'unknown' (considered missing) data
    #Replaced 'unknown' data into 'blue-collar' since 'blue-collar' data is the mode
    banks6['job'] = banks6['job'].replace('unknown', 'blue-collar')
```

```
[ ] #Checking the different types of job
    banks6['job'].head(15)
```

```
0       management
1       technician
2     entrepreneur
3      blue-collar
4      blue-collar
5       management
6       management
7     entrepreneur
8          retired
9       technician
10          admin.
11          admin.
12      technician
13      technician
14        services
Name: job, dtype: object
```

Figure 31. Python code for imputing missing values of the variable ‘job’.

ii. education

```
[ ]  #Percentages of each value under the variable 'education'
     banks6['education'].value_counts(normalize=True) * 100
```

```
secondary    52.831249
tertiary     28.349977
primary      14.891168
unknown       3.927606
Name: education, dtype: float64
```

```
[ ]  #Checking the different types of job
     banks6['education'].head(15)
```

```
0      tertiary
1     secondary
2     secondary
3     secondary
4     secondary
5      tertiary
6      tertiary
7      tertiary
8       primary
9     secondary
10    secondary
11    secondary
12    secondary
13    secondary
14    secondary
Name: education, dtype: object
```

```
[ ]  #Imputing 'unknown' (considered missing) data
     #Replaced 'unknown' data into 'secondary' since 'secondary' data is the mode
     banks6['education'] = banks6['education'].replace('unknown', 'secondary')
```

Figure 32. Python code for imputing missing values of the variable ‘education’.

iii. contact

```
[ ]  #Percentages of each value under the variable 'contact'
     banks6['contact'].value_counts(normalize=True) * 100
```

```
cellular     65.053144
unknown      29.514605
telephone     5.432251
Name: contact, dtype: float64
```

```
[ ]  #Imputing 'unknown' (considered missing) data
     #Replaced 'unknown' data into 'no
     banks6['contact'] = banks6['contact'].replace('unknown', 'no')
```

Figure 33. Python code for imputing missing values of the variable ‘contact’.

After imputing missing values by replacing the ‘unknown’ data to the mode of the variable, a summary check for missing values were done for verification. Since only ‘p-outcome’ appears to have ‘unknown’ data of 30,409, representing more than 60% of the data set, we considered dropping the column as treating it in a different way or imputing it would cause bias in our modeling.

```
#Investigating 'unknown' values
print("# Missing value 'age' variable: {0}".format(len(banks6.loc[banks6['age'] == "unknown"])))
print("# Missing value 'job' variable: {0}".format(len(banks6.loc[banks6['job'] == "unknown"])))
print("# Missing value 'marital' variable: {0}".format(len(banks6.loc[banks6['marital'] == "unknown"])))
print("# Missing value 'education' variable: {0}".format(len(banks6.loc[banks6['education'] == "unknown"])))
print("# Missing value 'default' variable: {0}".format(len(banks6.loc[banks6['default'] == "unknown"])))
print("# Missing value 'balance' variable: {0}".format(len(banks6.loc[banks6['balance'] == "unknown"])))
print("# Missing value 'housing' variable: {0}".format(len(banks6.loc[banks6['housing'] == "unknown"])))
print("# Missing value 'loan' variable: {0}".format(len(banks6.loc[banks6['loan'] == "unknown"])))
print("# Missing value 'contact' variable: {0}".format(len(banks6.loc[banks6['contact'] == "unknown"])))
print("# Missing value 'day' variable: {0}".format(len(banks6.loc[banks6['day'] == "unknown"])))
print("# Missing value 'month' variable: {0}".format(len(banks6.loc[banks6['month'] == "unknown"])))
print("# Missing value 'campaign' variable: {0}".format(len(banks6.loc[banks6['campaign'] == "unknown"])))
print("# Missing value 'pdays' variable: {0}".format(len(banks6.loc[banks6['pdays'] == "unknown"])))
print("# Missing value 'previous' variable: {0}".format(len(banks6.loc[banks6['previous'] == "unknown"])))
print("# Missing value 'poutcome' variable: {0}".format(len(banks6.loc[banks6['poutcome'] == "unknown"])))
```

```
# Missing value 'age' variable: 0
# Missing value 'job' variable: 0
# Missing value 'marital' variable: 0
# Missing value 'education' variable: 0
# Missing value 'default' variable: 0
# Missing value 'balance' variable: 0
# Missing value 'housing' variable: 0
# Missing value 'loan' variable: 0
# Missing value 'contact' variable: 0
# Missing value 'day' variable: 0
# Missing value 'month' variable: 0
# Missing value 'campaign' variable: 0
# Missing value 'pdays' variable: 0
# Missing value 'previous' variable: 0
# Missing value 'poutcome' variable: 30409
```

Figure 34. Python code for checking missing values after imputation.

11.3. Dropping Unnecessary Columns

```
#Dropping the column 'poutcome' since
banks6 = banks6.drop('poutcome', axis=1)

#Investigating 'unknown' values
#Variable 'poutcome' dropped
print("# Missing value 'age' variable: {0}".format(len(banks6.loc[banks6['age'] == "unknown"])))
print("# Missing value 'job' variable: {0}".format(len(banks6.loc[banks6['job'] == "unknown"])))
print("# Missing value 'marital' variable: {0}".format(len(banks6.loc[banks6['marital'] == "unknown"])))
print("# Missing value 'education' variable: {0}".format(len(banks6.loc[banks6['education'] == "unknown"])))
print("# Missing value 'default' variable: {0}".format(len(banks6.loc[banks6['default'] == "unknown"])))
print("# Missing value 'balance' variable: {0}".format(len(banks6.loc[banks6['balance'] == "unknown"])))
print("# Missing value 'housing' variable: {0}".format(len(banks6.loc[banks6['housing'] == "unknown"])))
print("# Missing value 'loan' variable: {0}".format(len(banks6.loc[banks6['loan'] == "unknown"])))
print("# Missing value 'contact' variable: {0}".format(len(banks6.loc[banks6['contact'] == "unknown"])))
print("# Missing value 'day' variable: {0}".format(len(banks6.loc[banks6['day'] == "unknown"])))
print("# Missing value 'month' variable: {0}".format(len(banks6.loc[banks6['month'] == "unknown"])))
print("# Missing value 'campaign' variable: {0}".format(len(banks6.loc[banks6['campaign'] == "unknown"])))
print("# Missing value 'pdays' variable: {0}".format(len(banks6.loc[banks6['pdays'] == "unknown"])))
print("# Missing value 'previous' variable: {0}".format(len(banks6.loc[banks6['previous'] == "unknown"])))

# Missing value 'age' variable: 0
# Missing value 'job' variable: 0
# Missing value 'marital' variable: 0
# Missing value 'education' variable: 0
# Missing value 'default' variable: 0
# Missing value 'balance' variable: 0
# Missing value 'housing' variable: 0
# Missing value 'loan' variable: 0
# Missing value 'contact' variable: 0
# Missing value 'day' variable: 0
# Missing value 'month' variable: 0
# Missing value 'campaign' variable: 0
# Missing value 'pdays' variable: 0
# Missing value 'previous' variable: 0
```

Figure 35. Python code for confirming the existence of ‘unknown’ data.

After dealing with the outliers of the data set and handling missing (‘unknown’) values, we now have 0 missing values. A correlation table and heatmaps were produced to understand correlation of each variables.

11.4. Correlation Table and Heatmaps

```
#Calculating correlation
corr_matrix = banks6.corr()
print(corr_matrix)
```

```
                age   balance       day  campaign     pdays  previous         y
age        1.000000  0.083464 -0.008343  0.038840 -0.029990 -0.006752 -0.016984
balance    0.083464  1.000000  0.007037 -0.024644  0.027265  0.034060  0.087320
day       -0.008343  0.007037  1.000000  0.102572 -0.083853 -0.044462 -0.025958
campaign   0.038840 -0.024644  0.102572  1.000000 -0.068166 -0.005904 -0.063334
pdays     -0.029990  0.027265 -0.083853 -0.068166  1.000000  0.442701  0.100359
previous  -0.006752  0.034060 -0.044462 -0.005904  0.442701  1.000000  0.091997
y         -0.016984  0.087320 -0.025958 -0.063334  0.100359  0.091997  1.000000
```

Figure 36. Python code for calculating correlation.

```
#Creating correlation heat map
plt.figure(figsize=(16,10))
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1,fmt='.2f')
plt.title("Correlation Heatmap", fontsize = 15)
plt.show()
```

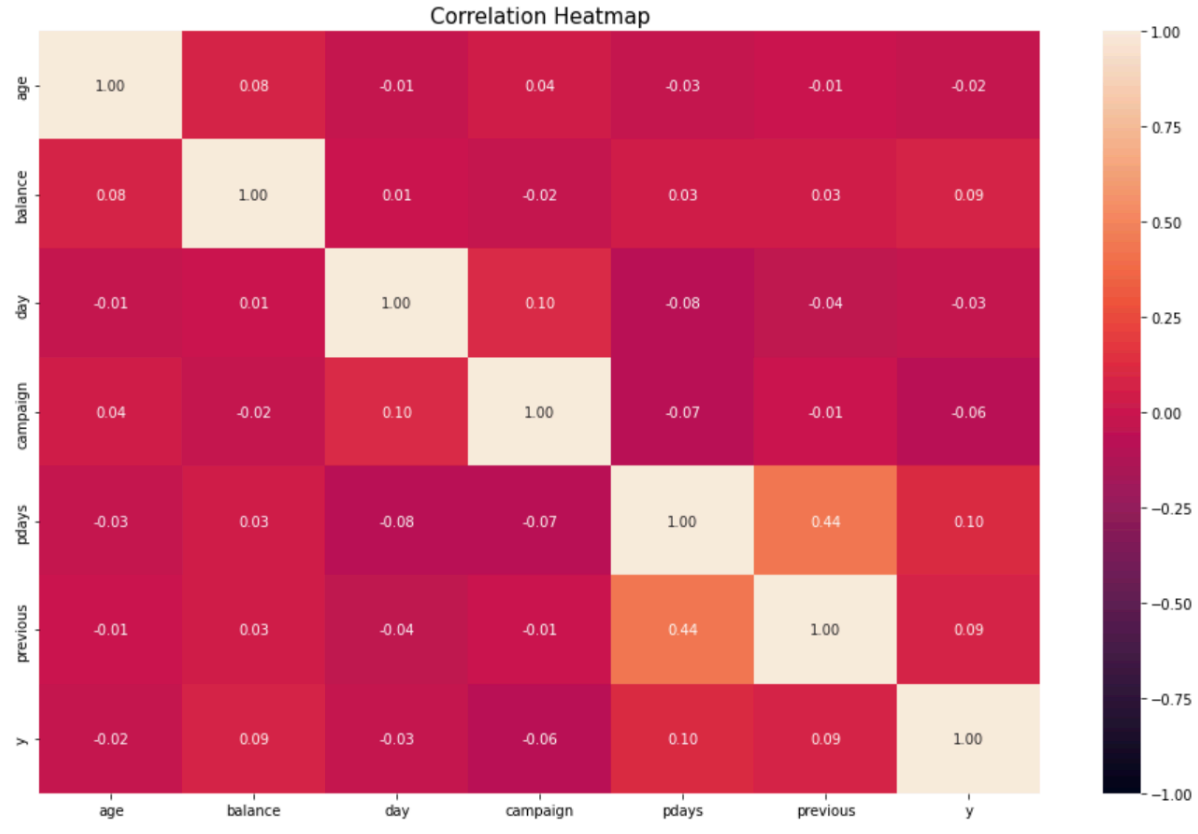


Figure 37. Python code for producing a correlation heatmap.

```
#Plotting the heatmap of only highly correlated varibles with threshold value 0.4
plt.figure(figsize=(16,10))
sns.heatmap(corr_matrix[corr_matrix > 0.4], annot=True)
plt.title("Correlation Heatmap",  fontsize = 15)
plt.show()
```

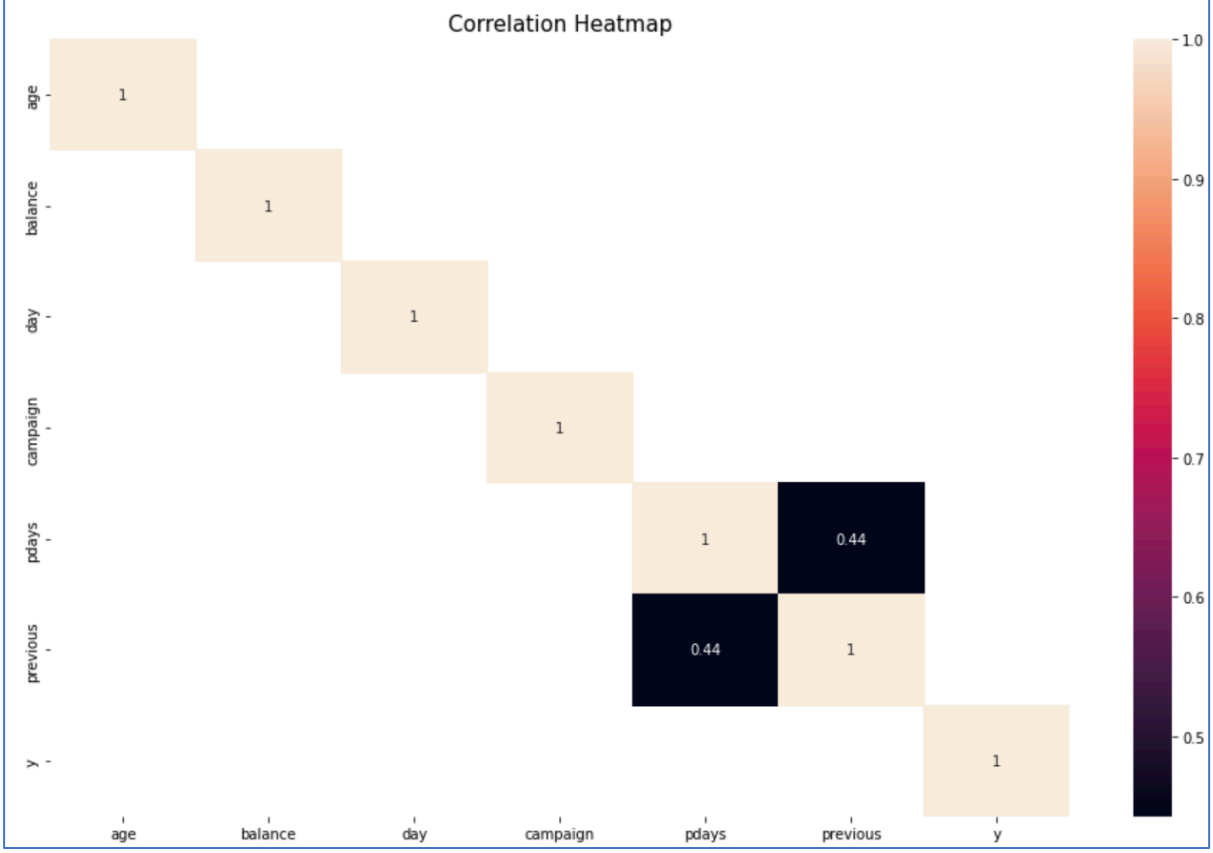


Figure 38. Python code for producing a correlation heatmap with a certain threshold.

Based from the correlation and heatmaps produced, ‘p-days’ and ‘previous’ have the highest correlation coefficient of 0.44 and is considered a moderate correlation. Most of the relationships of two variables was found out to have negative correlation. Positive correlations exist and they are ‘balance’ & ‘age’ (0.08), ‘balance’ & ‘p-days’ (0.03), ‘campaign’ & ‘age’ (0.04), ‘balance’ & ‘day’ (0.01) ; all of which have a very weak positive correlation.

12.0. Analysis Preparation

In preparation for the analysis and modeling part of our project and after preparing and cleaning the data and variables, we checked for class imbalance, transformed variables and get dummy variables, and performed a train test split.

12.1. Class Imbalance

Based on the python code ran, we can see than there is a class imbalance. Diving deep to the results of the bank marketing campaign, around 89% of the clients did not subscribe to a term deposit while only around 11% of them subscribed for a term deposit. As the ratio is approximately 1:8, we regard this as imbalance of class. We will treat this imbalance in the following sections by using an oversampling technique.

```
[ ]  #Checking for class imbalance
     class_values = (banks6['y'].value_counts()/banks6['y'].value_counts().sum())*100
     print(class_values)

     0    88.688389
     1    11.311611
     Name: y, dtype: float64
```

Note: The class distribution in the target is 89:11. This is a clear indication of imbalance.

Figure 39. Python code for checking class imbalance for the target variable.

12.2. Transforming Variables

To transform variables, we put the categorical variables into a list. The list include the variables ‘job’, ‘marital’, ‘education’, ‘default’, ‘housing’, ‘loan’, ‘contact’, and ‘month’. After that , we get the dummies of those variables. Dummy variables are the most common approach to add categorical variables as predictors in statistical and machine learning models (Bock, 2021).

```
[ ]  #Create a list of element containing the string 'purpose, job,etc.'. Call this list cat_list.
     cat_list = ['job','marital','education','default','housing','loan','contact','month']

[ ]  #Getting dummies
     new_banks = pd.get_dummies(banks6, columns = cat_list)

[ ]  banks6 = new_banks.dropna()
```

Figure 40. Python code for getting dummies.

After getting the dummies, there were 44 columns in consideration for modelling and analysis (see Figure 41).

```
banks6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 37351 entries, 0 to 45210
Data columns (total 45 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  37351 non-null  float64
 1   balance              37351 non-null  float64
 2   day                  37351 non-null  int64  
 3   campaign             37351 non-null  float64
 4   pdays                37351 non-null  int64  
 5   previous             37351 non-null  int64  
 6   y                    37351 non-null  int64  
 7   job_admin            37351 non-null  uint8  
 8   job_blue-collar      37351 non-null  uint8  
 9   job_entrepreneur     37351 non-null  uint8  
 10  job_housemaid        37351 non-null  uint8  
 11  job_management       37351 non-null  uint8  
 12  job_retired          37351 non-null  uint8  
 13  job_self-employed    37351 non-null  uint8  
 14  job_services         37351 non-null  uint8  
 15  job_student          37351 non-null  uint8  
 16  job_technician       37351 non-null  uint8  
 17  job_unemployed       37351 non-null  uint8  
 18  marital_divorced     37351 non-null  uint8  
 19  marital_married      37351 non-null  uint8  
 20  marital_single       37351 non-null  uint8  
 21  education_primary    37351 non-null  uint8  
 22  education_secondary  37351 non-null  uint8  
 23  education_tertiary   37351 non-null  uint8  
 24  default_no           37351 non-null  uint8  
 25  default_yes          37351 non-null  uint8  
 26  housing_no           37351 non-null  uint8  
 27  housing_yes          37351 non-null  uint8  
 28  loan_no              37351 non-null  uint8  
 29  loan_yes             37351 non-null  uint8  
 30  contact_cellular     37351 non-null  uint8  
 31  contact_no           37351 non-null  uint8  
 32  contact_telephone    37351 non-null  uint8  
 33  month_apr            37351 non-null  uint8  
 34  month_aug            37351 non-null  uint8  
 35  month_dec            37351 non-null  uint8  
 36  month_feb            37351 non-null  uint8  
 37  month_jan            37351 non-null  uint8  
 38  month_jul            37351 non-null  uint8  
 39  month_jun            37351 non-null  uint8  
 40  month_mar            37351 non-null  uint8  
 41  month_may            37351 non-null  uint8  
 42  month_nov            37351 non-null  uint8  
 43  month_oct            37351 non-null  uint8  
 44  month_sep            37351 non-null  uint8  
dtypes: float64(3), int64(4), uint8(38)
```

Figure 41. Python code for extracting some information about the data and columns.

12.3. Train Test Split

The train-test split procedure is used to measure the performance of machine learning algorithms when they make predictions on data that was not used to train the model.

Brownlee (2020) describes train-test split as a quick and simple strategy for comparing the performance of machine learning algorithms for your predictive modelling task. Although the method is simple to use and interpret, there are times when it should not be utilised, such as when you have a small dataset or when further configuration is needed, such as when it is used for classification and the dataset is not balanced.

We used random state 1 and a test size of 0.40 for this project.

```
[ ] #Split the datasets into training and test data
    X = banks6.drop('y', axis=1)
    y = banks6.y
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1, test_size= 0.40)

[ ] X_train.shape

    (22410, 44)

[ ] X_test.shape

    (14941, 44)

[ ] y_train.shape

    (22410,)

[ ] y_test.shape

    (14941,)
```

Figure 41. Python code for performing train test split.

12.4. Oversampling Technique

Based on the findings we had earlier (section 12.1), we found out that there was a clear indication of class imbalance since the result of the bank marketing campaign yielded to a 1:8 ratio, that is, 89% of clients did not subscribe for a term deposit compared to only 11% who subscribed. Confirming this again, we produced a bar plot of the results (Figure 42).

```
#Check how many clients subscribed for a term deposit
ax = banks6['y'].value_counts().plot(kind='bar', figsize=(12, 10), fontsize=13, color='#087E8B')
ax.set_title('Term deposit subscription', size=20, pad=30)
ax.set_ylabel('Count', fontsize=14)

for i in ax.patches:
    ax.text(i.get_x() + 0.19, i.get_height() + 700, str(round(i.get_height(), 2)), fontsize=15)
```

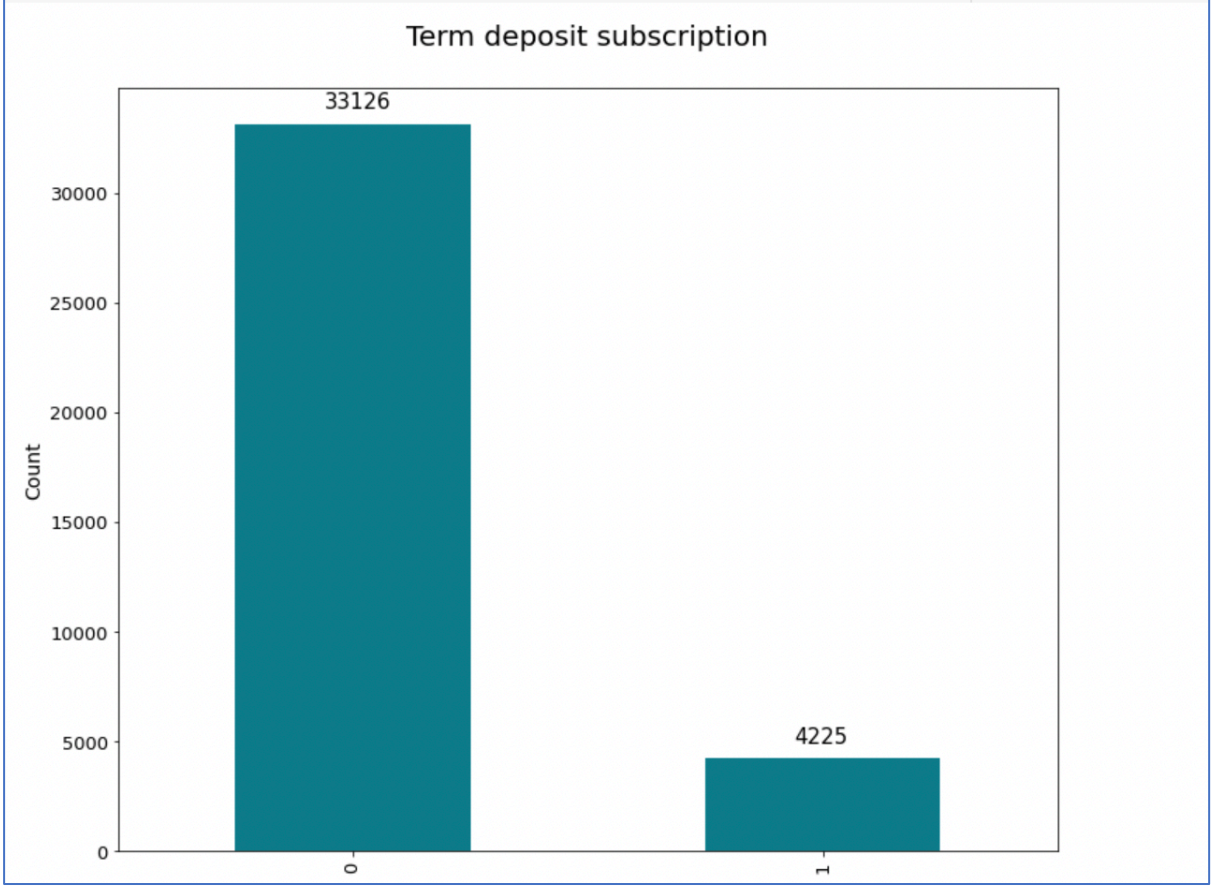


Figure 42. Python code for checking clients.

To handle the class imbalance we have on our target variable, we applied a an oversampling technique called Synthetic Minority Oversampling Technique (SMOTE).

The minority class was oversampled using SMOTE. This is a form of data augmentation approach that creates new samples from old ones. SMOTE selects a random sample, and then an algorithm selects neighbours to whom lines are drawn. We can produce as many synthetic samples as we need using this approach. As a result, SMOTE is ideal for large datasets. The only significant disadvantage is that synthetic examples are constructed without "consulting" the majority class. This could result in samples from both classes overlapping (Radecic, 2020).

```
#Applying smote to the imbalanced proportion
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)

X_sm, y_sm = sm.fit_resample(X, y)

print(f'''Shape of X before SMOTE: {X.shape}
Shape of X after SMOTE: {X_sm.shape}''')

print('\nBalance of positive and negative classes (%):')
y_sm.value_counts(normalize=True) * 100
```

```
Shape of X before SMOTE: (37351, 44)
Shape of X after SMOTE: (66252, 44)

Balance of positive and negative classes (%):
0    50.0
1    50.0
Name: y, dtype: float64
```

Figure 43. Python code for applying SMOTE to the imbalanced proportion.

12.5. Helper Function

To be even more prepared for the modeling part of the project, a helper function in python is coded so that we may reuse it in other areas of our code, allowing us to focus on the broader goals while keeping all of the smaller blocks of code in an implementable state. This makes the codes more comprehensible and understandable as well. For the helper function coded, we included functions that will print the different results of the metrics to be considered (to be discussed on the next section). Below are the following helper functions.

```
def model_perf_v1(model,X_train, X_val, y_train, y_val):

    pred_dt = model.predict(X_val)
    print("Accuracy on training set:")
    pred = model.predict(X_train)
    print(metrics.accuracy_score(y_true = y_train, y_pred = pred))
    #
    print("Accuracy on testing set:")
    accuracy = (metrics.accuracy_score(y_true = y_val, y_pred = pred_dt))
    print(accuracy)
    #confusion matrix
    confusion_matrix_ =  pd.crosstab(index=y_val, columns=pred_dt.ravel(), rownames=['Expected'], colnames=['Predicted'])
    #visualization
    sns.heatmap(confusion_matrix_, annot=True, square=False, fmt='', cbar=False)
    plt.title("Confusion Matrix", fontsize = 15)
    plt.show()
    #
    print("Recall:")
    recall = (metrics.recall_score(y_val,pred_dt))
    #recall_no = (metrics.recall_score(y_val,pred_dt))
    print(recall)
    #print(recall_no)
    # #
    print("Specificity:")
    tn, fp, fn, tp = confusion_matrix(y_val,pred_dt).ravel()
    spec = tn/(tn+fp)
    Specificity = (spec)
    print(Specificity)

    # #
    print("Precision:")
    Precision = (metrics.precision_score(y_val,pred_dt))
    print(Precision)
    # #
    print("Balanced Accuracy:")
    Balanced_Accuracy = (metrics.balanced_accuracy_score(y_val,pred_dt))
    print(Balanced_Accuracy)
    # #
    print("F1 score:")
    F1_score = (metrics.f1_score(y_val,pred_dt))
    print(F1_score)
    #classification_report
    # print(metrics.classification_report(y_test, pred_dt))
    return accuracy,recall,Specificity,Precision,F1_score,Balanced_Accuracy
```

Figure 44a. Python code for a Helper Function.

```python
def model_perf_to_lst(model,X_val, y_val):
    lst = [str(model)]
    pred_dt = model.predict(X_val)
    #print("Accuracy on testing set:")
    lst.append(metrics.accuracy_score(y_true = y_val, y_pred = pred_dt))
    #print("Recall:")
    lst.append(metrics.recall_score(y_val,pred_dt))
    #
    #print("Specificity:")
    tn, fp, fn, tp = confusion_matrix(y_val,pred_dt).ravel()
    spec = tn/(tn+fp)
    lst.append(spec)
    #
    #print("Precision:")
    lst.append(metrics.precision_score(y_val,pred_dt))
    #
    #print("Balanced Accuracy:")
    lst.append(metrics.balanced_accuracy_score(y_val,pred_dt))
    #
    #print("F1 score:")
    lst.append(metrics.f1_score(y_val,pred_dt))
    return lst

best_cl_normal = pd.DataFrame(columns = ['Model','Accuracy','Recall', 'Specificity', 'Precision', 'Balanced Accuracy', 'F1 score'])
```

Figure 44b. Python code for a Helper Function.

Model Exploration

13.0. Modeling Approach/Introduction

The approach to modeling in this investigation was divided in two groups, for both of them the set of techniques utilized were the same (Random Forest Classifier, Decision Tree Classifier, Logistic Regression Classifier, and Gradient Boost Classifier), with the difference that in the second group, we did a hyperparameter tuning.

Random Forest Classifier

Random Forest was the project's first modelling technique; it is an extremely handy method, because it can be used for both classification and regression tasks. It can easily handle binary and numerical variables, as well as categorical, without the need for transformation or scaling. in research from Yiu (2021). There are many decision tree classifier models in the Random Forest classifier model, and they work together as a unit.

Decision Tree Classifier:

Decision Tree is one of the more user-friendly and forgiving modelling tools, because they are straightforward to understand and interpret. They can be useful with or without obvious data, and any data only requires a minimal amount of preparation. Additions can be made to the existing trees.

Logistic Regression Classifier

Logistic Regression was the third modelling technique employed. It is more difficult to obtain high-quality data for Logistic Regressions than for Decision Trees Classifier and Random Forest Classifier, missing values are not handled well, despite the fact that Logistic Regressions accept incomplete data.

Gradient Boost Classifier

Gradient Boost Classifier is a methodology for improving the strength of a slight supposition or learning algorithm by making a series of adjustments. In research from Nelson (2022) As the previous methodologies mentioned above, the Gradient Boost can be used to predict or classify, depending on the analysis and business requirements.

One strength of this technique is the fact that its results are interpretable, and it works very well, when there is a lot of data and the interrelationships between them are not very complex.

13.1. Metrics to be Considered

Each classification model's performance is assessed using three statistical measures: classification accuracy, sensitivity, and specificity. It employs true positive (TP), true negative (TN), false positive (FP), and false negative (FN) signals (FN). The difference between the actual and anticipated values of variables is the proportion of correct/incorrect categorization. True Positive (TP) is the number of correct predictions that an instance is true, or when a classifier's positive prediction matches a target attribute's positive prediction. True Negative (TN) is a set of true predictions that an instance is false; it occurs when both the classifier and the target attribute indicate the

absence of a positive prediction. False Positive (FP) is the number of wrongly predicted true cases. Finally, the number of incorrect predictions that an instance is untrue is referred to as the False Negative (FN) (Vyas, 2019).

The number of correctly classified cases is used to measure *accuracy*. This number is equal to the sum of TP and TN divided by the total number of cases (TN + FN + TP + FP).

Recall is the ratio of the number of correct answers (TP) to the number of correct answers plus the number of wrong answers (FN).

Specificity is defined as the proportion of correctly categorized negatives and equals the ratio of TN to the sum of TN and FP.

Precision is the ratio of the number of true positives (TP) to the sum of the number of true positives and false positives (FP).

Balanced accuracy measures machine learning error for binary and multi-class models. It adjusts the conventional accuracy metric to perform better on imbalanced datasets. We can balance skewed datasets by averaging sensitivity and specificity.

F1 score is a harmonic mean of precision and recall, where 1 is best and 0 is worse. Precision and memory equally affect F1 score.

Moreover, we will consider AUC-ROC Curve as one of the metrics. This will be further discussed in the governance section of this paper.

14.0. Default Classifiers

In this section, we ran four different predictive modelling techniques – Random Forest Classifier, Decision Tree Classifier, Logistic Regression Classifier, and Gradient Boost Classifier. Procedure for running each model can be seen on the following sections.

14.1. Random Forest Classifier

In the evaluation of the Banking Marketing Campaign database, the number of estimators found were 100 and its depth has 38 leaves as Figure 45 shows.

```
print(len(rf.estimators_))
print(rf.estimators_[0].tree_.max_depth)
```

```
100
38
```

Figure 45. Python code for printing the number of estimators and maximum depth.

A figure of the tree would be impossible to read because of its size and depth, for this reason the plot has been limited to a depth of three leaves only (see Figure 46).

```
# Plotting random forest with 3 levels only
rf = RandomForestClassifier(n_estimators=100, max_depth=3)
rf.fit(X_sm,y_sm)
fig=plt.figure(figsize=(23,15))
_ = tree.plot_tree(rf.estimators_[0], feature_names=X.columns, filled=True,fontsize=10)
plt.title('Random Forest');
```

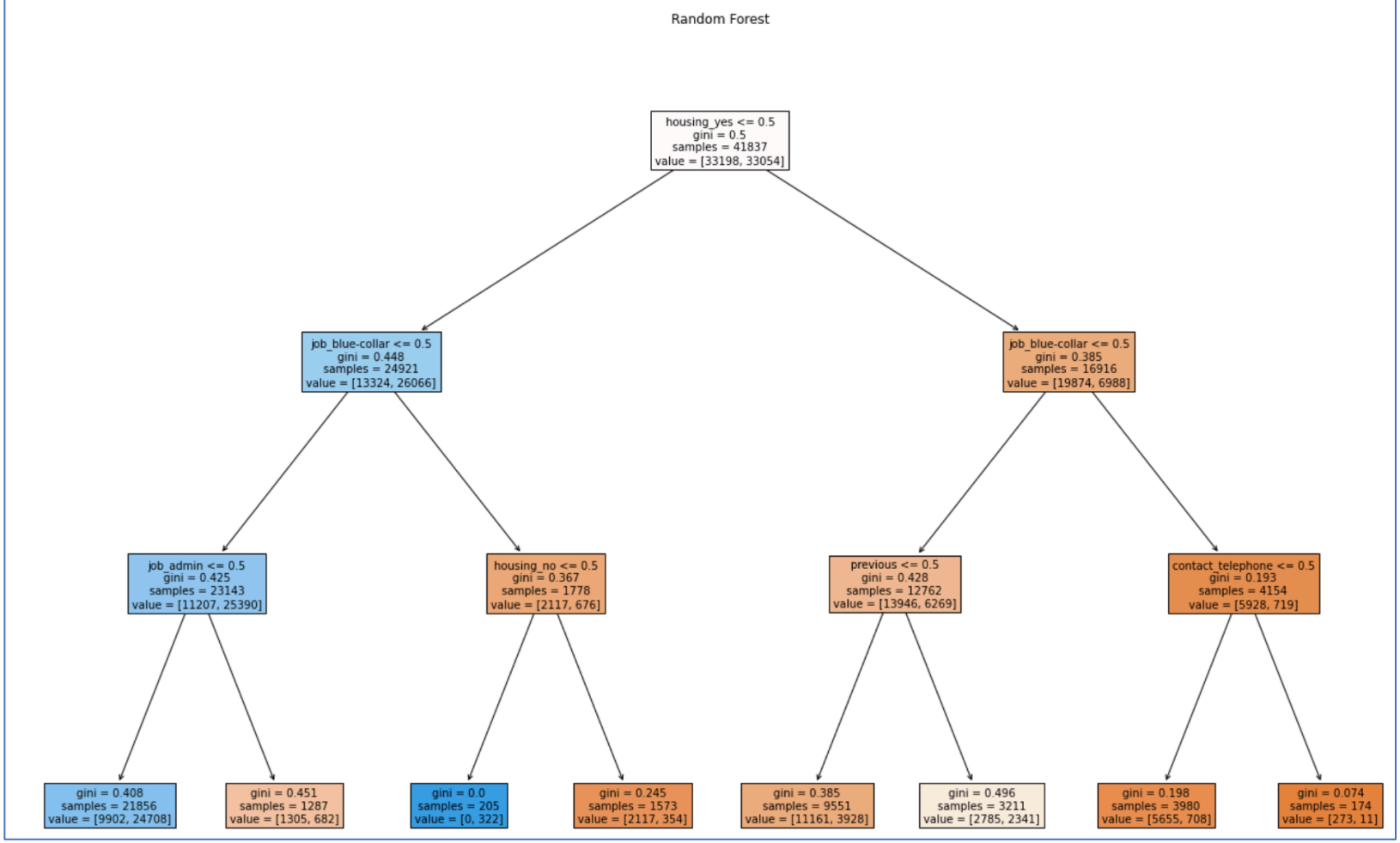
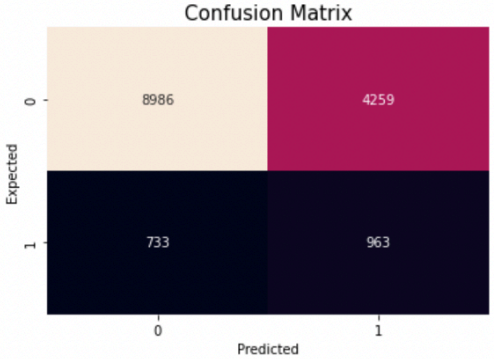


Figure 46. Random Forest Classifier plot with 3 levels.

```
rf = RandomForestClassifier()
#Fitting the model
rf.fit(X_sm,y_sm)
Accuracy_rf_sm, Recall_rf_sm, Specificity_rf_sm, Precision_rf_sm, F1_score_rf_sm, Balanced_Accuracy_rf_sm = model_perf_v1(rf,X_sm, X_test, y_sm, y_test)
rf_perf = model_perf_to_lst(rf, X_test, y_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  "X does not have valid feature names, but"
Accuracy on training set:
0.9999698122320836
Accuracy on testing set:
0.6658858175490262
```




```
Recall:
0.5678066037735849
Specificity:
0.6784446961117403
Precision:
0.18441210264266564
Balanced Accuracy:
0.6231256499426626
F1 score:
0.27840416305290544
```

Figure 47. Random Forest Classifier Confusion Matrix.

14.2. Decision Tree Classifier

```
# Plotting Decision Tree with 3 levels only
dt = DecisionTreeClassifier(max_depth=3)
fig=plt.figure(figsize=(23,15))
tree.plot_tree(decision_tree=dt.fit(X_sm, y_sm),feature_names=X.columns, filled=True,fontsize=10);
plt.title('Decision Tree');
```

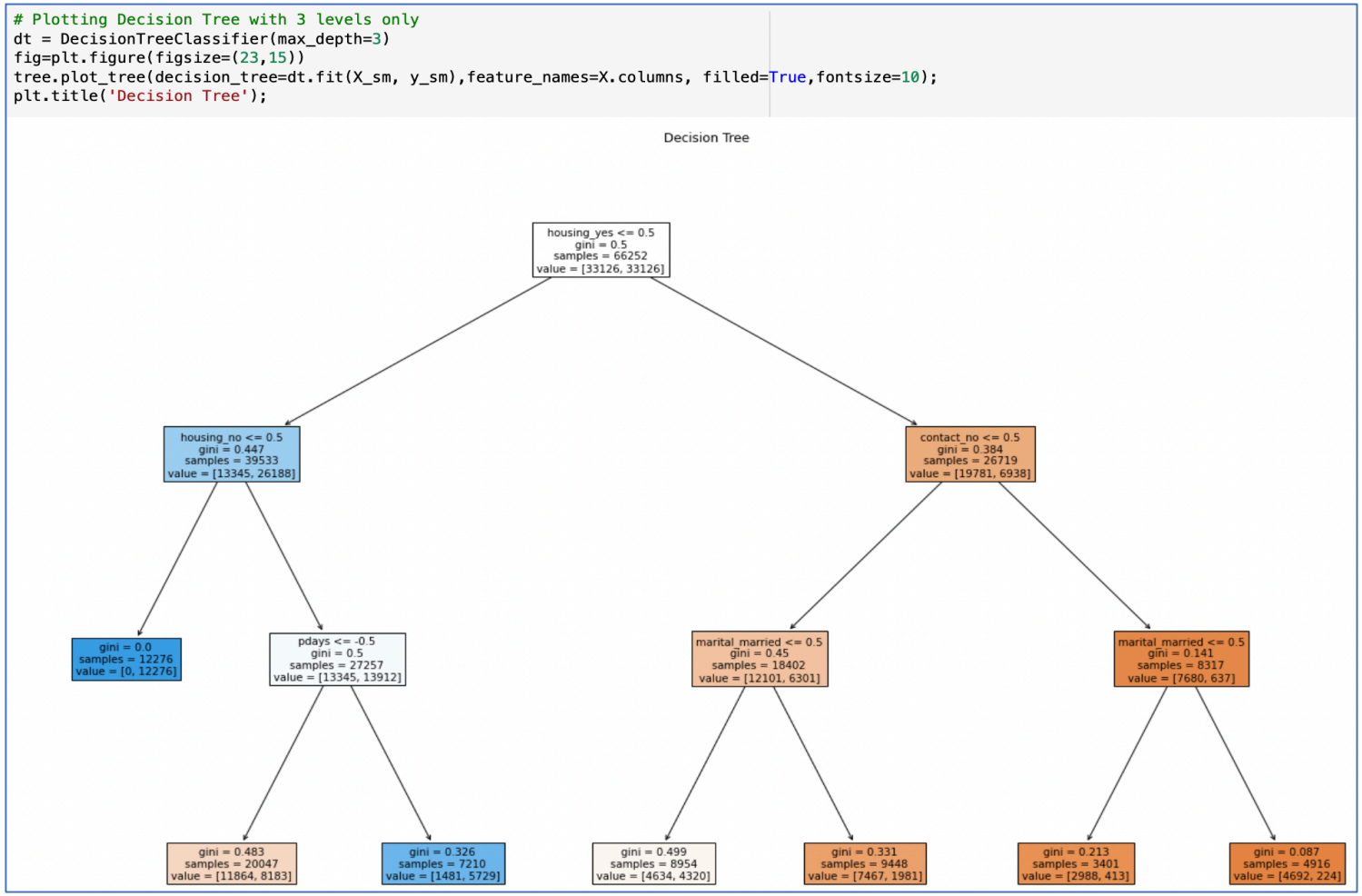
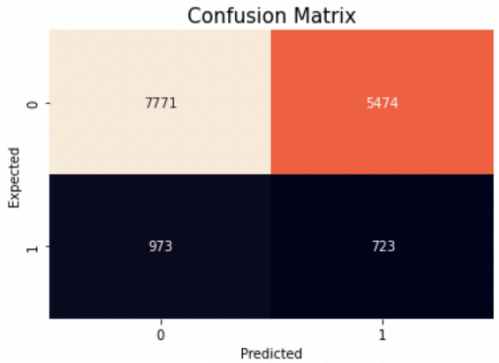


Figure 48. Decision Tree Classifier plot with 3 levels.

```
dt = DecisionTreeClassifier()
dt.fit(X_sm, y_sm)
Accuracy_dt_sm,Recall_dt_sm,Specificity_dt_sm,Precision_dt_sm,F1_score_dt_sm,Balanced_Accuracy_dt_sm = model_perf_v1(dt,X_sm, X_test,y_sm,y_test)
dt_perf = model_perf_to_lst(dt, X_test, y_test)
```

```
Accuracy on training set:
1.0
Accuracy on testing set:
0.5685027775918613
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
  "X does not have valid feature names, but"
```



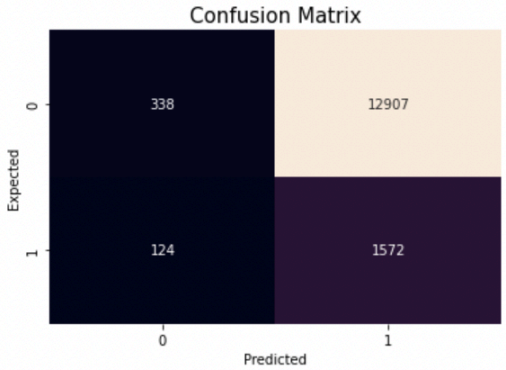

```
Recall:
0.42629716981132076
Specificity:
0.586711966779917
Precision:
0.11666935614006778
Balanced Accuracy:
0.5065045682956189
F1 score:
0.18320030406689475
```

Figure 49. Decision Tree Classifier Confusion Matrix.

14.3. Logistic Regression Classifier

```
log = LogisticRegression(max_iter=5000)
#Fitting the model
log.fit(X_sm, y_sm)
Accuracy_log_sm, Recall_log_sm, Specificity_log_sm, Precision_log_sm, F1_score_log_sm, Balanced_Accuracy_log_sm = model_perf_v1(log,X_sm,X_test,y_sm,y_test)
log_perf = model_perf_to_lst(log, X_test, y_test)
```

```
Accuracy on training set:
0.9338435066111211
Accuracy on testing set:
0.12783615554514424
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  "X does not have valid feature names, but"
```



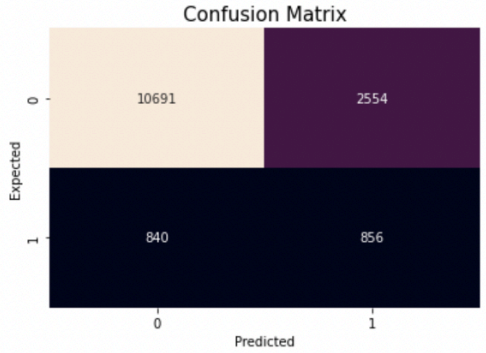

```
Recall:
0.9268867924528302
Specificity:
0.025519063797659496
Precision:
0.10857103391118171
Balanced Accuracy:
0.47620292812524484
F1 score:
0.19437403400309117
```

Figure 50. Logistic Regression Classifier Confusion Matrix.

14.4. Gradient Boost Classifier

```
grb = GradientBoostingClassifier()
grb.fit(X_sm, y_sm)
Accuracy_grb_sm, Recall_grb_sm, Specificity_grb_sm, Precision_grb_sm, F1_score_grb_sm, Balanced_Accuracy_grb_sm = model_perf_v1(grb,X_sm,X_test,y_sm,y_test)
grb_perf = model_perf_to_lst(grb, X_test, y_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but GradientBoostingClassifier was fitted with feature names
  "X does not have valid feature names, but"
Accuracy on training set:
0.9300851295055244
Accuracy on testing set:
0.7728398366909845
```




```
Recall:
0.5047169811320755
Specificity:
0.8071725179312949
Precision:
0.25102639296187684
Balanced Accuracy:
0.6559447495316852
F1 score:
0.33529181355268317
```

Figure 51. Gradient Boost Classifier Confusion Matrix.

15.0. Hyperparameter Tuning

The tuning of hyperparameters (optimization) is an important part of the machine learning process. A smart selection of hyperparameters can either help a model attain the intended metric value or lead to an endless loop of continuous training and tuning.

According to DeepAI (2020), hypertuned parameters are specified before the learning process begins. The parameters have an instantaneous effect on a model's trainability and are customizable. Grid Search with helper function was used to improve the project's hypertuned parameters. Scikit-learn (2022b) describes grid search as exhaustive supposition and validation, where it attempts all conceivable combinations of domain values to find the ideal combination.

For this project, hyperparameter tuning via grid search was applied. The hyperparameters' domain is grid-divided, and then we cross-validate every grid value combination to calculate performance metrics. The hyperparameter settings that maximize the cross-validation average are ideal. Grid search is an exhaustive technique that can locate the best location in the domain; it is pretty slow. Checking every space combination takes too much time and every grid point needs k-fold cross-validation and k training steps. Tuning a model's hyperparameters can be hard and costly but grid search is useful for finding the optimal hyperparameter values (Malato, 2021).

Earlier, we ran our models with the use of default classifiers (no hyperparameter tuning). For the following section, we ran the same predictive modeling techniques after performing a hyperparameter tuning.

A helper function is again constructed for conducting the grid search.

```
# Helper function for grid search
def grid_search_helper():

    pipeline1 = Pipeline((
    ('clf', DecisionTreeClassifier()),
    ))

    pipeline2 = Pipeline((
    ('clf', LogisticRegression()),
    ))

    pipeline3 = Pipeline((
    ('clf', GradientBoostingClassifier()),
    ))

    parameters1 = {
    'clf__min_samples_split': [5, 10, 20, 30, 40, 50],
    'clf__max_depth': list(range(1,15))
    }

    parameters2 = {
    'clf__C': np.logspace(0, 4, 10)
     }

    parameters3 ={
    'clf__n_estimators':[1, 2, 5, 10, 20, 50],
    'clf__learning_rate':[0.1, 0.3, 0.5, 0.7, 1]
    }

    pars = [parameters1, parameters2, parameters3]
    pips = [pipeline1, pipeline2, pipeline3]

    print("starting Gridsearch")
    dict_best_params ={}
    for i in range(len(pars)):
        print(pars[i])
        print(pips[i])
        gs = GridSearchCV(pips[i], pars[i], cv= 3, n_jobs=-1)
        gs.fit(X_train, y_train)
        print("finished Gridsearch\n")
        #print(gs.best_estimator_)
        dict_best_params[i]= gs.best_estimator_
    return dict_best_params

best_params_dict = grid_search_helper()
```

Figure 52. Python code for a Helper Function for Grid Search.

```
best_params_dict = grid_search_helper()

starting Gridsearch
{'clf__min_samples_split': [5, 10, 20, 30, 40, 50], 'clf__max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]}
Pipeline(steps=(('clf', DecisionTreeClassifier()),))
finished Gridsearch

{'clf__C': array([1.00000000e+00, 2.78255940e+00, 7.74263683e+00, 2.15443469e+01,
       5.99484250e+01, 1.66810054e+02, 4.64158883e+02, 1.29154967e+03,
       3.59381366e+03, 1.00000000e+04])}
Pipeline(steps=(('clf', LogisticRegression()),))
finished Gridsearch

{'clf__n_estimators': [1, 2, 5, 10, 20, 50], 'clf__learning_rate': [0.1, 0.3, 0.5, 0.7, 1]}
Pipeline(steps=(('clf', GradientBoostingClassifier()),))
finished Gridsearch
```

Figure 53. Python code for a Helper Function.

Similar to what we did above, we will apply the four predictive modeling techniques to the hypertuned parameters. Results for each of the model can be seen and will be discussed in the following section.

15.1. Random Forest Classifier

```
# 3 fold cross validation
k=3
params = dict(
    min_samples_split = [5, 10, 20, 30, 40, 50],
    max_depth = list(range(1,15)),
    n_estimators = [1, 2, 5, 10, 20, 50]
)
params
```

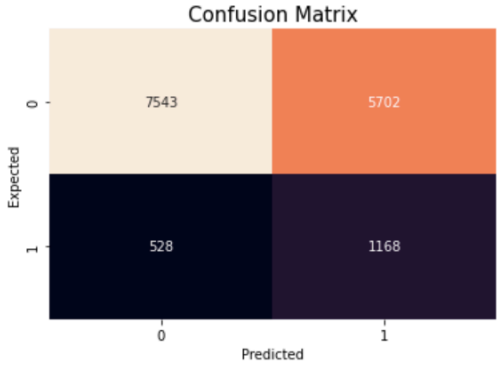
```
{'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14],
 'min_samples_split': [5, 10, 20, 30, 40, 50],
 'n_estimators': [1, 2, 5, 10, 20, 50]}
```

```
rf_1 = RandomForestClassifier()
rf_gs = GridSearchCV(estimator=rf_1, param_grid=params, cv=k, n_jobs=-1 )
# fitting the random forest model
rf_gs.fit(X_sm, y_sm)
best_estimator_rf = rf_gs.best_estimator_
```

Figure 54. Hypertuned Random Forest Classifier

```
Accuracy_rf_h, Recall_rf_h, Specificity_rf_h, Precision_rf_h, F1_score_rf_h, Balanced_Accuracy_rf_h = model_perf_v1(best_estimator_rf,X_sm,X_test,y_sm,y_test)
```

```
Accuracy on training set:
0.725864879550806
Accuracy on testing set:
0.5830265711799746
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  "X does not have valid feature names, but"
```




```
Recall:
0.6886792452830188
Specificity:
0.5694979237448093
Precision:
0.1700145560407569
Balanced Accuracy:
0.629088584513914
F1 score:
0.27270604716320335
```

Figure 55. Hypertuned Random Forest Classifier Confusion Matrix.

```
# Plotting random forest with 3 levels only
rf_1 = RandomForestClassifier(n_estimators=100, max_depth=3)
rf_1.fit(X_sm,y_sm)
fig=plt.figure(figsize=(23,15))
_rf1 = tree.plot_tree(rf_1.estimators_[0], feature_names=X.columns, filled=True,fontsize=10)
plt.title('Random Forest_Hypertuned');
```

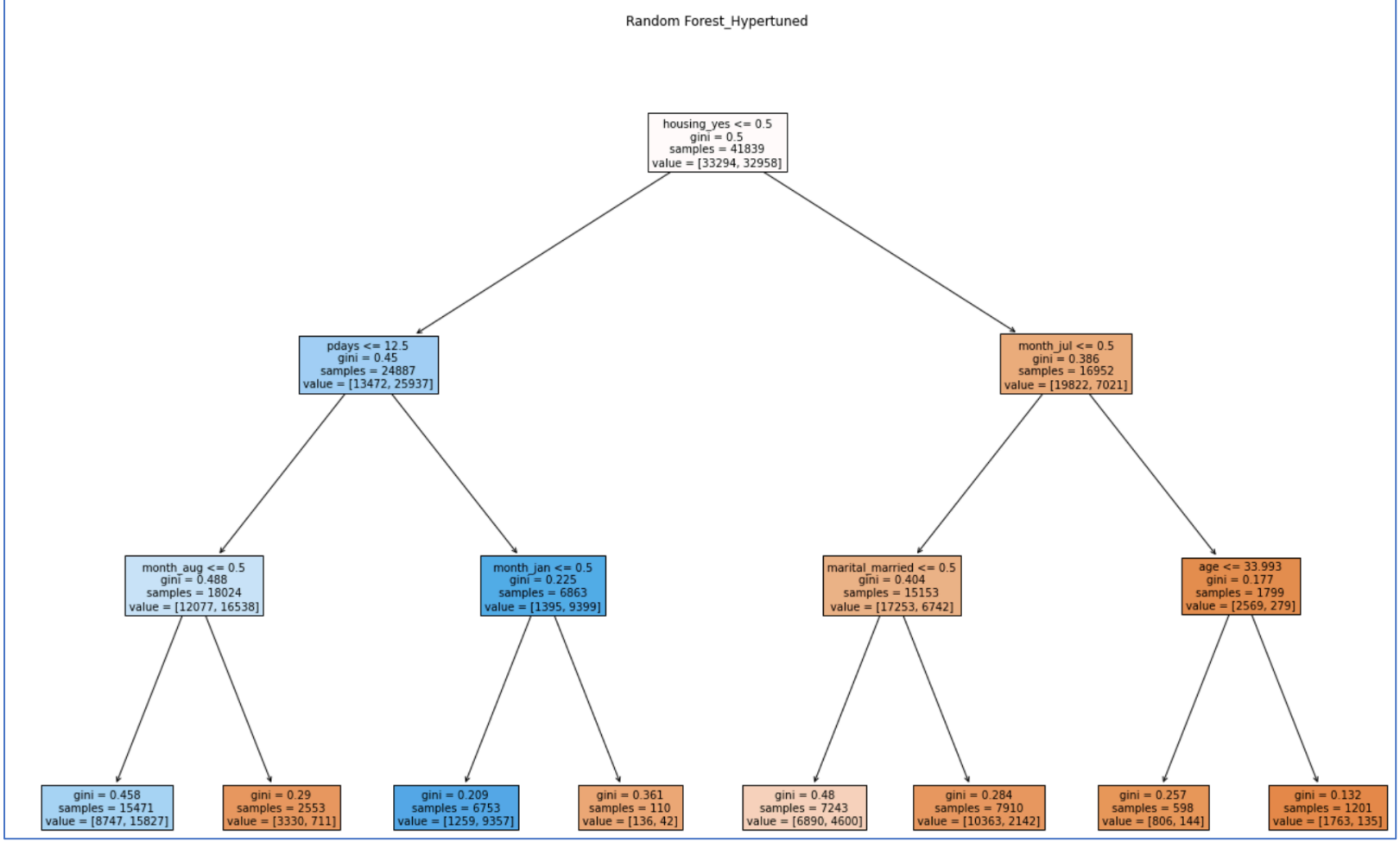
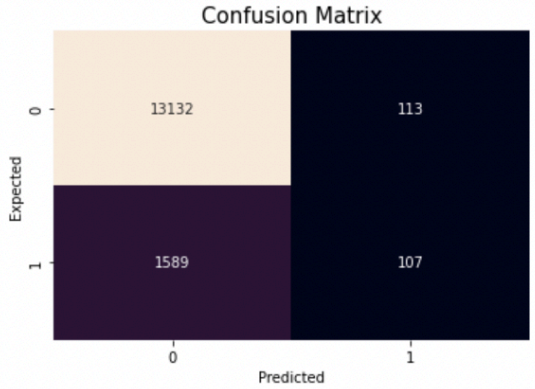


Figure 56. Hypertuned Random Forest Classifier Confusion Matrix.

15.2. Decision Tree Classifier

```
Accuracy_dt_h,Recall_dt_h,Specificity_dt_h,Precision_dt_h,F1_score_dt_h,Balanced_Accuracy_dt_h = model_perf_v1(best_params_dict[0],X_sm,X_test,y_sm,y_test)
```

```
Accuracy on training set:
0.5355159089536919
Accuracy on testing set:
0.8860852687236463
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: UserWarning: X has feature names, but DecisionTreeClassifier was fitted without feature names
  f"X has feature names, but {self.__class__.__name__} was fitted without"
```

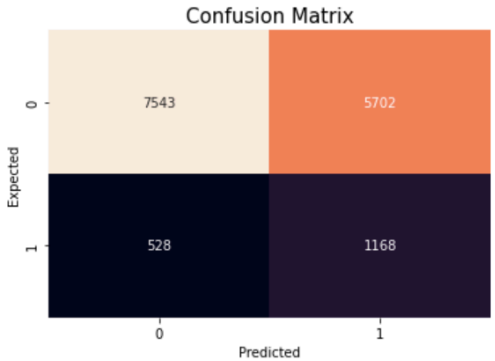



```
Recall:
0.06308962264150944
Specificity:
0.9914684786711967
Precision:
0.4863636363636364
Balanced Accuracy:
0.527279050656353
F1 score:
0.1116910229645094
```

Figure 57. Hypertuned Decision Tree Classifier.

15.3. Logistic Regression Classifier

```
Accuracy_rf_h, Recall_rf_h, Specificity_rf_h, Precision_rf_h, F1_score_rf_h, Balanced_Accuracy_rf_h = model_perf_v1(best_estimator_rf,X_sm,X_test,y_sm,y_test)

Accuracy on training set:
0.725864879550806
Accuracy on testing set:
0.5830265711799746
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  "X does not have valid feature names, but"
```



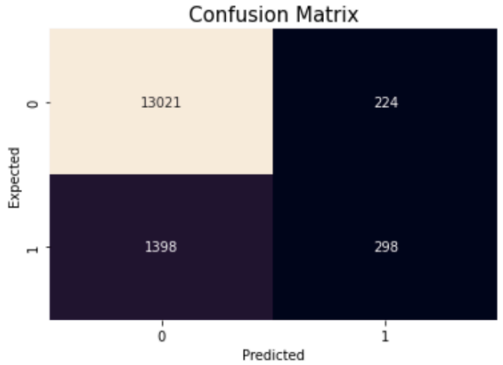

```
Recall:
0.6886792452830188
Specificity:
0.5694979237448093
Precision:
0.1700145560407569
Balanced Accuracy:
0.629088584513914
F1 score:
0.27270604716320335
```

Figure 58. Hypertuned Logistic Regression Classifier.

15.4. Gradient Boost Classifier

```
Accuracy_grb_h, Recall_grb_h, Specificity_grb_h, Precision_grb_h, F1_score_grb_h, Balanced_Accuracy_grb_h = model_perf_v1(best_params_dict[2],X_sm,X_test,y_sm,y_test)
```

```
Accuracy on training set:
0.4997132162047938
Accuracy on testing set:
0.8914396626731812
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: UserWarning: X has feature names, but GradientBoostingClassifier was fitted without feature names
  f"X has feature names, but {self.__class__.__name__} was fitted without"
```




```
Recall:
0.17570754716981132
Specificity:
0.9830879577198943
Precision:
0.5708812260536399
Balanced Accuracy:
0.5793977524448528
F1 score:
0.26871055004508565
```

Figure 59. Hypertuned Gradient Boost Classifier.

16.0. Model Comparison

As our modeling were divided into two groups, one that we use the default classifiers, and one that we used hypertuned parameters, we will compare each of the predictive modeling techniques based on the metrics mentioned in section 13.1.

16.1. Comparison for Default Classifiers

	Model	Accuracy	Recall	Specificity	Precision	Balanced Accuracy	F1 score
0	RandomForestClassifier()	0.665886	0.567807	0.678445	0.184412	0.623126	0.278404
1	DecisionTreeClassifier()	0.568503	0.426297	0.586712	0.116669	0.506505	0.183200
2	LogisticRegression(max_iter=5000)	0.127836	0.926887	0.025519	0.108571	0.476203	0.194374
3	GradientBoostingClassifier()	0.772840	0.504717	0.807173	0.251026	0.655945	0.335292

Figure 60a. Comparison table for the result of the models.

The Gradient Boost Classifier has the greatest accuracy with 0.772840, followed by the Random Forest Classifier with 0.665886 and the Decision Tree Classifier with 0.568503. The accuracy of the Logistic Regression Classifier was only 0.127836.

The accuracy metric yielded the same results as the specificity metric, with Gradient Boost Classifier ranking first with 0.807173, followed by Random Forest Classifier at 0.678445, Decision Tree Classifier at 0.586712, and Logistic Regression Classifier at 0.025519.

The metric balance accuracy ranking from highest to lowest is the same as the two metrics accuracy and specificity.

Logistic Regression Classifier has the highest Recall with 0.926887, and Decision Tree Classifier has the lowest with 0.426297.

In order of precision, we have Gradient Boosting Classifier (0.251026), Random Forest Classifier (0.184412), Decision Tree Classifier (0.116669), and Logistic Regression Classifier (0.108571), in that order.

Finally, the Gradient Boost Classifier had the greatest F1 score of 0.335292, while the Decision Tree Classifier had the lowest F1 score of 0.183200.

16.2. Comparison for Hypertuned Parameters

	classifier	accuracy	recall	specificity	precision	f1-score	balanced
0	RandomForest_h	0.583027	0.688679	0.569498	0.170015	0.272706	0.629089
0	DecisionTree_h	0.886085	0.063090	0.991468	0.486364	0.111691	0.527279
0	LogisticRegression_h	0.886554	0.075472	0.990411	0.501961	0.131215	0.532942
0	GradientBoosting_h	0.891440	0.175708	0.983088	0.570881	0.268711	0.579398

Figure 60b. Comparison table for the result of the models – Hyperparameter Tuning.

The accuracy of the various models improved after a hyperparameter adjustment using grid search. The Gradient Boost Classifier achieved the best accuracy and precision, respectively, of 0.8991440 and 0.570881, while the Random Forest Classifier had the highest recall, F1 score, and balanced accuracy of 0.688679, 0.27206, and 0.629089. The Decision Tree Classifier had the highest specificity with 0.991468, followed by the Logistic Regression Classifier with 0.990411.

Model Recommendation

17.0. Model Selection

Since we are considering the balanced data set after hyperparameter tuning, we consider accuracy as our primary metric for choosing the best model. Before hyperparameter tuning the parameters, we have a 1:8 ratio of classes. After applying the oversampling technique SMOTE, we achieved a 1:1 ratio of the classes. Thus, making the data set now balanced. Below is a summary of the accuracy score of all classification models.

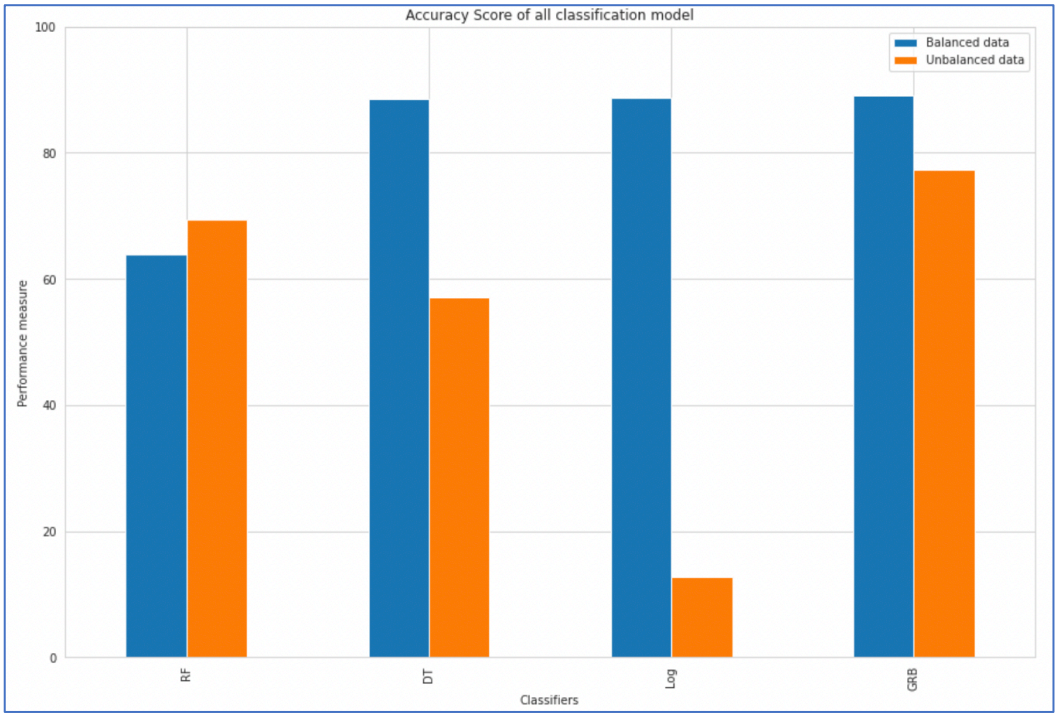


Figure 61. Accuracy score of all classification models.

Figure 61 clearly shows that the Gradient Boosting Classifier performed the best for using both the balanced data and unbalanced data. It can also be observed that all of the models except the Random Forest Classifier improved after balancing the data.

	classifier	accuracy	recall	specificity	precision	f1-score	balanced
0	RandomForest_h	0.583027	0.688679	0.569498	0.170015	0.272706	0.629089
0	DecisionTree_h	0.886085	0.063090	0.991468	0.486364	0.111691	0.527279
0	LogisticRegression_h	0.886554	0.075472	0.990411	0.501961	0.131215	0.532942
0	GradientBoosting_h	0.891440	0.175708	0.983088	0.570881	0.268711	0.579398

Figure 60b. Comparison table for the result of the models – Hyperparameter Tuning.

According to the figure above, the model Gradient Boost Classifier has the greatest accuracy of 0.8991440. Although we are only looking at metric accuracy, we will compare the Gradient Boost Classifier and Logistic Regression Classifier because they have a high accuracy of 0.886554 as well. In terms of precision, the Gradient Boost Classifier had the greatest with 0.570881 compared to Logistic Regression's precision of 0.131215. Gradient Boosting outperformed the Logistic Regression Classifier in terms of recall and recall and F1 score. Furthermore, considering the performance of all models for both the unbalanced data and balanced data after hyperparameter tuning through grid search, the Gradient Boost Classifier performed the best. As a result, we select the Gradient Boost Classifier as the best predictive model.

Conclusions and Recommendations

18.0. Conclusion

In this project, we created a classification model to predict whether or not a bank marketing campaign will be effective. Because the main focus was on prediction, our main goal was to create a predictive model that could properly forecast as many positive examples as possible without sacrificing too much accuracy.

We reached to the conclusion through the analysis that variables like education and martial status do effect the term deposit as we noticed that professionals with admin and blue collar occupations are more ready to accept the term deposit as these were categorical variables. Customers who are married are also more likely to accept the term deposit. When we look at education variables, we can find that customers with a higher educational degree are better educated in terms of comprehending the policies and terms of accepting the term deposit. Furthermore, clients with administrative jobs have subscribed to the bank deposit term more frequently. Based on our findings, we concluded that material status and education had an impact on term deposit.

Because the classes were extremely uneven, we had to evaluate the following performance metrics (in addition to the Accuracy score): Positive Class Precision, Recall, and F1-score.

To address the imbalanced classes, we used the SMOTE (Synthetic Minority Oversampling Technique) algorithm to oversample the minority class. We plotted ROC and Precision/Recall curves to increase classifier performance even further. We found no duplicates or incorrectly reported data, and the data quality was fair. There were

no missing values, although there was 'unknown' data from a few variables, as well as outliers.

With an overall accuracy of 89%, the chosen model (Gradient Boost Classifier) can accurately predict more than half of the observations belonging to the positive class. As a result, the model's performance can still be improved, particularly in terms of minority class predictions. Additional data is required for this purpose in order to train the model on a larger dataset and capture the previously unseen underlying correlations between variables.

19.0. Recommended Next Steps

We will try to develop a realistic model by removing the column 'duration' because there is no way a client will subscribe to a term deposit if the duration is zero. We can say that the duration attribute is strongly related to the target variable.

We are considering categorizing the age variable into distinct intervals of age groups and determining which interval age group subscribes to term deposits the most. Furthermore, hyperparameter adjustment will be beneficial because it will alter the model's parameters during the training phase. As a result, selecting the appropriate hyperparameters is critical. Concerning outliers, we must address them before dealing with missing numbers. It is recommended to conduct a deep-dive examination of the data for each variable because some of them may contain 'unknown' data that could skew the analysis. Outliers can also have an impact on the model's accuracy.

GOVERNANCE AND VALIDATION

20.0. Variable Level Monitoring

As we have seen from the data exploration part of this paper, we had variables with outliers and data that is unacceptable within the range of data we wanted. For example, ages above 70 were declared as outliers as most of our data are within the range of 18 to 70. For future marketing campaigns, we can target ages of the same range. In addition, we should try to develop a more realistic model by removing the column 'duration,' because there is no way a client will sign up for a term deposit if the duration is zero. We can say that the duration attribute is strongly related to the target variable. We must also investigate categorising the age variable into intervals of age groups and determining which interval age group subscribes to term deposits the most.

As for the missing values, we have observed that 4 of the variables have data containing ‘unknown’. For future marketing campaigns, it would be better if we don’t have a non-specific entries or answers.

21.0. Risks and Model Stability

Risks will develop when deploying machine learning models and algorithms if suitable governance mechanisms are not in place. The goal of this section is to assume and try to predict what risks may affect the model's results as well as the decision-making process. The risks could stem from three major sources: input data, algorithm design, and output data. Because we are dealing with a sale dataset that contains limited information about client characteristics, these hazards are classed as Minimal Risk.

Input

Let's go over the origins of our dataset and how it was collected and kept. The information was obtained from a Portuguese bank and included our clients' profiles and details, personal bank balances, credit histories, and responses to our marketing campaign. Some risks are possible:

- Since we are predicting term deposit subscription of current clients, information could change such as marital, education, age, and balance. Marital status could change anytime, as well as education and age. Similar to these variables, the balance variable is always changing depends on the spending behavior of clients, especially if they use this bank to purchase things. The result could be misleading and it is risky to apply it to the next marketing campaign. We need to update the data regularly or in real-time and retrain all the models for better precision and choose the right model for a better prediction.
- This dataset missed an important part which is the information about our customers' purchasing behaviors and previous balances when using their bank from time to time. It could be highly correlated to the target of subscription toward opening a term deposit.

Algorithm (AUC-ROC Curve)

We used another method to measure the performance of the classifier models called ROC curve or Receiver Operating Characteristic which shows the relationship between TPR or the proportion positive correctly classified (True Positive/ Positive) and False positive rate (FPR) is calculated by 1-TNR (True Negative/Negative) after

comparing the accuracy and precision of all the applied models and deciding to go with the Gradient Boosting Classifier.

The Area Under the Curve (AUC) of the four models is computed, as illustrated in the figures below. The AUC scores of each model are as follows, from highest to lowest: Gradient Boost Classifier (0.73), Random Forest Classifier (0.69), Decision Tree Classifier (0.50), and Logistic Regression Classifier (0.50). (0.43). The Gradient Boost model has the highest AUC while also being the most accurate and precise.

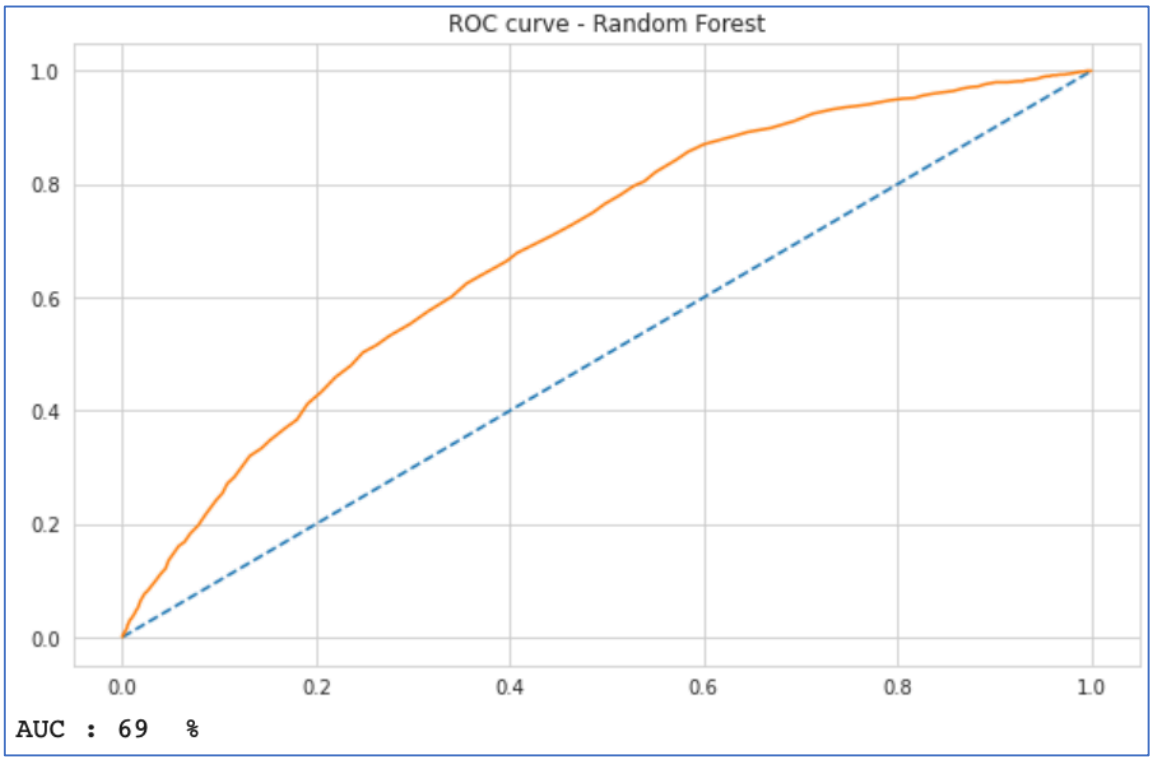


Figure 61. ROC Curve of the Random Forest Classifier.

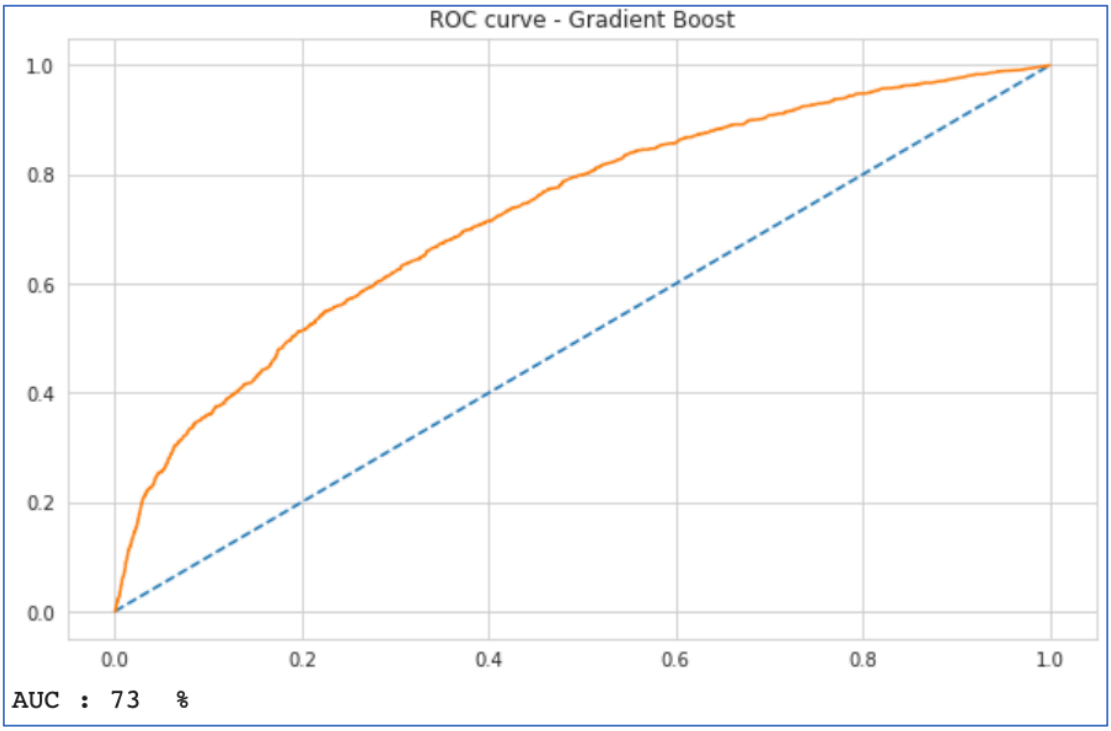


Figure 62. ROC Curve of the Gradient Boost Classifier.

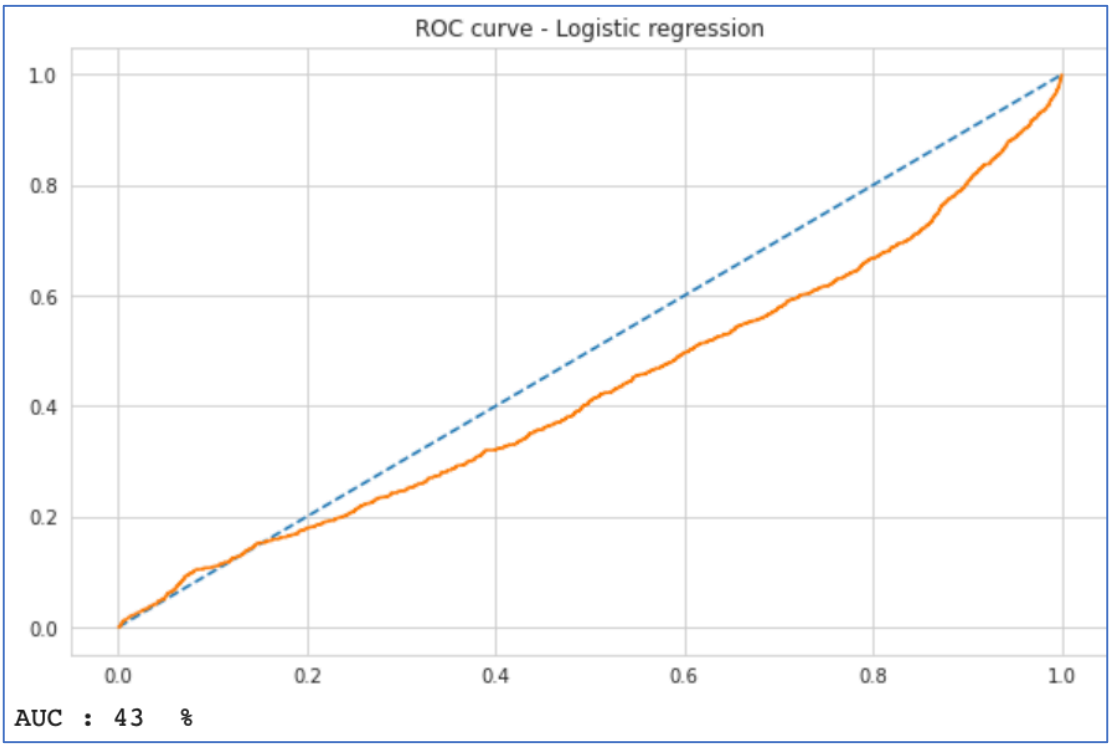


Figure 63. ROC Curve of the Logistic Regression Classifier.

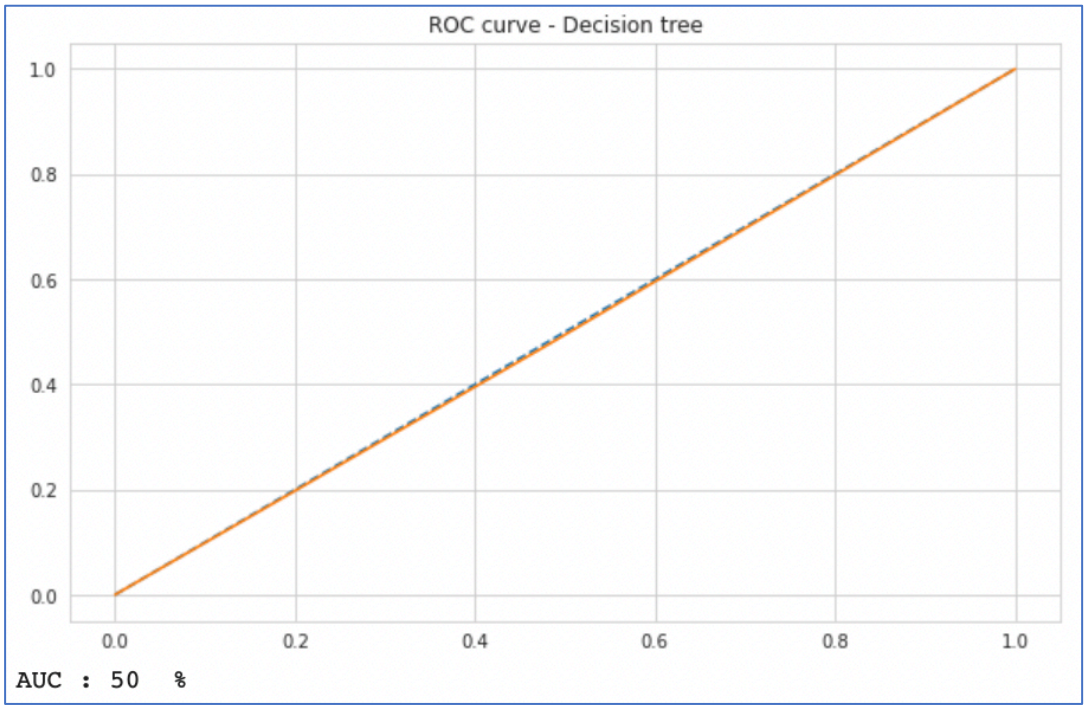


Figure 64. ROC Curve of the Decision Tree Classifier.

The chosen model performs well with this dataset and the current input data to forecast the next campaign. We advocate gathering more input data connected to each marketing campaign and retraining all models with the new input data for better prediction. It would be time-consuming and expensive because we would need to collect enough data to feed the model for retraining purposes.

Output

Because we are predicting the likelihood of subscription to a marketing campaign, yet the input data do not offer much information about the marketing campaign, applying this output to a completely different campaign may result in an incorrect conclusion. Furthermore, the use of this model may result in a poor success rate. As a result, there should be enough data input, and the application of our model's output should be relevant to the incoming data.

References

Brownlee, J. (2020, August 26). *Train-Test Split for Evaluating Machine Learning Algorithms*. Machine Learning Mastery. https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/

DeepAI. (2020, June 25). *Hyperparameter*. https://deepai.org/machine-learning-glossary-and-terms/hyperparameter

Malato, G. (2022, May 27). *Hyperparameter tuning. Grid search and random search*. Your Data Teacher. https://www.yourdatateacher.com/2021/05/19/hyperparameter-tuning-grid-search-and-random-search/

Moro, S., Cortez, P., & Rita, P. (2014). A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, *62*, 22–31. https://doi.org/10.1016/j.dss.2014.03.001

Nelson, D. (2022, July 21). *Gradient Boosting Classifiers in Python with Scikit-Learn*. Stack Abuse. https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/

NumPy. (2022). *numpy.logspace — NumPy v1.23 Manual*. NumPy Developers. https://numpy.org/doc/stable/reference/generated/numpy.logspace.html

Radečić, D. (2022, March 21). *How to Effortlessly Handle Class Imbalance with Python and SMOTE*. Medium. https://towardsdatascience.com/how-to-effortlessly-handle-class-imbalance-with-python-and-smote-9b715ca8e5a7

Scikit-learn. (2022a). *3.2.3.3.5. sklearn.ensemble.GradientBoostingClassifier — Scikit-learn 0.15-git documentation*. https://scikit-

learn.org/0.15/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

Scikit-learn. (2022b). *sklearn.ensemble.RandomForestClassifier*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Vyas, M. (2021, December 11). *Bank Marketing Campaign Prediction Project - Madhav Vyas*. Medium. https://medium.com/@madhavvyas_10553/bank-marketing-campaign-prediction-project-99fe1c3029a4

Yiu, T. (2021, December 10). *Understanding Random Forest - Towards Data Science*. Medium. https://towardsdatascience.com/understanding-random-forest-58381e0602d2