Capstone Project Report

Modeling to Predict the Subscription of Term Deposit

HINNA MARY STEEPHEN

Aug, 2022

Contents

1.0. Executive Summary

This report focuses on determining how to increase the effectiveness of the bank's telemarketing campaign. A detailed predictive analysis was undertaken as part of this project using python to develop a machine learning model for forecasting the variables influencing the term deposit subscription of the clients of a bank.

UCI Machine Learning Repository provided the dataset for this project, which included data related with direct marketing campaigns (phone calls) of a Portuguese banking institution to access if the bank term deposit would be subscribed.

After detailed analysis variables such as age, duration, campaign, pdays, previous, cons_conf_idx, euribor3m, housing, loan, marital, education, contact, poutcome were considered for the study and variables such as default, month, job, day_of_week, emp_var_rate, cons_price_idx, nr_employed were excluded from the study. The dataset contains a column or feature called y later renamed as subscription which has two values: yes and no which specifies whether a customer has subscribed to the term deposit in the previous campaign

Outcome: Based on the analysis, the Logistic Regression Model was selected as the best model to predict customers' response to the bank’s telemarketing campaign.

Recommendation: Based on the metrics, the telemarketing campaigns needs to focus their efforts on clients who are single, with an age between 30 and 50 and do not have a personal or housing loan. They should also have an educational qualification of degree or professional course

2.0. Introduction

2.1. Background

Portugal has one of the most sophisticated interbank networks in the world as part of its contemporary banking system. Portugal now has more than 150 banks. This comprises a variety of Portugal's newest mobile banks as well as public and cooperative banks, private national retail banks, and international banks. The Banco de Portugal, Portugal's central bank, also acts as the country's banking regulatory body. The majority of banks provide online banking services, and there are ATMs scattered over the nation. Almost any type of transaction is possible, including deposits (cash and checks), interbank transfers, and payments for services like mobile phones and the internet.

Term deposit is one of the services provided by the bank to its clients. A fixed-term investment known as a term deposit entails the deposit of money into an account at a financial institution for a fixed period. When purchasing a term deposit, the investor must be aware that they can only withdraw their money once the period has expired. If the investor gives a few days' notice, the account holder might in some situations permit an early termination—or withdrawal—of their investment. A fee will be charged for early termination as well.

2.2. Problem Statement

Bank has multiple products that it sells to customer such as saving account, credit cards, term deposit etc. Even today bank rely on telemarketing campaign to contact the clients and sell its products.

A Portuguese bank wants to determine which customer will purchase its term deposit, so that they can focus the marketing efforts into the client list who is most probable to subscribe to the term deposit. Using the information, the bank has about its clients and from the data collected from previous marketing campaigns, we need to predict the target client such that we can achieve maximum conversion rate i.e., get higher number of term deposit subscribers. This will increase the campaign performance.

2.3. Objective and Measurement

The objective of this project is to build a Machine learning Models and determine the best model that will predicts which client of the bank will subscribe to the term deposit at the bank. The measurements used to measure the model and rate its success are accuracy score and recall score.

2.4. Assumptions and Limitation

- Limitation - this model was built on data collected in 2014
- The recommendation is transferable to current business requirement.
- The data sample is representative of the population.
- It is ethically sourced data
- The data is reliable, original and comprehensive.
- All input variables are independent of each other

3.0. Data sources

3.1. Data Set Introduction

This data set is the outcome of a direct marketing campaigns of a Portuguese banking institution conducted in 2014 to access if the client would subscribe to the banks term deposit. This data set is downloaded from the open source's website: UCI Machine Learning Repository (S. Moro, P. Cortez and P. Rita, A Data-Driven Approach to Predict the Success of Bank Telemarketing June 2014). The data set includes 41188 rows and 20 columns that represent data collected from previous campaign. The detailed information is shown in Table 1:

3.2. Data Dictionary

No.	Type	Label	Description	Code
1	Integer	Age	Age of the client	
2	Categorical	Job	Type of job	admin., blue-collar, entrepreneur, housemaid, management, retired, self-employed, services, student, technician, unemployed, unknown
3	Categorical	Marital	Marital status	divorced, married, single, unknown; note:

				divorced means divorced or widowed
4	Categorical	Education	Educational Qualification of the client	basic.4y, basic.6y, basic.9y, high.school, illiterate, professional.course, university.degree, unknown
5	Categorical	Default	Does the client have any credit in default?	no, yes, unknown
6	Categorical	Housing	Does the client have any housing loan?	no, yes, unknown
7	Categorical	Loan	Does the client have any personal loan?	no, yes, unknown
Data related with the last contact of the current campaign				
8	Categorical	Contact	Contact communication type	cellular, telephone
9	Categorical	Month	last contact month of year	jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec

10	Categorical	day_of_week	last contact day of the week	mon, tue, wed, thu, fri
11	Integer	duration	last contact duration, in seconds	
Other attributes				
12	Integer	campaign	number of contacts performed during this campaign and for this client	
13	Integer	pdays	number of days that passed by after the client was last contacted from a previous campaign	999 means client was not previously contacted
14	Integer	previous	number of contacts performed before this campaign and for this client	
15	Categorical	poutcome	outcome of the previous marketing campaign	failure, nonexistent, success

Social and economic context attributes				
16	Float	emp.var.rate	employment variation rate	
17	Float	cons.price.idx	consumer price index	
18	Float	cons.conf.idx	consumer confidence index	
19	Float	euribor3m	euribor 3 month rate	
20	Integer	nr.employed	number of employees	
Output variable (desired target)				
21	Binary	Y	Has the client subscribed a term deposit	Yes, no

Table 1: Data Dictionary

3.3. Initial Data Preparation

3.3.1. Import Libraries for data Exploration

Pandas is a Python module used to expedite data cleaning, pre-processing, and analysis. Numpy, Python's mathematical library, is the foundation upon which Pandas is constructed. Make sure numpy is set up on your system before installing pandas. Python's Matplotlib toolkit provides a complete tool for building static, animated, and interactive visualizations. Matplotlib makes difficult things possible and simple things easy. A matplotlib-based Python data visualization library is called Seaborn. It offers a sophisticated drawing tool for creating eye-catching and educational statistical visuals.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Figure 1: Libraries for Data Exploration

3.3.2. Upload data file

```
#upload dataset
bankdf = pd.read_csv("bank-additional-full.csv")
bankdf.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	p
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	

5 rows × 21 columns

Figure 2: code to upload data ;view first five rows of dataset

3.3.3. Dimension and datatypes

The shape function helps understand the dimension of the data set. It helps us determine the number of columns and number of rows in the dataset. We are working with a dataset that has 41188 rows and 21 columns in this project. The info function gives us a summary of the data frame. We can conclude that the data set has float, int and object datatypes.

```python
print(bankdf.shape)
bankdf.info()
```

```
(41188, 21)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

Figure 3: code to see the number of rows, columns and datatype of each column

4.0. Data Cleaning

4.1. Standardized nomenclature

Column names are standardized to ensure consistent usage and reduce error when referring to the column in the code. Looking at the column names we see that the column names are well structured except for the usage of both dot and dash in the nomenclature. We are going to rename the columns to have only dash. We are also going to rename the dependent variable from ‘y’ to subscription to give a more meaningful name.

```
bankdf.columns

Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

Figure 4: code to see the column name

```
[162] bankdf.columns = [c.replace('.','_') for c in bankdf.columns]
```

```
[163] # rename y to subscription
      bankdf.rename(columns={'y': 'subscription'}, inplace=True)
```

Figure 5: code to rename the columns

```
[89] bankdf.columns

Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp_var_rate', 'cons_price_idx',
       'cons_conf_idx', 'euribor3m', 'nr_employed', 'subscription'],
      dtype='object')
```

Figure 6: code to see the renamed columns

4.2. Remove duplicates

To minimize redundancy, we need to remove duplicate entries. Checking for duplicates we see that there are 12 duplicate copies of rows already existing. After removing the duplicates there are 41176 rows to records remaining in the data set.

```
# check duplicate - this code is only returning a single entry(1 copy) of the duplicate rows
duplicateRows = bankdf[bankdf.duplicated()]
duplicateRows
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week
1266	39	blue-collar	married	basic.6y	no	no	no	telephone	may	thu
12261	36	retired	married	unknown	no	no	no	telephone	jul	thu
14234	27	technician	single	professional.course	no	no	no	cellular	jul	mon
16956	47	technician	divorced	high.school	no	yes	no	cellular	jul	thu
18465	32	technician	single	professional.course	no	yes	no	cellular	jul	thu
20216	55	services	married	high.school	unknown	no	no	cellular	aug	mon
20534	41	technician	married	professional.course	no	yes	no	cellular	aug	tue
25217	39	admin.	married	university.degree	no	no	no	cellular	nov	tue
28477	24	services	single	high.school	no	yes	no	cellular	apr	tue
32516	35	admin.	married	university.degree	no	yes	no	cellular	may	fri
36951	45	admin.	married	university.degree	no	no	no	cellular	jul	thu
38281	71	retired	single	university.degree	no	no	no	telephone	oct	tue

12 rows × 21 columns

Figure 7: checking for duplicates

```
[91] bankdf = bankdf.drop_duplicates()
     bankdf.shape

     (41176, 21)
```

Figure 8:Drop duplicate

5.0. Data Exploration

5.1. Missing values

First, we need to check to see if the dataset has any null values. It is determined that this dataset has no null or NaN values. Looking into the unique values of the categorical value we see that certain variables has values as unknown which needs to be further investigated

```
# checking for missing values
bankdf.isnull().sum()
```

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp_var_rate      0
cons_price_idx    0
cons_conf_idx     0
euribor3m         0
nr_employed       0
subscription      0
dtype: int64
```

Figure 9: checking for missing values

```
obj_col = []
for i, x in enumerate(bankdf.dtypes.tolist()):
    if x == 'object':
        obj_col.append(bankdf.columns[i])

for x in obj_col:
    print(f'Unique Values in {x}:', bankdf[x].unique())
```

```
Unique Values in job: ['housemaid' 'services' 'admin.' 'blue-collar' 'technician' 'retired'
 'management' 'unemployed' 'self-employed' 'unknown' 'entrepreneur'
 'student']
Unique Values in marital: ['married' 'single' 'divorced' 'unknown']
Unique Values in education: ['basic.4y' 'high.school' 'basic.6y' 'basic.9y' 'professional.course'
 'unknown' 'university.degree' 'illiterate']
Unique Values in default: ['no' 'unknown' 'yes']
Unique Values in housing: ['no' 'yes' 'unknown']
Unique Values in loan: ['no' 'yes' 'unknown']
Unique Values in contact: ['telephone' 'cellular']
Unique Values in month: ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'sep']
Unique Values in day_of_week: ['mon' 'tue' 'wed' 'thu' 'fri']
Unique Values in poutcome: ['nonexistent' 'failure' 'success']
Unique Values in subscription: ['no' 'yes']
```

Figure 10: Printing unique values in categorical variable

5.2. Data Exploration of dependent variable

The dependent variable of this project is subscription. Looking at the count plot and unique values of the dependent variable we can make the following inference:

- The dependent variable has 2 unique values: Yes and no
- There are 36537 records where the entry is no and 4639 entries is yes.
- Looking at the count plot we can see that the dataset is unbalanced as more that 85% entries have no in it.

```python
# unique values in subscription variable

print(f'number of unique values:{bankdf.subscription.nunique()}')
print(f'Unique values: {bankdf.subscription.unique()}')
print(f'count:- \n{bankdf.subscription.value_counts()}')


fig = plt.gcf()
fig.set_size_inches(5, 5)
sns.countplot(x ='subscription', data = bankdf)
# Show the plot
plt.show()
```

```
number of unique values:2
Unique values: ['no' 'yes']
count:-
no     36548
yes     4640
Name: subscription, dtype: int64
```

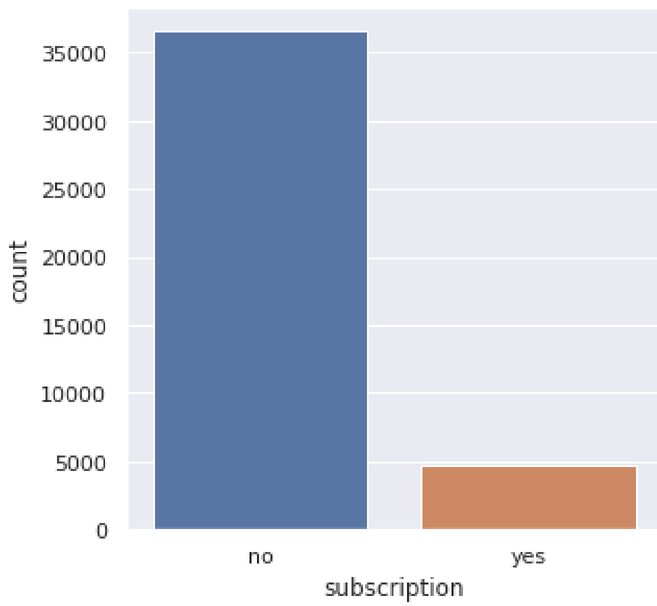


Figure 11: EDA of subscription variable

5.3. Data Exploration of categorical independent variables

As part of the data exploratory analysis of the categorical variable we are looking into the unique values and count plot of each variable. We are also looking at bivariant plot of each variable with the dependent variable to make some valuable inference

5.3.1 Data exploration of job variable

The job variable gives information about the profession of the client. The following inference can be made looking at the job variable:

- The job variable has 12 unique values: admin., blue-collar, entrepreneur, housemaid, management, retired, self-employed, services, student, technician, unemployed, unknown
- Looking at the count plot we can see that over 10000 calls were made to clients working as an admin and the least calls were made to students and unknown
- Looking at the bivariate plot we can see that we have got maxing term deposit conversion from people working as admin and least conversion from entrepreneurs, housemaids, self-employed, unemployed and unknown. We can also see that although the least number of calls were made to students, they have subscribed to term deposit more than entrepreneurs, housemaids, self-employed, unemployed and unknown.
- Since there is a possibility that people of other profession could have denied to give their professional information, we will treat unknown in this case as a unique value.

```
# unique values in job variable

print(f'number of unique values:{bankdf.job.nunique()}\n')
print(f'Unique values: {bankdf.job.unique()}\n')
print(f'count:- \n{bankdf.job.value_counts()}\n')

plt.figure(figsize=[30,12])

plt.subplot(2,2,1)
sns.countplot(x ='job',hue = 'subscription', data = bankdf)
plt.subplot(2,2,2)
sns.countplot(x ='job', data = bankdf)
```

Figure 12: code for job variable data exploration

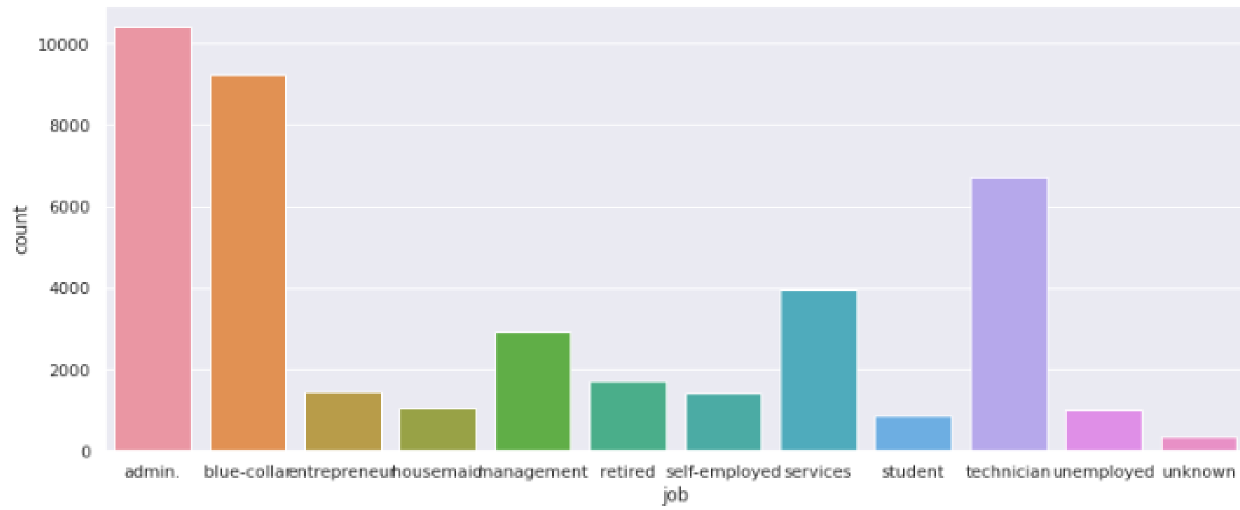


Figure 13: Count plot of job variable

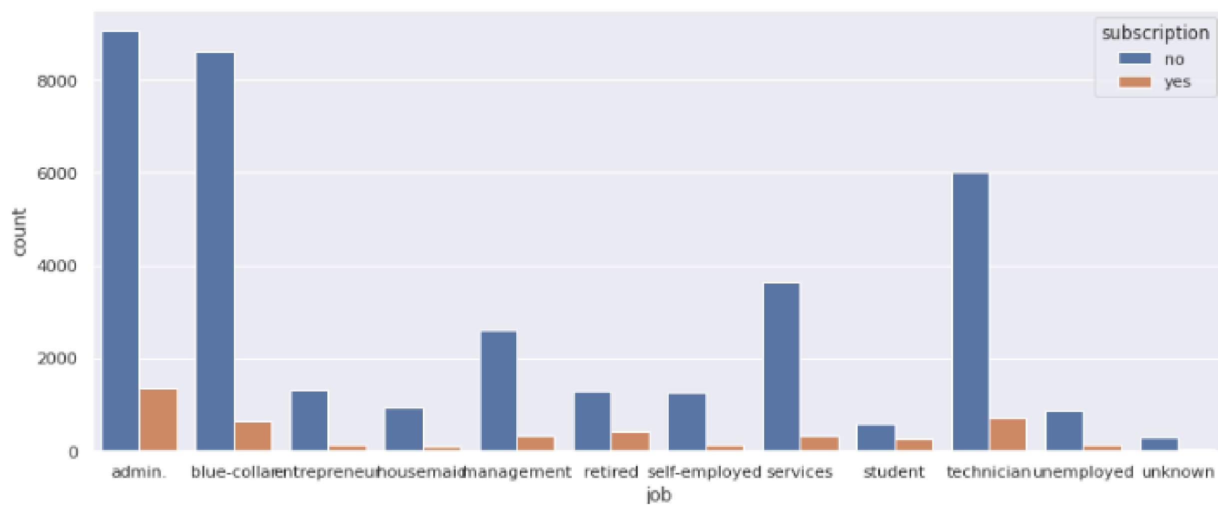


Figure 14: Bivariate count plot of job with subscription

5.3.2. Data exploration of marital variable

The marital variable gives information about the marital status of the client. The following inference can be made looking at the marital variable:

- The marital variable has 4 unique values: married, single, divorced, unknown

- Looking at the count plot we can see that over 20000 calls were made to who are married and the least calls were made to clients who were divorced.
- There are 80 entries where the marital status of the client is unknown. Since the number is very low, we will be removing these entries from our study.
- Looking at the bivariate plot we can see that we have got maximum term deposit conversion from people who are married and least conversion from divorcees

```
[389] # unique values in marital variable

      print(f'number of unique values:{bankdf.marital.nunique()}\n')
      print(f'Unique values: {bankdf.marital.unique()}\n')
      print(f'count:- \n{bankdf.marital.value_counts()}\n')
      print("count-groupBy :\n",bankdf.groupby(['subscription','marital']).size())

      #plot
      plt.figure(figsize=[30,12])

      plt.subplot(2,2,1)
      sns.countplot(x ='marital',hue = 'subscription', data = bankdf)
      plt.subplot(2,2,2)
      sns.countplot(x ='marital', data = bankdf)
```

Figure 15: code for marital variable data exploration

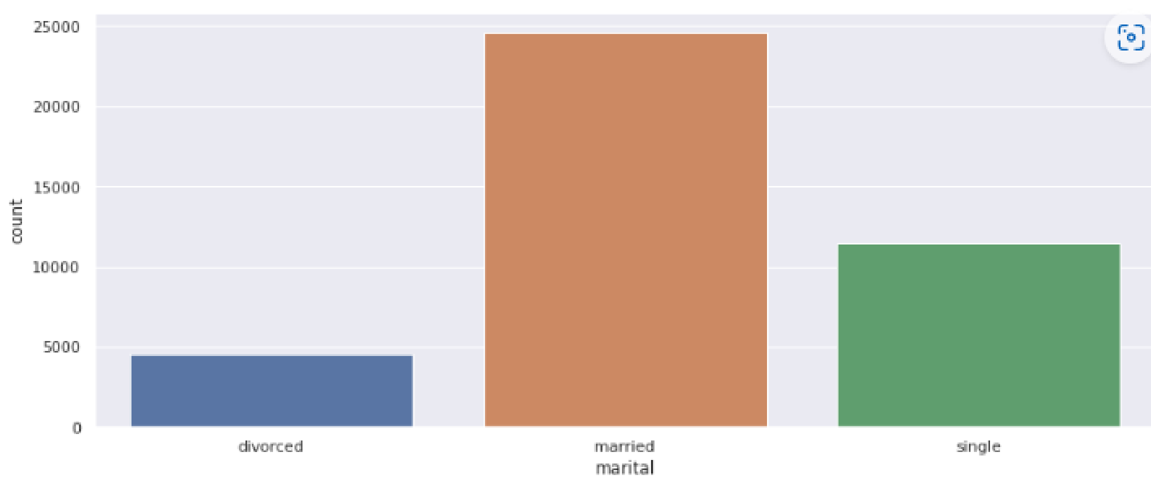


Figure 16: Count plot of marital variable

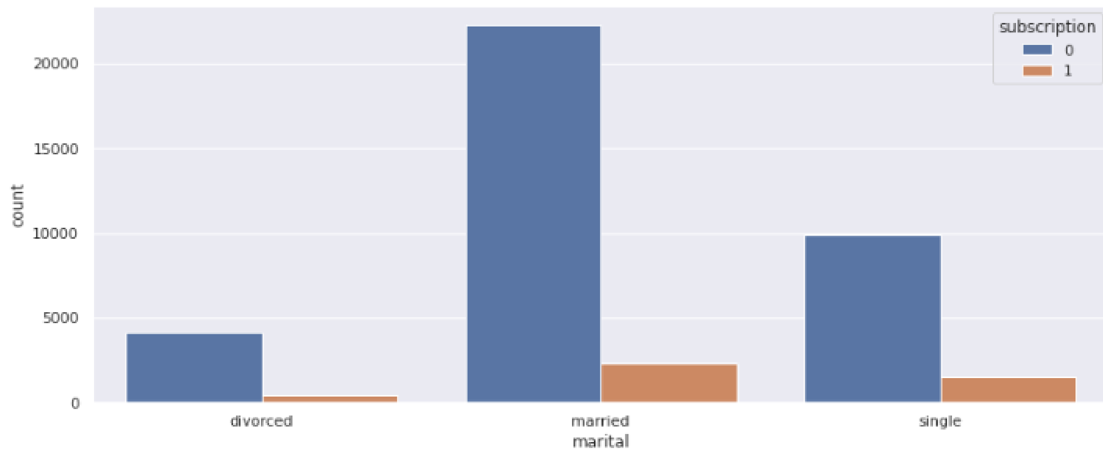


Figure 17: Bivariate count plot of marital with subscription

5.3.3. Data exploration of education variable

The education variable gives information about the education level of the client. The following inference can be made looking at the marital variable:

- The education variable has 8 unique values: basic.4y, basic.6y, basic.9y, high. School, illiterate, professional.course, university.degree, unknown
- Looking at the count plot we can see that over 12000 calls were made to clients with university degree and the least calls were made to clients who were illiterate.
- There are 1730 entries where the education level of the client is unknown. Since there is a possibility that some people could have denied to give their educational information, we will treat unknown in this case as a unique value.
- Looking at the bivariate plot we can see that we have got maximum term deposit conversion from people who has university degree and least conversion from illiterates

```python
print(f'number of unique values:{bankdf.education.nunique()}\n')
print(f'Unique values: {bankdf.education.unique()}\n')
print(f'count:- \n{bankdf.education.value_counts()}\n')
print("count-groupBy :\n",bankdf.groupby(['subscription','education']).size())

plt.figure(figsize=[30,12])

plt.subplot(2,2,1)
sns.countplot(x ='education',hue = 'subscription', data = bankdf)
plt.subplot(2,2,2)
sns.countplot(x ='education', data = bankdf)
```

Figure 18: code for education variable data exploration

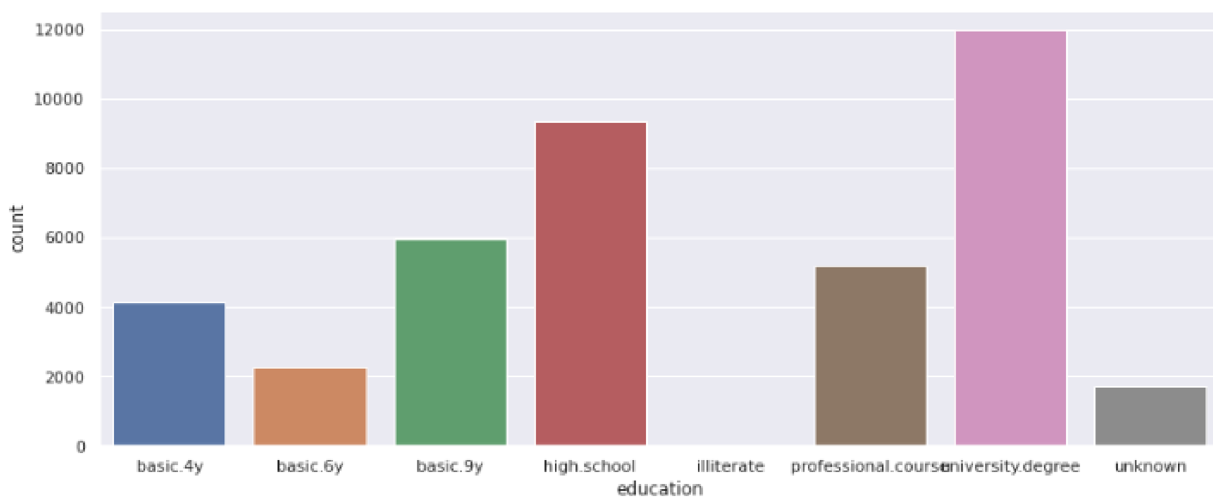


Figure 19: Count plot of education variable

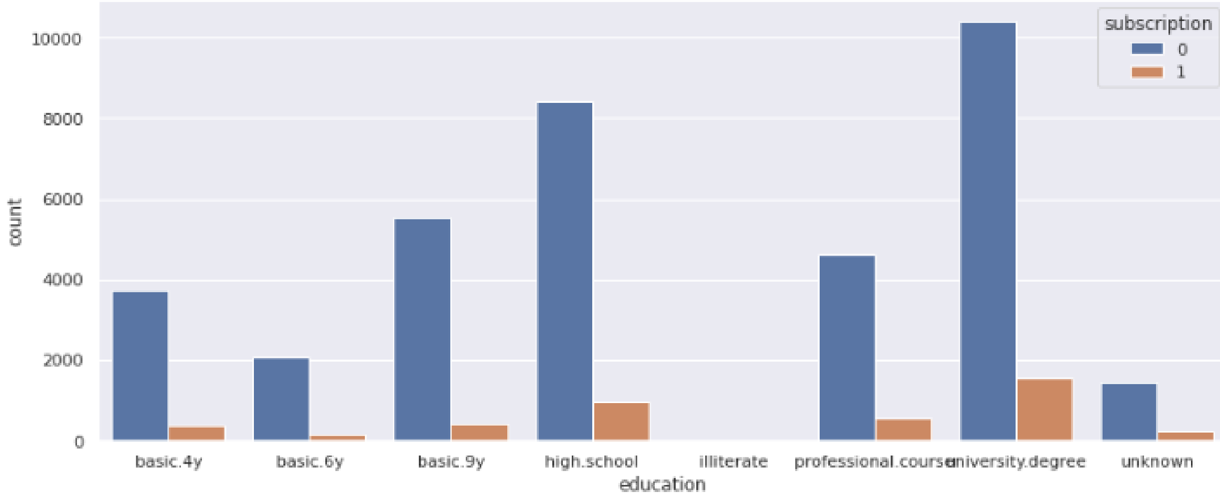


Figure 20: Bivariate count plot of education with subscription

5.3.4. Data exploration of day_of_week variable

The day_of_week variable gives information about the day of the week when the client was contacted last. The following inference can be made looking at the day_of_week variable:

- The day_of_week variable has 5 unique values: mon, tue, wed, thu, fri
- Looking at the count plot we can see that on all the week days around 8000 clients were contacted
- Looking at the bivariate plot we can see that nearly same number of clients have subscribed to the term deposit on all the days
- Since this variable does not vary much between the values. It's considered to be near zero variance.
- Removing this variable since this variable will add little value to the algorithm

```python
print(f'number of unique values:{bankdf.day_of_week.nunique()}\n')
print(f'Unique values: {bankdf.day_of_week.unique()}\n')
print(f'count:- \n{bankdf.day_of_week.value_counts()}\n')

plt.figure(figsize=[30,12])

plt.subplot(2,2,1)
sns.countplot(x ='day_of_week',hue = 'subscription', data = bankdf)
plt.subplot(2,2,2)
sns.countplot(x ='day_of_week', data = bankdf)
```

Figure 21: code for day_of_week variable data exploration

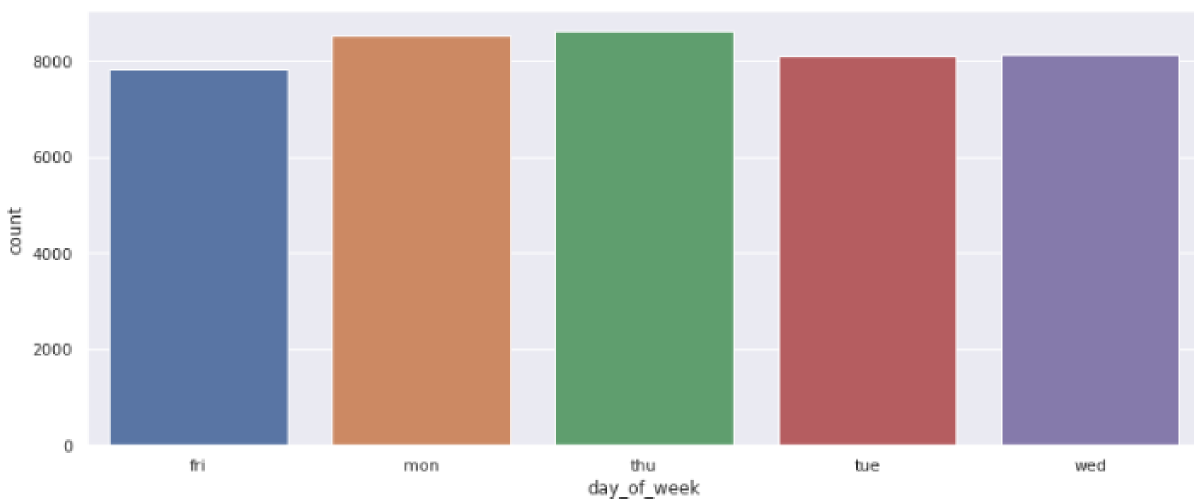


Figure 22: Count plot of day_of_week variable

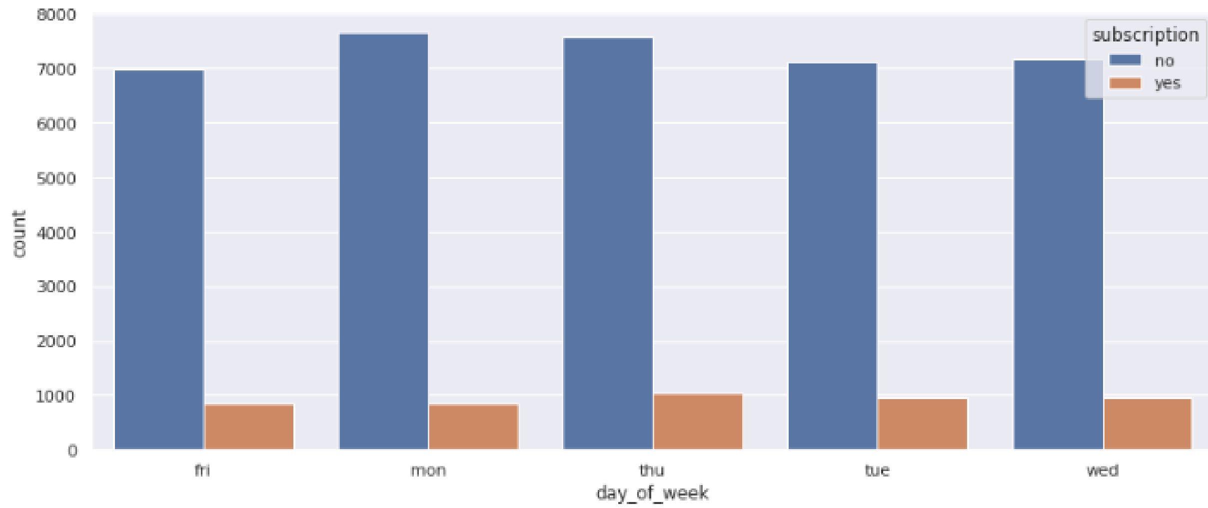


Figure 23: Bivariate count plot of day_of_week with subscription

5.3.5. Data exploration of default variable

The default variable gives information whether the client has credit in default. The following inference can be made looking at the default variable:

- The default variable has 3 unique values: yes, no, unknown
- Looking at the count plot we can see that 32577 clients has no default; 8596 clients default information is unknown and 3 clients had credit default.
- However, information regarding default should be available with the bank as credit is one of the service the bank provides and the credit history should be available with the bank
- Since only 3 entries are yes and all other entries are no, this variable is also considered as a near zero variance.
- Removing this variable since this variable will add little value to the algorithm

```
[325] print(f'number of unique values:{bankdf.default.nunique()}\n')
      print(f'Unique values: {bankdf.default.unique()}\n')
      print(f'count:- \n{bankdf.default.value_counts()}\n')
      print("count-groupBy :\n",bankdf.groupby(['subscription','default']).size())

      plt.figure(figsize=[30,12])

      plt.subplot(2,2,1)
      sns.countplot(x ='default',hue = 'subscription', data = bankdf)
      plt.subplot(2,2,2)
      sns.countplot(x ='default', data = bankdf)
```

Figure 24: code for default variable data exploration

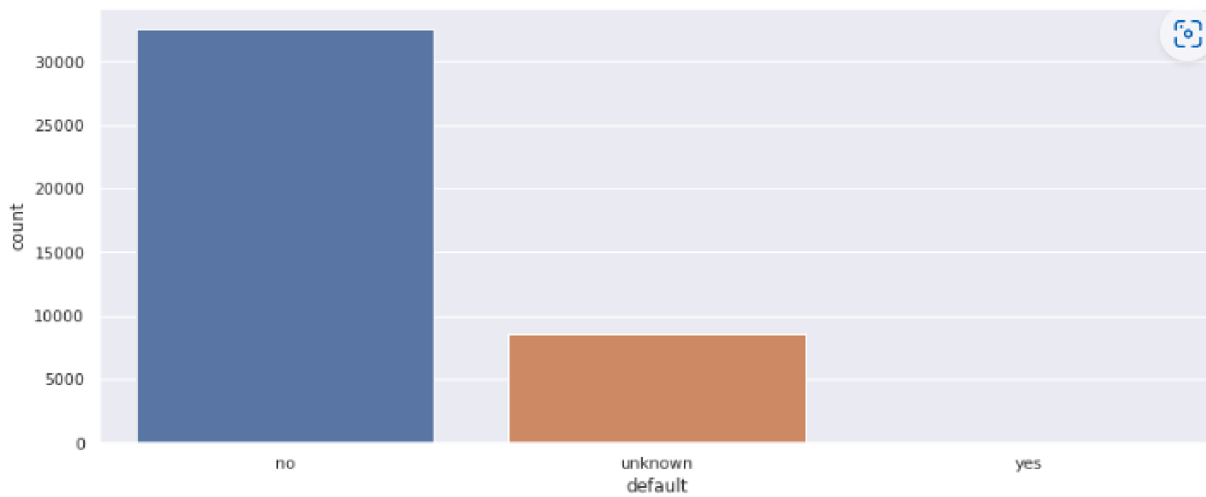


Figure 25: Count plot of default variable

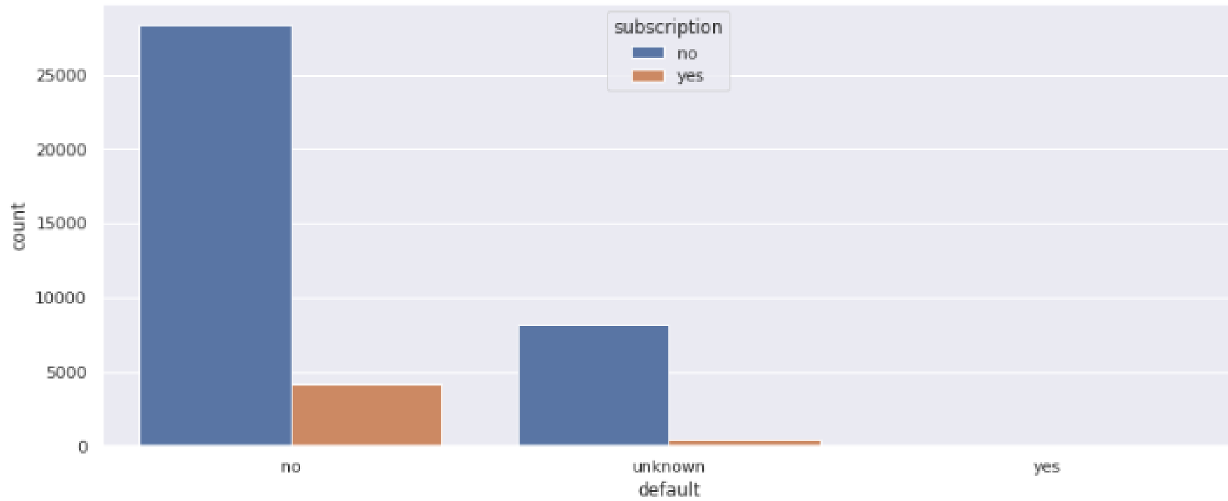


Figure 26: Bivariate count plot of default with subscription

5.3.6. Data exploration of housing variable

The housing variable gives information whether the client has housing loan. The following inference can be made looking at the housing variable:

- The housing variable has 3 unique values: yes, no, unknown
- Looking at the count plot we can see that 21571 clients has housing loan; 18615 clients has no housing loan and nearly 990 clients housing loan information is unknown
- However, information regarding housing loan should be available with the bank as housing loan is one of the service the bank provides.
- Hence, we are going to impute unknown values of housing loan to no, because if its yes then then information should be known to the bank

```
print(f'number of unique values:{bankdf.housing.nunique()}\n')
print(f'Unique values: {bankdf.housing.unique()}\n')
print(f'count:- \n{bankdf.housing.value_counts()}\n')
print("count-groupBy :\n",bankdf.groupby(['subscription','housing']).size())

plt.figure(figsize=[15,10])

plt.subplot(2,2,1)
sns.countplot(x ='housing',hue = 'subscription', data = bankdf)
plt.subplot(2,2,2)
sns.countplot(x ='housing', data = bankdf)
```

Figure 27:code for housing variable data exploration

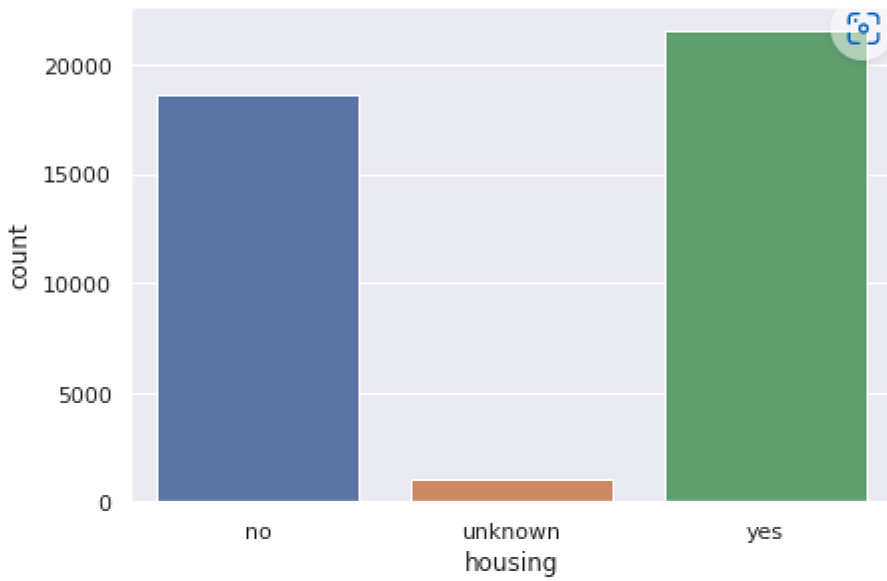


Figure 28: Count plot of housing variable

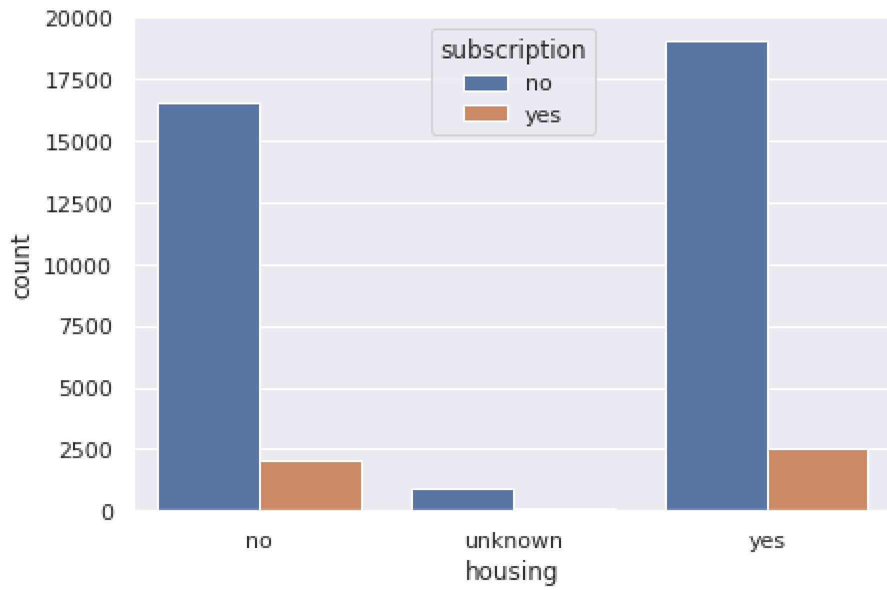


Figure 29: Bivariate count plot of housing with subscription

5.3.7. Data exploration of loan variable

The **loan** variable gives information whether the client has personal loan. The following inference can be made looking at the loan variable:

- The loan variable has 3 unique values: yes, no, unknown
- Looking at the count plot we can see that 6248 clients has personal loan; 33938 clients has no personal loan and nearly 990 clients personal loan information is unknown
- However, information regarding personal loan should be available with the bank as personal loan is one of the service the bank provides.
- Hence, we are going to impute unknown values of personal loan to no, because if its yes then then information should be known to the bank

```
print(f'number of unique values:{bankdf.loan.nunique()}\n')
print(f'Unique values: {bankdf.loan.unique()}\n')
print(f'count:- \n{bankdf.loan.value_counts()}\n')
print("count-groupBy :\n",bankdf.groupby(['subscription','loan']).size())

plt.figure(figsize=[30,12])

plt.subplot(2,2,1)
sns.countplot(x ='loan',hue = 'subscription', data = bankdf)
plt.subplot(2,2,2)
sns.countplot(x ='loan', data = bankdf)
```

Figure 30: code for loan variable data exploration

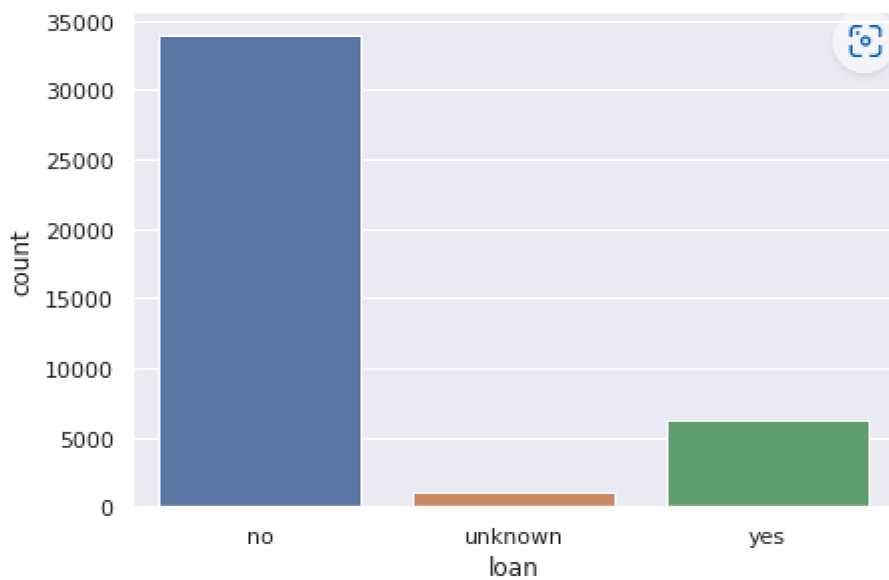


Figure 31: Count plot of loan variable

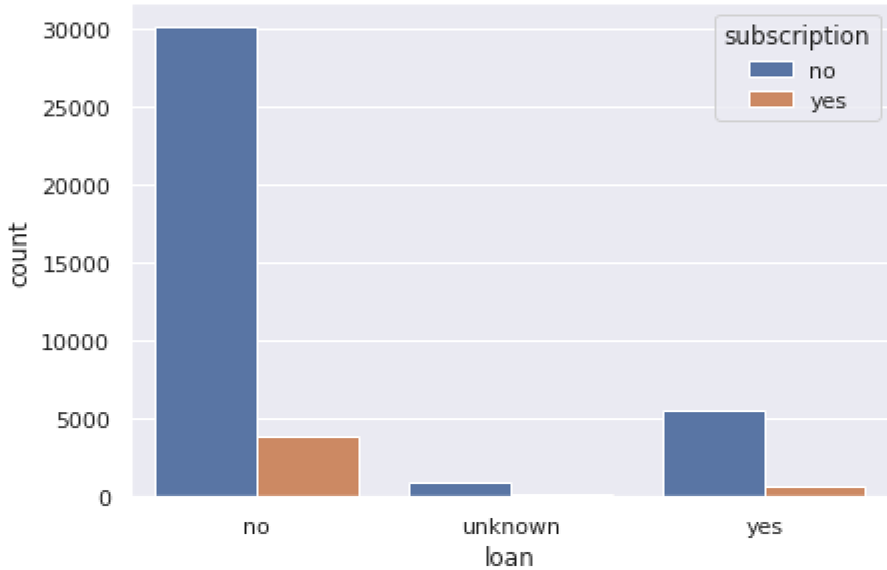


Figure 32:Bivariate count plot of loan with subscription

5.3.8. Data exploration of contact variable

The contact variable gives information how the client was contacted. The following inference can be made looking at the contact variable:

- The contact variable has 2 unique values: telephone and cellular
- Looking at the count plot we can see that 26135 clients were contacted through cell calls and 15041 clients were contacted through telephone
- Looking at the bivariate plot we can see that maximum conversion rate for term deposit was achieved from clients contacted through cell phone.

```
print(f'number of unique values:{bankdf.contact.nunique()}\n')
print(f'Unique values: {bankdf.contact.unique()}\n')
print(f'count:- \n{bankdf.contact.value_counts()}\n')
print("count-groupBy :\n",bankdf.groupby(['subscription','contact']).size())

plt.figure(figsize=[30,12])

plt.subplot(2,2,1)
sns.countplot(x ='contact',hue = 'subscription', data = bankdf)
plt.subplot(2,2,2)
sns.countplot(x ='contact', data = bankdf)
```

Figure 33: code for contact variable data exploration

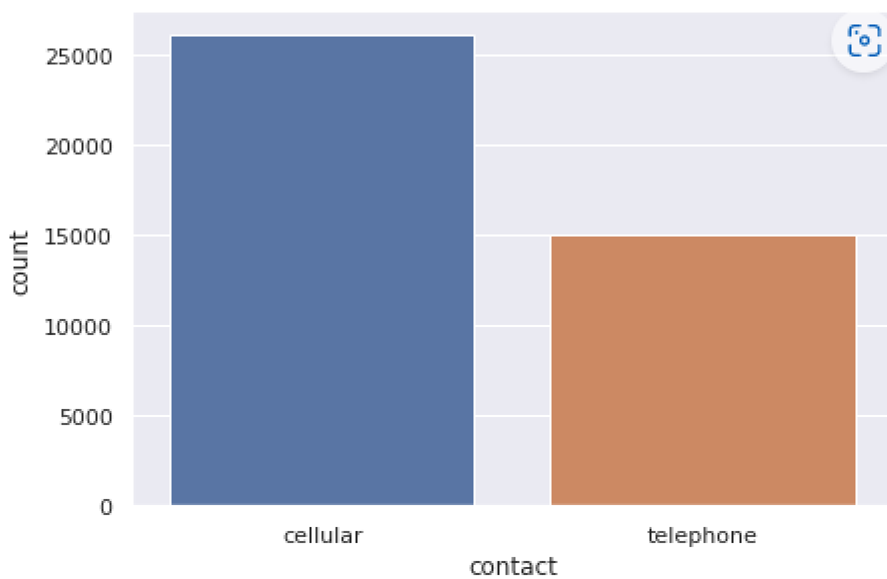


Figure 34: Count plot of contact variable

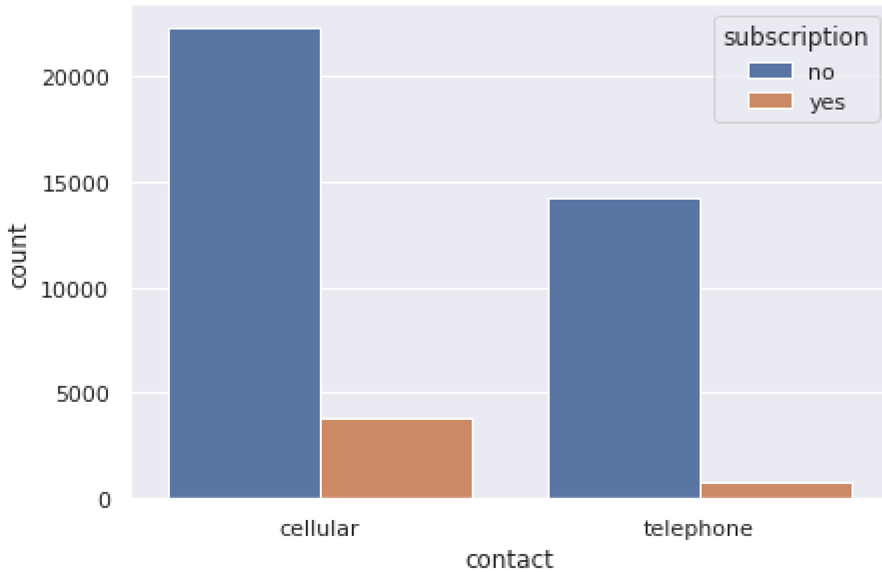


Figure 35: Bivariate count plot of contact with subscription

5.3.9. Data exploration of month variable

The month variable gives information on which month the client was last contacted. The following inference can be made looking at the month variable:

- The month variable has 10 unique values: may, jul, jun, aug, nov, apr, oct, sep, mar, dec
- Clients were not contacted in jan and feb
- Looking at the count plot we can see that maximum number of clients were contacted in May and least number of clients were contacted in December
- Looking at the bivariate plot we can see that maximum conversion rate for term deposit was achieved from clients contacted in may and least in month December

```
print(f'number of unique values:{bankdf.month.nunique()}\n')
print(f'Unique values: {bankdf.month.unique()}\n')
print(f'count:- \n{bankdf.month.value_counts()}\n')
print("count-groupBy :\n",bankdf.groupby(['subscription','month']).size())

plt.figure(figsize=[30,12])

plt.subplot(2,2,1)
sns.countplot(x ='month',hue = 'subscription', data = bankdf)
plt.subplot(2,2,2)
sns.countplot(x ='month', data = bankdf)
```

Figure 36: code for month variable data exploration

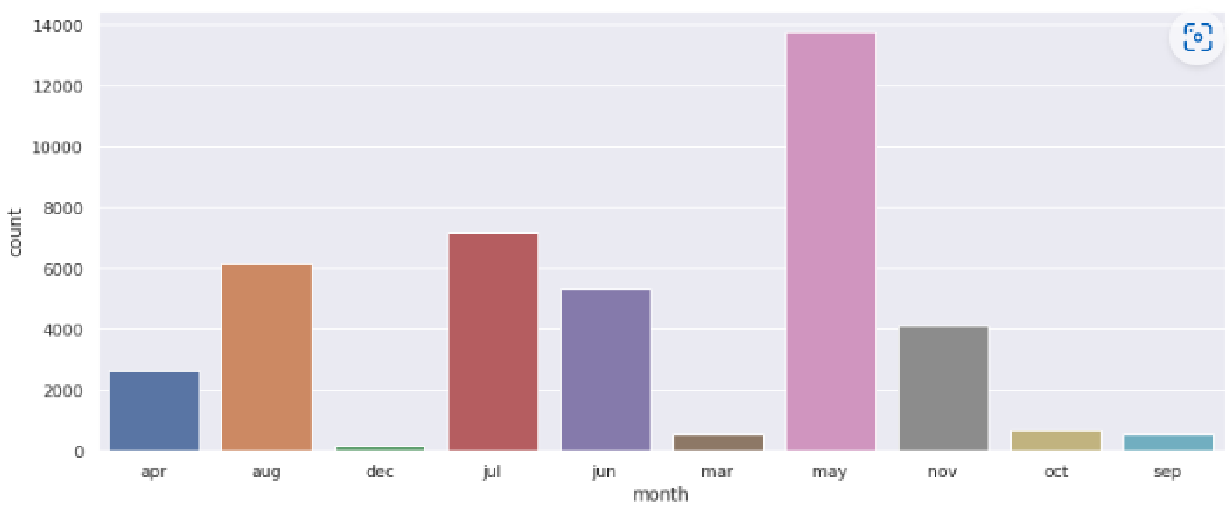


Figure 37: Count plot of month variable

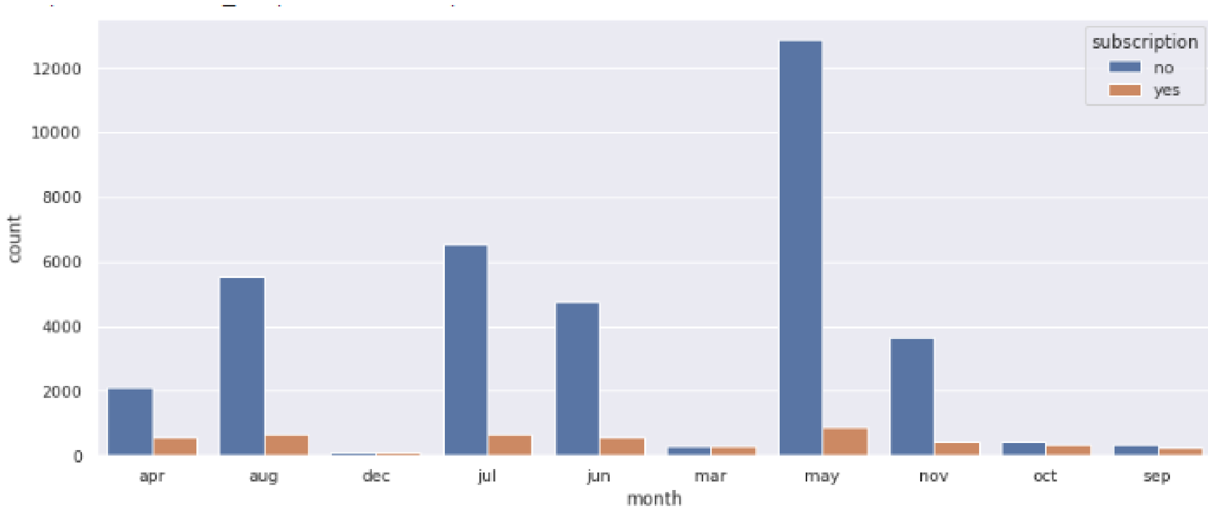


Figure 38:Bivariate count plot of month with subscription

5.3.10. Data exploration of poutcome variable

The poutcome variable gives information about the outcome of the previous marketing campaign. The following inference can be made looking at the poutcome variable:

- The poutcome variable has 3 unique values: success, failure and nonexistent.
- Nonexistent means that these contacted was not previous contacted. This value is to be treated as a unique value
- Looking at the count plot we can see that maximum number of clients were not contacted previously.
- Looking at the bivariate plot we can see that around 10% of the newly contacted clients subscribed to the term deposit

```python
print(f'number of unique values:{bankdf.poutcome.nunique()}\n')
print(f'Unique values: {bankdf.poutcome.unique()}\n')
print(f'count:- \n{bankdf.poutcome.value_counts()}\n')
print("count-groupBy :\n",bankdf.groupby(['subscription','poutcome']).size())

plt.figure(figsize=[15,10])

plt.subplot(2,2,1)
sns.countplot(x ='poutcome',hue = 'subscription', data = bankdf)
plt.subplot(2,2,2)
sns.countplot(x ='poutcome', data = bankdf)
```

Figure 39: code for poutcome variable data exploration

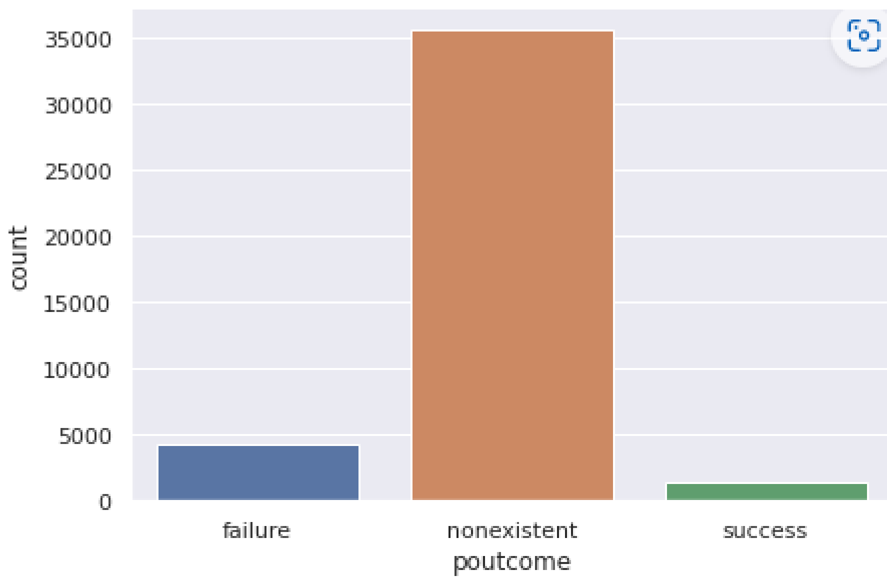


Figure 40: Count plot of poutcome variable

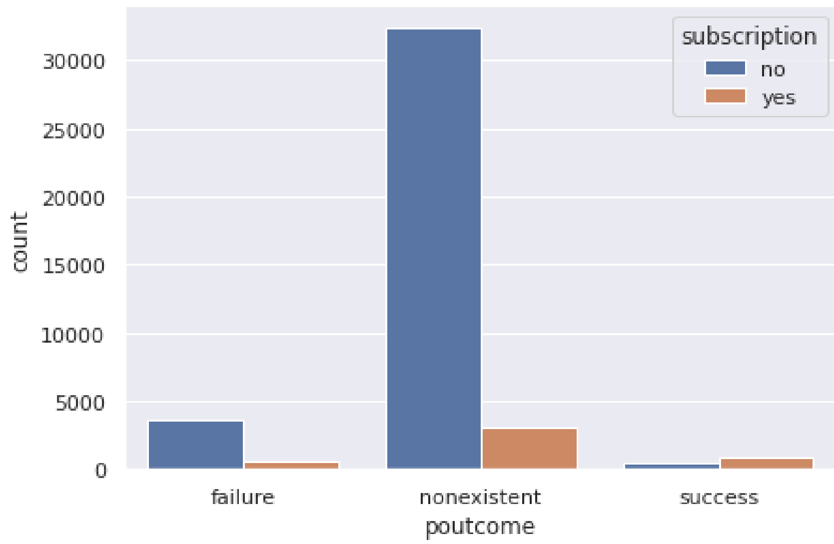


Figure 41:Bivariate count plot of poutcome with subscription

5.4. Data Exploration of numerical variable

The following are the 10 numerical variable in this data set:

- age: age of the client
- duration: last contact duration in seconds
- campaign: number of contacts performed during this campaign and for this client
- pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
- previous: number of contacts performed before this campaign and for this client
- emp.var.rate: employment variation rate - quarterly indicator
- cons.price.idx: consumer price index - monthly indicator
- cons.conf.idx: consumer confidence index - monthly indicator
- euribor3m: euribor 3 month rate - daily indicator
- nr.employed: number of employees - quarterly indicator

```
numerical =[ var for var in bankdf.columns if((bankdf[var].dtypes != 'category')
and (bankdf[var].dtypes != 'O'))]

numerical
```

```
['age',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'emp_var_rate',
 'cons_price_idx',
 'cons_conf_idx',
 'euribor3m',
 'nr_employed']
```

Figure 42: code to list all numerical variables

For the data exploration of numerical data are looking inot the histogram plots and box plot for bivariate analysis. The following inference can be made looking at the numerical variables:

- age value is mainly distributed between 17 to 98.
- Duration is skewed to the right. We will convert the duration variable from seconds to minutes and remove entries with duration more than 20 min.
- Similarly, campaign variable is also skewed to the right. This variable is also imputed by removing entries with campaign value more than 20.
- We will further study the numerical variable by looking at the correlation matrix

```
bankdf.describe()
```

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed
count	41176.00000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000
mean	40.02380	258.315815	2.567879	962.464810	0.173013	0.081922	93.575720	-40.502863	3.621293	5167.034870
std	10.42068	259.305321	2.770318	186.937102	0.494964	1.570883	0.578839	4.627860	1.734437	72.251364
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000

Figure 43: Summary statistics of numerical variables

```
plt.figure(figsize=[30,60])
sns.set(rc={"figure.figsize": (8, 4)})
plotnumber =1
for num in numerical:
  plt.subplot(12,3,plotnumber)
  ax = sns.distplot(bankdf[num])
  plotnumber +=1
plt.show()
```

Figure 44: code to draw histogram of numerical variable

```
plt.figure(figsize=[30,60])
sns.set(rc={"figure.figsize": (8, 4)})
plotnumber =1
for num in numerical:
  plt.subplot(12,3,plotnumber)
  ax = sns.boxplot(x="subscription",y= bankdf[num], data = bankdf)
  plotnumber +=1
plt.show()
```

Figure 45: code to draw bivariate box plot of numerical variable

5.4.1. Histogram of numerical variable

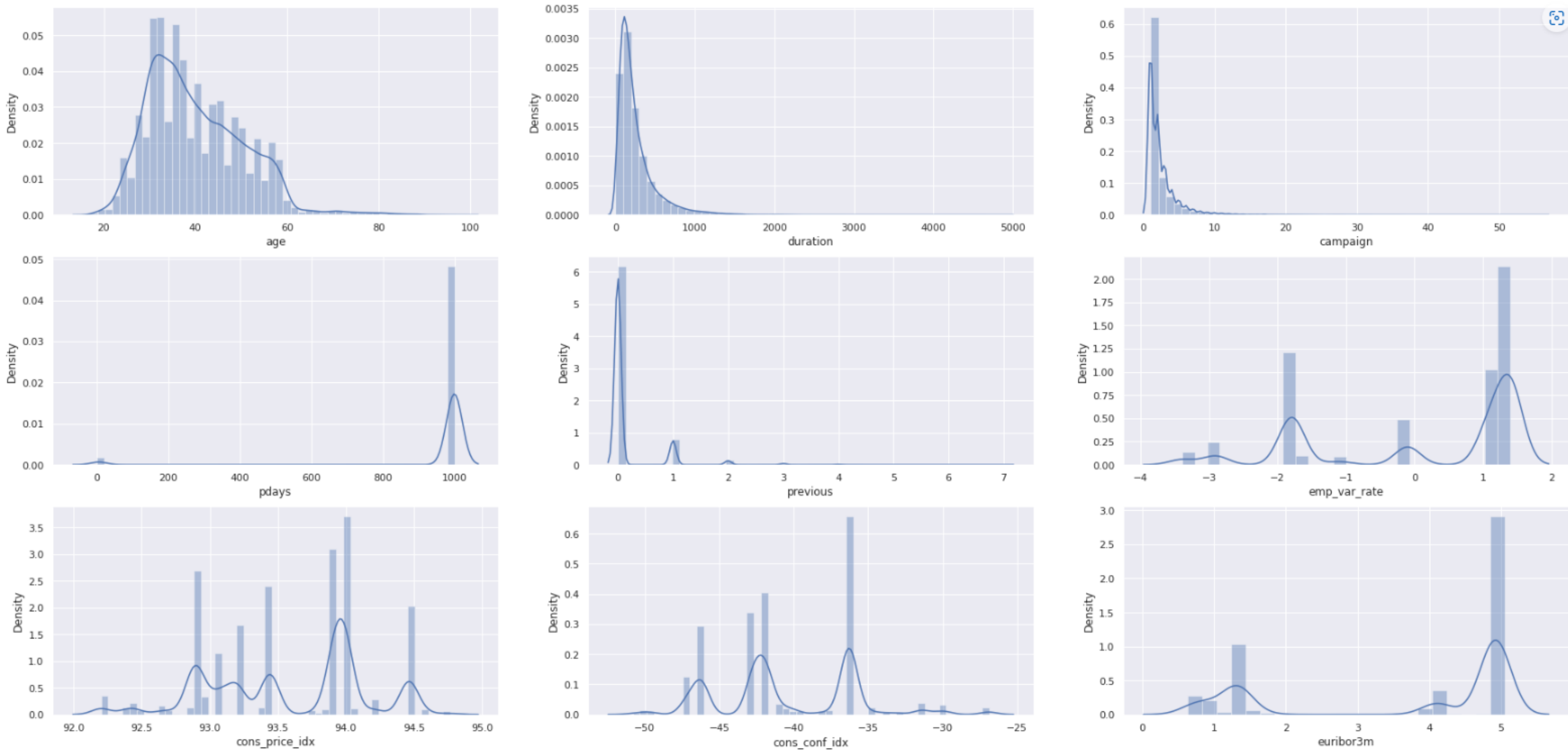


Figure 46: Histogram plot of numerical variables

5.4.2. bivariate box plot of numerical variable

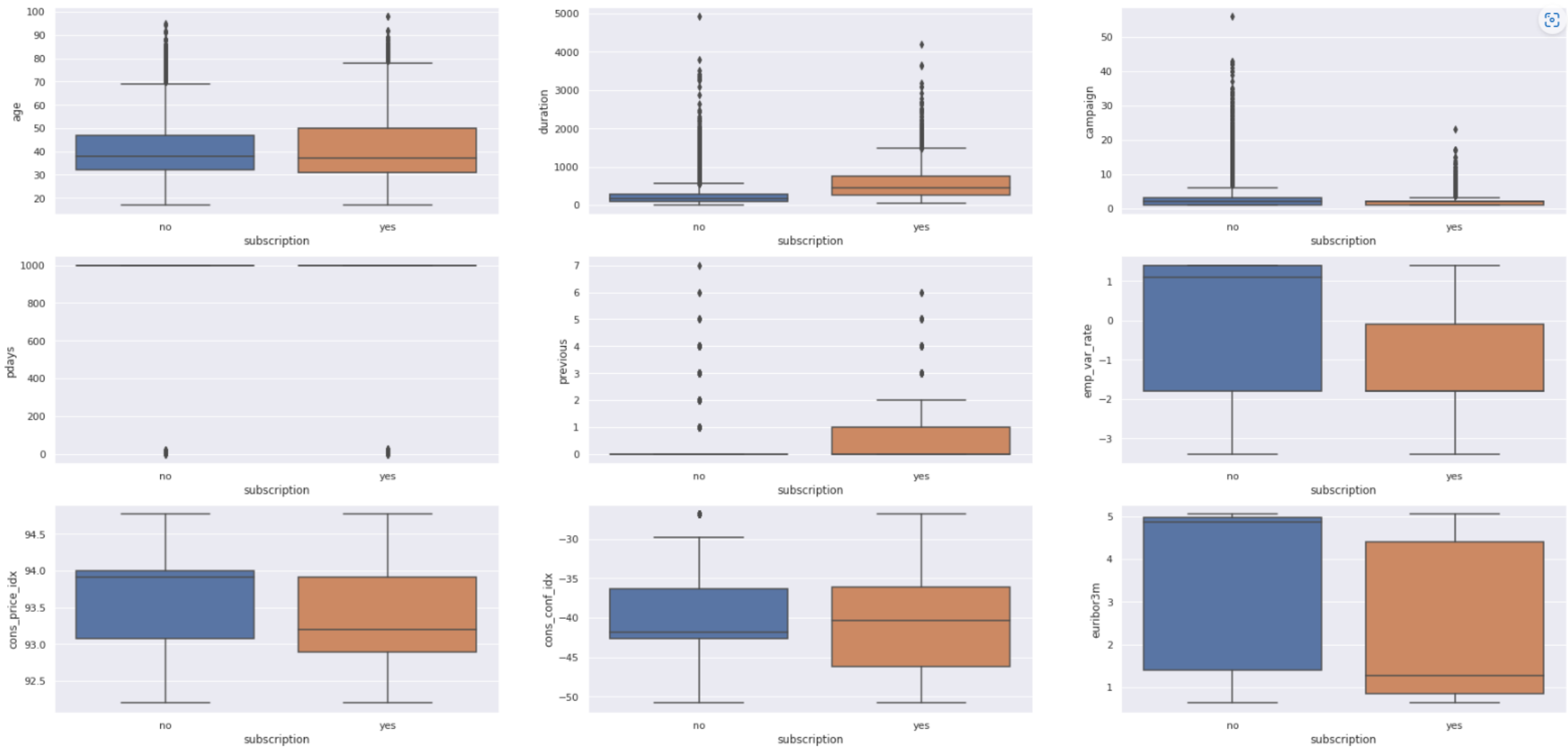


Figure 47: Bivariate box plot of numerical variables

5.4.3. Correlation matrix

From the correlation matrix it is evident that nr_employes, emp_var_rate, cons_price_idx, and euribor3m are highly correlated. To reduce redundancy we will be removing nr_employed , emp_var_rate and cons_price_idx.

```
[918] cor_matrix = bankdf.corr()
      fig = plt.figure(figsize=(15,7))
      sns.heatmap(cor_matrix,annot=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f376ed01090>

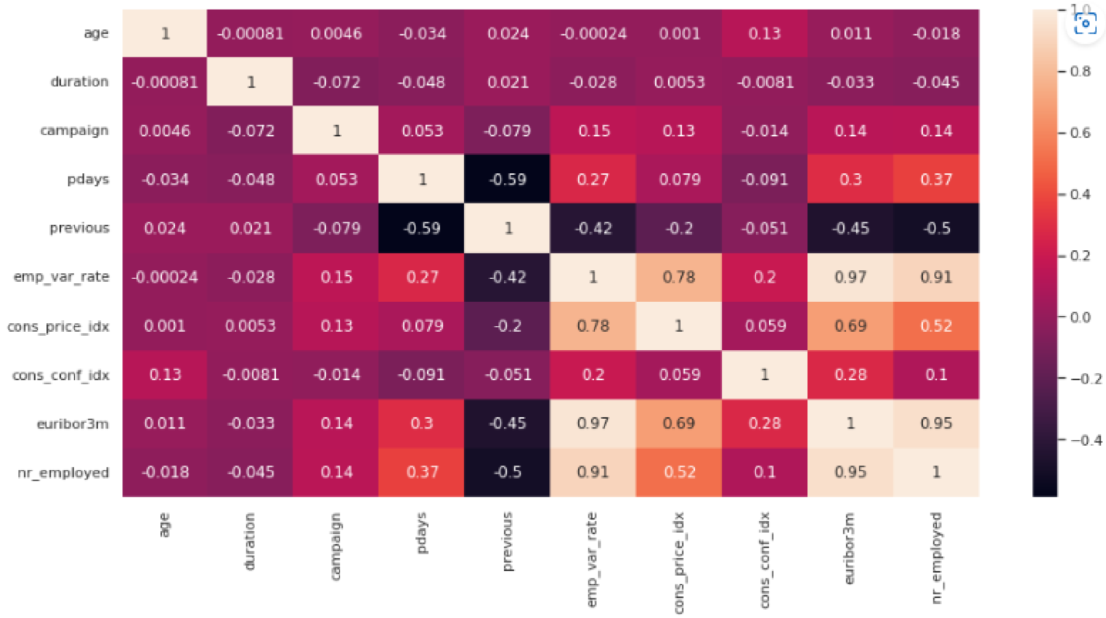


Figure 48: Correlation matrix

5.5. Data exploration summary

- There are No null values in the dataset.
- Some categorical variable has unknown values which needs to be handled
- The dataset is unbalanced as more that 85% entries have no value in it so we need to oversample the dataset with yes values
- Remove unknown entries from the marital variable

- Remove day_of_week, default variable because of near zero variance
- Impute unknown values in housing loan and personal loan to no.
- All the categorical variable should be dummy encoded
- Remove nr_employes, emp_var_rate and cons_price_idx to avoid correlation redundancy
- Convert the duration variable from seconds to minutes and remove entries with duration more than 20 min.
- Duration and campaign variable are skewed to the left. These variables are imputed by removing entries with value more than 20.

6.0. Data Preprocessing

6.1. Excluding columns

From the data exploration we have determined to remove the following columns:

- Default – near zero variance
- Day_of_week – near zero variance
- Emp_var_rate – correlated
- Nr_employed – correlated
- Cons_price_idx – correlated

```
# excluding columns due to near zero variance and correlation
bankdf.drop(['default','day_of_week','emp_var_rate','nr_employed','cons_price_idx'],axis=1, inplace=True)
```

Figure 49: Code to exclude columns

6.2. Outlier handling

Campaign and duration variable is skewed right due to the presence of outliers. To handle it, first convert duration to minutes and delete entries with campaign and duration value greater than 20

```
bankdf['duration'] = bankdf['duration'].apply(lambda n:n/60).round(2)
```

```
cond = (bankdf['duration']> 20 )
bankdf = bankdf.drop(bankdf[cond].index, axis = 0, inplace = False)
```

```
cond = (bankdf['campaign']> 20 )
bankdf = bankdf.drop(bankdf[cond].index, axis = 0, inplace = False)
```

Figure 50:Code to handle outliers

6.3. Converting datatypes

There are 8 object data types in this dataset. From the data dictionary it is clear that they are categorical variable. We need to set these variables as categorical variable.

```
bankdf['job'] = bankdf['job'].astype('category')
bankdf['marital'] = bankdf['marital'].astype('category')
bankdf['education'] = bankdf['education'].astype('category')
bankdf['housing'] = bankdf['housing'].astype('category')
bankdf['loan'] = bankdf['loan'].astype('category')
bankdf['contact'] = bankdf['contact'].astype('category')
bankdf['month'] = bankdf['month'].astype('category')
bankdf['poutcome'] = bankdf['poutcome'].astype('category')
bankdf['subscription'] = bankdf['subscription'].astype('category')
```

Figure 51: code to convert datatype from object to Category

6.3. Missing data Handling and dummy encoding

The unknown values in marital variables needs to be removed and unknown values in housing and loan need to be imputed to no. All categorical variable except dependent variable needs to be dummy encoded.

```
[1198] bankdf = bankdf[bankdf["marital"].str.contains("unknown")== False]
```

```
[1155] # bank should have information able a client having loan hence treating unknown as no
bool_columns = ['housing', 'loan']
for col in  bool_columns:
    bankdf[col+'_new']=bankdf[col].apply(lambda x : 1 if x == 'yes' else 0)
    bankdf.drop(col, axis=1, inplace=True)
```

```
[1199] bankdf1 = bankdf.copy()
```

```
cat_columns = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']
for col in  cat_columns:
    bankdf1 = pd.concat([bankdf1.drop(col, axis=1),pd.get_dummies(bankdf1[col], prefix=col, prefix_sep='_',drop_first=True, dummy_na=False)], axis=1)
```

Figure 52: Code to handle missing values; dummy encode

7.0. Modeling

7.1. Libraries for modeling

```
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_curve, roc_auc_score,recall_score
from sklearn import metrics
```

Figure 53: Code to import modeling libraries

7.2. Splitting dataset

A model that is tested using the same data that it was trained on will perform poorly and overfit in real-world situations. In order to avoid that, split your data into 2 pieces: train set and test set

```
X = bankdf1.drop(['subscription'],axis=1)
y = bankdf1['subscription']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=0)
```

Figure 54: code to split dataset

7.3. Normalization of Data

One of the most popular methods for preparing data is normalization, which enables us to convert the values of the dataset's numerical columns to a common scale. Normalization is employed if the attributes in the dataset have diverse ranges. It helps to enhance the performance and reliability of a machine learning model.

```
scaler = StandardScaler()
model = scaler.fit(X)
scaled_data_X = model.transform(X)
```

Figure 55: Code to normalize numerical data

7.4. Models

7.4.1. Random Forest classifier

Random Forest is a supervised learning model. The basic idea in random forests is to take several replacement samples at random from the data and create a "forest" by fitting a classification (or regression) tree to each sample using a random selection of predictors at each stage and combine the categories and predictions from the various trees to produce more accurate classifications.

Results from a random forest cannot be shown in a diagram that resembles a tree, hence they lack the interpretability that a single tree offers. Random forests, on the other hand, can generate "variable significance" ratings, which gauge the relative relevance of the many predictors. A specific predictor's importance score is calculated by adding the Gini index decline for that predictor over all of the tress in the forest

```
model_rfc = RandomForestClassifier(n_estimators=500, random_state=1)
model_rfc.fit(X_train, y_train)
pred_rfc = model_rfc.predict(X_test)
acc_rfc = accuracy_score(pred_rfc, y_test)
rec_rfc = recall_score(pred_rfc, y_test)
print(classification_report(pred_rfc, y_test))
print("Accuracy score:",acc_rfc)
print("Recall score:",rec_rfc)
```

```
              precision    recall  f1-score   support

           0       0.97      0.93      0.95     11235
           1       0.45      0.68      0.54       894

    accuracy                           0.92     12129
   macro avg       0.71      0.81      0.75     12129
weighted avg       0.94      0.92      0.92     12129

Accuracy score: 0.9153269024651661
Recall score: 0.6823266219239373
```

Figure 56: Code to build Random Forest Model

```
from seaborn.external.docscrape import header
header = ["name","score"]
values = sorted(zip(X.columns,model_rfc.feature_importances_),key = lambda x: x[1] * -1)
model_rfc_feature_importance = pd.DataFrame(values,columns = header)

fig = plt.figure(figsize =(15,7))
x_pos = np.arange(0,len(model_rfc_feature_importance))
plt.bar(x_pos, model_rfc_feature_importance['score'] )
plt.xticks(x_pos, model_rfc_feature_importance['name'] )
plt.xticks(rotation = 90)
plt.title('feature importance')

plt.show()
```

Figure 57: Code to plot variable importance

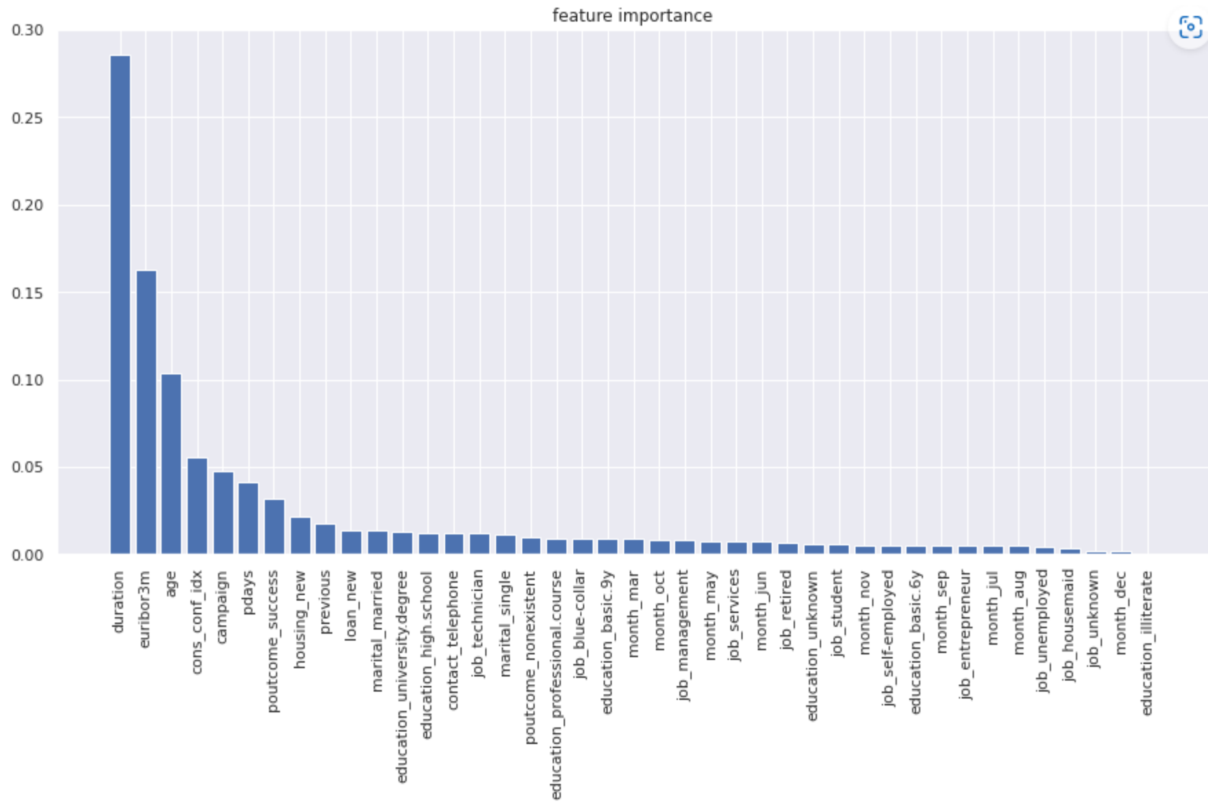


Figure 58: Variable importance plot

7.4.2. Logistic regression

Logistic regression is a supervised classification algorithm. For a specific collection of features (or inputs), X, the target variable (or output), y, can only take discrete values in a classification problem. Only when a decision threshold is included does logistic regression become a classification technique.

When the outcome variable, Y, is categorical, logistic regression applies the concepts of linear regression. A categorical variable can be thought of as classifying the records. Logistic regression can be used to place a new record into one of the classes when its class is unknown. its predictor variables' values (called classification).

The concept underlying logistic regression is simple: We utilize a function of Y called the logit as the outcome variable rather than Y itself. It turns out that one can represent the logit as a linear function of the predictors. The logit can then be converted back to a probability after being predicted.

Logistic regression is used in applications such as:

- Based on factors including annual salary, monthly credit card payments, and the number of defaults, a credit card business can use logistic regression to divide people into two groups: those with good credit and those with bad credit.
- Based on particular health characteristics, a hospital can use this test to divide patients into critical and non-critical groups.
- Insurance providers employ logistic regression to calculate the likelihood that a policyholder would pass away before the policy's term has run out based on factors including gender, age, and physical examination.

```
model_lr = LogisticRegression(max_iter=5000)
model_lr.fit(X_train, y_train)
pred_lr = model_lr.predict(X_test)
acc_lr = accuracy_score(pred_lr, y_test)
rec_lr = recall_score(pred_lr, y_test)
print(classification_report(pred_lr, y_test))
print("Accuracy score:",acc_lr)
print("Recall score:",rec_lr)
```

```
              precision    recall  f1-score   support

           0       0.98      0.93      0.95     11338
           1       0.40      0.68      0.50       791

    accuracy                           0.91     12129
   macro avg       0.69      0.80      0.73     12129
weighted avg       0.94      0.91      0.92     12129

Accuracy score: 0.9121114683815649
Recall score: 0.6814159292035398
```

Figure 59: code to build logistic regression model

7.4.3. Naïve Bayes Model

Naive Bayes classifiers in statistics work by applying the Bayes theorem while making strong assumptions about the independence of the features.

Naive Bayes is a straightforward method for building classifiers. These models assign class labels to problem cases, which are represented as vectors of feature values, and the class labels are chosen from a finite set. For training such classifiers, there isn't just one technique, but rather a family of algorithms built on the premise that, given the class variable, the value of one feature is independent of the value of every other feature.

Naive Bayes has the benefit of just requiring a little amount of training data to estimate the classification-related parameters.

Bayes' Theorem: The Bayes Theorem determines the likelihood of an event occurring given the likelihood of an earlier event occurring. The mathematical formulation of Bayes' theorem is given by the equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Assumption: Each variable is independent of each other

The classifier used to develop this model is a Gaussian Naïve Bayes Classifier. After building this model, we achieved 87.97% accuracy and 45.97% Recall Score

```
model_gnb = GaussianNB()
model_gnb.fit(X_train, y_train)
pred_gnb = model_gnb.predict(X_test)
acc_gnb = accuracy_score(pred_gnb, y_test)
rec_gnb = recall_score(pred_gnb, y_test)
print(classification_report(pred_gnb, y_test))
print("Accuracy score:",acc_gnb)
print("Recall score:",rec_gnb)
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93     10816
           1       0.45      0.46      0.45      1313

    accuracy                           0.88     12129
   macro avg       0.69      0.70      0.69     12129
weighted avg       0.88      0.88      0.88     12129

Accuracy score: 0.8797922334899827
Recall score: 0.46001523229246
```

Figure 60: Code to build Naive Bayes Model

7.4.4. k-nearest neighbors' algorithm

Finding k records in the training dataset that are comparable to a new record that we desire to categorize is the goal of k-nearest-neighbors' algorithms. The new record is then put into a class using these similar nearby records, with the new record being put into the class that is most common among its neighbors. Use x1, x2..., xp to denote the predictor values for this new record. In our training data, we search for records that are comparable to or "near" the record that has to be classified in the predictor space (i.e., records with values close to x1, x2..., xp). Then, we give the record that we want to categorize a class based on the classes to which those nearby records belong.

The k-nearest neighbors' algorithm (k-NN) in statistics is a non-parametric supervised learning technique. Regression and classification are two uses for it. The input in both situations consists of a data set's k closest training samples. Whether k-NN is applied for classification or regression determines the results:

The result of k-NN classification is a class membership. The class that an object is assigned to base on the majority vote of its k closest neighbors is determined by the item's neighbors (k is a positive integer, typically small). The object is simply put into the class of its one nearest neighbor if k = 1.

In our model, we will be using the normalized independent variables to run the knn model. After running the model, we achieved an accuracy rate of 89.97% and recall score of 58.61%

```
model_knn = KNeighborsClassifier()
model_knn.fit(X_train, y_train)
pred_knn = model_knn.predict(X_test)
acc_knn = accuracy_score(pred_knn, y_test)
rec_knn = recall_score(pred_knn, y_test)
print(classification_report(pred_knn, y_test))
print(acc_knn)
print(rec_knn)
```

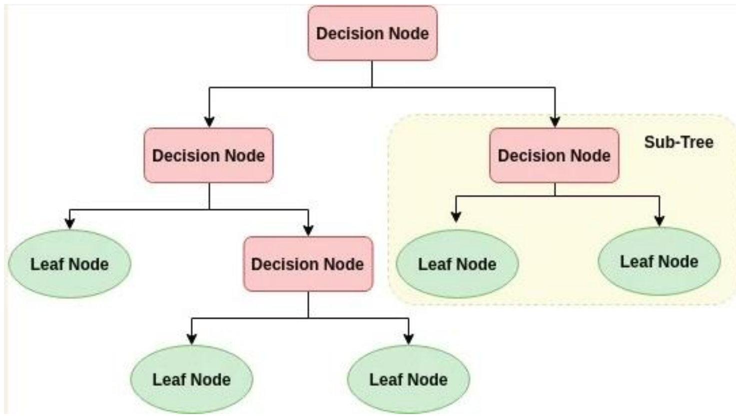
```
              precision    recall  f1-score   support

           0       0.97      0.92      0.94     11409
           1       0.31      0.59      0.41       720

    accuracy                           0.90     12129
   macro avg       0.64      0.75      0.68     12129
weighted avg       0.93      0.90      0.91     12129

0.8986726028526671
0.5861111111111111
```

Figure 61: Code to run the knn model

7.4.5. Decision Tree

A decision tree is a simple representation for classifying examples. This method of supervised machine learning continuously divides the data based on a given parameter. Both classification and regression issues can be resolved via decision tree analysis. A decision tree is incrementally built-in conjunction with the decision tree algorithm, which divides a dataset into smaller subgroups.

An internal node represents a feature (or property), a branch represents a decision rule, and each leaf node indicates the conclusion in a decision tree, which resembles a flowchart. The root node in a decision tree is the first node from the top. It gains the ability to divide data according to attribute values. Recursive partitioning is the process of repeatedly dividing a tree. This framework, which resembles a flowchart, aids in decision-making. It is a flowchart-like representation that perfectly replicates how people think. Decision trees are simple to grasp and interpret because of this.



After running the model, we achieved an accuracy rate of 89.77% and recall score of 54.15%

```
model_dtc= DecisionTreeClassifier(random_state=1)
model_dtc.fit(X_train, y_train)
pred_dtc = model_dtc.predict(X_test)
acc_dtc = accuracy_score(pred_dtc, y_test)
rec_dtc = recall_score(pred_dtc, y_test)
print(classification_report(pred_dtc, y_test))
print(acc_dtc)
print(rec_dtc)
```

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94     10766
           1       0.55      0.54      0.54      1363

    accuracy                           0.90     12129
   macro avg       0.74      0.74      0.74     12129
weighted avg       0.90      0.90      0.90     12129

0.8977656855470361
0.541452677916361
```

Figure 62: Code to build decision tree

7.4.6. Sampled Models

We will be over sampling the input variables using SMOTE to resolve the unbalanced dataset issue and run the 5 models again. To see if there is an improvement in the model efficiency.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=1)
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
```

Figure 63: Code to oversample input variable

```
sam_model_lr = LogisticRegression(max_iter=5000)
sam_model_lr.fit(X_train_sm, y_train_sm)
sam_pred_lr = sam_model_lr.predict(X_test)
sam_acc_lr = accuracy_score(sam_pred_lr, y_test)
sam_rec_lr = recall_score(sam_pred_lr, y_test)
print(classification_report(sam_pred_lr, y_test))
print(sam_acc_lr)
print(sam_rec_lr)
```

```
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      9496
           1       0.88      0.45      0.59      2633

    accuracy                           0.87     12129
   macro avg       0.87      0.72      0.76     12129
weighted avg       0.87      0.87      0.85     12129

0.8667656031000083
0.4500569692366122
```

Figure 64: sampled logistic regression

```
sam_model_gnb = GaussianNB()
sam_model_gnb.fit(X_train_sm, y_train_sm)
sam_pred_gnb = sam_model_gnb.predict(X_test)
sam_acc_gnb = accuracy_score(sam_pred_gnb, y_test)
sam_rec_gnb = recall_score(sam_pred_gnb, y_test)
print(classification_report(sam_pred_gnb, y_test))
print(sam_acc_gnb)
print(sam_rec_gnb)
```

```
              precision    recall  f1-score   support

           0       0.92      0.93      0.93     10654
           1       0.47      0.44      0.45      1475

    accuracy                           0.87     12129
   macro avg       0.70      0.68      0.69     12129
weighted avg       0.87      0.87      0.87     12129

0.8727017891005029
0.4352542372881356
```

Figure 64: sampled Naive Bayes model

```
sam_model_knn = KNeighborsClassifier()
sam_model_knn.fit(X_train_sm, y_train_sm)
sam_pred_knn = sam_model_knn.predict(X_test)
sam_acc_knn = accuracy_score(sam_pred_knn, y_test)
sam_rec_knn = recall_score(sam_pred_knn, y_test)
print(classification_report(sam_pred_knn, y_test))
print(sam_acc_knn)
print(sam_rec_knn)
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.91      9853
           1       0.69      0.41      0.52      2276

    accuracy                           0.86     12129
   macro avg       0.78      0.68      0.72     12129
weighted avg       0.84      0.86      0.84     12129

0.8553054662379421
0.4116871704745167
```

Figure 66: sampled knn model

```
sam_model_dtc= DecisionTreeClassifier(random_state=1)
sam_model_dtc.fit(X_train_sm, y_train_sm)
sam_pred_dtc = sam_model_dtc.predict(X_test)
sam_rec_dtc = recall_score(sam_pred_dtc, y_test)
sam_acc_dtc = accuracy_score(sam_pred_dtc, y_test)
print(classification_report(sam_pred_dtc, y_test))
print(sam_acc_dtc)
print(sam_rec_dtc)
```

```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93     10566
           1       0.56      0.48      0.52      1563

    accuracy                           0.88     12129
   macro avg       0.74      0.71      0.73     12129
weighted avg       0.88      0.88      0.88     12129

0.8837496908236459
0.4817658349328215
```

Figure 657: sampled decision tree

```
sam_model_rfc = RandomForestClassifier()
sam_model_rfc.fit(X_train_sm, y_train_sm)
sam_pred_rfc = sam_model_rfc.predict(X_test)
sam_acc_rfc = accuracy_score(sam_pred_rfc, y_test)
sam_rec_rfc = recall_score(sam_pred_rfc, y_test)
print(classification_report(sam_pred_rfc, y_test))
print(sam_acc_rfc)
print(sam_rec_rfc)
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     10763
           1       0.60      0.59      0.59      1366

    accuracy                           0.91     12129
   macro avg       0.77      0.77      0.77     12129
weighted avg       0.91      0.91      0.91     12129

0.9090609283535328
0.5915080527086384
```

Figure 66:Sampled random forest model

7.4.7. Model optimization

Looking at the feature importance from the random forest model. We see that month and job has least importance. Hence, we will be removing those variables and building the models again to see if we can further optimize the models and improve its efficiency.

```
[194] bankdf2 = bankdf.copy()
```

```
[196] bankdf2.drop(['month','job'],axis=1, inplace=True)
```

```
[197] cat_columns = ['marital', 'education', 'contact', 'poutcome']
      for col in  cat_columns:
          bankdf2 = pd.concat([bankdf2.drop(col, axis=1),pd.get_dummies(bankdf2[col], prefix=col, prefix_sep='_',drop_first=True, dummy_na=False)], axis=1)
```

Figure 67: data preprocessing for selected feature modeling

```
Xf = bankdf2.drop(['subscription'],axis=1)
yf = bankdf2['subscription']


Xf_train, Xf_test, yf_train, yf_test = train_test_split(Xf,yf,test_size=0.3, random_state=0)
```

Figure 68: data splitting for selected feature modeling

```
fea_model = LogisticRegression(max_iter=5000)
fea_model.fit(Xf_train, yf_train)
fea_pred_lr = fea_model.predict(Xf_test)
fea_rec_lr = recall_score(fea_pred_lr, yf_test)
fea_acc_lr = accuracy_score(fea_pred_lr, yf_test)
print(classification_report(fea_pred_lr, yf_test))
print(fea_acc_lr)
print(fea_rec_lr)
```

```
              precision    recall  f1-score   support

           0       0.98      0.93      0.95     11404
           1       0.37      0.70      0.49       725

    accuracy                           0.91     12129
   macro avg       0.68      0.81      0.72     12129
weighted avg       0.94      0.91      0.92     12129

0.9117816802704263
0.6951724137931035
```

```
fea_model_gnb = GaussianNB()
fea_model_gnb.fit(Xf_train, yf_train)
fea_pred_gnb = fea_model_gnb.predict(Xf_test)
fea_acc_gnb = accuracy_score(fea_pred_gnb, yf_test)
fea_rec_gnb = recall_score(fea_pred_gnb, yf_test)
print(classification_report(fea_pred_gnb, yf_test))
print(fea_acc_gnb)
print(fea_rec_gnb)
```

```
              precision    recall  f1-score   support

           0       0.95      0.92      0.94     11146
           1       0.37      0.50      0.42       983

    accuracy                           0.89     12129
   macro avg       0.66      0.71      0.68     12129
weighted avg       0.91      0.89      0.90     12129

0.888861406546294
0.5025432349949135
```

Figure 69: feature selected logistic regression Figure 70: feature selected sampled NBayes

```python
fea_model_knn = KNeighborsClassifier()
fea_model_knn.fit(Xf_train, yf_train)
fea_pred_knn = fea_model_knn.predict(Xf_test)
fea_acc_knn = accuracy_score(fea_pred_knn, yf_test)
fea_rec_knn = recall_score(fea_pred_knn, yf_test)
print(classification_report(fea_pred_knn, yf_test))
print(fea_acc_knn)
print(fea_rec_knn)
```

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.95     11146
           1       0.44      0.61      0.51       983

    accuracy                           0.91     12129
   macro avg       0.70      0.77      0.73     12129
weighted avg       0.92      0.91      0.91     12129

0.9065050704922087
0.6113936927772126
```

Figure 66: feature selected knn model

```python
fea_model_dtc= DecisionTreeClassifier(random_state=1)
fea_model_dtc.fit(Xf_train, yf_train)
fea_pred_dtc = fea_model_dtc.predict(Xf_test)
fea_acc_dtc = accuracy_score(fea_pred_dtc, yf_test)
fea_rec_dtc = recall_score(fea_pred_dtc, yf_test)
print(classification_report(fea_pred_dtc, yf_test))
print(fea_acc_dtc)
print(fea_rec_dtc)
```

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94     10771
           1       0.52      0.52      0.52      1358

    accuracy                           0.89     12129
   macro avg       0.73      0.73      0.73     12129
weighted avg       0.89      0.89      0.89     12129

0.8924066287410339
0.5176730486008837
```

Figure 69: feature selected decision tree

```python
fea_model_rfc = RandomForestClassifier()
fea_model_rfc.fit(Xf_train, yf_train)
fea_pred_rfc = fea_model_rfc.predict(Xf_test)
fea_acc_rfc = accuracy_score(fea_pred_rfc, yf_test)
fea_rec_rfc = recall_score(fea_pred_rfc, yf_test)
print(classification_report(fea_pred_rfc, yf_test))
print(fea_acc_rfc)
print(fea_rec_rfc)
```

```
              precision    recall  f1-score   support

           0       0.97      0.93      0.95     11246
           1       0.44      0.67      0.53       883

    accuracy                           0.91     12129
   macro avg       0.71      0.80      0.74     12129
weighted avg       0.93      0.91      0.92     12129

0.9135955148816886
0.6727066817667045
```

Figure 70: feature selected random forest model

7.5. Model comparison

Looking at the roc curve, accuracy rate and recall score, we can conclude that feature selected logistic regression is the best model. With an accuracy score of 0.911782 and recall score of 0.695172. Hence logistic regression model is selected to predict the subscription of the term deposit by the bank clients.

```
acc_table2 = pd.DataFrame({'Model': ['Logistic Regression','Naive Bayes','KNN','Decision Tree','Random Forest Tree',
                                     'sampled Logistic Regression','sampled Naive Bayes','sampled KNN','sampled Decision Tree',
                                     'sampled Random Forest Tree','featured selected Logistic Regression','featured selected Naive Bayes',
                                     'feature selected KNN','featured selected Decision Tree','featured selected Random Forest Tree'],
                                     'Accuracy Score': [acc_lr,acc_gnb,acc_knn,acc_dtc,acc_rfc,
                                            sam_acc_lr,sam_acc_gnb,sam_acc_knn,sam_acc_dtc,sam_acc_rfc,
                                            fea_acc_lr,fea_acc_gnb,fea_acc_knn,fea_acc_dtc,fea_acc_rfc],
                                     'Recall Score': [rec_lr,rec_gnb,rec_knn,rec_dtc,rec_rfc,
                                            sam_rec_lr,sam_rec_gnb,sam_rec_knn,sam_rec_dtc,sam_rec_rfc,
                                            fea_rec_lr,fea_rec_gnb,fea_rec_knn,fea_rec_dtc,fea_rec_rfc]})
acc_table2 = acc_table2.sort_values(by='Recall Score', ascending=False)
acc_table2
```

Figure 71: code to compare the accuracy score and recall score of the models

```
from sklearn.metrics import plot_roc_curve

lr_disp = metrics.plot_roc_curve(model_lr, X_test, y_test)
gnb_disp = metrics.plot_roc_curve(model_gnb, X_test, y_test, ax=lr_disp.ax_)
knn_disp = metrics.plot_roc_curve(model_knn, X_test, y_test, ax=gnb_disp.ax_)
dtc_disp = metrics.plot_roc_curve(model_dtc, X_test, y_test, ax=knn_disp.ax_)
rfc_disp = metrics.plot_roc_curve(model_rfc, X_test, y_test, ax=dtc_disp.ax_)
rfc_disp.figure_.suptitle("ROC curve comparison")

plt.show()
```

Figure 72: code to plot roc curve for comparison

	Model	Accuracy Score	Recall Score
10	featured selected Logistic Regression	0.911782	0.695172
4	Random Forest Tree	0.915327	0.682327
0	Logistic Regression	0.912111	0.681416
14	featured selected Random Forest Tree	0.913596	0.672707
12	feature selected KNN	0.906505	0.611394
9	sampled Random Forest Tree	0.909061	0.591508
2	KNN	0.898673	0.586111
3	Decision Tree	0.897766	0.541453
13	featured selected Decision Tree	0.892407	0.517673
11	featured selected Naive Bayes	0.888861	0.502543
8	sampled Decision Tree	0.883750	0.481766
1	Naive Bayes	0.879792	0.460015
5	sampled Logistic Regression	0.866766	0.450057
6	sampled Naive Bayes	0.872702	0.435254
7	sampled KNN	0.855305	0.411687

Figure 73: Accuracy score and recall score comparison of models

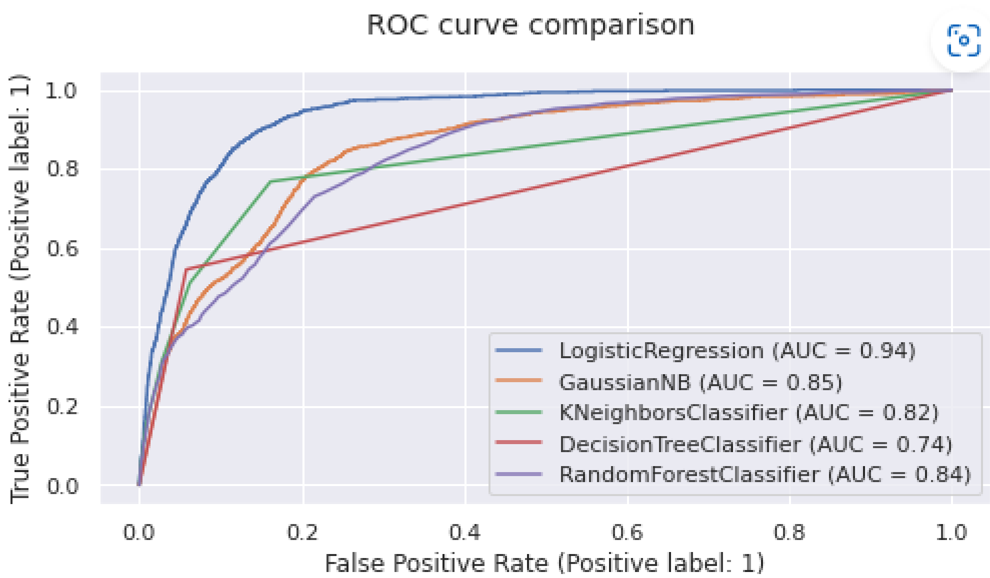


Figure 74: ROC curve of different models

8.0. Model Recommendation

8.1. Model Selection

After completing a thorough analysis and model building. Logistic regression was selected as the best model for this dataset.

8.2. Model Theory

Logistic regression is a supervised classification algorithm. For a specific collection of features (or inputs), X, the target variable (or output), y, can only take discrete values in a classification problem. Only when a decision threshold is included does logistic regression become a classification technique.

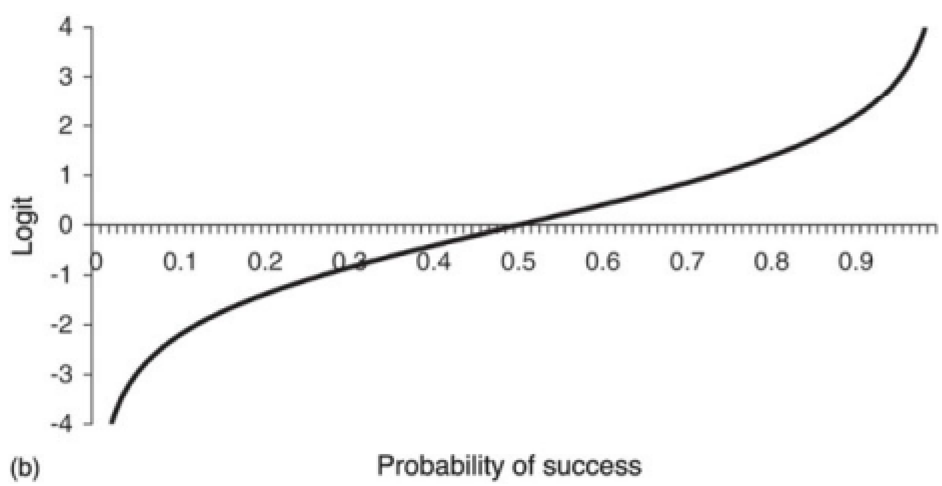
When the outcome variable, Y, is categorical, logistic regression applies the concepts of linear regression. A categorical variable can be thought of as classifying the records. Logistic regression can be used to place a new record into one of the classes when its class is unknown. its predictor variables' values (called classification).

The concept underlying logistic regression is simple: We utilize a function of Y called the logit as the outcome variable rather than Y itself. It turns out that one can represent the logit as a linear function of the predictors. The logit can then be converted back to a probability after being predicted.

Logit function:

$$p(x) = \frac{1}{1 + e^{-(x-\mu)/s}}$$

Logit as a function of p:



8.3. Model Assumptions and Limitation

- The outcome variable is a binary variable 2 possible values
- There is a linear relationship between the logit function and each predictor variable, this is one of the major limitations of logistic regression
- The are no outliers in the independent variable which could affect the model
- There are no highly correlated independent variables
- The sample size is large enough to run the model

8.3. Model Outcome

```
table = pd.DataFrame({'coef' : fea_model.coef_[0],'odds': np.e**fea_model.coef_[0]},index=Xf.columns)
table = table.sort_values(by='coef', ascending=False)
table
```

Figure 75: code to display odds ratio

	coef	odds
poutcome_nonexistent	0.706955	2.027808
poutcome_success	0.522763	1.686681
education_university.degree	0.398437	1.489495
marital_single	0.380491	1.463002
duration	0.329041	1.389635
education_professional.course	0.258542	1.295040
education_high.school	0.188084	1.206935
previous	0.164310	1.178580
education_unknown	0.149349	1.161078
marital_married	0.072988	1.075718
cons_conf_idx	0.059445	1.061248
age	0.011818	1.011888
education_illiterate	0.006763	1.006786
pdays	-0.001609	0.998393
housing_new	-0.038273	0.962450
campaign	-0.041145	0.959690
education_basic.6y	-0.092368	0.911769
loan_new	-0.110990	0.894948
education_basic.9y	-0.139661	0.869653
contact_telephone	-0.375063	0.687246
euribor3m	-0.609437	0.543657

Figure 76: Odds Ratio estimate

According to odds ratio estimate following inference can be made:

- **Poutcome** –
  - The subscription to term deposit is 2 times more likely to be predicted by per unit change in nonexistent value in poutcome
  - The subscription to term deposit is 68.66% more likely to be predicted by per unit change in success value in poutcome
- **Education** –

- The subscription to term deposit is 48.95% more likely to be predicted by per unit change in university degree education
- The subscription to term deposit is 29.50% more likely to be predicted by per unit change in professional course education
- The subscription to term deposit is 20.69% more likely to be predicted by per unit change in high school education
- The subscription to term deposit is 16.11% more likely to be predicted by per unit change in unknown education

- **Marital status –**
  - The subscription to term deposit is 46.30% more likely to be predicted by per unit change in marital status single
  - The subscription to term deposit is 7.57% more likely to be predicted by per unit change in marital status married

- **Duration –**
  - The subscription to term deposit is 38.96% more likely to be predicted by per unit change in duration
- **Previous –**
  - The subscription to term deposit is 17.85% more likely to be predicted by per unit change in previous
- **Conf.conf.idx -**

    - The subscription to term deposit is 6.12% more likely to be predicted by per unit change in consumer confidence index
- **Age –**
    - The subscription to term deposit is 1.1% more likely to be predicted by per unit change in age
- **Euribor3m** –
    - The subscription to term deposit is 45.64% less likely to be predicted by per unit change in euribor 3-month rate
- **Contact** –
    - The subscription to term deposit is 31.28% less likely to be predicted by per unit change in contact by telephone
- **Loan** –
    - The subscription to term deposit is 10.51% less likely to be predicted by per unit change in personal loan
- **Campaign –**
    - The subscription to term deposit is 4.04% less likely to be predicted by per unit change in campaign
- **Housing –**
    - The subscription to term deposit is 2.76% less likely to be predicted by per unit change in housing loan
- **Pdays –**
    - The subscription to term deposit is 2% less likely to be predicted by per unit change in pdays

	Variable	Odds Ratio	Interpretation
1	Poutcome_nonexistent	2.0278	The subscription to term deposit is 2 times more likely to be predicted by per unit change in nonexistent value in poutcome
2	Poutcome_success	1.6866	The subscription to term deposit is 68.66% more likely to be predicted by per unit change in success value in poutcome
3	Education_university_degree	1.4894	The subscription to term deposit is 48.95% more likely to be predicted by per unit change in university degree education
4	Marital_single	1.4630	The subscription to term deposit is 46.30% more likely to be predicted by per unit change in marital status single
5	duration	1.3896	The subscription to term deposit is 38.96% more likely to be predicted by per unit change in duration
6	education_professional.course	1.2950	The subscription to term deposit is 29.50% more likely to be predicted by per unit change in professional course education
7	education_high.school	1.2069	The subscription to term deposit is 20.69% more likely to be predicted by per unit change in high school education

8	previous	1.1785	The subscription to term deposit is 17.85% more likely to be predicted by per unit change in previous
9	education_unknown	1.1610	The subscription to term deposit is 16.11% more likely to be predicted by per unit change in unknown education
10	marital_married	1.0757	The subscription to term deposit is 7.57% more likely to be predicted by per unit change in marital status married
11	cons_conf_idx	1.0612	The subscription to term deposit is 6.12% more likely to be predicted by per unit change in consumer confidence index
12	Age	1.0118	The subscription to term deposit is 1.1% more likely to be predicted by per unit change in age
13	education_illiterate	1.0067	The subscription to term deposit is 0.68% more likely to be predicted by per unit change in illiterate education
14	pdays	0.9983	The subscription to term deposit is 2% less likely to be predicted by per unit change in pdays

15	housing_new	0.9624	The subscription to term deposit is 2.76% less likely to be predicted by per unit change in housing loan
16	campaign	0.9596	The subscription to term deposit is 4.04% less likely to be predicted by per unit change in campaign
17	education_basic.6y	0.9117	The subscription to term deposit is 8.82% less likely to be predicted by per unit change 6 years of basic education
18	loan_new	0.8949	The subscription to term deposit is 10.51% less likely to be predicted by per unit change in personal loan
19	education_basic.9y	0.869653	The subscription to term deposit is 13.04% less likely to be predicted by per unit change 9 years of basic education
20	contact_telephone	0.687246	The subscription to term deposit is 31.28% less likely to be predicted by per unit change in contact by telephone
21	euribor3m	0.543657	The subscription to term deposit is 45.64% less likely to be predicted by per unit change in euribor 3-month rate

9.0. Validation and Governance

9.1. Variable level monitoring

The data quality is evaluated using variable level monitoring, which is also used to measure variable drift and exceptions. We need to specify:

- handling of data that falls outside the range
- handling of missing value.
- We need to also consider if the variable is standard or drifting due to socio economic factors

9.1.1. Build statistics

	age	duration	campaign	pdays	previous	cons.conf.idx	euribor3m
count	40510.000000	40510.000000	40510.000000	40510.000000	40510.000000	40510.000000	40510.000000
mean	40.022859	4.035271	2.473068	10.882868	0.173784	-40.497944	3.615609
std	10.425949	3.455124	2.307423	6.526829	0.496017	4.634957	1.735849
min	17.000000	0.000000	1.000000	0.000000	0.000000	-50.800000	0.634000
25%	32.000000	1.700000	1.000000	5.000000	0.000000	-42.700000	1.344000
50%	38.000000	2.970000	2.000000	10.000000	0.000000	-41.800000	4.857000
75%	47.000000	5.200000	3.000000	20.000000	0.000000	-36.400000	4.961000
max	98.000000	20.000000	20.000000	20.000000	7.000000	-26.900000	5.045000

9.1.2. Acceptable range

Based on the Exploratory analysis, certain assumptions were made and certain variables outliers were handled and imputed. Based on criteria set to build this model, these are the acceptable variable range

Acceptable input Range			
	Variable	**Lower limit**	**Upper limit**

1	Age	17	98
2	Duration	0	20
3	Campaign	1	20
4	Pdays	3	20
5	Cons.conf.idx	-50.8	-26.9
6	Euribor3m	0.634	5.045
7	Previous	0	7
Categorical Variables			
	variable	**Accepted Values**	
1	Housing	Yes, no	
2	Loan	Yes, no	
3	Education	basic.4y, basic.6y, basic.9y, high.school, illiterate, professional.course, university.degree, unknown	
4	Marital	Single, divorce, married, unknown	
5	Contact	Cellular, telephone	
6	Poutcome	Failure, success, nonexistent	

In this dataset there were some outliers in duration variable and campaign variable. Since the number of entries with outlier entries were low, they were removed from the study. The duration and campaign variable are cap and floored between the values 0 and 20 are accepted.

9.1.3. Missing Values

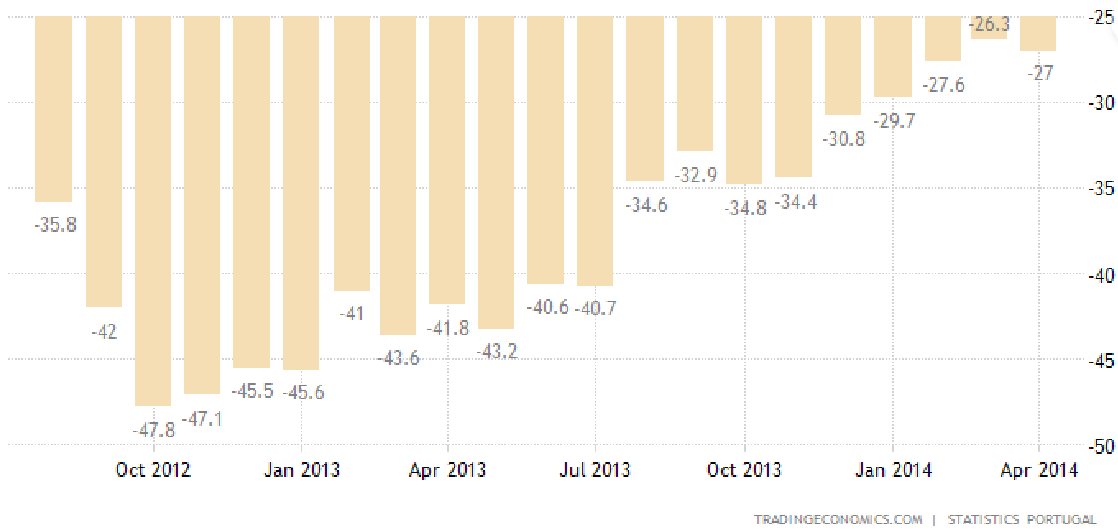
This section is doing to describe how to handling missing values. In the dataset provided we did not face any missing value. But if missing values are encountered in the future the following actions needs to be taken for the respective variables.

	Variable	**Missing values Handling**
1	Age	Impute the missing values with absolute value of the mean
2	Duration	Impute the missing values with absolute value of the mean
3	Campaign	Impute the missing values with absolute value of the mean
4	Pdays	Impute the missing values with absolute value of the mean
5	Cons.conf.idx	Impute the missing values with the median
6	Euribor3m	Impute the missing values with the median
7	Previous	Impute the missing values with absolute value of the mean

8	Housing	Impute the missing values with no. information about housing loan should be available with the bank, if its missing then it means they do not have a housing loan.
9	Loan	Impute the missing values with no. information about personal loan should be available with the bank, if its missing then it means they do not have a personal loan.
10	Education	Impute the missing values as unknown. There is an unknown category in Education variable.
11	Marital	Impute the missing values as unknown. There is an unknown category in marital variable.
12	Poutcome	Impute the missing values as nonexistent. If a previous campaign was conducted that information should be available, if its missing then it means they were not contacted previously

9.1.4. Variable drift Monitoring

model drift occurs when the statistical patterns present in the data a model was trained on have altered. In this model, variable such as consumer confidence index and Euribor 3-month rate are subjected to inflation rates as the cost-of-living and banking offer rate increase. Hence the value of these variable is continuously changing. Regular monitoring of these variables is needed to determine when the model needs to be retrained. The below chart shows the fluctuation of consumer confidence index in Portugal during the time the data was collected



2

Figure 77: Consumer confidence index in Portugal

This table shows that forecasted Euribor rate, some time the rate might fall between the range in which the model was build, as in this scenario the values is till within the expect range but sometime if could go out of range hence regular monitoring is needed.

Year	Mo	Min	Max	Close	Mo,%	Total%
2022	Aug	0.921	1.152	**0.988**	7.3%	7.3%
2022	Sep	0.975	1.099	**1.037**	5.0%	12.6%
2022	Oct	0.926	1.044	**0.985**	-5.0%	6.9%
2022	Nov	0.972	1.096	**1.034**	5.0%	12.3%
2022	Dec	1.021	1.151	**1.086**	5.0%	17.9%
2023	Jan	0.970	1.094	**1.032**	-5.0%	12.1%
2023	Feb	0.921	1.039	**0.980**	-5.0%	6.4%
2023	Mar	0.967	1.091	**1.029**	5.0%	11.7%
2023	Apr	1.015	1.145	**1.080**	5.0%	17.3%
2023	May	1.024	1.154	**1.089**	0.8%	18.2%
2023	Jun	0.973	1.097	**1.035**	-5.0%	12.4%
2023	Jul	1.022	1.152	**1.087**	5.0%	18.0%
2023	Aug	1.048	1.182	**1.115**	2.6%	21.1%
2023	Sep	1.050	1.184	**1.117**	0.2%	21.3%
2023	Oct	1.009	1.137	**1.073**	-3.9%	16.5%
2023	Nov	1.009	1.137	**1.073**	0.0%	16.5%
2023	Dec	1.005	1.133	**1.069**	-0.4%	16.1%
2024	Jan	1.011	1.141	**1.076**	0.7%	16.8%
2024	Feb	1.010	1.138	**1.074**	-0.2%	16.6%
2024	Mar	1.060	1.196	**1.128**	5.0%	22.5%
2024	Apr	1.033	1.165	**1.099**	-2.6%	19.3%
2024	May	1.008	1.136	**1.072**	-2.5%	16.4%
2024	Jun	1.011	1.141	**1.076**	0.4%	16.8%
2024	Jul	0.961	1.083	**1.022**	-5.0%	11.0%
2024	Aug	0.913	1.029	**0.971**	-5.0%	5.4%

Figure 78: Euribor monthly forecast

Drift Tolerance – We are going to make an assumption of the drift percentage as 10%. If the drift for these variables is more than 10% then we need to look at the impact of this variable drift on model evaluation scores and if there is significant impact then we need to retrain the model and refit the parameters of the model

9.2. model monitoring

9.2.1 Health and stability

For this model, to predict the health of this model, the RMSE, MAE, AUC are calculated and monitored.

Root Mean Square Error - a measurement of the discrepancies between values (in a sample or population) predicted by a model or an estimate and the values observed

Mean Absolute Error - a measurement of the discrepancies corresponding to the average absolute difference predicted by a model or an estimate and the values observed

AUC - Area Under the Receiver Operating Characteristic Curve is a statistic used to evaluate the effectiveness of machine learning models for categorization. The area under the ROC curve (AUC) score measures how well the model can predict classes, hence the score reflects this.

For the logistic regression model selected, the RMSE, MAE and AUC are as follows:

Root Mean Square Error	0.2970
Mean Absolute Error	0.0882
AUC score	0.94

```
print('Feature selected Logistic regression model')
print(regressionSummary(fea_pred_lr,yf_test))
```

```
Feature selected Logistic regression model

Regression statistics

               Mean Error (ME) : -0.0518
Root Mean Squared Error (RMSE) : 0.2970
     Mean Absolute Error (MAE) : 0.0882
None
```

Figure 79: Code to print summary

```
metrics.plot_roc_curve(model_lr, X_test, y_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py
  warnings.warn(msg, category=FutureWarning)
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7efd8cb98790>
```

Figure 80: roc curve

9.2.2. Initial Model Fit Statistics

For the selected logistic regression model

Accuracy score	**0.9117**
Precision score	0.3725
Recall score	0.6951
F1-score	0.4850

```
              precision    recall  f1-score   support

           0       0.98      0.93      0.95     11404
           1       0.37      0.70      0.49       725

    accuracy                           0.91     12129
   macro avg       0.68      0.81      0.72     12129
weighted avg       0.94      0.91      0.92     12129

0.9117816802704263
0.6951724137931035
0.37250554323725055
0.48508180943214624
```

Figure 81: logistic regression statistics

9.3. Risk Tiering

Risk Tiering seeks to categorize potential harm in order to make the model trustworthy. For this model, actions are decided for the risk tiering based on the following:

- Root Mean square value
- Accuracy
- Drift of input variable
-

Risk Tiering	Statistics level	Actions
Minimal Risk	• RMSE below or equal to the RMSE of the build model measure. • Accuracy is above or equal to the Accuracy of the build model measure.	No action need

	• no drift of input variable euro rate and consumer confidence index	
Limited Risk	• up to 5% variation in RMSE score • up to 5% variation in accuracy score • Up to 10% drift of input variable euro rate and consumer confidence index	Report the variation
High Risk	• RMSE more than 5 % higher than build model score. • Accuracy more than 5 % lower than build model score • Greater 10% drift of input variable euro rate and consumer confidence index	Refit the model and monitor
Unacceptable Risk	• RMSE higher build model measure. • Greater 10% drift of input variable euro rate and consumer confidence index	After refitting the model if the model becomes unstable or the statistic vary significantly then rebuild the model

10.0 Conclusion and Recommendation

10.1. Business recommendation

Based on the analysis, The logistic regression is the model which best fits the dataset. By looking at the odds ratio of the model we can make the following recommendation:

1. Clients who are single is more likely to subscribe to term deposit when compared to divorcees and married people
2. Clients who have university degree, professional course are more likely to subscribe to term deposit when compared to other educational qualification
3. Clients who have a housing loan or personal loan are less likely to subscribe to term deposit.
4. Clients are more likely to subscribe to term deposit if there is a positive change in consumer confidence index.
5. Clients within the age group of 30 to 50 are more likely to subscribe to term deposit

10.2 Future Works.

- Try building few more models such as XGboost Classifier and Support Vector Machine and see if the performance of the model increases.
- We could also try dimension reduction and Check if the performance of the model increases with dimension reduction techniques

11.0. Reference

UCI Machine Learning Repository: Bank Marketing Data Set. (n.d.). Retrieved August 12, 2022, from http://archive.ics.uci.edu/ml/datasets/Bank+Marketing

Banking in Portugal. Expatica. (2022, August 4). Retrieved August 12, 2022, from https://www.expatica.com/pt/finance/banking/banking-in-portugal-1101590/#System

Visualization with python. Matplotlib. (n.d.). Retrieved August 12, 2022, from https://matplotlib.org/

Chen, J. (2022, July 8). *Term deposit definition*. Investopedia. Retrieved August 12, 2022, from https://www.investopedia.com/terms/t/termdeposit.asp

Statistical Data Visualization¶. seaborn. (n.d.). Retrieved August 12, 2022, from https://seaborn.pydata.org/

Joy, A. (2019, February 28). *A beginner's Guide to Pandas Library [with examples]*. Pythonista Planet. Retrieved August 12, 2022, from https://pythonistaplanet.com/pandas/#:~:text=Pandas%20is%20a%20Python%20library%20that%20is%20used,need%20to%20have%20a%20look%20at%20this%20article.

GeeksforGeeks. (2022, June 28). *Understanding logistic regression*. GeeksforGeeks. Retrieved August 12, 2022, from https://www.geeksforgeeks.org/understanding-logistic-regression/#:~:text=Logistic%20regression%20is%20basically%20a%20supervised%20classification%20algorithm.,popular%20belief%2C%20logistic%20regression%20IS%20a%20regression%20model.

DataCamp. (2018, December 28). *Python decision tree classification tutorial: Scikit-Learn Decisiontreeclassifier*. DataCamp. Retrieved August 12, 2022, from https://www.datacamp.com/tutorial/decision-tree-classification-python#decision-tree-algorithm

Euribor Forecast 2022, 2023 and 2024. Long Forecast. (n.d.). Retrieved August 12, 2022, from https://longforecast.com/euribor-forecast-2017-2018-2019

Consumer price index (CPI) forecast. Consumer Price Index Forecast. (n.d.). Retrieved August 12, 2022, from https://conferenceboard.ca/e-data/data/consumerpriceindex.aspx

12.0 Table of Images