Classification Task

April 8, 2024

1 Classification Task

Name: ***R.M. Nipuna Upeksha***

IIT ID: ***20230106***

RGU ID: ***2322823***

1.1 Introduction

1. Clearly define the data mining problem and objectives.
2. Formulates the problem in a way suitable for data mining techniques.

a)For the classification task, the following data set was chosen from Kaggle. https://www.kaggle.com/datasets/rashmiranu/banking-dataset-classification. The dataset is comprised of banking details of the Portuguese bank.

- `age` → Age of a person
- `job` → Type of job(‘admin.’,‘blue-collar’,‘entrepreneur’,‘housemaid’,‘management’,‘retired’,‘self-employed’,‘services’,‘student’,‘technician’,‘unemployed’,‘unknown’)
- `marital` → Marital status(‘divorced’,‘married’,‘single’,‘unknown’)
- `education` → Education status(‘basic.4y’,‘basic.6y’,‘basic.9y’,‘high.school’,‘illiterate’,‘professional.course’,‘un
- `default` → Has credit in default?(‘no’, ‘yes’, ‘unknown’)
- `housing` → Has a housing loan?(‘no’, ‘yes’, ‘unknown’)
- `loan` → Has personal loan?(‘no’, ‘yes’, ‘unknown’)
- `contact` → Contact communication type(‘cellular’, ‘telephone’)
- `month` → Last contact month of year
- `day_of_week` → Last contact day of the week
- `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
- `previous` → Number of contacts performed before this campaign and for this client
- `poutcome` → Outcome of the previous marketing campaign
- `y` → Has the client subscribed to a term deposit? (‘yes’,‘no’)

1.1.1 Data Mining Problem and Objectives

The data mining problem is to develop a predictive model to predict whether a client will subscribe to a term deposit or not, based on various attributes collected by the Portuguese Bank. The objective is to create a robust classification model that accurately predicts whether a client will subscribe to a term deposit or not using the attributes given by the Portuguese Bank.

The primary goal is to analyze and understand the relationships between these features and whether a client subscribed to a term deposit. By doing so, we can determine to build a model that can provide accurate estimates for new clients. This predictive model can be utilized to find customer segmentation and improve campaign effectiveness and risk management.

In summary, the problem statement for data mining is to develop a classification model that utilizes the provided dataset's features to forecast whether a client will subscribe to a term deposit accurately, enabling informed decision-making for the bankers.

1.1.2 Problem Formulation with Data Mining Techniques

We can divide the problem formulation and the necessary steps that can be taken to solve it into the following steps.

Description

The problem formulated for data mining techniques is to create a classification model that predicts whether a customer will subscribe to a deposit based on various features provided in the dataset.This involves using attributes such as age, job, education, marital status, and other relevant factors to estimate the selling price of houses

Methodology

We need to follow the data mining techniques mentioned below to achieve our goal. 1. Data Pre-processing
(a) Handling NA/missing values if any.
(b) Handling outliers.
(c) Feature scaling for uniformity.
(d) Encoding categorical variables, if present. 3. Exploratory Data Analysis(EDA)
(a) Understanding the data and feature distribution. 4. Feature Selection
(a) Identifying relevant features for segmentation.
(b) Removing redundant or less impactful features. 6. Classification Algorithm
(a) Utilizing classification algorithms like Logistic Regression & Support Vector Machines.
(b) Predict whether a customer will subscribe to a deposit based on the algorithms.

1.2 Pre-process the Dataset

1. Handle missing values and outliers if any.
2. Produce Q-Q plots and histograms of the features and apply the transformations if required.
3. If it is required, apply suitable feature coding techniques.
4. Scale and/or standardize the features and produce relevant graphs to show the scaling/standardizing effect.
5. If required, apply suitable feature discretization techniques.

Before applying the algorithms to our dataset, we need to pre-process the dataset to remove NA/missing values, duplications, and outliers.

Let's import the necessary libraries and our dataset first.

The following code is used to ignore the warnings that may pop up in the libraries.

```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

[1]:

```
# Import libraries
# Data structures
import pandas as pd
import numpy as np

# Data processing
import scipy.stats as stats
from sklearn.preprocessing import FunctionTransformer, MinMaxScaler,␣
↪KBinsDiscretizer, PolynomialFeatures
from scipy.sparse import csr_matrix
from sklearn.decomposition import PCA, TruncatedSVD
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# Visualization and analysis
import matplotlib.pyplot as plt
from yellowbrick.regressor import PredictionError, ResidualsPlot
import seaborn as sns
from prettytable import PrettyTable

# Suppress warnings
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

Next let's import the dataset as a `Pandas` data frame.

```python
[2]: # Constants
PORTUGUESE_BANK_DATASET = './datasets/banking.csv'
```

Let's check the number of records and features in the dataset. Also let's print the first 10 records to check the data in our dataset.

```python
[3]: # Read the dataset
df = pd.read_csv(PORTUGUESE_BANK_DATASET)

# Count the number of records in the data frame
number_of_records = len(df.index)
print(f'Number of records in the dataset: {number_of_records}')

# Count the number of features in the data frame
number_of_features = len(df.columns)
print(f'Number of features in the dataset: {number_of_features}')

# Show first 10 records
df.head(10)
```

```
Number of records in the dataset: 32950
Number of features in the dataset: 16
```

```
[3]:    age           job   marital          education  default housing loan  \
     0   49   blue-collar   married           basic.9y  unknown      no   no
     1   37  entrepreneur   married  university.degree       no      no   no
     2   78       retired   married           basic.4y       no      no   no
     3   36        admin.   married  university.degree       no     yes   no
     4   59       retired  divorced  university.degree       no      no   no
     5   29        admin.    single  university.degree       no      no   no
     6   26       student    single           basic.9y       no      no   no
     7   30   blue-collar   married           basic.4y       no     yes   no
     8   50   blue-collar   married           basic.4y  unknown      no   no
     9   33        admin.    single        high.school       no     yes   no

          contact month day_of_week  duration  campaign  pdays  previous  \
     0   cellular   nov         wed       227         4    999         0
     1  telephone   nov         wed       202         2    999         1
     2   cellular   jul         mon      1148         1    999         0
     3  telephone   may         mon       120         2    999         0
     4   cellular   jun         tue       368         2    999         0
     5   cellular   aug         wed       256         2    999         0
     6  telephone   aug         wed       449         1    999         0
     7   cellular   nov         wed       126         2    999         0
     8  telephone   may         fri       574         1    999         0
     9   cellular   jul         tue       498         5    999         0

           poutcome    y
```

```
0  nonexistent   no
1      failure   no
2  nonexistent  yes
3  nonexistent   no
4  nonexistent   no
5  nonexistent   no
6  nonexistent  yes
7  nonexistent   no
8  nonexistent   no
9  nonexistent   no
```

The **'duration'** column is deleted before the execution as it is included for benchmark purposes and it will be helpful to have a realistic predictive model.

To get a general idea about the dataset, we can use the `df.describe()` function, which will provide the metrics like `mean`, `count`, and `standard deviation` of the dataset.

[4]:
```python
# Remove duration column
df = df.drop(columns=['duration'],axis=1)

# Checking descriptive stats
df.describe()
```

[4]:

	age	campaign	pdays	previous
count	32950.000000	32950.000000	32950.000000	32950.000000
mean	40.014112	2.560607	962.052413	0.174719
std	10.403636	2.752326	187.951096	0.499025
min	17.000000	1.000000	0.000000	0.000000
25%	32.000000	1.000000	999.000000	0.000000
50%	38.000000	2.000000	999.000000	0.000000
75%	47.000000	3.000000	999.000000	0.000000
max	98.000000	56.000000	999.000000	7.000000

[5]:
```python
# Get a general idea about the dataset
df.describe()
```

[5]:

	age	campaign	pdays	previous
count	32950.000000	32950.000000	32950.000000	32950.000000
mean	40.014112	2.560607	962.052413	0.174719
std	10.403636	2.752326	187.951096	0.499025
min	17.000000	1.000000	0.000000	0.000000
25%	32.000000	1.000000	999.000000	0.000000
50%	38.000000	2.000000	999.000000	0.000000
75%	47.000000	3.000000	999.000000	0.000000
max	98.000000	56.000000	999.000000	7.000000

The following code also provides a general idea about the dataset. It provides information on the data types in the dataset and the non-null data count.

[6]:
```python
# Get a general idea about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32950 entries, 0 to 32949
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   age          32950 non-null  int64
 1   job          32950 non-null  object
 2   marital      32950 non-null  object
 3   education    32950 non-null  object
 4   default      32950 non-null  object
 5   housing      32950 non-null  object
 6   loan         32950 non-null  object
 7   contact      32950 non-null  object
 8   month        32950 non-null  object
 9   day_of_week  32950 non-null  object
 10  campaign     32950 non-null  int64
 11  pdays        32950 non-null  int64
 12  previous     32950 non-null  int64
 13  poutcome     32950 non-null  object
 14  y            32950 non-null  object
dtypes: int64(4), object(11)
memory usage: 3.8+ MB
```

1.2.1 Removing Duplicated Records, Missing Values, and Outliers from the Dataset

The following code snippets show how you can remove the duplicate records, and process the missing values and outliers from the dataset.

Step 1 - Removing Duplicated Records Let's first look at the data frame and remove the duplicate records if present. To check whether we have duplicate records we can use `df.duplicated().sum()` function.

[7]:
```python
# Get a copy of the df data frame
df1 = df.copy()

# Count the number of records in the duplicates removed data frame
number_of_records_before = len(df1.index)
print(f'Number of records in the dataset before removing duplicates:␣
 ↪{number_of_records_before}')

# Count the number of features in the duplicates removed data frame
number_of_features_before = len(df1.columns)
print(f'Number of features in the dataset before removing duplicates:␣
 ↪{number_of_features_before}')
```

```python
# Drop the duplicates
df1 = df1.drop_duplicates(subset=df1.columns)

# Reset index after removing the duplicates
df1 = df1.reset_index(drop=True)

# Count the number of records in the duplicates removed data frame
number_of_records_after = len(df1.index)
print(f'Number of records in the dataset after removing duplicates: {number_of_records_after}')

# Count the number of features in the duplicates removed data frame
number_of_features_after = len(df1.columns)
print(f'Number of features in the dataset after removing duplicates: {number_of_features_after}')

# Show first 10 records in the new data frame
df1.head()
```

```
Number of records in the dataset before removing duplicates: 32950
Number of features in the dataset before removing duplicates: 15
Number of records in the dataset after removing duplicates: 31329
Number of features in the dataset after removing duplicates: 15
```

[7]:

	age	job	marital	education	default	housing	loan
0	49	blue-collar	married	basic.9y	unknown	no	no
1	37	entrepreneur	married	university.degree	no	no	no
2	78	retired	married	basic.4y	no	no	no
3	36	admin.	married	university.degree	no	yes	no
4	59	retired	divorced	university.degree	no	no	no

	contact	month	day_of_week	campaign	pdays	previous	poutcome	y
0	cellular	nov	wed	4	999	0	nonexistent	no
1	telephone	nov	wed	2	999	1	failure	no
2	cellular	jul	mon	1	999	0	nonexistent	yes
3	telephone	may	mon	2	999	0	nonexistent	no
4	cellular	jun	tue	2	999	0	nonexistent	no

We can see that there are only 8 duplicated values. Since this does not affect the dataset much, we can remove them from the dataset.

$$percentage\_of\_duplications = \frac{8}{32950} \times 100\%$$
$$percentage\_of\_duplications = 0.00024279\%$$

Step 2 - Handling Missing Values Since there are no duplications in the dataset, next we need to check for the missing values(NA/null values). To get an idea about whether we have missing values, we can iterate the data rows and check for the unique values in each column.

[8]:
```python
# Checking the unique data in the dataset
for col_name in df1:
    print(f'==========={col_name}===========')
    print(df1[col_name].unique())
    print('\n')
```

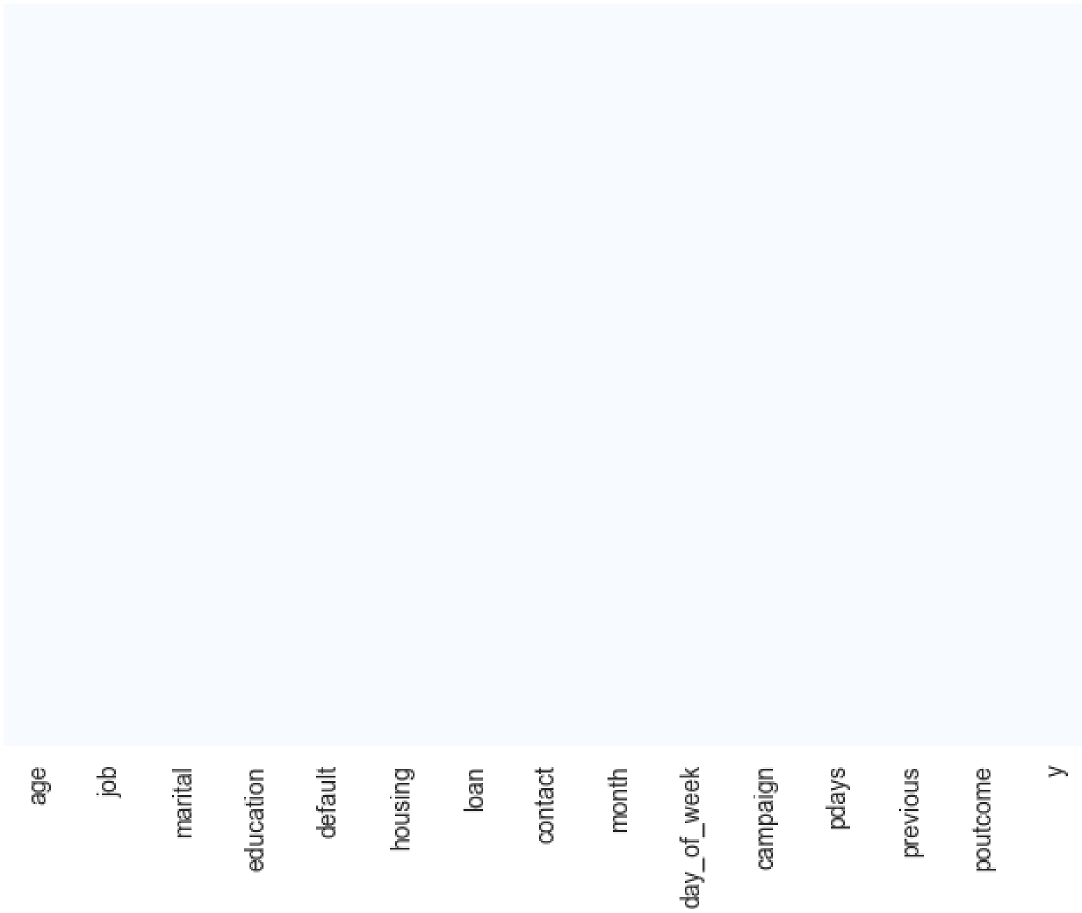
```
===========age===========
[49 37 78 36 59 29 26 30 50 33 44 32 43 56 40 47 34 46 39 41 55 38 63 23
 48 53 35 51 71 58 21 45 25 77 28 52 80 57 22 60 27 24 31 42 54 81 64 79
 20 76 82 19 68 65 73 66 85 74 61 86 69 18 83 88 70 87 84 75 62 72 67 89
 17 91 98]


===========job===========
['blue-collar' 'entrepreneur' 'retired' 'admin.' 'student' 'services'
 'technician' 'self-employed' 'management' 'unemployed' 'unknown'
 'housemaid']


===========marital===========
['married' 'divorced' 'single' 'unknown']


===========education===========
['basic.9y' 'university.degree' 'basic.4y' 'high.school'
 'professional.course' 'unknown' 'basic.6y' 'illiterate']


===========default===========
['unknown' 'no' 'yes']


===========housing===========
['no' 'yes' 'unknown']


===========loan===========
['no' 'yes' 'unknown']


===========contact===========
['cellular' 'telephone']


===========month===========
['nov' 'jul' 'may' 'jun' 'aug' 'mar' 'oct' 'apr' 'sep' 'dec']
```

```
===========day_of_week===========
['wed' 'mon' 'tue' 'fri' 'thu']


===========campaign===========
[ 4  2  1  5  9  3  7  6 13  8 12 10 19 11 31 17 16 29 43 20 14 21 35 15
 33 28 22 25 18 23 27 26 24 34 32 37 30 42 40 56]


===========pdays===========
[999   3   6  10   8   4   9  11   7  12   5   2  22  25  15  17   0  14
  13   1  16  18  19  21  20  27  26]


===========previous===========
[0 1 3 4 2 6 5 7]


===========poutcome===========
['nonexistent' 'failure' 'success']


===========y===========
['no' 'yes']
```

As you can see, to check for missing values in each column, we have to check the data manually, and it is a laborious task. Therefore, what we can do first is not check for the unique values, but plot the missing values using the `Seaborn` library's `heatmap()` function. This will give us an idea about whether we have missing values or not.

[9]:
```python
# View the heatmap to check null/NaN values
sns.heatmap(df1.isnull(), yticklabels=False, cbar=False, cmap="Blues")
```

[9]:
```
<Axes: >
```



Since we can see that there are missing values present in the dataset(because of the graph), next we need to check how many missing values are present in each column. To get that information you can use the below given options. - `df.isnull().sum()` - `df.info()`

[10]:
```
# Check the missing values in the data frame
df1.isnull().sum()
```

[10]:
```
age            0
job            0
marital        0
education      0
default        0
housing        0
loan           0
contact        0
month          0
day_of_week    0
campaign       0
```

```
pdays          0
previous       0
poutcome       0
y              0
dtype: int64
```

[11]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31329 entries, 0 to 31328
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   age          31329 non-null  int64 
 1   job          31329 non-null  object
 2   marital      31329 non-null  object
 3   education    31329 non-null  object
 4   default      31329 non-null  object
 5   housing      31329 non-null  object
 6   loan         31329 non-null  object
 7   contact      31329 non-null  object
 8   month        31329 non-null  object
 9   day_of_week  31329 non-null  object
 10  campaign     31329 non-null  int64 
 11  pdays        31329 non-null  int64 
 12  previous     31329 non-null  int64 
 13  poutcome     31329 non-null  object
 14  y            31329 non-null  object
dtypes: int64(4), object(11)
memory usage: 3.6+ MB
```

As you can see there are no null/NaN values in the dataset. But in the categorical data, we can see some data is listed as 'unknown'. Let's check the percentage of those.

[12]:
```
def check_missing_values_percentage(col):
    unknown_val_count = (df1[col]=='unknown').sum()
    if unknown_val_count > 0:
        print(f'Unknown value count in column {col}: {round((unknown_val_count␣
 ↪* 100/number_of_records_after),2)}%')

print('========== Missing values percentages in categorical columns ==========')
for col in df1.columns:
    check_missing_values_percentage(col)
```

```
========== Missing values percentages in categorical columns ==========
Unknown value count in column job: 0.83%
Unknown value count in column marital: 0.21%
Unknown value count in column education: 4.37%
Unknown value count in column default: 21.23%
```

```
Unknown value count in column housing: 2.52%
Unknown value count in column loan: 2.52%
```

There are two options to handle missing data in a data set.

1. If the missing value count of a column is greater than 50% - 70%, then we need to remove that column from the dataset.
2. If the missing value count of a column is less than 50% - 70%, then we need to impute the missing values.

The `marital` field has a small percentage of `unknown` data. Therefore, we can delete those rows. This will make a small reduction of data in the dataset.

The following code indicates how to remove the `unknown` values in `marital` data from the dataset.

[13]:
```
# Copy the data frame
df2 = df1.copy()

# Filter the data set from the unknown marital data
df2 = df2[df2['marital']!='unknown']

# Reset index after removing the duplicates
df2 = df2.reset_index(drop=True)

# Get the information of the dataframe
df2.head()
```

[13]:
```
   age           job   marital          education  default housing loan  \
0   49   blue-collar   married           basic.9y  unknown      no   no
1   37  entrepreneur   married  university.degree       no      no   no
2   78       retired   married           basic.4y       no      no   no
3   36        admin.   married  university.degree       no     yes   no
4   59       retired  divorced  university.degree       no      no   no

     contact month day_of_week  campaign  pdays  previous     poutcome    y
0   cellular   nov         wed         4    999         0  nonexistent   no
1  telephone   nov         wed         2    999         1      failure   no
2   cellular   jul         mon         1    999         0  nonexistent  yes
3  telephone   may         mon         2    999         0  nonexistent   no
4   cellular   jun         tue         2    999         0  nonexistent   no
```

[14]:
```
# Get the information on the data frame
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31264 entries, 0 to 31263
Data columns (total 15 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   age           31264 non-null  int64
```

```
 1   job          31264 non-null  object
 2   marital      31264 non-null  object
 3   education    31264 non-null  object
 4   default      31264 non-null  object
 5   housing      31264 non-null  object
 6   loan         31264 non-null  object
 7   contact      31264 non-null  object
 8   month        31264 non-null  object
 9   day_of_week  31264 non-null  object
 10  campaign     31264 non-null  int64
 11  pdays        31264 non-null  int64
 12  previous     31264 non-null  int64
 13  poutcome     31264 non-null  object
 14  y            31264 non-null  object
dtypes: int64(4), object(11)
memory usage: 3.6+ MB
```

There are another 5 variables `job`, `education`, `default`, `housing`,and `loan`. For them, the `unknown` records are too high. And also we can assume that customers have not provided their private information. Therefore, we will consider `unknown` as a value in this dataset for those columns.

We can do a cross-tabulation to check that.

[15]:
```python
# Cross tabulation between two columns
def cross_tabulation(data, col_1, col_2):
    # Find the count of unique values in the first column
    col1 = list(data[col_1].unique())
    # Find the count of unique values in the second column
    col2 = list(data[col_2].unique())
    df_list = []
    for col in col2:
        col1_df = data[data[col_2]==col]
        comb_df = col1_df.groupby(col_1).count()[col_2]
        df_list.append(comb_df)
    cross_tab_res = pd.concat(df_list, axis=1)
    cross_tab_res.columns = col2

    # Fill null values with zero(0)
    cross_tab_res = cross_tab_res.fillna(0)
    return cross_tab_res
```

Imputation 1 - Job and Education

[16]:
```python
# Cross tabulation between job and education
cross_tabulation(df2, 'job', 'education')
```

[16]:
```
             basic.9y  university.degree  basic.4y  high.school  \
job
admin.            385               4236        59         2518
```

```
blue-collar      2697      76   1734    690
entrepreneur      165     471    100    180
housemaid          72     115    359    136
management        128    1591     80    231
retired           114     231    454    214
self-employed     159     571     74     96
services          306     127    108   2033
student            77     135     16    276
technician        295    1357     47    654
unemployed        148     194     91    192
unknown            27      34     44     28

               professional.course  unknown  basic.6y  illiterate
job
admin.                         281      191       115         0.0
blue-collar                    350      358      1116         8.0
entrepreneur                   104       51        56         2.0
housemaid                       49       34        64         1.0
management                      70       87        66         0.0
retired                        198       69        52         2.0
self-employed                  129       22        18         3.0
services                       163      123       178         0.0
student                         34      144        11         0.0
technician                    2470      173        71         0.0
unemployed                     112       15        29         0.0
unknown                          8       92        20         0.0
```

- Inferring education from jobs: From the cross-tabulation, we can note that people with management jobs usually have some sort of college degree. Hence wherever 'job' = management and 'education' = unknown, we can replace 'education' with 'university.degree'. Similarly,
  - 'job' = 'services' -> 'education' = 'high.school'
  - 'job' = 'housemaid' -> 'education' = 'basic.4y'.
- Inferring jobs from education: If 'education' = 'basic.4y' or 'basic.6y' or 'basic.9y' then the 'job' is usually 'blue-collar'. If 'education' = 'professional.course', then the 'job' = 'technician'.
- While imputing the values for job and education, we want to make sure the correlations should make real-world sense. If it didn't make real-world sense, we don't need to replace the missing values.

[17]:
```python
# Replace the unknown values of education and job
df2.loc[(df2['education']=='unknown') & (df2['job']=='management'), 'education'] = 'university.degree'
df2.loc[(df2['education']=='unknown') & (df2['job']=='services'), 'education'] = 'high.school'
df2.loc[(df2['education']=='unknown') & (df2['job']=='housemaid'), 'education'] = 'basic.4y'
```

```python
df2.loc[(df2['job'] == 'unknown') & (df2['education']=='basic.4y'), 'job'] =␣
↪'blue-collar'
df2.loc[(df2['job'] == 'unknown') & (df2['education']=='basic.6y'), 'job'] =␣
↪'blue-collar'
df2.loc[(df2['job'] == 'unknown') & (df2['education']=='basic.9y'), 'job'] =␣
↪'blue-collar'
df2.loc[(df2['job']=='unknown') & (df2['education']=='professional.course'),␣
↪'job'] = 'technician'

# Check cross-tabulation between job and education
cross_tabulation(df2, 'job', 'education')
```

[17]:

```
               basic.9y  university.degree  basic.4y  high.school  \
job
admin.            385.0               4236      59.0         2518
blue-collar      2724.0                 76    1778.0          690
entrepreneur      165.0                471     100.0          180
housemaid          72.0                115     393.0          136
management        128.0               1678      80.0          231
retired           114.0                231     454.0          214
self-employed     159.0                571      74.0           96
services          306.0                127     108.0         2156
student            77.0                135      16.0          276
technician        295.0               1357      47.0          654
unemployed        148.0                194      91.0          192
unknown             0.0                 34       0.0           28

               professional.course  basic.6y  unknown  illiterate
job
admin.                       281.0     115.0    191.0         0.0
blue-collar                  350.0    1136.0    358.0         8.0
entrepreneur                 104.0      56.0     51.0         2.0
housemaid                     49.0      64.0      0.0         1.0
management                    70.0      66.0      0.0         0.0
retired                      198.0      52.0     69.0         2.0
self-employed                129.0      18.0     22.0         3.0
services                     163.0     178.0      0.0         0.0
student                       34.0      11.0    144.0         0.0
technician                  2478.0      71.0    173.0         0.0
unemployed                   112.0      29.0     15.0         0.0
unknown                        0.0       0.0     92.0         0.0
```

Now we can observe that the `unknown` values in the dataset is reduced a bit.

Imputation 2 - House and Job We can use the cross-tabulation between `housing` and `job` one more time to impute values for the `unknown` values. It's because the housing loan status should directly affect each job category.

[18]:
```
# Check cross-tabulation between job and housing
cross_tabulation(df2, 'job', 'housing')
```

[18]:

job	no	yes	unknown
admin.	3473	4133	179
blue-collar	3351	3573	196
entrepreneur	503	600	26
housemaid	386	420	24
management	1041	1154	58
retired	612	691	31
self-employed	483	554	35
services	1388	1571	79
student	292	384	17
technician	2237	2720	118
unemployed	339	421	21
unknown	66	85	3

[19]:
```
# Imputation via cross-tabulation for housing
def fill_housing(df, job_housing):
    jobs = ['housemaid','services','admin.
↪','blue-collar','technician','retired','management','unemployed','self-employed','entrepren
    house = ['no','yes']
    for j in jobs:
        # Here we are taking the value where housing is unknown and job value␣
↪is known
        ind = df[np.logical_and(np.array(df['housing']=='unknown'),np.
↪array(df['job']==j))].index
        mask = np.random.rand(len(ind))<((job_housing.loc[j]['no'])/
↪(job_housing.loc[j]['no']+job_housing.loc[j]['yes']))
        ind1 = ind[mask]
        ind2 = ind[~mask]
        df.loc[ind1,'housing'] = 'no'
        df.loc[ind2,'housing'] = 'yes'
    return df

# Fill housing
job_housing_df = cross_tabulation(df2,'job','housing')
df2=fill_housing(df2, job_housing_df)

# Check again with cross-tabulation
cross_tabulation(df2,'job','housing')
```

[19]:

job	no	yes	unknown
admin.	3556	4229	0.0
blue-collar	3446	3674	0.0

```
entrepreneur    516   613      0.0
housemaid       398   432      0.0
management     1063  1190      0.0
retired         628   706      0.0
self-employed   498   574      0.0
services       1421  1617      0.0
student         299   394      0.0
technician     2291  2784      0.0
unemployed      348   433      0.0
unknown          66    85      3.0
```

Imputation 3 - Loan and Job We are using the cross tabulation between `loan` and `job` to fill the `unknown` values. Its because the house loan status should be proportional to each job category

[20]:
```python
cross_tabulation(df2,'job','loan')
```

[20]:
```
                 no   yes  unknown
job
admin.         6256  1350      179
blue-collar    5806  1118      196
entrepreneur    936   167       26
housemaid       684   122       24
management     1847   348       58
retired        1115   188       31
self-employed   888   149       35
services       2482   477       79
student         556   120       17
technician     4190   767      118
unemployed      640   120       21
unknown         126    25        3
```

[21]:
```python
# Imputation via cross-tabulation for loan
def fill_loan(df, job_loan):
    jobs = ['housemaid','services','admin.
 ↪','blue-collar','technician','retired','management','unemployed','self-employed','entrepren
    loan = ['no','yes']
    for j in jobs:
        ind = df[np.logical_and(np.array(df['loan']=='unknown'),np.
 ↪array(df['job']==j))].index
        mask = np.random.rand(len(ind))<((job_loan.loc[j]['no'])/(job_loan.
 ↪loc[j]['no']+job_loan.loc[j]['yes']))
        ind1 = ind[mask]
        ind2 = ind[~mask]
        df.loc[ind1,'loan']='no'
        df.loc[ind2,'loan']='yes'
    return df
```

```python
# Fill housing
job_loan_df = cross_tabulation(df2,'job','loan')
df2 = fill_loan(df2,job_loan_df)

# Check again with cross-tabulation
cross_tabulation(df2,'job','loan')
```

[21]:

job	no	yes	unknown
admin.	6404	1381	0.0
blue-collar	5969	1151	0.0
entrepreneur	961	168	0.0
housemaid	705	125	0.0
management	1894	359	0.0
retired	1142	192	0.0
self-employed	921	151	0.0
services	2555	483	0.0
student	571	122	0.0
technician	4285	790	0.0
unemployed	658	123	0.0
unknown	126	25	3.0

Checking Numerical Variables It has been mentioned that the missing values, or NaNs are encoded as '999' because they were never contacted. Let's check that.

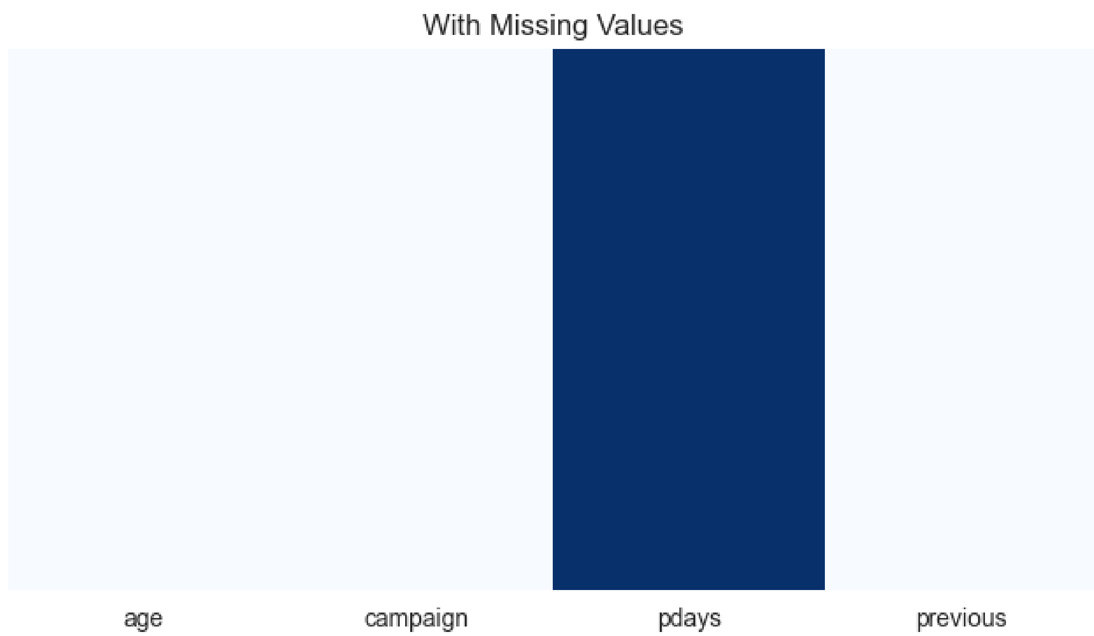
[22]:
```python
numerical_variables = ['age','campaign', 'pdays', 'previous']
for var in numerical_variables:
    count = df2[df2[var]==999][var].count()
    print(f'Number of missing values for column {var}: {count}')
```

```
Number of missing values for column age: 0
Number of missing values for column campaign: 0
Number of missing values for column pdays: 30043
Number of missing values for column previous: 0
```

We can see that, there is a large number of missing values for `pdays`. We can use snsheatmap to view that as well.

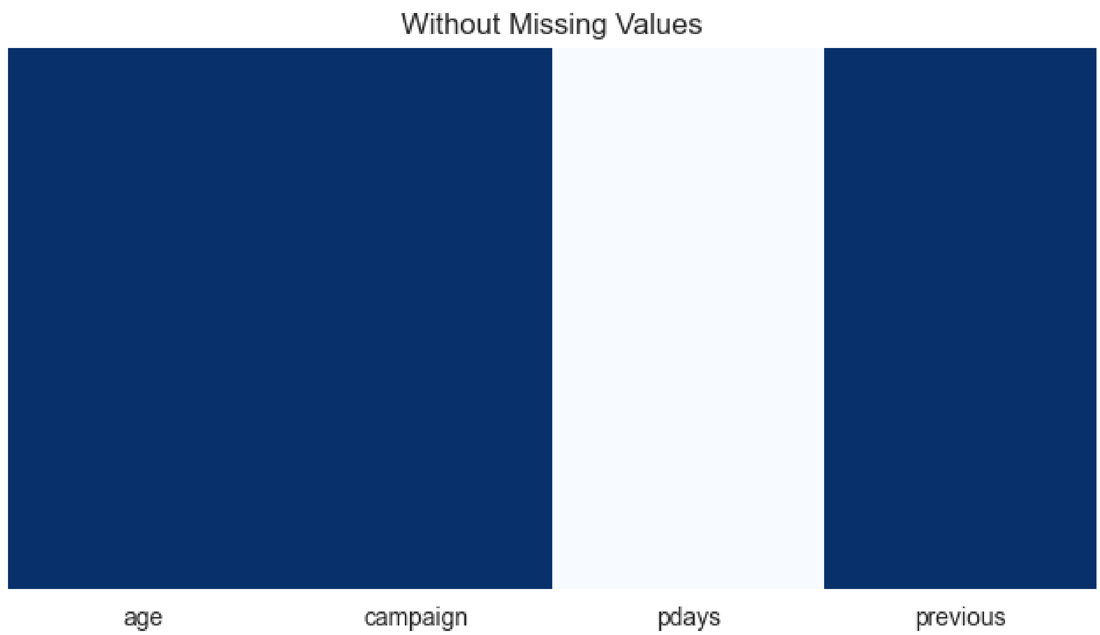
[23]:
```python
# Create a mask to check whether there are values that equal to 999
mask = df2[numerical_variables]==999

# Count the true values along the rows
count_999 = mask.sum()

# Reshape the data for heatmap
count_999 = count_999.to_frame().T

# Create the heatmap
```

```
plt.figure(figsize=(8,4))
sns.heatmap(count_999, yticklabels=False, cbar=False, cmap="Blues")
plt.title('With Missing Values')
plt.show()
```



[24]:

```
# Create a mask to check whether there are values that not-equal to 999
mask = df2[numerical_variables]!=999

# Count the true values along the rows
non_count_999 = mask.sum()

# Reshape the data for heatmap
non_count_999 = non_count_999.to_frame().T

# Create the heatmap
plt.figure(figsize=(8,4))
sns.heatmap(non_count_999, yticklabels=False, cbar=False, cmap="Blues")
plt.title('Without Missing Values')
plt.show()
```



[25]:
```
# Check cross-tabulation between pdays and poutcome
pd.crosstab(df2['pdays'], df2['poutcome'], normalize=True)
```

[25]:

poutcome pdays	failure	nonexistent	success
0	0.000000	0.000000	0.000384
1	0.000000	0.000000	0.000704
2	0.000000	0.000000	0.001663
3	0.000128	0.000000	0.011195
4	0.000064	0.000000	0.003007
5	0.000096	0.000000	0.001183
6	0.000768	0.000000	0.009948
7	0.000352	0.000000	0.001151
8	0.000192	0.000000	0.000256
9	0.000576	0.000000	0.000864
10	0.000192	0.000000	0.001183
11	0.000096	0.000000	0.000544
12	0.000320	0.000000	0.000960
13	0.000192	0.000000	0.000704
14	0.000128	0.000000	0.000448
15	0.000256	0.000000	0.000416
16	0.000032	0.000000	0.000256
17	0.000160	0.000000	0.000096
18	0.000160	0.000000	0.000032
19	0.000032	0.000000	0.000032
20	0.000032	0.000000	0.000000

```
21          0.000064       0.000000   0.000000
22          0.000000       0.000000   0.000096
25          0.000032       0.000000   0.000000
26          0.000000       0.000000   0.000032
27          0.000000       0.000000   0.000032
999         0.103538       0.857408   0.000000
```

As we can see from the above table and the graphs, there is a majority of the values for `pdays` are missing. The majority of these missing values occur when the `poutcome` is `non-existent`. Therefore, we can deduce that majority of the values in `pdays` are missing because the customer was never contacted before. Therefore, we can remove this `pdays` numerical value and add the columns `pdays` and `pdays2` based on the findings.

[26]:
```python
# Copy the data frame
df3 = df2.copy()

def pdays_2_creation(row):
    if row['pdays'] == 999:
        return 'no'
    return 'yes'

df3['pdays2'] = df3.apply(pdays_2_creation, axis = 1)

# Change the values 999 to 30 because number of days in a month can be␣
 ↪considered 30
def pdays_change(row):
    if row['pdays'] == 999:
        return 30
    return row['pdays']

df3['pdays'] = df3.apply(pdays_change, axis = 1)

# View the data
df3.head()
```

[26]:
```
   age           job   marital          education  default housing loan  \
0   49   blue-collar   married           basic.9y  unknown      no   no
1   37  entrepreneur   married  university.degree       no      no   no
2   78       retired   married           basic.4y       no      no   no
3   36        admin.   married  university.degree       no     yes   no
4   59       retired  divorced  university.degree       no      no   no

     contact month day_of_week  campaign  pdays  previous     poutcome    y  \
0   cellular   nov         wed         4     30         0  nonexistent   no
1  telephone   nov         wed         2     30         1      failure   no
2   cellular   jul         mon         1     30         0  nonexistent  yes
3  telephone   may         mon         2     30         0  nonexistent   no
4   cellular   jun         tue         2     30         0  nonexistent   no
```

```
  pdays2
0     no
1     no
2     no
3     no
4     no
```

Step 3 - Removing/Capping Outliers Now, what we need to do is check for the outliers in the dataset and remove them. There are a few ways that you can do to remove the outliers in a dataset.

1. **Z-Score** → Calculate the Z-score for each observation and remove the data points with a Z-Score beyond the threshold.
2. **Inter-quartile Range(IQR) Method** → Compute the IQR for the data and remove data points that fall below $Q_1 - 1.5 \times IQR$ or above $Q_3 + 1.5 \times IQR$ where is the Q_1 25th percentile and Q_3 is the 75th percentile.
3. **Percentile Method** → Rather than using the 25th and 75th percentile, we can define custom ranges and use them.

Using Percentile Method We want to check the outliers with numerical values. Therefore, we first need to remove the categorical values and other binary values present in the data frame. To do that, we can copy the data frame using `df.copy()` and select a slice from that where we have the necessary numerical columns. The following code snippet shows how to get the results using the percentile method.

[27]:
```
numerical_features = df3.dtypes!=object
num_fields = df3.columns[numerical_features].tolist()

def remove_fields_from_list(col):
    num_fields.remove(col)

# We are using pdays2 since we made a categorical variable out of it.
remove_fields = ['pdays', 'previous']

for field in remove_fields:
    remove_fields_from_list(field)

print(f'Numerical Columns: {num_fields}')
```

```
Numerical Columns: ['age', 'campaign']
```

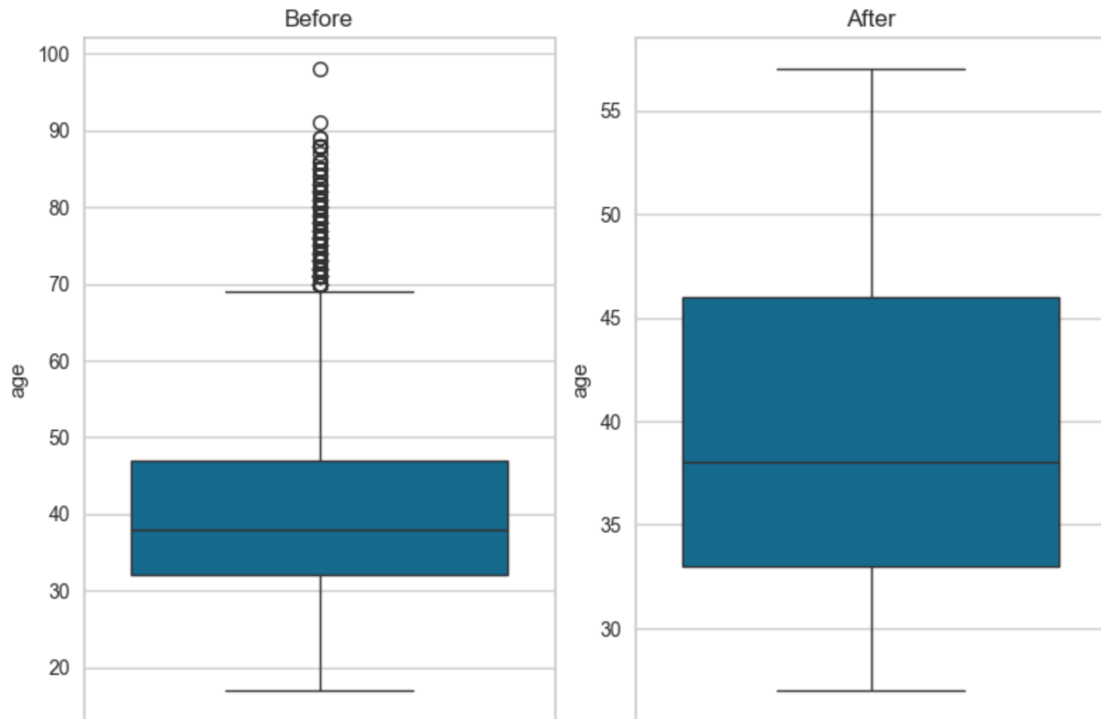
[28]:
```
# Check percentiles before handling the outliers
def check_percentiles(df):
    for col in num_fields:
        print(f'For column {col}')
        print('Min:', df[col].quantile(q = 0))
        print('25% Quartile:', df[col].quantile(q = 0.25))
```

```
        print('50% Quartile:', df[col].quantile(q = 0.50))
        print('75% Quartile:', df[col].quantile(q = 0.75))
        print('Max:', df[col].quantile(q = 1.00),'\n')
check_percentiles(df3)
```

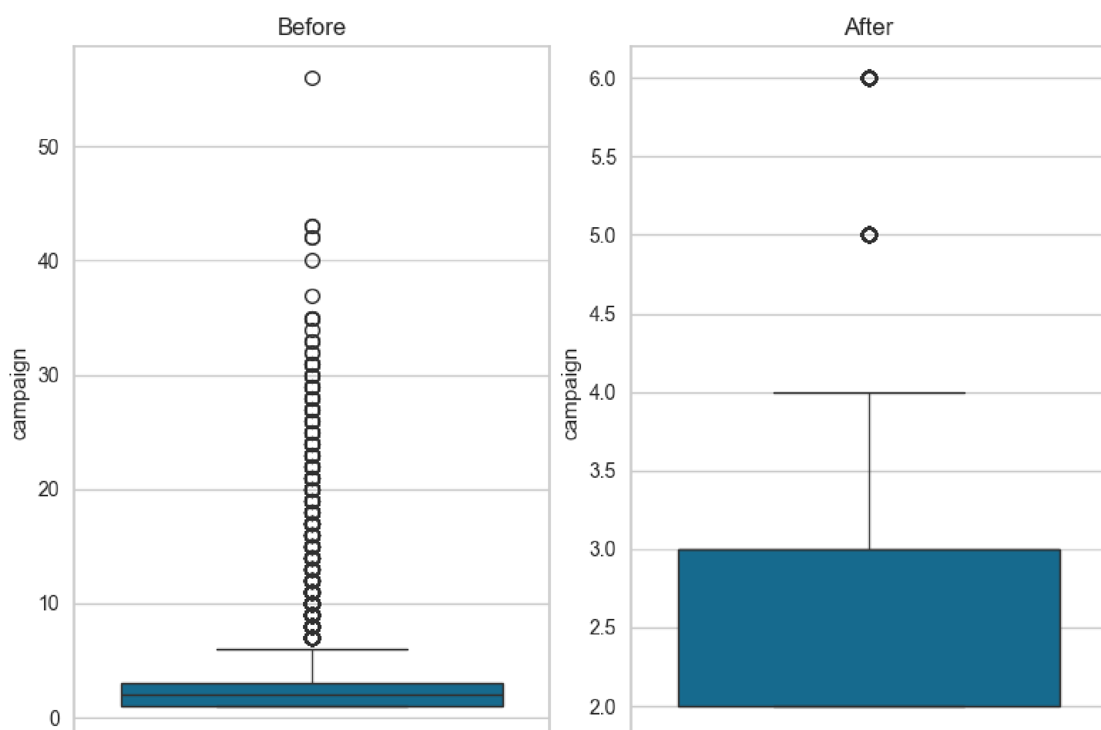
```
For column age
Min: 17.0
25% Quartile: 32.0
50% Quartile: 38.0
75% Quartile: 47.0
Max: 98.0

For column campaign
Min: 1.0
25% Quartile: 1.0
50% Quartile: 2.0
75% Quartile: 3.0
Max: 56.0
```

[29]:
```
# Handling outliers using the percentile method
# Copy the data frame
df4 = df3.copy()

def outliers_using_percentile(field):
  fig, axes = plt.subplots(1,2)
  plt.tight_layout()
  print("Before shape:",df3.shape);
  ## Max and Min Quantile
  max_val = df3[field].quantile(0.95);
  min_val = df3[field].quantile(0.05);
  ## Removing all the outliers
  df4 = df3[(df3[field]>min_val) & (df3[field]<max_val)];
  ## Visulization
  print("After shape:",df4.shape);
  sns.boxplot(df3[field],orient='v',ax=axes[0]);
  axes[0].title.set_text("Before");
  sns.boxplot(df4[field],orient='v',ax=axes[1]);
  axes[1].title.set_text("After");
  plt.show();

for field in num_fields :
  outliers_using_percentile(field)
```

```
Before shape: (31264, 16)
After shape: (27678, 16)
```




```
Before shape: (31264, 16)
After shape: (16384, 16)
```

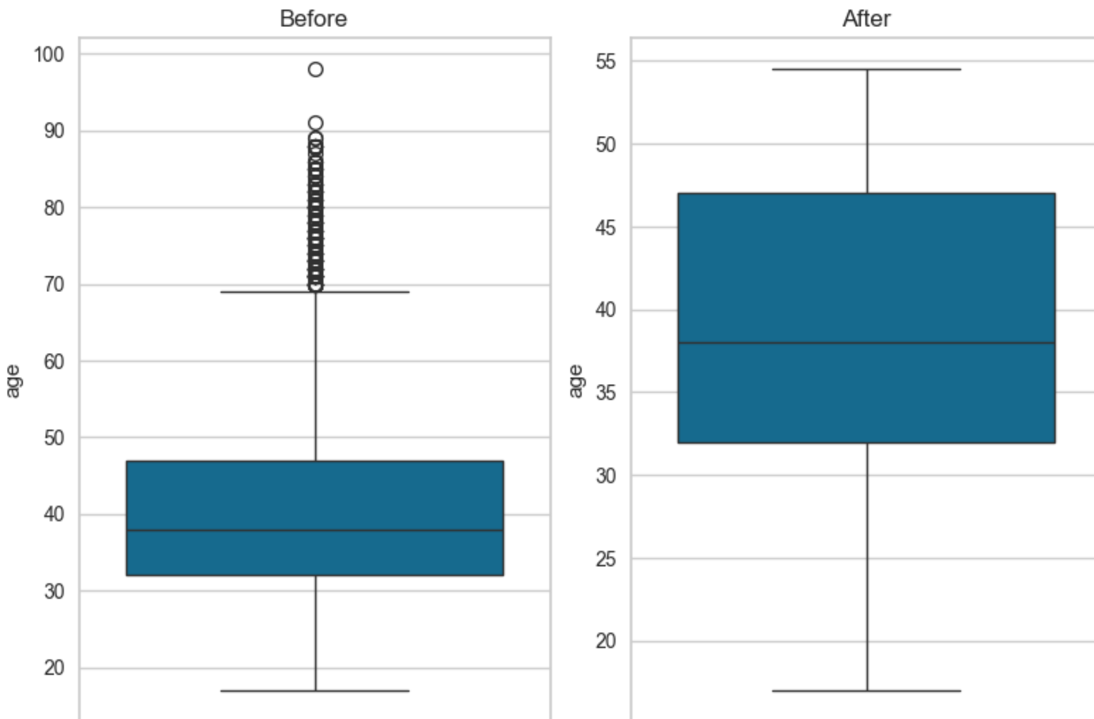


It can be seen that if we use the `Percentile Method` there will be a huge loss in data. Therefore, we need to find an alternative option to handle the outliers. Now, let's check what kind of results we can get from using the `Inter-quartile Range` method.

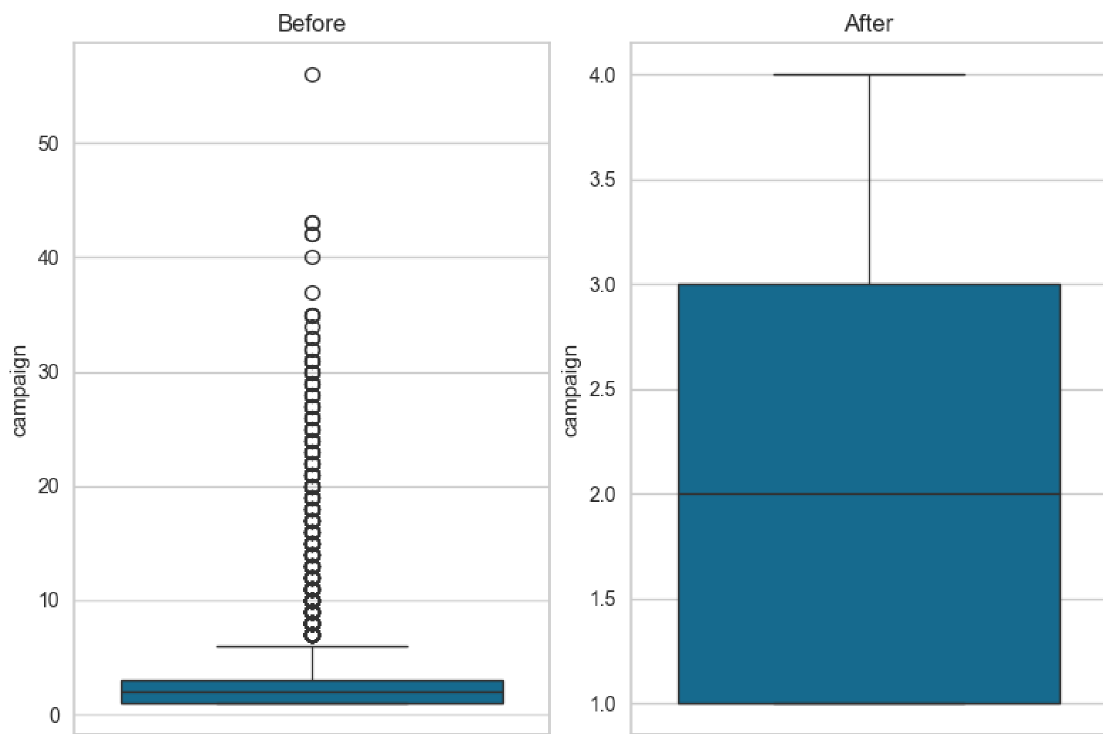
Inter-quartile Range Method The following code snippet is used to get the calculated `Inter-quartile Range` and get the necessary results. Next, let's check the results we have obtained from the above code snippet.

[30]:
```
# Handling outliers using IQR method
df4 = df3.copy()

def outliers_usinq_iqr(field):
    fig, axes = plt.subplots(1,2)
    plt.tight_layout()
    print("Previous shape with outlier: ",df3.shape)
    sns.boxplot(df3[field],orient='v',ax=axes[0])
    axes[0].title.set_text("Before")

    # calculate IQR
    q1 = df3[field].quantile(0.25)
    q3 = df3[field].quantile(0.75)
    print(f'Q1 : {q1} Q3 : {q3}')
    iqr = q3 - q1
    print('IQR : ', iqr)
    lower_limit = q1 - 1.5*iqr
    upper_limit = q1 + 1.5*iqr
    print('Lower limit : {lower_limit} Upper limit : {upper_limit}')
    df4[field] = np.where(df4[field] > upper_limit,upper_limit,df4[field])
    df4[field] = np.where(df4[field]<lower_limit,lower_limit,df4[field])
    print("Shape after removing outliers:", df4.shape)
    sns.boxplot(df4[field],orient='v',ax=axes[1])
    axes[1].title.set_text("After")
    plt.show()

for field in num_fields :
  outliers_usinq_iqr(field)
```

```
Previous shape with outlier:  (31264, 16)
Q1 : 32.0 Q3 : 47.0
IQR :  15.0
Lower limit : {lower_limit} Upper limit : {upper_limit}
Shape after removing outliers: (31264, 16)
```




```
Previous shape with outlier:  (31264, 16)
Q1 : 1.0 Q3 : 3.0
IQR :  2.0
Lower limit : {lower_limit} Upper limit : {upper_limit}
Shape after removing outliers: (31264, 16)
```



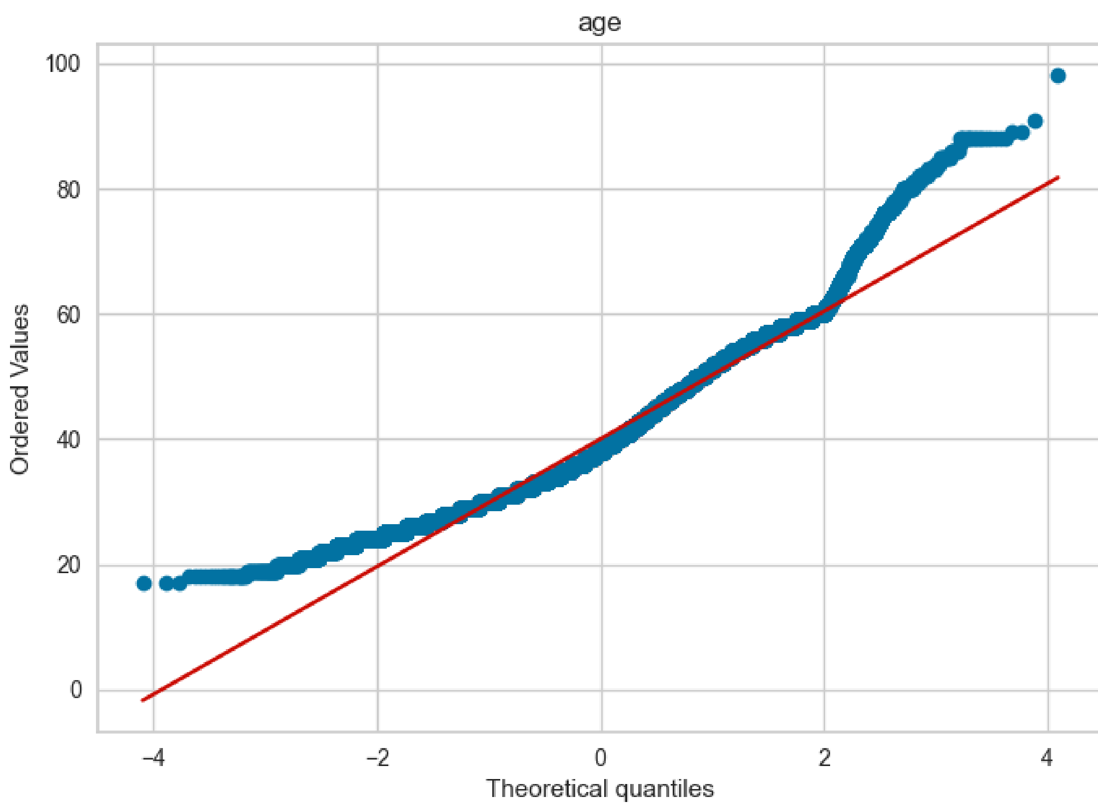
From the results, we can see that `IQR` is better since it does not reduce the dataset drastically.

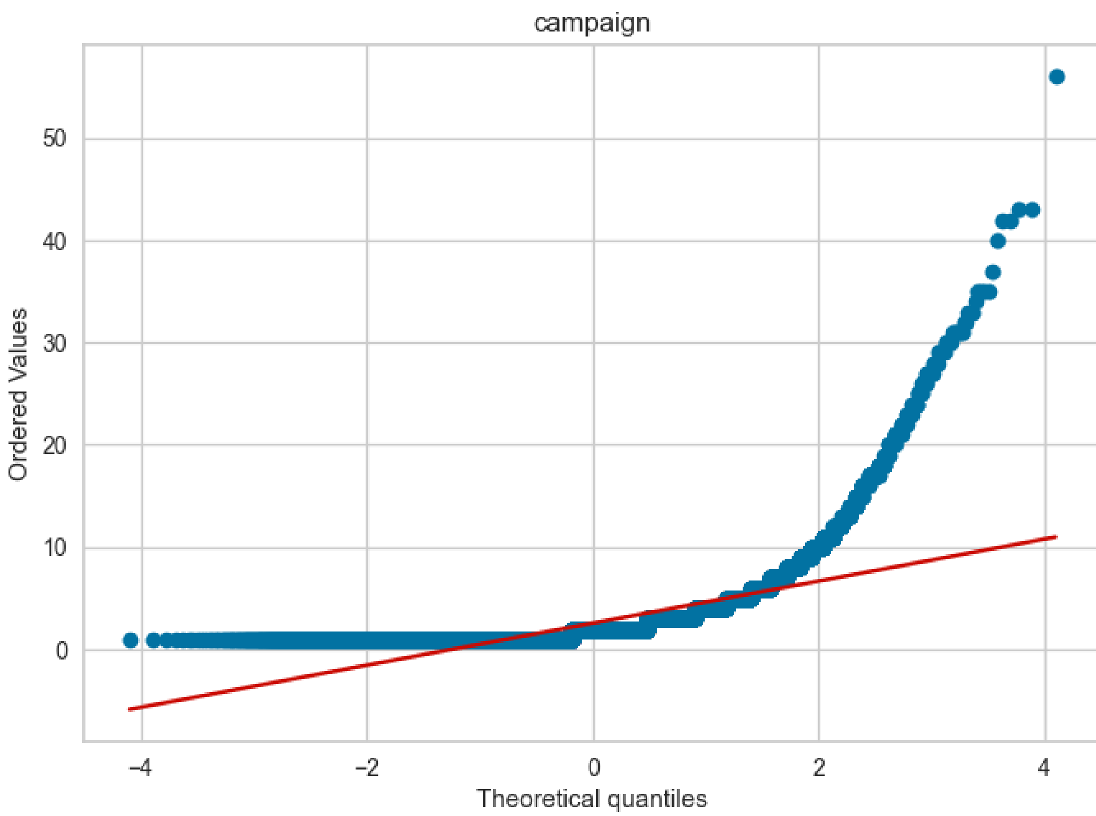
1.2.2 Producing Q-Q Plots, Histograms, and Applying Necessary Transformations

First, we need to produce Q-Q plots and histograms to check whether the data distribution in the dataset is normal or not.

Step 1 - Q-Q Plots Q-Q(Quantile-Quantile) plots are typically used to assess whether a given dataset follows a particular distribution or not. It usually checks whether a dataset follows a normal(Gaussian) distribution and based on the exploratory data extracted from it, we can transform the data and normalize the dataset for more accurate results. The following code snippet is used to make Q-Q plots for the dataset.

[31]:
```
# We are using the outliers removed data frame in here
def draw_qq_plots(df, col):
    # Create and show the plot
    stats.probplot(df[col], dist='norm', plot=plt)
    plt.title(col)
    plt.show()

for field in num_fields:
    draw_qq_plots(df,field)
```

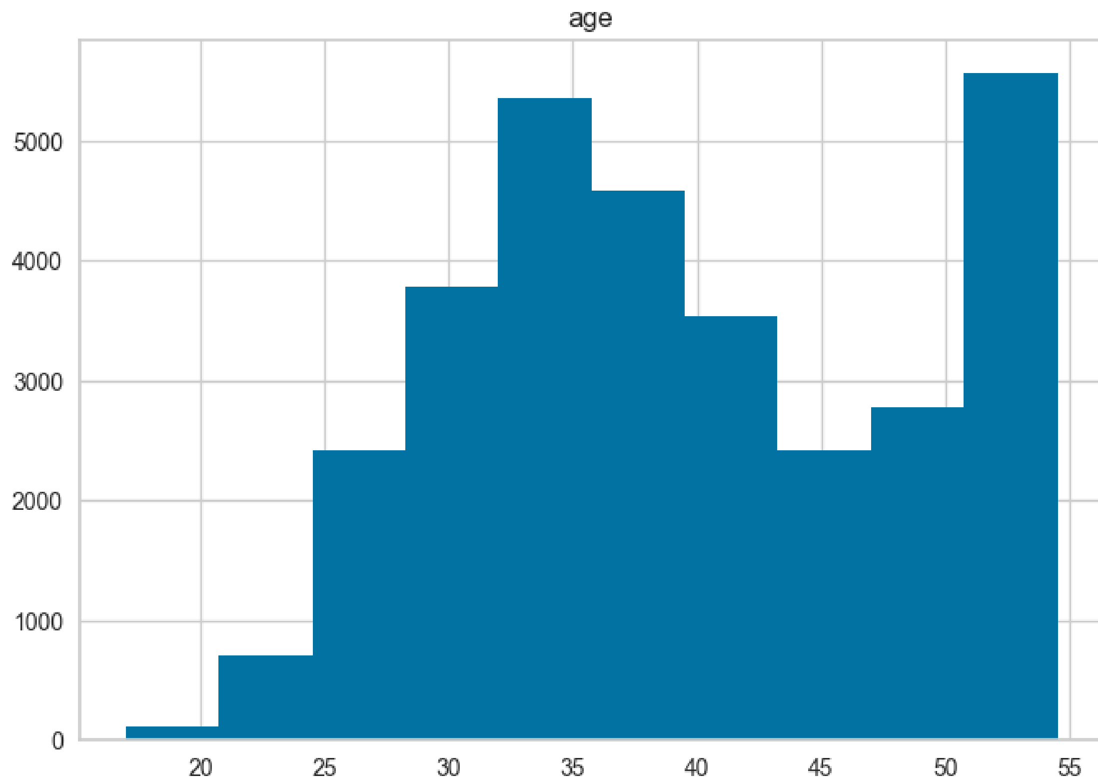

age
Ordered Values
Theoretical quantiles
100
80
60
40
20
0
−4
−2
0
2
4



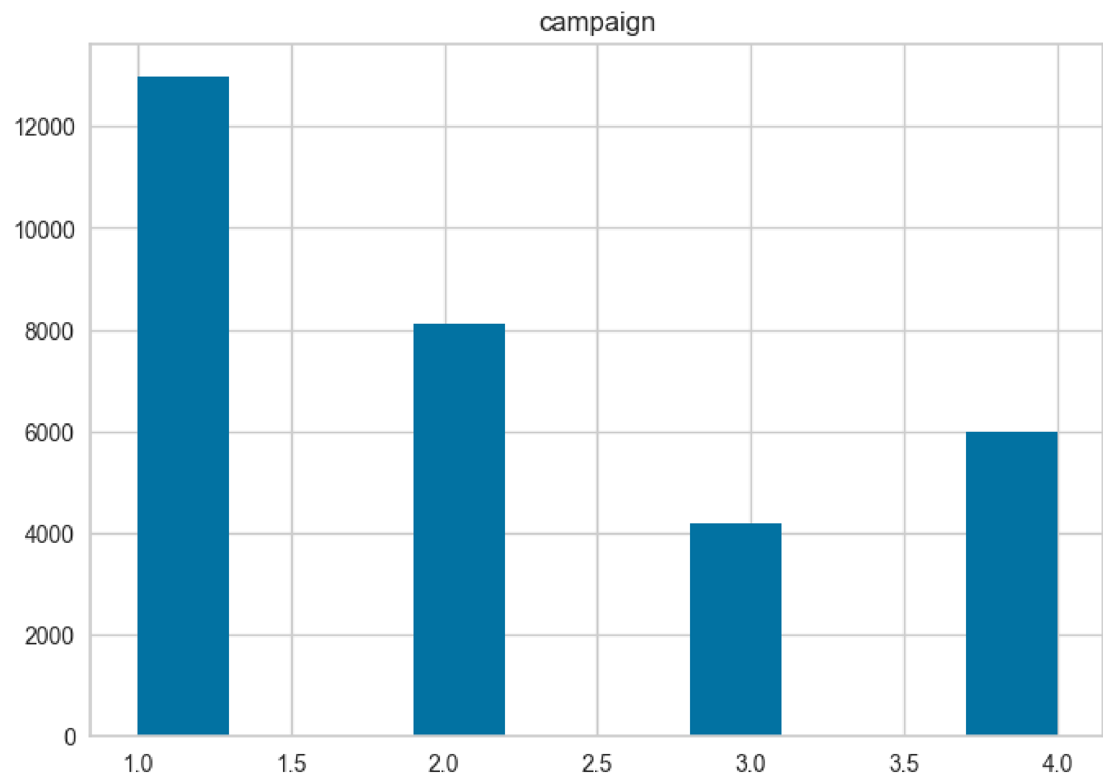
As you can see most of the Q-Q plot data are above the 45-degree line, which indicates that there is normal distribution in the dataset. Now, let's check the histograms to see whether there are any skewed columns in the dataset.

Step 2 - Histograms Histograms allow us to check the data distribution. And using them we can find out whether a dataset is right-skewed or left-skewed. The following code is used to plot the histograms.

[32]:
```
def draw_histograms(df, field):
    plt.hist(df[field], color='b')
    plt.title(field)
    plt.figure()
    plt.show()

# Draw histograms
for field in num_fields:
    draw_histograms(df4, field)
```




```
<Figure size 800x550 with 0 Axes>
```

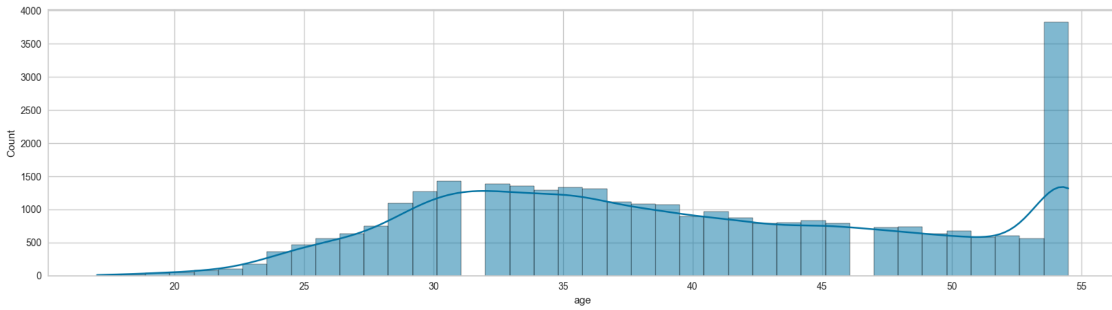


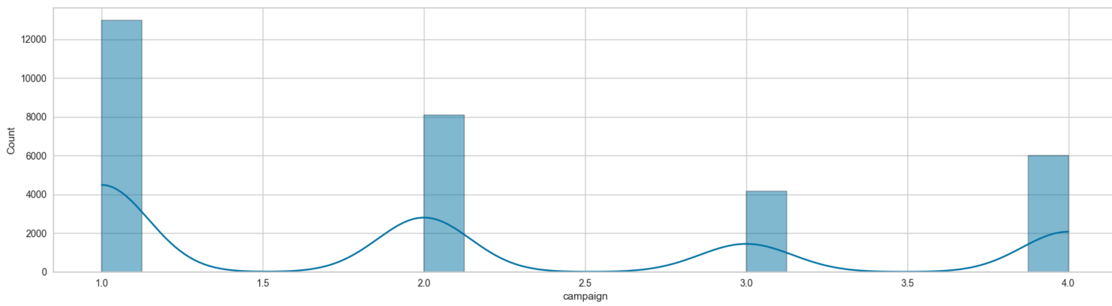

```
<Figure size 800x550 with 0 Axes>
```

Since it is hard to understand whether there is a skew or not in the dataset, we can use the `histplot()` function in the `Yellowbrick` library.

Step 3 - Histplots The function to get histplots is shown below.

[33]:
```
# Generate histplot to analyze more closely
def draw_histplots(df, field):
    plt.figure(figsize=(20,5))
    sns.histplot(df[field], kde=True)
    plt.show()

# Draw histplots
for field in num_fields:
    draw_histplots(df4, field)
```





Step 4 - Applying Transformations From the graphs, we can see that the following columns are right-skewed.

- `age`

The following techniques are used to address the skewed data. 1. Right Skewed Data →
(a) Logarithmic Transformation
(b) Square-root Transformation 3. Left Skewed Data →
(a) Exponential Transformation The `Logarithmic Transformation` cannot be used with 0 and negative values. Therefore, to address those values we can use the `Square-root Transformation` with a little tweak as shown below.

```
FunctionTransformer(lambda x:np.sqrt(abs(x)), validate=True)
```

The following code explains how you can transform the necessary data.

[34]:
```
# Identified columns for transformation by looking at the histograms
right_skewed = ['age']
identified_cols = right_skewed
```

[35]:
```
# Copy the data frame
data = df4.copy()

# Create a FunctionTransformer object with square root transformation for␣
↪right-skewed data
```

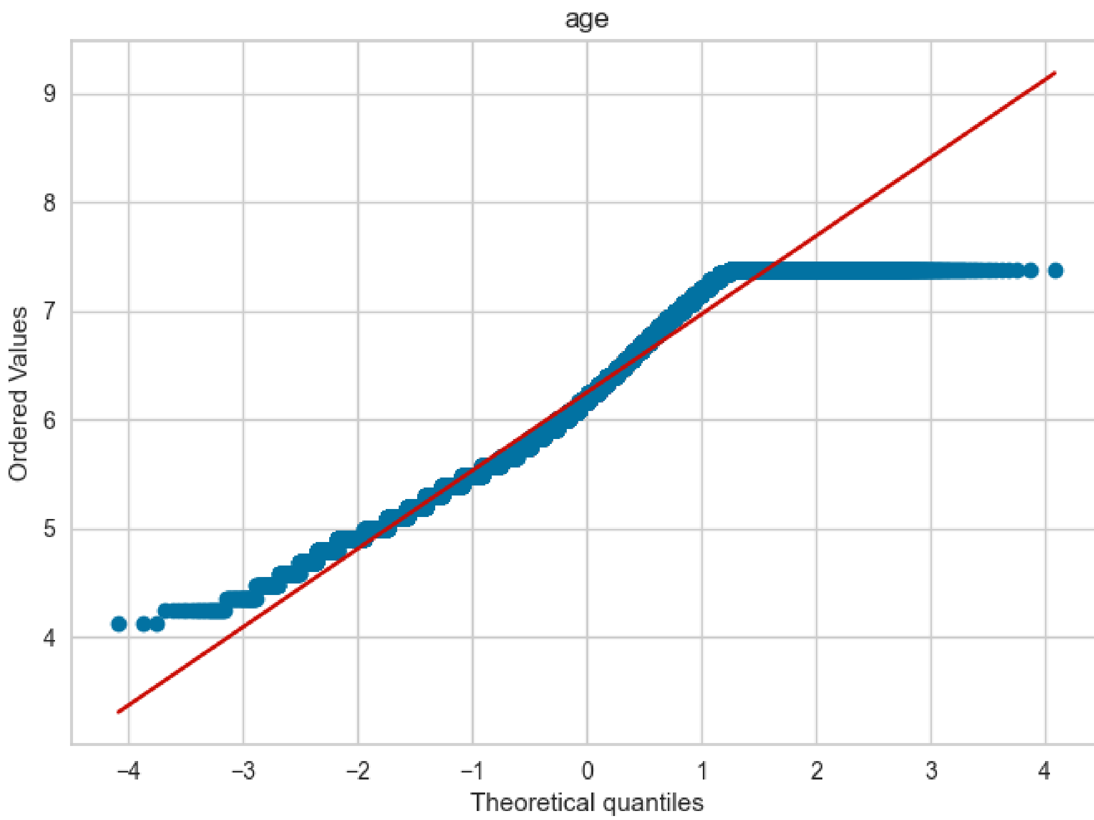
```
square_root_transformer = FunctionTransformer(lambda x: np.sqrt(x), validate =
 ↪True)

# Apply the transformations to the data
data_new = square_root_transformer.transform(data[right_skewed])

# Create a new data frame
df5 = pd.DataFrame(data_new, columns=right_skewed)
```

[36]:
```
df5.head(100)
```

[36]:
```
          age
0   7.000000
1   6.082763
2   7.382412
3   6.000000
4   7.382412
..       …
95  7.071068
96  6.324555
97  6.928203
98  5.656854
99  5.385165

[100 rows x 1 columns]
```

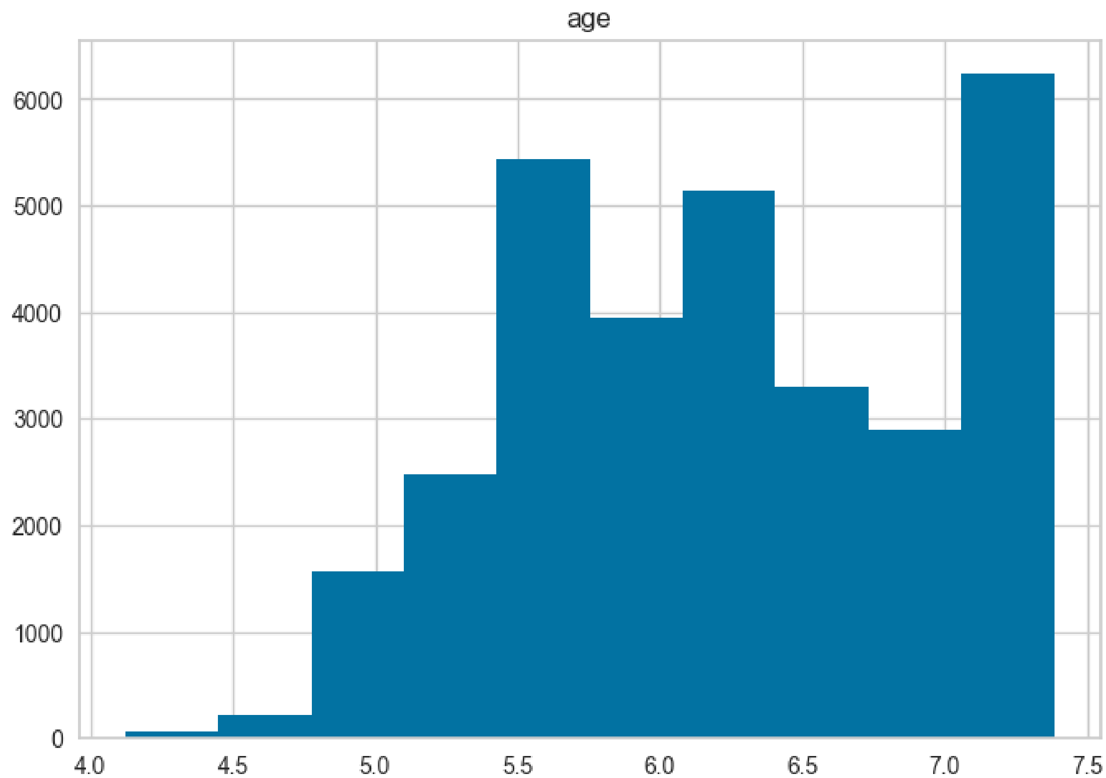
Step 5 - Verifying Let's check the Q-Q plots again to see if the transformation applied properly for the transformed columns.

[37]:
```
# Check q-q plots for the transformed data
for field in identified_cols:
    draw_qq_plots(df5, field)
```



Similarly, let's check the Histograms to see if the transformation applied properly for the transformed columns.

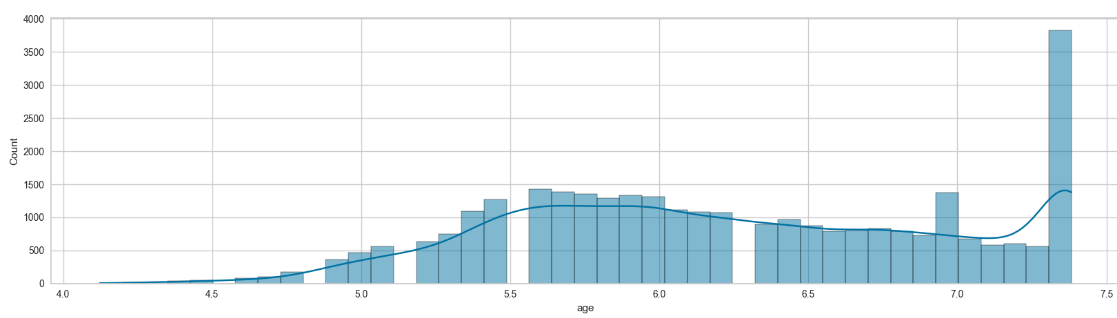
[38]:
```
# Check histograms for the transformed data
for field in identified_cols:
    draw_histograms(df5, field)
```




```
<Figure size 800x550 with 0 Axes>
```

Similarly, let's check the Histplots to see if the transformation applied properly for the transformed columns.

[39]:
```python
# Check histplots for the transformed data
for field in identified_cols:
    draw_histplots(df5, field)
```



1.2.3 Applying Suitable Feature Encoding Techniques

Common Techniques for Feature Coding • **One Hot Encoding**: Use some variables to predict unknown values of other variables.
- **Label Encoding**: Find human-readable patterns that describe the data.
- **Ordinal Encoding**: Ordinal encoding is used when the categories in a variable have a natural ordering.

Since the `waterfront` and `view` variables are label encoded for the boolean values, we can simply concatenate them to the current data frame.

Therefore, our data frame contains only numerical values(according to `df.info()` function)and, we do not need to do feature encodings.

[40]:
```
df6 = df4.copy()
df6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31264 entries, 0 to 31263
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   age          31264 non-null  float64
 1   job          31264 non-null  object
 2   marital      31264 non-null  object
 3   education    31264 non-null  object
 4   default      31264 non-null  object
 5   housing      31264 non-null  object
 6   loan         31264 non-null  object
 7   contact      31264 non-null  object
 8   month        31264 non-null  object
 9   day_of_week  31264 non-null  object
 10  campaign     31264 non-null  float64
 11  pdays        31264 non-null  int64
 12  previous     31264 non-null  int64
 13  poutcome     31264 non-null  object
 14  y            31264 non-null  object
 15  pdays2       31264 non-null  object
dtypes: float64(2), int64(2), object(12)
memory usage: 3.8+ MB
```

Filter the categorical columns using a mask and turn it into a list.

[41]:
```
# Filter the categorical columns using a mask and turn it to a list
categorical_features = df6.dtypes == object
categorical_columns = df6.columns[categorical_features].tolist()

# Don't use y column as well
categorical_columns.remove('y')
print(f'Categorical columns: {categorical_columns}')
```

```
Categorical columns: ['job', 'marital', 'education', 'default', 'housing',
'loan', 'contact', 'month', 'day_of_week', 'poutcome', 'pdays2']
```

Encode values using One Hot Encoder

[42]:
```python
df_one_hot_encoder = df6.copy()
df_one_hot_encoder = pd.get_dummies(df_one_hot_encoder, columns=categorical_columns)

print(f'Shape of one hot encoded data frame: {df_one_hot_encoder.shape}')
```

```
Shape of one hot encoded data frame: (31264, 59)
```

View the results

[43]:
```python
df_one_hot_encoder.head()
```

[43]:
```
    age  campaign  pdays  previous    y  job_admin.  job_blue-collar  \
0  49.0       4.0     30         0   no       False             True
1  37.0       2.0     30         1   no       False            False
2  54.5       1.0     30         0  yes       False            False
3  36.0       2.0     30         0   no        True            False
4  54.5       2.0     30         0   no       False            False

   job_entrepreneur  job_housemaid  job_management  …  day_of_week_fri  \
0             False          False           False  …            False
1              True          False           False  …            False
2             False          False           False  …            False
3             False          False           False  …            False
4             False          False           False  …            False

   day_of_week_mon  day_of_week_thu  day_of_week_tue  day_of_week_wed  \
0            False            False            False             True
1            False            False            False             True
2             True            False            False            False
3             True            False            False            False
4            False            False             True            False

   poutcome_failure  poutcome_nonexistent  poutcome_success  pdays2_no  \
0             False                  True             False       True
1              True                 False             False       True
2             False                  True             False       True
3             False                  True             False       True
4             False                  True             False       True

   pdays2_yes
0       False
1       False
2       False
```

```
3       False
4       False

[5 rows x 59 columns]
```

Change the y column using boolean encoding.

[44]:
```python
# Copy the data frame
df_one_hot_encoder_copy = df_one_hot_encoder.copy()

# First change the values of y (yes, no) to True and False
def change_y_values(row):
    if row['y'].casefold() == 'yes':
        return 1
    return 0

df_one_hot_encoder_copy['y'] = df_one_hot_encoder.apply(change_y_values, axis=1)

# Get all columns
df_one_hot_encoder_copy = df_one_hot_encoder_copy.astype(int)

# Check the values
df_one_hot_encoder_copy.head()
```

[44]:
```
   age  campaign  pdays  previous  y  job_admin.  job_blue-collar  \
0   49         4     30         0  0           0                1
1   37         2     30         1  0           0                0
2   54         1     30         0  1           0                0
3   36         2     30         0  0           1                0
4   54         2     30         0  0           0                0

   job_entrepreneur  job_housemaid  job_management  …  day_of_week_fri  \
0                 0              0               0  …                0
1                 1              0               0  …                0
2                 0              0               0  …                0
3                 0              0               0  …                0
4                 0              0               0  …                0

   day_of_week_mon  day_of_week_thu  day_of_week_tue  day_of_week_wed  \
0                0                0                0                1
1                0                0                0                1
2                1                0                0                0
3                1                0                0                0
4                0                0                1                0

   poutcome_failure  poutcome_nonexistent  poutcome_success  pdays2_no  \
0                 0                     1                 0          1
1                 1                     0                 0          1
```

```
2                  0                      1                   0           1
3                  0                      1                   0           1
4                  0                      1                   0           1

   pdays2_yes
0            0
1            0
2            0
3            0
4            0

[5 rows x 59 columns]
```

Drop the columns that were used for transformations

[45]:
```python
# Drop the columns that were used for transformations
df_one_hot_encoder_copy = df_one_hot_encoder_copy.drop(columns = right_skewed)
df_one_hot_encoder_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31264 entries, 0 to 31263
Data columns (total 58 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   campaign                       31264 non-null  int64
 1   pdays                          31264 non-null  int64
 2   previous                       31264 non-null  int64
 3   y                              31264 non-null  int64
 4   job_admin.                     31264 non-null  int64
 5   job_blue-collar                31264 non-null  int64
 6   job_entrepreneur               31264 non-null  int64
 7   job_housemaid                  31264 non-null  int64
 8   job_management                 31264 non-null  int64
 9   job_retired                    31264 non-null  int64
 10  job_self-employed              31264 non-null  int64
 11  job_services                   31264 non-null  int64
 12  job_student                    31264 non-null  int64
 13  job_technician                 31264 non-null  int64
 14  job_unemployed                 31264 non-null  int64
 15  job_unknown                    31264 non-null  int64
 16  marital_divorced               31264 non-null  int64
 17  marital_married                31264 non-null  int64
 18  marital_single                 31264 non-null  int64
 19  education_basic.4y             31264 non-null  int64
 20  education_basic.6y             31264 non-null  int64
 21  education_basic.9y             31264 non-null  int64
 22  education_high.school          31264 non-null  int64
 23  education_illiterate           31264 non-null  int64
```

```
 24   education_professional.course  31264 non-null  int64
 25   education_university.degree    31264 non-null  int64
 26   education_unknown              31264 non-null  int64
 27   default_no                     31264 non-null  int64
 28   default_unknown                31264 non-null  int64
 29   default_yes                    31264 non-null  int64
 30   housing_no                     31264 non-null  int64
 31   housing_unknown                31264 non-null  int64
 32   housing_yes                    31264 non-null  int64
 33   loan_no                        31264 non-null  int64
 34   loan_unknown                   31264 non-null  int64
 35   loan_yes                       31264 non-null  int64
 36   contact_cellular               31264 non-null  int64
 37   contact_telephone              31264 non-null  int64
 38   month_apr                      31264 non-null  int64
 39   month_aug                      31264 non-null  int64
 40   month_dec                      31264 non-null  int64
 41   month_jul                      31264 non-null  int64
 42   month_jun                      31264 non-null  int64
 43   month_mar                      31264 non-null  int64
 44   month_may                      31264 non-null  int64
 45   month_nov                      31264 non-null  int64
 46   month_oct                      31264 non-null  int64
 47   month_sep                      31264 non-null  int64
 48   day_of_week_fri                31264 non-null  int64
 49   day_of_week_mon                31264 non-null  int64
 50   day_of_week_thu                31264 non-null  int64
 51   day_of_week_tue                31264 non-null  int64
 52   day_of_week_wed                31264 non-null  int64
 53   poutcome_failure               31264 non-null  int64
 54   poutcome_nonexistent           31264 non-null  int64
 55   poutcome_success               31264 non-null  int64
 56   pdays2_no                      31264 non-null  int64
 57   pdays2_yes                     31264 non-null  int64
dtypes: int64(58)
memory usage: 13.8 MB
```

Concatenate the transformed data and one hot encoded data

[46]:
```
# Concat the transformed data and one hot encoded data
df_preprocessed = pd.concat([df5, df_one_hot_encoder_copy], join='outer',␣
↪axis=1)
df_preprocessed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31264 entries, 0 to 31263
Data columns (total 59 columns):
 #   Column                          Non-Null Count  Dtype
```

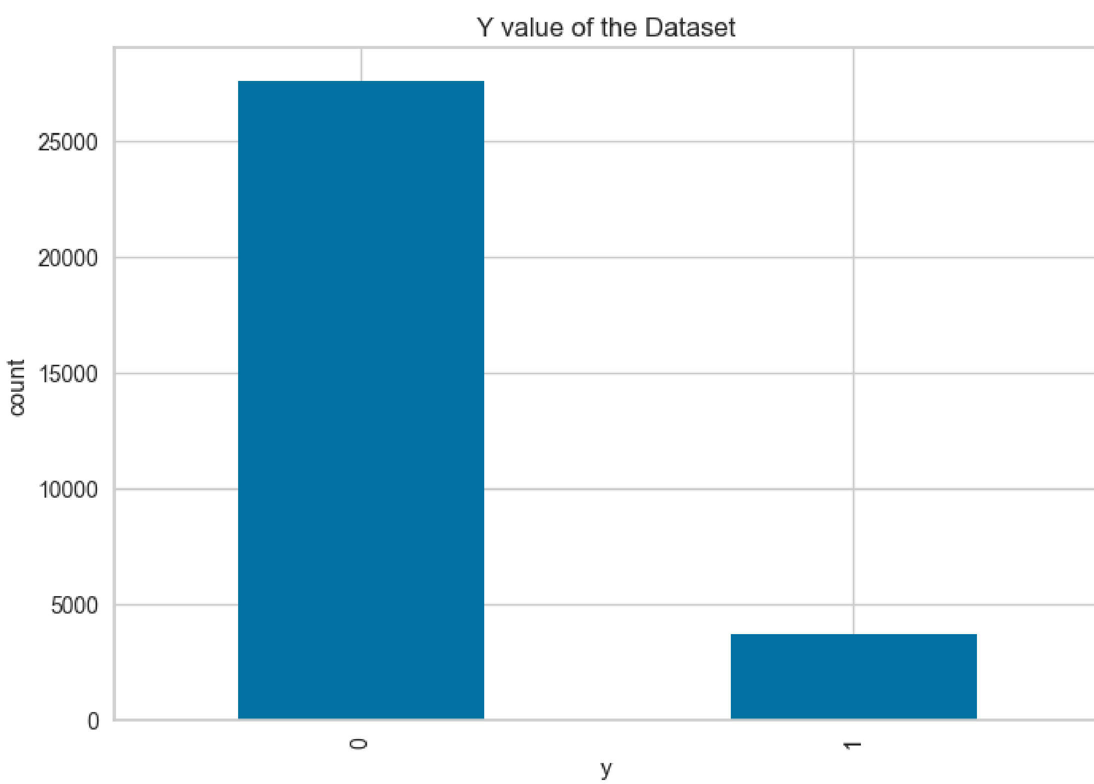
```
---  ------                         --------------  -----
 0   age                            31264 non-null  float64
 1   campaign                       31264 non-null  int64
 2   pdays                          31264 non-null  int64
 3   previous                       31264 non-null  int64
 4   y                              31264 non-null  int64
 5   job_admin.                     31264 non-null  int64
 6   job_blue-collar                31264 non-null  int64
 7   job_entrepreneur               31264 non-null  int64
 8   job_housemaid                  31264 non-null  int64
 9   job_management                 31264 non-null  int64
 10  job_retired                    31264 non-null  int64
 11  job_self-employed              31264 non-null  int64
 12  job_services                   31264 non-null  int64
 13  job_student                    31264 non-null  int64
 14  job_technician                 31264 non-null  int64
 15  job_unemployed                 31264 non-null  int64
 16  job_unknown                    31264 non-null  int64
 17  marital_divorced               31264 non-null  int64
 18  marital_married                31264 non-null  int64
 19  marital_single                 31264 non-null  int64
 20  education_basic.4y             31264 non-null  int64
 21  education_basic.6y             31264 non-null  int64
 22  education_basic.9y             31264 non-null  int64
 23  education_high.school          31264 non-null  int64
 24  education_illiterate           31264 non-null  int64
 25  education_professional.course  31264 non-null  int64
 26  education_university.degree    31264 non-null  int64
 27  education_unknown              31264 non-null  int64
 28  default_no                     31264 non-null  int64
 29  default_unknown                31264 non-null  int64
 30  default_yes                    31264 non-null  int64
 31  housing_no                     31264 non-null  int64
 32  housing_unknown                31264 non-null  int64
 33  housing_yes                    31264 non-null  int64
 34  loan_no                        31264 non-null  int64
 35  loan_unknown                   31264 non-null  int64
 36  loan_yes                       31264 non-null  int64
 37  contact_cellular               31264 non-null  int64
 38  contact_telephone              31264 non-null  int64
 39  month_apr                      31264 non-null  int64
 40  month_aug                      31264 non-null  int64
 41  month_dec                      31264 non-null  int64
 42  month_jul                      31264 non-null  int64
 43  month_jun                      31264 non-null  int64
 44  month_mar                      31264 non-null  int64
 45  month_may                      31264 non-null  int64
 46  month_nov                      31264 non-null  int64
```

```
 47  month_oct                    31264 non-null  int64
 48  month_sep                    31264 non-null  int64
 49  day_of_week_fri              31264 non-null  int64
 50  day_of_week_mon              31264 non-null  int64
 51  day_of_week_thu              31264 non-null  int64
 52  day_of_week_tue              31264 non-null  int64
 53  day_of_week_wed              31264 non-null  int64
 54  poutcome_failure             31264 non-null  int64
 55  poutcome_nonexistent         31264 non-null  int64
 56  poutcome_success             31264 non-null  int64
 57  pdays2_no                    31264 non-null  int64
 58  pdays2_yes                   31264 non-null  int64
dtypes: float64(1), int64(58)
memory usage: 14.1 MB
```

Checking Imbalance Since this is a binary classification problem, we need to check whether the dataset has the same amount of values for both binary values. If not we have to address that imbalance problem. To check that, we can simply plot a graph.

[47]:
```
ax = df_preprocessed['y'].value_counts().plot(kind='bar', color='b')
ax.set_title('Y value of the Dataset')
ax.set_ylabel('count')
ax.set_xlabel('y')
```

[47]: `Text(0.5, 0, 'y')`

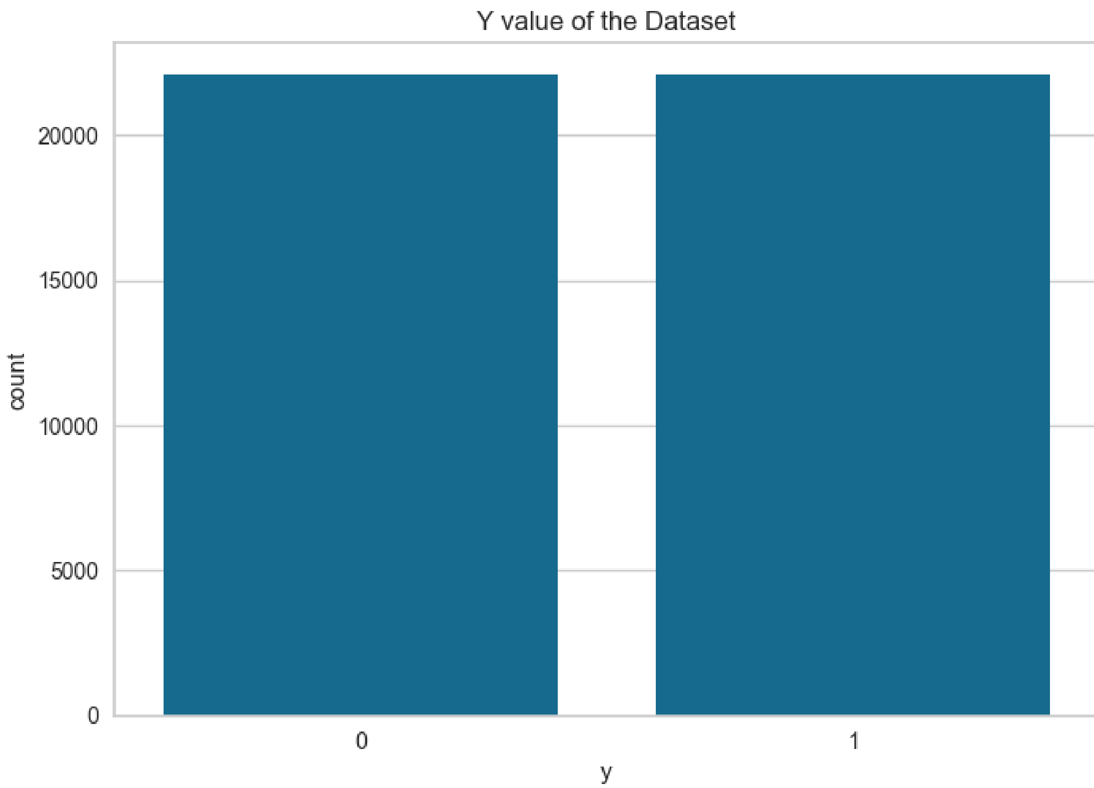


Since the values in category 1 are lacking we can deduce that there is a data imbalance. To address data imbalances we can do the following. - Oversampling → Increase the number of instances in the minority class by generating synthetic samples or duplicating existing ones. - Undersampling → Reduce the number of instances in the majority class to match the minority class.

The following code shows how you can perform oversampling with `SMOTE` algorithm.

[48]:
```python
# Copy the data frame
processed_data = df_preprocessed.copy()

# Separating features and result vectors
y = processed_data[['y']]
X = processed_data.drop(['y'], axis=1)

os = SMOTE(random_state = 42)
X_class_train, X_test, y_class_train, y_test = train_test_split(X, y,
 ↪test_size=0.2, random_state=42)

columns = X_class_train.columns

data_X, data_y = os.fit_resample(X_class_train, y_class_train)

smoted_X = pd.DataFrame(data=data_X, columns=columns)
smoted_y = pd.DataFrame(data=data_y, columns=['y'])

sns.countplot(x='y', data=smoted_y)
plt.title('Y value of the Dataset')
```

[48]:
```
Text(0.5, 1.0, 'Y value of the Dataset')
```



1.2.4 Scale/Standardize Data

Feature Scaling, also known as data normalization, is a technique used to standardize the range of independent variables of features of data. There are a few ways to scale or standardize data.

- Min/Max Scaling(Normalization) → Rescales features to a range between 0 and 1.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- Standardization(Z-score Normalization) → Centers the data around 0 with a standard deviation of 1.

$$X_{std} = \frac{X - mean(X)}{std(X)}$$

- Robust Scaling → Similar to standardization but uses the median and the inter-quartile range.

$$X_{robust} = \frac{X - median(X)}{Q_3(X) - Q_1(X)}$$

In this case, we are using the `Min-max Scaling` to rescale features to a range between 0 and 1, since it helps more in the classification process. The code snippet used to scale the data is shown below.

[49]:
```python
# Copy the data frame
df_scaled = smoted_X.copy()

df_scaled = df_scaled[numerical_variables]

# Create the scaler object
scaler = MinMaxScaler()
df_scaled[df_scaled.columns] = scaler.fit_transform(df_scaled[df_scaled.
 ↪columns])

# Summarize
print(df_scaled.columns)
df_scaled.head()
```

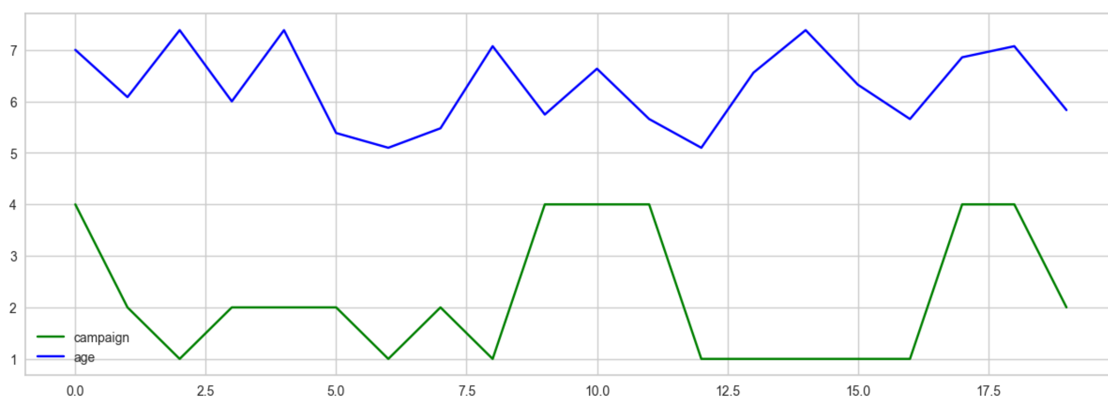
```
Index(['age', 'campaign', 'pdays', 'previous'], dtype='object')
```

[49]:

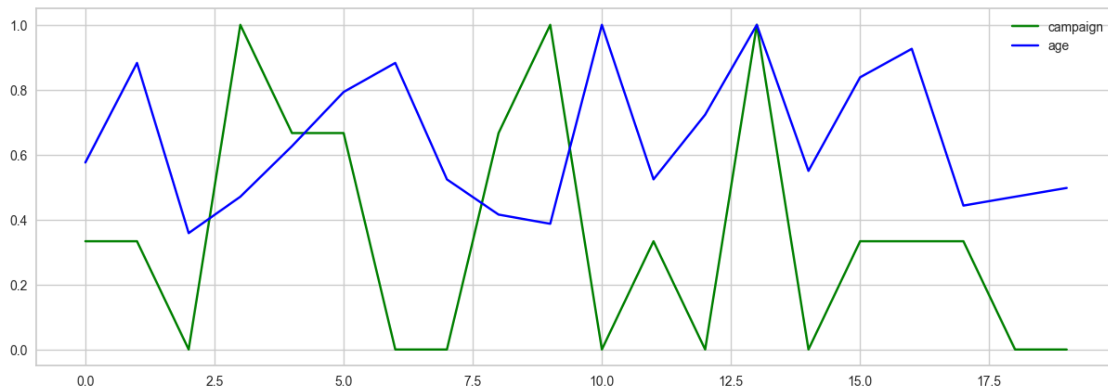
	age	campaign	pdays	previous
0	0.575857	0.333333	1.0	0.0
1	0.882671	0.333333	1.0	0.0
2	0.358480	0.000000	1.0	0.0
3	0.470575	1.000000	1.0	0.0
4	0.626302	0.666667	1.0	0.0

The graphs of before and after scaling the data are shown below.

[50]:
```python
# Before scaling
plt.figure(figsize=(15, 5));
plt.plot(df_preprocessed.campaign[:20], color='green', label='campaign')
plt.plot(df_preprocessed.age[:20], color='blue', label='age')
plt.legend(loc='best')
plt.show()
```



[51]:
```python
# After scaling
plt.figure(figsize=(15, 5));
plt.plot(df_scaled.campaign[:20], color='green', label='campaign')
plt.plot(df_scaled.age[:20], color='blue', label='age')
plt.legend(loc='best')
plt.show()
```



1.2.5 Data Discretization

Feature discretization, also known as binning or discretization, is the process of transforming continuous numerical features into discrete intervals or bins. This helps to handle continuous data more effectively, simplify models, and improve their interpretability. In the classification process, this allows us to classify better.. - Equal-Width Binning(Fixed Width) → Divides the range of values into equally sized bins. - Equal-Frequency Binning(Fixed Frequency) → Divides the data into bins with approximately the same number of data points in each bin. - Custom Binning → Bins are defined based on domain knowledge or specific requirements. - Quantile Binning → Bins are created based on quantiles of the data. - Cluster-Based Binning → Uses clustering algorithms to group similar data points into bins. In our case, we will be using `KBinsDiscretizer` for data discretization.

[52]:
```python
## Apply feature discretization

# Copy the data frame
df_discretization = df_scaled.copy()

kbins = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='uniform')
df_discretization[df_discretization.columns] = kbins.
 ↪fit_transform(df_discretization[df_discretization.columns])

df_discretization.head()
```

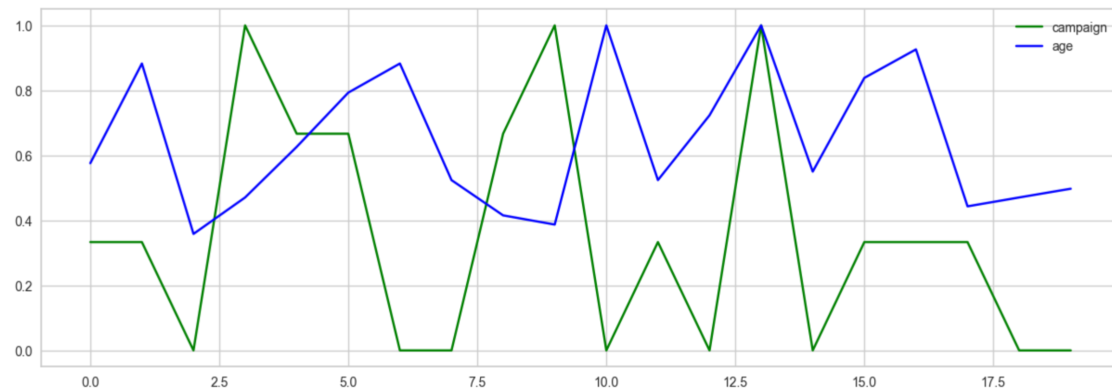
[52]:

	age	campaign	pdays	previous
0	5.0	3.0	9.0	0.0

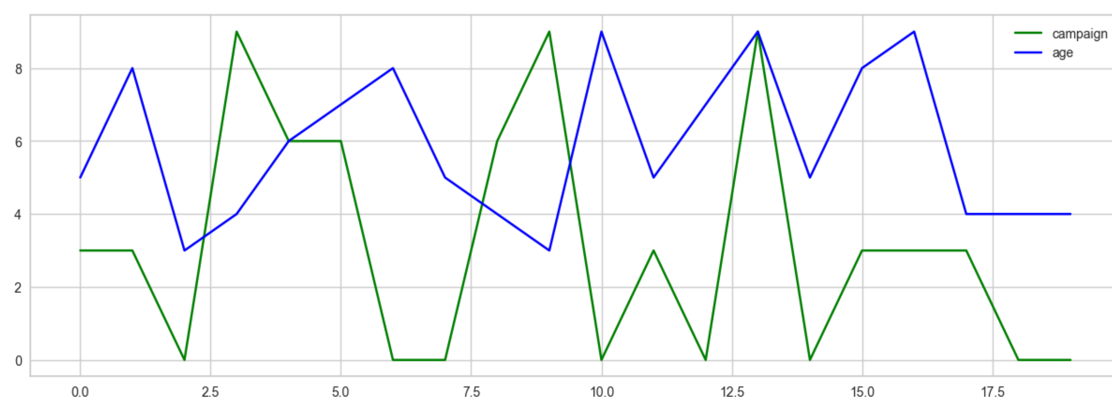
```
1  8.0        3.0     9.0        0.0
2  3.0        0.0     9.0        0.0
3  4.0        9.0     9.0        0.0
4  6.0        6.0     9.0        0.0
```

The graphs of before and after discretization the data are shown below.

[53]:
```python
# Before discretization
plt.figure(figsize=(15, 5));
plt.plot(df_scaled.campaign[:20], color='green', label='campaign')
plt.plot(df_scaled.age[:20], color='blue', label='age')
plt.legend(loc='best')
plt.show()
```



[54]:
```python
# After scaling
plt.figure(figsize=(15, 5));
plt.plot(df_discretization.campaign[:20], color='green', label='campaign')
plt.plot(df_discretization.age[:20], color='blue', label='age')
plt.legend(loc='best')
plt.show()
```



1.3 Perform Feature Engineering

1. Identify significant and independent features using appropriate techniques.
2. Show how you selected the features using suitable graphs.

1.3.1 Identifying Significant and Independent Variables

We can identify the significant features in the dataset by performing feature engineering. We are using the following feature engineering methods on our dataset.

1. PCA(Principal Component Analysis) → PCA is used for dimension reduction in the dataset. It is useful when dealing with dealing with datasets that have many highly correlating variables.
2. SVD(Single Value Decomposition) → SVD is a matrix factorization technique that decomposes a matrix into three simple matrices, ∪, ∨, and $\sum$. This also allows us to reduce dimensionality.

Step 1 - PCA(Principal Component Analysis) PCA is used for dimension reduction in the dataset. It is useful when dealing with dealing with datasets that have many highly correlating variables.

The following code is used to perform PCA.

[55]:
```
# PCA
numerical_variables = ['age', 'campaign', 'pdays', 'previous']
pca_categorical = smoted_X.copy().drop(numerical_variables, axis=1)
df_pca = pd.concat([df_discretization, pca_categorical], join='outer', axis=1)
pca = PCA()

# Apply the transform to dataset
df_pca[df_pca.columns] = pca.fit_transform(df_pca[df_pca.columns])

df_pca.shape
```

[55]:
```
(44168, 58)
```

[56]:
```
df_pca.head()
```

[56]:
```
         age  campaign     pdays  previous  job_admin.  job_blue-collar  \
0  0.449857 -0.895972  0.737189 -0.987929   -1.338503         0.551558
1  0.521929 -0.919739 -2.020773 -0.561270   -1.223406         0.038820
2 -2.526307 -1.528352  2.682445 -1.680693   -0.180347         0.514728
3  6.186656  0.584194  2.105616  0.833874   -0.294076        -0.185213
4  3.345977 -0.147590 -0.075700  0.049416    0.787846        -0.338023

   job_entrepreneur  job_housemaid  job_management  job_retired  …  \
0         -0.191148       0.193159        0.468848    -0.469203  …
```

```
1          1.084545      -0.993970        0.593381      1.172973  …
2         -0.376714       0.573585        0.147982      1.210334  …
3         -0.059947      -0.584749       -0.156442     -0.339331  …
4         -0.616373       0.117620       -0.334118      1.554171  …

   day_of_week_fri  day_of_week_mon  day_of_week_thu  day_of_week_tue  \
0        -0.033134         0.008459         0.001450        -0.000885
1         0.174633         0.005482        -0.017505        -0.003612
2        -0.011521        -0.000498         0.000487        -0.007789
3        -0.010923         0.001711         0.009337         0.005113
4        -0.020775        -0.008455        -0.001219        -0.003901

   day_of_week_wed  poutcome_failure  poutcome_nonexistent  poutcome_success  \
0        -0.010094          0.000905             -0.000325         -0.000426
1        -0.005428         -0.001132             -0.000114          0.001323
2         0.002835          0.000460             -0.000406          0.000213
3         0.001642         -0.000268              0.000599         -0.001510
4        -0.001538         -0.000226              0.000499         -0.000008

   pdays2_no    pdays2_yes
0  -0.000180  3.380492e-16
1   0.000240  2.210104e-16
2   0.000091 -1.011014e-16
3  -0.000149 -5.422392e-17
4  -0.000038 -4.607034e-16

[5 rows x 58 columns]
```

Step 2 - SVD(Single Value Decomposition) SVD is a matrix factorization technique that decomposes a matrix into three simple matrices, ∪, ∨, and $\sum$. This also allows us to reduce dimensionality.

The following code is used to perform SVD. First, we get the optimal number of components and perform the SVD using them.

[57]:
```python
# Copy the data frame
df_svd_temp = df_discretization.copy()

# Make sparse matrix
X_sparse = csr_matrix(df_svd_temp)

tsvd = TruncatedSVD(n_components=X_sparse.shape[1]-1)
X_tsvd = tsvd.fit(df_svd_temp)

tsvd_var_ratios = tsvd.explained_variance_ratio_

def select_n_components(var_ratio, goal_var: float) -> int:
```

```python
    # Set initial variance explained so far
    total_variance = 0.0

    # Set the initial number of features
    n_components = 0

    # For the explained variance of each feature:
    for explained_variance in var_ratio:
        # Add the explained variance to the total
        total_variance += explained_variance
        n_components += 1

        # If we reach our goal level of explained variance
        if total_variance >= goal_var:
            break

    return n_components

number_of_components = select_n_components(tsvd_var_ratios, 0.95)
print("Number of components : ", number_of_components)
```

```
Number of components :  3
```

[58]:
```python
## SVD

pca_categorical = smoted_X.copy().drop(numerical_variables,axis=1)

df_for_svd = pd.concat([df_discretization, pca_categorical],join="outer",axis=1)

svd = TruncatedSVD(n_components = number_of_components)
#svd = TruncatedSVD()
# prepare transform on dataset
svd.fit(df_for_svd)

# apply transform to dataset
transformed = svd.transform(df_for_svd)

df_svd = pd.DataFrame(transformed)
df_svd.head()
```

[58]:

	0	1	2
0	10.880007	0.104965	-1.125842
1	12.417177	-0.559836	1.057044
2	8.951060	-2.279542	-3.002415
3	12.047219	6.034403	-1.410222
4	12.289766	2.703436	-0.069434

Step 3 - Correlation Values Next, we need to check for the correlation values. Because the higher the correlation values are, it has a good impact on the final outcome. The following code shows how you can get the correlation values.

[59]:
```
### Identify significant and independent features
df_for_svd['y'] = smoted_y['y']
df_for_svd.corr()
```

[59]:
```
                                    age  campaign     pdays  previous  \
age                            1.000000  0.018282  0.002866 -0.013805
campaign                       0.018282  1.000000  0.137454 -0.129833
pdays                          0.002866  0.137454  1.000000 -0.664626
previous                      -0.013805 -0.129833 -0.664626  1.000000
job_admin.                    -0.103712  0.009111 -0.005841  0.002634
job_blue-collar                0.028553  0.034726  0.126298 -0.105300
job_entrepreneur               0.039013  0.012447  0.042620 -0.035433
job_housemaid                  0.081735  0.011484  0.013767 -0.018684
job_management                 0.075566 -0.004100  0.035306 -0.016838
job_retired                    0.329973 -0.035568 -0.064446  0.045390
job_self-employed              0.000115  0.018253  0.039275 -0.032088
job_services                  -0.048397  0.031643  0.068897 -0.053095
job_student                   -0.281954 -0.048603 -0.069609  0.085672
job_technician                -0.060928  0.025667  0.050101 -0.046725
job_unemployed                 0.004466  0.006509  0.006059 -0.002203
job_unknown                    0.024425  0.000960  0.006893 -0.010362
marital_divorced               0.157949  0.028019  0.054091 -0.037345
marital_married                0.378531  0.016163  0.055162 -0.062719
marital_single                -0.517358 -0.026603 -0.033275  0.035533
education_basic.4y             0.249442  0.007163  0.023526 -0.037685
education_basic.6y             0.024078  0.023984  0.053719 -0.046453
education_basic.9y            -0.016490  0.021712  0.088977 -0.070113
education_high.school         -0.105947  0.004903  0.071122 -0.052916
education_illiterate           0.013251  0.003684  0.005698 -0.004814
education_professional.course -0.000252  0.019629  0.031079 -0.034946
education_university.degree   -0.079455 -0.008034 -0.047149  0.043902
education_unknown             -0.000639 -0.008002  0.001485  0.010691
default_no                    -0.174750 -0.075864 -0.125922  0.130071
default_unknown                0.171026  0.078432  0.133640 -0.131029
default_yes                    0.000448 -0.002608  0.002326  0.001891
housing_no                     0.007381  0.021562  0.063705 -0.054883
housing_unknown                0.001764 -0.004453  0.002849 -0.003351
housing_yes                   -0.007317 -0.003564 -0.003971  0.000137
loan_no                        0.006398  0.004334  0.008956  0.001668
loan_unknown                   0.001764 -0.004453  0.002849 -0.003351
loan_yes                       0.000104  0.016555  0.055030 -0.054651
contact_cellular              -0.039058 -0.102353 -0.153025  0.198113
contact_telephone              0.049984  0.110434  0.179685 -0.212423
```

month_apr	0.015018	-0.063352	0.015269	-0.001785
month_aug	0.075095	0.066143	0.043322	-0.051809
month_dec	0.016796	-0.008408	-0.033564	0.020139
month_jul	-0.021359	0.106241	0.096343	-0.114423
month_jun	-0.015955	0.060803	0.059338	-0.087273
month_mar	-0.023399	-0.008782	-0.036419	0.039102
month_may	-0.034848	0.032984	0.147153	-0.089476
month_nov	0.036599	-0.058230	0.016559	0.022664
month_oct	0.007104	-0.079204	-0.084864	0.067465
month_sep	-0.007354	-0.044954	-0.128671	0.141055
day_of_week_fri	0.005677	0.048929	0.063228	-0.040799
day_of_week_mon	0.019785	0.072905	0.070347	-0.050177
day_of_week_thu	-0.022789	-0.022342	0.014478	-0.024063
day_of_week_tue	0.018995	-0.011973	0.029271	-0.027225
day_of_week_wed	-0.018853	-0.016555	0.043880	-0.037440
poutcome_failure	-0.006925	-0.069310	0.072635	0.427947
poutcome_nonexistent	0.008723	0.150077	0.655711	-0.771285
poutcome_success	0.001671	-0.128088	-0.950980	0.598489
pdays2_no	0.002972	0.136317	0.984764	-0.667563
pdays2_yes	-0.002355	-0.136974	-0.985990	0.668453
y	-0.044847	-0.193574	-0.296144	0.212807

	job_admin.	job_blue-collar	job_entrepreneur \
age	-0.103712	0.028553	0.039013
campaign	0.009111	0.034726	0.012447
pdays	-0.005841	0.126298	0.042620
previous	0.002634	-0.105300	-0.035433
job_admin.	1.000000	-0.259210	-0.087888
job_blue-collar	-0.259210	1.000000	-0.070490
job_entrepreneur	-0.087888	-0.070490	1.000000
job_housemaid	-0.077311	-0.062007	-0.021024
job_management	-0.144144	-0.115610	-0.039199
job_retired	-0.137679	-0.110424	-0.037441
job_self-employed	-0.088628	-0.071083	-0.024102
job_services	-0.159338	-0.127796	-0.043331
job_student	-0.096860	-0.077686	-0.026340
job_technician	-0.228001	-0.182867	-0.062003
job_unemployed	-0.072299	-0.057987	-0.019661
job_unknown	-0.031711	-0.025434	-0.008624
marital_divorced	0.003251	-0.034946	0.007149
marital_married	-0.107755	0.134549	0.064683
marital_single	0.113495	-0.109841	-0.061107
education_basic.4y	-0.166003	0.230587	-0.006527
education_basic.6y	-0.088732	0.249537	0.001941
education_basic.9y	-0.146629	0.375552	0.013857
education_high.school	0.082944	-0.131254	-0.022848
education_illiterate	-0.009372	0.014319	0.006550

education_professional.course	-0.153909	-0.102966	-0.011314
education_university.degree	0.329074	-0.299575	0.037885
education_unknown	-0.040655	0.028382	0.016766
default_no	0.121590	-0.215331	-0.020232
default_unknown	-0.114577	0.216750	0.024653
default_yes	-0.003826	-0.003068	-0.001040
housing_no	-0.021660	0.038180	-0.002608
housing_unknown	-0.004685	-0.003758	-0.001274
housing_yes	0.019175	-0.023164	0.008712
loan_no	-0.018864	0.004601	0.009151
loan_unknown	-0.004685	-0.003758	-0.001274
loan_yes	0.018967	0.014240	0.002670
contact_cellular	0.057056	-0.133823	-0.041411
contact_telephone	-0.059473	0.146155	0.048182
month_apr	-0.001777	-0.028268	0.011945
month_aug	0.069004	-0.096053	-0.029703
month_dec	0.006478	-0.020719	-0.006900
month_jul	-0.009618	0.062337	0.011836
month_jun	-0.007733	0.054014	0.013964
month_mar	0.011472	-0.041558	-0.020250
month_may	-0.059102	0.175110	0.028359
month_nov	-0.001833	-0.036050	0.059299
month_oct	-0.000443	-0.051657	-0.013450
month_sep	0.006217	-0.052034	-0.012588
day_of_week_fri	-0.006332	0.021924	0.007009
day_of_week_mon	-0.001273	-0.011868	0.023746
day_of_week_thu	0.011863	0.010076	0.005965
day_of_week_tue	0.000927	0.001748	0.004453
day_of_week_wed	-0.003354	0.035019	-0.001777
poutcome_failure	-0.010025	-0.016639	-0.008200
poutcome_nonexistent	0.000257	0.104564	0.039696
poutcome_success	0.009422	-0.120423	-0.039839
pdays2_no	-0.004319	0.128763	0.043230
pdays2_yes	0.004777	-0.128528	-0.043150
y	0.008537	-0.176577	-0.092819

	job_housemaid	job_management	job_retired \
age	0.081735	0.075566	0.329973
campaign	0.011484	-0.004100	-0.035568
pdays	0.013767	0.035306	-0.064446
previous	-0.018684	-0.016838	0.045390
job_admin.	-0.077311	-0.144144	-0.137679
job_blue-collar	-0.062007	-0.115610	-0.110424
job_entrepreneur	-0.021024	-0.039199	-0.037441
job_housemaid	1.000000	-0.034481	-0.032935
job_management	-0.034481	1.000000	-0.061406
job_retired	-0.032935	-0.061406	1.000000

job_self-employed	-0.021201	-0.039528	-0.037756
job_services	-0.038116	-0.071066	-0.067879
job_student	-0.023170	-0.043200	-0.041263
job_technician	-0.054541	-0.101690	-0.097129
job_unemployed	-0.017295	-0.032246	-0.030800
job_unknown	-0.007586	-0.014143	-0.013509
marital_divorced	0.025352	0.003927	0.062768
marital_married	0.046498	0.060999	0.078603
marital_single	-0.054540	-0.064722	-0.141632
education_basic.4y	0.192722	-0.056082	0.225688
education_basic.6y	0.014459	-0.023840	-0.029418
education_basic.9y	-0.014645	-0.053130	-0.050818
education_high.school	-0.018705	-0.080814	-0.059813
education_illiterate	0.008047	-0.004180	0.008018
education_professional.course	-0.022017	-0.068133	-0.008434
education_university.degree	-0.054890	0.257706	-0.098639
education_unknown	-0.022026	-0.041066	0.008435
default_no	-0.047185	0.023917	0.026390
default_unknown	0.046023	-0.025675	-0.023430
default_yes	-0.000915	-0.001706	-0.001630
housing_no	0.017038	0.012781	-0.009064
housing_unknown	-0.001121	-0.002090	-0.001996
housing_yes	-0.009138	-0.012930	-0.001800
loan_no	0.007651	0.009381	0.015471
loan_unknown	-0.001121	-0.002090	-0.001996
loan_yes	0.002679	0.002356	-0.025887
contact_cellular	-0.031695	0.001783	0.058555
contact_telephone	0.037343	0.002870	-0.063839
month_apr	0.006132	-0.002849	0.034912
month_aug	0.014437	-0.015338	0.008762
month_dec	0.002969	-0.007524	0.034422
month_jul	0.028279	-0.019475	-0.030105
month_jun	-0.001505	-0.000995	-0.038630
month_mar	-0.009912	-0.000002	0.024383
month_may	0.004743	-0.005837	-0.089969
month_nov	-0.002572	0.088438	-0.020058
month_oct	-0.004740	-0.004092	0.048301
month_sep	-0.006576	0.001643	0.046242
day_of_week_fri	0.003234	-0.004171	-0.018205
day_of_week_mon	0.010183	0.021513	-0.026880
day_of_week_thu	-0.008088	0.001321	-0.040615
day_of_week_tue	0.014374	0.003563	0.009861
day_of_week_wed	0.006264	-0.004346	0.024586
poutcome_failure	-0.011940	0.008491	0.009268
poutcome_nonexistent	0.019898	0.018475	-0.056827
poutcome_success	-0.010615	-0.033743	0.066394
pdays2_no	0.012466	0.039432	-0.064472

```
pdays2_yes                           -0.012370        -0.039269     0.064730
y                                    -0.071218        -0.052079     0.086599

                              …  day_of_week_mon  day_of_week_thu  \
age                           …         0.019785        -0.022789
campaign                      …         0.072905        -0.022342
pdays                         …         0.070347         0.014478
previous                      …        -0.050177        -0.024063
job_admin.                    …        -0.001273         0.011863
job_blue-collar               …        -0.011868         0.010076
job_entrepreneur              …         0.023746         0.005965
job_housemaid                 …         0.010183        -0.008088
job_management                …         0.021513         0.001321
job_retired                   …        -0.026880        -0.040615
job_self-employed             …         0.005511         0.021997
job_services                  …         0.023098        -0.001478
job_student                   …        -0.020341         0.004191
job_technician                …         0.009925         0.004331
job_unemployed                …         0.011909         0.011080
job_unknown                   …         0.007752        -0.000203
marital_divorced              …         0.016278        -0.011897
marital_married               …         0.016604         0.000525
marital_single                …        -0.021263         0.013949
education_basic.4y            …        -0.005271        -0.012888
education_basic.6y            …        -0.000417         0.000421
education_basic.9y            …         0.002214         0.014375
education_high.school         …         0.016066        -0.008452
education_illiterate          …        -0.007557         0.002813
education_professional.course …        -0.006880         0.003149
education_university.degree   …         0.012329         0.007160
education_unknown             …         0.017031        -0.000967
default_no                    …        -0.018911        -0.000153
default_unknown               …         0.022227         0.004914
default_yes                   …        -0.003085        -0.003194
housing_no                    …        -0.003928         0.001621
housing_unknown               …         0.003475        -0.003912
housing_yes                   …         0.012230        -0.002722
loan_no                       …        -0.002798        -0.011857
loan_unknown                  …         0.003475        -0.003912
loan_yes                      …         0.013729         0.014296
contact_cellular              …        -0.010739         0.030317
contact_telephone             …         0.016299        -0.025940
month_apr                     …         0.008088         0.101280
month_aug                     …        -0.006906        -0.006473
month_dec                     …         0.019648         0.009871
month_jul                     …         0.017607         0.019349
month_jun                     …         0.060759        -0.031647
```

month_mar	…	-0.002022	-0.006353
month_may	…	0.010984	-0.031526
month_nov	…	-0.014150	0.007595
month_oct	…	-0.024363	-0.009386
month_sep	…	-0.025732	-0.001570
day_of_week_fri	…	-0.200680	-0.207816
day_of_week_mon	…	1.000000	-0.217586
day_of_week_thu	…	-0.217586	1.000000
day_of_week_tue	…	-0.206458	-0.213799
day_of_week_wed	…	-0.209020	-0.216453
poutcome_failure	…	-0.002393	-0.006868
poutcome_nonexistent	…	0.054007	0.019729
poutcome_success	…	-0.067602	-0.022551
pdays2_no	…	0.071197	0.020374
pdays2_yes	…	-0.070902	-0.020015
y	…	-0.104420	-0.056457

	day_of_week_tue	day_of_week_wed \
age	0.018995	-0.018853
campaign	-0.011973	-0.016555
pdays	0.029271	0.043880
previous	-0.027225	-0.037440
job_admin.	0.000927	-0.003354
job_blue-collar	0.001748	0.035019
job_entrepreneur	0.004453	-0.001777
job_housemaid	0.014374	0.006264
job_management	0.003563	-0.004346
job_retired	0.009861	0.024586
job_self-employed	-0.002680	-0.000158
job_services	0.012902	-0.009409
job_student	-0.007355	-0.013216
job_technician	-0.008601	-0.004598
job_unemployed	0.017121	-0.009111
job_unknown	0.009672	0.004764
marital_divorced	0.012223	0.001115
marital_married	0.012041	0.018378
marital_single	-0.011782	-0.015347
education_basic.4y	0.019191	0.016193
education_basic.6y	0.001993	0.022678
education_basic.9y	0.002073	0.015893
education_high.school	-0.001246	0.007290
education_illiterate	0.007250	-0.000239
education_professional.course	0.009210	-0.005152
education_university.degree	-0.012445	-0.010433
education_unknown	0.001759	-0.011531
default_no	-0.017069	0.010273
default_unknown	0.017516	-0.001990

default_yes	0.014941	-0.003069
housing_no	-0.003423	0.013441
housing_unknown	0.003625	-0.003758
housing_yes	0.007938	-0.009179
loan_no	0.022154	0.007854
loan_unknown	0.003625	-0.003758
loan_yes	-0.009058	-0.003372
contact_cellular	-0.012548	-0.031628
contact_telephone	0.011550	0.030670
month_apr	-0.068982	-0.040540
month_aug	0.023301	0.007278
month_dec	-0.006323	-0.002864
month_jul	0.023084	0.021863
month_jun	-0.007313	-0.012973
month_mar	0.018948	-0.035975
month_may	0.026299	0.032484
month_nov	0.007504	0.012195
month_oct	-0.007581	-0.012554
month_sep	-0.001238	0.006182
day_of_week_fri	-0.197187	-0.199634
day_of_week_mon	-0.206458	-0.209020
day_of_week_thu	-0.213799	-0.216453
day_of_week_tue	1.000000	-0.205382
day_of_week_wed	-0.205382	1.000000
poutcome_failure	-0.000213	-0.020758
poutcome_nonexistent	0.021678	0.048470
poutcome_success	-0.025977	-0.040461
pdays2_no	0.032915	0.045102
pdays2_yes	-0.032588	-0.045936
y	-0.068265	-0.057757

	poutcome_failure	poutcome_nonexistent \
age	-0.006925	0.008723
campaign	-0.069310	0.150077
pdays	0.072635	0.655711
previous	0.427947	-0.771285
job_admin.	-0.010025	0.000257
job_blue-collar	-0.016639	0.104564
job_entrepreneur	-0.008200	0.039696
job_housemaid	-0.011940	0.019898
job_management	0.008491	0.018475
job_retired	0.009268	-0.056827
job_self-employed	-0.007551	0.038189
job_services	0.000146	0.054825
job_student	0.028267	-0.075921
job_technician	-0.011910	0.047004
job_unemployed	0.002245	0.001262

job_unknown	-0.008558	0.013613
marital_divorced	-0.000067	0.043358
marital_married	-0.023021	0.055705
marital_single	0.013884	-0.030832
education_basic.4y	-0.024488	0.032083
education_basic.6y	-0.009160	0.047834
education_basic.9y	-0.005516	0.068107
education_high.school	0.006514	0.049970
education_illiterate	-0.001191	0.005360
education_professional.course	-0.011678	0.031658
education_university.degree	-0.013580	-0.019581
education_unknown	0.019545	-0.013730
default_no	0.070470	-0.149747
default_unknown	-0.069324	0.155242
default_yes	0.008640	-0.004609
housing_no	-0.022840	0.065234
housing_unknown	-0.002831	0.004345
housing_yes	0.009875	-0.009973
loan_no	0.021141	-0.007251
loan_unknown	-0.002831	0.004345
loan_yes	-0.023590	0.057351
contact_cellular	0.172839	-0.246755
contact_telephone	-0.173594	0.267342
month_apr	0.067112	-0.038986
month_aug	-0.060594	0.078830
month_dec	0.004760	-0.031568
month_jul	-0.107733	0.148362
month_jun	-0.082113	0.107614
month_mar	0.014525	-0.038353
month_may	0.040615	0.078743
month_nov	0.076338	-0.034887
month_oct	0.039369	-0.091698
month_sep	0.028837	-0.114774
day_of_week_fri	0.012501	0.034341
day_of_week_mon	-0.002393	0.054007
day_of_week_thu	-0.006868	0.019729
day_of_week_tue	-0.000213	0.021678
day_of_week_wed	-0.020758	0.048470
poutcome_failure	1.000000	-0.651568
poutcome_nonexistent	-0.651568	1.000000
poutcome_success	-0.114122	-0.630264
pdays2_no	0.065623	0.665856
pdays2_yes	-0.065414	-0.665007
y	0.009948	-0.245386

	poutcome_success	pdays2_no	pdays2_yes	\
age	0.001671	0.002972	-0.002355	

campaign	-0.128088	0.136317	-0.136974
pdays	-0.950980	0.984764	-0.985990
previous	0.598489	-0.667563	0.668453
job_admin.	0.009422	-0.004319	0.004777
job_blue-collar	-0.120423	0.128763	-0.128528
job_entrepreneur	-0.039839	0.043230	-0.043150
job_housemaid	-0.010615	0.012466	-0.012370
job_management	-0.033743	0.039432	-0.039269
job_retired	0.066394	-0.064472	0.064730
job_self-employed	-0.038871	0.040918	-0.040834
job_services	-0.066335	0.071061	-0.070908
job_student	0.063124	-0.073213	0.073422
job_technician	-0.044991	0.049559	-0.051387
job_unemployed	-0.004165	0.006435	-0.006339
job_unknown	-0.007647	0.007854	-0.007817
marital_divorced	-0.054927	0.056821	-0.056635
marital_married	-0.048023	0.053705	-0.054168
marital_single	0.027241	-0.029983	0.030533
education_basic.4y	-0.020739	0.024571	-0.024350
education_basic.6y	-0.051829	0.055251	-0.055143
education_basic.9y	-0.084680	0.090540	-0.090350
education_high.school	-0.069873	0.068542	-0.068206
education_illiterate	-0.005477	0.005787	-0.005779
education_professional.course	-0.030036	0.027840	-0.030185
education_university.degree	0.043249	-0.037011	0.037587
education_unknown	-0.001074	0.001765	-0.001638
default_no	0.121871	-0.129018	0.128787
default_unknown	-0.128747	0.136010	-0.135812
default_yes	-0.002236	0.002362	-0.002359
housing_no	-0.066163	0.064827	-0.065656
housing_unknown	-0.002738	0.002893	-0.002889
housing_yes	0.007468	0.000392	0.000438
loan_no	-0.003679	0.013043	-0.013411
loan_unknown	-0.002738	0.002893	-0.002889
loan_yes	-0.053073	0.054534	-0.054282
contact_cellular	0.144187	-0.156422	0.156222
contact_telephone	-0.171027	0.182975	-0.182675
month_apr	-0.016210	0.022806	-0.022611
month_aug	-0.043971	0.041562	-0.041303
month_dec	0.038670	-0.035030	0.035114
month_jul	-0.085954	0.091950	-0.092361
month_jun	-0.056919	0.063769	-0.063556
month_mar	0.041093	-0.037288	0.037429
month_may	-0.140877	0.150354	-0.150047
month_nov	-0.027486	0.020037	-0.019832
month_oct	0.078122	-0.087814	0.087536
month_sep	0.128550	-0.125224	0.125451

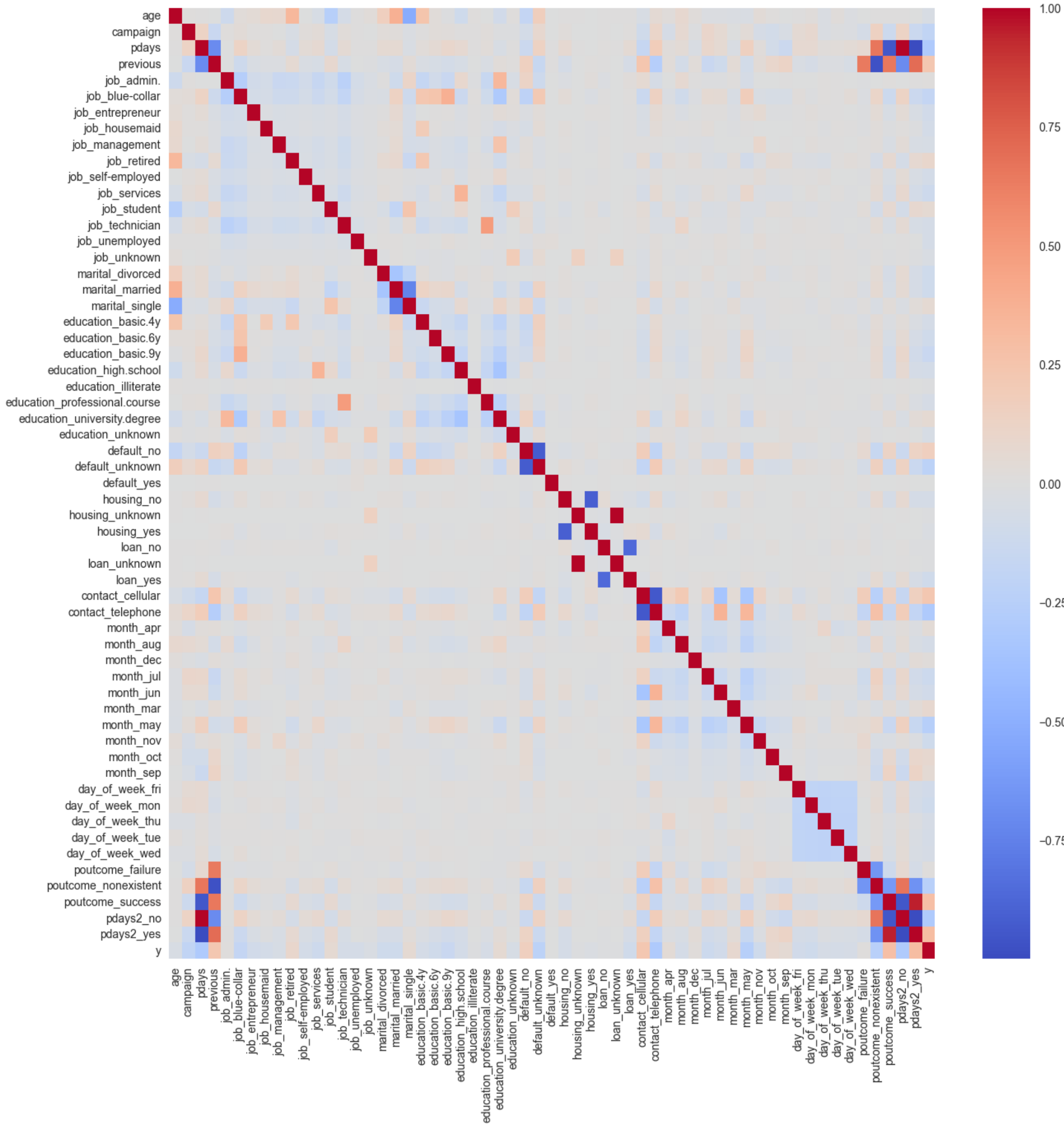
day_of_week_fri	-0.057926	0.063552	-0.063464
day_of_week_mon	-0.067602	0.071197	-0.070902
day_of_week_thu	-0.022551	0.020374	-0.020015
day_of_week_tue	-0.025977	0.032915	-0.032588
day_of_week_wed	-0.040461	0.045102	-0.045936
poutcome_failure	-0.114122	0.065623	-0.065414
poutcome_nonexistent	-0.630264	0.665856	-0.665007
poutcome_success	1.000000	-0.946547	0.947756
pdays2_no	-0.946547	1.000000	-0.998724
pdays2_yes	0.947756	-0.998724	1.000000
y	0.286245	-0.300444	0.299946

	y
age	-0.044847
campaign	-0.193574
pdays	-0.296144
previous	0.212807
job_admin.	0.008537
job_blue-collar	-0.176577
job_entrepreneur	-0.092819
job_housemaid	-0.071218
job_management	-0.052079
job_retired	0.086599
job_self-employed	-0.077354
job_services	-0.113349
job_student	0.060440
job_technician	-0.067499
job_unemployed	-0.067116
job_unknown	-0.034608
marital_divorced	-0.107731
marital_married	-0.104316
marital_single	0.060282
education_basic.4y	-0.063369
education_basic.6y	-0.100036
education_basic.9y	-0.142131
education_high.school	-0.076922
education_illiterate	-0.013738
education_professional.course	-0.060399
education_university.degree	0.057803
education_unknown	-0.053796
default_no	0.157741
default_unknown	-0.222939
default_yes	-0.006729
housing_no	-0.067057
housing_unknown	-0.008242
housing_yes	-0.025242
loan_no	-0.013158

```
loan_unknown                -0.008242
loan_yes                    -0.097087
contact_cellular             0.231169
contact_telephone           -0.297994
month_apr                    0.047074
month_aug                   -0.063012
month_dec                    0.018726
month_jul                   -0.116731
month_jun                   -0.088072
month_mar                    0.073316
month_may                   -0.250956
month_nov                   -0.106967
month_oct                    0.075188
month_sep                    0.053505
day_of_week_fri             -0.087264
day_of_week_mon             -0.104420
day_of_week_thu             -0.056457
day_of_week_tue             -0.068265
day_of_week_wed             -0.057757
poutcome_failure             0.009948
poutcome_nonexistent        -0.245386
poutcome_success             0.286245
pdays2_no                   -0.300444
pdays2_yes                   0.299946
y                            1.000000

[59 rows x 59 columns]
```

To identify the correlation values better we can get a heatmap.

[60]:
```python
def draw_heatmap(df):
    f, ax = plt.subplots(figsize=(15, 15))
    sns.heatmap(df.corr(method='spearman'), annot=False, cmap='coolwarm')

draw_heatmap(df_for_svd)
```



Since all the features in the dataset have low correlation values with each other, we can determine all of the features are independent variables and they can be used for the classification algorithms. We can find the significant variables using the correlation matrix as well. The following code explains how you can find the significant variables.

[61]:
```
# Select most significant variables in descending order
correlation_matrix = df_for_svd.corr()

# Get them in descending order
correlation_with_price = correlation_matrix['y'].abs().
 ↪sort_values(ascending=False)
```

```python
# Print the values
print(correlation_with_price)
```

```
y                                1.000000
pdays2_no                        0.300444
pdays2_yes                       0.299946
contact_telephone                0.297994
pdays                            0.296144
poutcome_success                 0.286245
month_may                        0.250956
poutcome_nonexistent             0.245386
contact_cellular                 0.231169
default_unknown                  0.222939
previous                         0.212807
campaign                         0.193574
job_blue-collar                  0.176577
default_no                       0.157741
education_basic.9y               0.142131
month_jul                        0.116731
job_services                     0.113349
marital_divorced                 0.107731
month_nov                        0.106967
day_of_week_mon                  0.104420
marital_married                  0.104316
education_basic.6y               0.100036
loan_yes                         0.097087
job_entrepreneur                 0.092819
month_jun                        0.088072
day_of_week_fri                  0.087264
job_retired                      0.086599
job_self-employed                0.077354
education_high.school            0.076922
month_oct                        0.075188
month_mar                        0.073316
job_housemaid                    0.071218
day_of_week_tue                  0.068265
job_technician                   0.067499
job_unemployed                   0.067116
housing_no                       0.067057
education_basic.4y               0.063369
month_aug                        0.063012
job_student                      0.060440
education_professional.course    0.060399
marital_single                   0.060282
education_university.degree      0.057803
day_of_week_wed                  0.057757
day_of_week_thu                  0.056457
```

```
education_unknown                0.053796
month_sep                        0.053505
job_management                   0.052079
month_apr                        0.047074
age                              0.044847
job_unknown                      0.034608
housing_yes                      0.025242
month_dec                        0.018726
education_illiterate             0.013738
loan_no                          0.013158
poutcome_failure                 0.009948
job_admin.                       0.008537
loan_unknown                     0.008242
housing_unknown                  0.008242
default_yes                      0.006729
Name: y, dtype: float64
```

1.4 Classification

1. Justifies the choice of classification algorithm for the dataset.
2. Consider and apply alternative algorithms to dataset and explain why they were chosen.
3. Using suitable evaluation matrices, compare the applicability of different classification algorithms on the given dataset.
4. Relate classification results to the original problem and provide actionanle insights.

1.4.1 Chosen Classification Algorithm for the Dataset

The Classification algorithm that was chosen for this is the `Support Vector Machines(SVM)` algorithm. The reasons for choosing that algorithm for this dataset are mentioned below. - SVMs perform well in high-dimensional spaces, making them suitable for problems with many features, such as text classification or image recognition. - SVMs have a regularization parameter (C), which helps prevent overfitting. This parameter balances the trade-off between maximizing the margin and minimizing the classification error. - SVMs can use different kernel functions to transform the input space into higher-dimensional space. This ability allows SVMs to handle nonlinear relationships in the data. - SVMs can work well with small to medium-sized datasets. They are not as affected by the curse of dimensionality as some other algorithms. - SVMs can handle imbalanced datasets well by adjusting the class_weight parameter or using techniques like Synthetic Minority Over-sampling Technique (SMOTE).

Step 1 - Split Train and Test Set The following code explains how to divide our dataset into training and testing.

[62]:
```
processed_data = df_pca.copy()

X = processed_data

X_train, X_test, y_train, y_test = train_test_split(X, data_y, test_size=0.20,
    random_state=42)
```

```
print("Train dataset:", X_train.shape)
print("Test dataset:", X_test.shape)
```

```
Train dataset: (35334, 58)
Test dataset: (8834, 58)
```

Step 2 - Support Vector Machines (SVM)

[63]:
```
#Import svm model
from sklearn import svm
from sklearn import metrics

# Create a svm Classifier with Linear Kernel
classifier_svm = svm.SVC(kernel='linear')

# Train the model using the training sets
classifier_svm.fit(X_train, y_train)
```

[63]:
```
SVC(kernel='linear')
```

[64]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import␣
↪accuracy_score,precision_score,recall_score,confusion_matrix
import math

# Predict the response for test dataset
y_pred = classifier_svm.predict(X_test)

# evaluation
print("Classifier score : ", classifier_svm.score(X_test, y_test))
print("Confusion matrix : ", confusion_matrix(y_test,classifier_svm.
↪predict(X_test)))

# accuracy testing
accuracy_svm = metrics.accuracy_score(y_test, y_pred)
print("Accuracy score : ", accuracy_svm)
precision_svm = metrics.precision_score(y_test, y_pred)
print("Precision score : ", precision_svm)
recall_svm = metrics.recall_score(y_test, y_pred)
print("recall_score : ", recall_svm)
f1_svm = 2 * precision_svm * recall_svm / (precision_svm + recall_svm)
print("f1 score : ", f1_svm)
```

```
Classifier score :  0.9216662893366538

Confusion matrix :  [[4389   71]
 [ 621 3753]]
```

```
Accuracy score :  0.9216662893366538
Precision score :  0.9814330543933054
recall_score :  0.8580246913580247
f1 score :  0.9155891680897781
```

1.4.2 Alternative Classification Algorithms

Rather than using `Support Vector Machines` we can use other classification algorithms as well. The alternative classification algorithm chosen for this dataset is the `Logistic Regression` algorithm. The reasons for selecting `Logistic Regression` algorithm are mentioned below. - Logistic Regression is straightforward and easy to implement. It provides a clear interpretation of the coefficients, which can help understand the impact of each feature on the predicted outcome. - It performs well with small to medium-sized datasets, making it suitable for problems with limited data. - Logistic Regression models the probability of the outcome using a linear function. This results in a linear decision boundary, making it particularly useful when the relationship between features and the target variable is approximately linear. - Logistic Regression does not assume a specific distribution of the input variables, unlike some other algorithms such as Naive Bayes.

[65]:
```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

classifier_lr = LogisticRegression(random_state=0)

classifier_lr.fit(X_train,y_train)
```

[65]:
```
LogisticRegression(random_state=0)
```

[66]:
```
# Evaluation
print("Classifier score : ", classifier_lr.score(X_test,y_test))
print("Confusion matrix : ", confusion_matrix(y_test,classifier_lr.
 ↪predict(X_test)))

# accuracy testing
accuracy_lr = accuracy_score(y_test,classifier_lr.predict(X_test))
print("Accuracy score : ", accuracy_lr)
precision_lr = precision_score(y_test,classifier_lr.predict(X_test))
print("Precision score : ", precision_lr)
recall_lr = recall_score(y_test,classifier_lr.predict(X_test))
print("recall_score : ", recall_lr)
f1_lr = 2 * precision_lr * recall_lr / (precision_lr + recall_lr)
print("f1 score : ", f1_lr)
```

```
Classifier score :  0.9214398913289563
Confusion matrix :  [[4375   85]
 [ 609 3765]]
Accuracy score :  0.9214398913289563

Precision score :  0.977922077922078
```

```
recall_score :  0.8607681755829903
f1 score :  0.915612840466926
```

1.4.3 Comparing Classification Algorithms

As shown above, both algorithms can be used for supervised learning. Also, both models have been evaluated based on their accuracy, precision score, recall score, and F1 score.

Receiver Operating Characteristic(ROC) Curve is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: - True Positive Rate - False Positive Rate

The True Positive Rate(TPR) and False Positive Rate(FPR) are calculated using the following equations.

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

where,

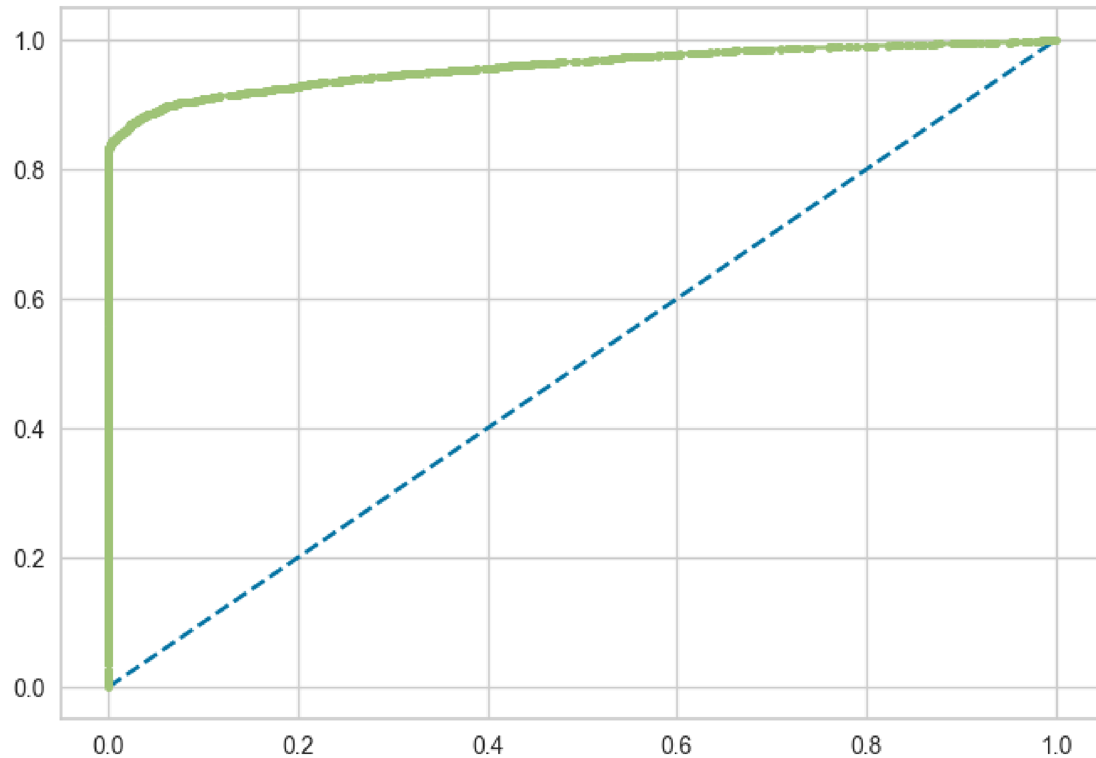
- TP → True Positive
- FP → False Positive
- FP → True Negative
- FP → False Negative

The higher the AUC(Area Under the Curve) value the model is better.

The below code is for the ROC curve of the Logistic Regression Model.

[67]:
```
# ROC curve forLR
probs = classifier_lr.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

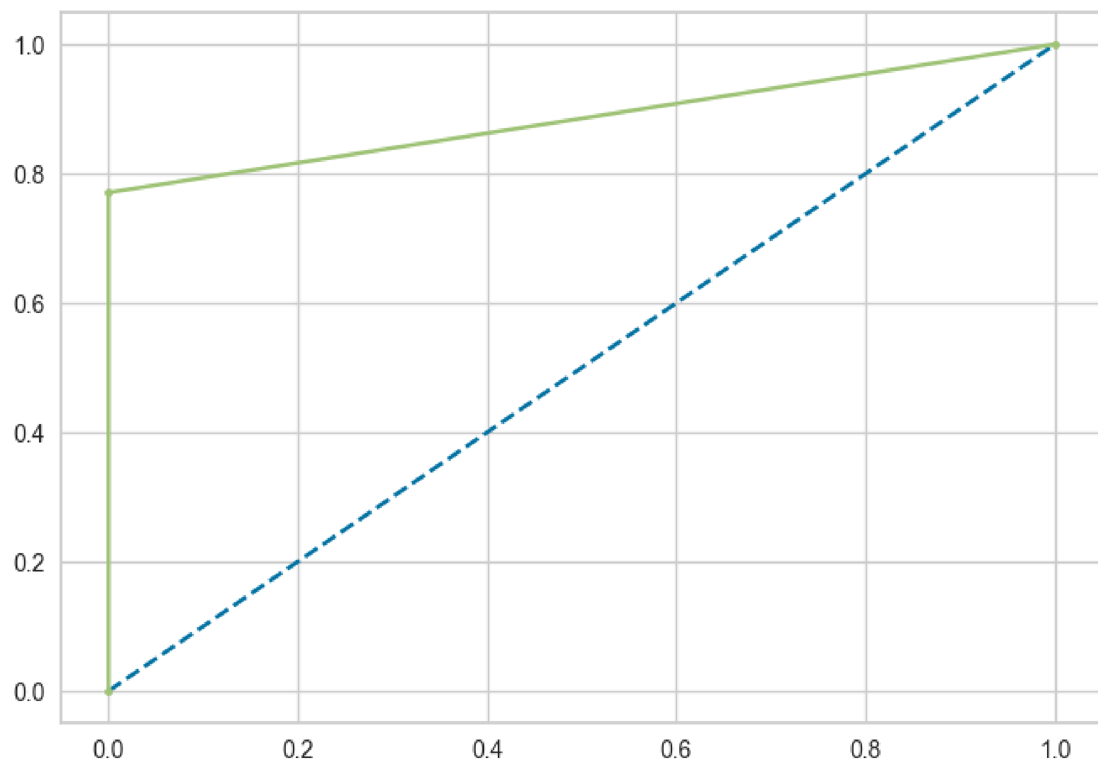
```
AUC: 0.958
```



As shown above, 95.8% is the ROC for the Logistic Regression Model. The following code is used to calculate the ROC curve for Support Vector Machines.

[68]:
```
# ROC Curve For SVM
probs = classifier_svm.predict(X_test)
# keep probabilities for the positive outcome only

# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

```
AUC: 0.885
```



[69]:
```
def train_and_evaluate(clf, X_train, X_test, y_train, y_test):
    clf.fit(X_train, y_train)
    print ("Accuracy on training set:")
    print (clf.score(X_train, y_train))
    print ("Accuracy on testing set:")
    print (clf.score(X_test, y_test))
    y_pred = clf.predict(X_test)
    print ("Classification Report:")
    print (metrics.classification_report(y_test, y_pred))
    print ("Confusion Matrix:")
    print (metrics.confusion_matrix(y_test, y_pred))
```

As shown above the Support Vector Machines have 88.5% in the ROC.

Classification Report The classification report has the precision, recall, F1 score and support scores mentioned there. This helps to evaluate the model based on several matrices.

The following code is for the classification report of Logistic Regression.

[70]:
```
train_and_evaluate(classifier_lr, X_train, X_test, y_train, y_test)
```

```
Accuracy on training set:
0.9246051961283749
```

```
Accuracy on testing set:
0.9214398913289563

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.98      0.93      4460
           1       0.98      0.86      0.92      4374

    accuracy                           0.92      8834
   macro avg       0.93      0.92      0.92      8834
weighted avg       0.93      0.92      0.92      8834

Confusion Matrix:
[[4375   85]
 [ 609 3765]]
```

The following code is for the classification report of Support Vector Machines.

[71]:
```
train_and_evaluate(classifier_svm, X_train, X_test, y_train, y_test)
```

```
Accuracy on training set:

0.9240391690722817
Accuracy on testing set:

0.9220058863482001

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.98      0.93      4460
           1       0.98      0.86      0.92      4374

    accuracy                           0.92      8834
   macro avg       0.93      0.92      0.92      8834
weighted avg       0.93      0.92      0.92      8834

Confusion Matrix:
[[4392   68]
 [ 621 3753]]
```

[72]:
```
table = PrettyTable()

table.field_names = ["MODEL", "Accuracy score","Precision score","Recall␣
↪score","F1 score"]
table.add_row(["Logistic Regression)", accuracy_lr.round(4), precision_lr.
↪round(3), recall_lr.round(3), f1_lr.round(3)])
table.add_row(["SVM classifier",accuracy_svm.round(4), precision_svm.
↪round(3),recall_svm.round(3),f1_svm.round(3)])
```

```python
print('Bank Marketing')
print(table)
```

```
Bank Marketing
+----------------------+----------------+-----------------+--------------+------
----+
|        MODEL         | Accuracy score | Precision score | Recall score | F1
score |
+----------------------+----------------+-----------------+--------------+------
----+
| Logistic Regression) |     0.9214     |      0.978      |    0.861     |
0.916   |
|    SVM classifier    |     0.9217     |      0.981      |    0.858     |
0.916   |
+----------------------+----------------+-----------------+--------------+------
----+
```

From the aforementioned values, ROC curve and classification report, we can see that the accuracies of both models are similar, but the ROC curve of Logistic Regression is better than Support Vector Machine. Therefore, we can deduce that Logistic Regression performs better thann Support Vector Machines for this dataset.

1.5 Insights

From the above classifiers and data analysis techniques, we can deduce some insights for the dataset.

- The contacts done by the bank are low and only have contacted literate clients.
- The customers with high-ranking jobs (e.g. Management, Admin) have a university degree.
- The bank has contacted people who have taken housing loans and not personal loans.
- The bank hasn't targeted possible clients in their 20s and over 60s. This oversight can be improved if the bank puts a considerable amount of effort into it.
- Although the duration was not considered, it can be seen that the higher the duration, the more clients will go for a deposit.
- Clients were contacted during the mid-months and have not been contacted considerably in December, this also can be improved.
- The bank should focus on clients whom they have previously reached out to during previous campaigns as well.

2 References

- www.ibm.com. (2023). What is support vector machine? | IBM. [online] Available at: https://www.ibm.com/topics/support-vector-machine.
- scikit learn (2018). 1.4. Support Vector Machines — scikit-learn 0.20.3 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/svm.html.
- An Idiot's guide to Support vector machines (SVMs) R. Berwick, Village Idiot SVMs: A New Generation of Learning Algorithms. (n.d.). Available at: https://web.mit.edu/6.034/wwwbob/svm.pdf.

- IBM (2022). What Is Logistic Regression? [online] IBM. Available at: https://www.ibm.com/topics/logistic-regression.
- Google Developers. (n.d.). Classification: ROC Curve and AUC | Machine Learning Crash Course. [online] Available at: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

[ ]:

```

```