Portugal Bank Marketing Campaign

The dataset for this project originates from the [UCI Portugal bank marketing campaigns Repository](#).

Abstract: Predicting whether client will agree to place deposit

In this project, we will evaluate the performance and predictive power of a model that has been trained and tested on data collected from customers of portugal bank. A model trained on this data that is seen as a good fit could then be used to predict if the client will subscribe a term deposit or not.

Attribute Information

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Input variables:

1) **age:** Age of the customer (numeric)

2) **job:** type of job(categorical:"admin.","bluecollar","entrepreneur","housemaid","management","retired","self-employed","services","student","technician","unemployed","unknown")

3) **marital:** marital status (categorical: "divorced","married","single","unknown"; note: "divorced" means divorced or widowed)

4) **education:** education of individual (categorical: "basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree","unk

5) **default:** has credit in default? (categorical: "no","yes","unknown")

6) **housing:** has housing loan? (categorical: "no","yes","unknown")

7) **loan:** has personal loan? (categorical: "no","yes","unknown")

8) **contact:** contact communication type (categorical: "cellular","telephone")

9) **month:** last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")

10) **dayofweek:** last contact day of the week (categorical: "mon","tue","wed","thu","fri")

11) **duration:** last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y="no"). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

12) **campaign:** number of contacts performed during this campaign and for this client (numeric, includes last contact)

13) **pdays:** number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14) **previous:** number of contacts performed before this campaign and for this client (numeric)

15) **poutcome:** outcome of the previous marketing campaign (categorical: "failure","nonexistent","success")

16) **emp.var.rate:** employment variation rate - quarterly indicator (numeric)

17) **cons.price.idx:** consumer price index - monthly indicator (numeric)

18) **cons.conf.idx:** consumer confidence index - monthly indicator (numeric)

19) **concave points_se:** standard error for number of concave portions of the contour

20) **euribor3m:** euribor 3 month rate - daily indicator (numeric)

21) **nr.employed:** number of employees - quarterly indicator (numeric)

Output variable (desired target):

22) **y:** has the client subscribed a term deposit? (binary: "yes","no")

Table of Contents

5. **Model Comparision**

1. Import Packages

Import required packages for developing the project. goto toc

In [1]:

```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
# Pretty display for notebooks
%matplotlib inline

import seaborn as sns
# Set default setting of seaborn
sns.set()
```

2. Load Dataset

Read data from *bank.csv* file using pandas method *read_csv()*.

...goto toc

In [2]:

```
# read the data
raw_data = pd.read_csv('bank.csv',delimiter=";")

# print the first five rows of the data
raw_data.head()
```

Out[2]:

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

5 rows × 21 columns

3. Data Preprocessing

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.

Steps:

1. Understand datatypes and dimensions
2. Data Cleaning
3. Exploratory Analysis
4. Feature Selection
5. Data Transformation

...goto toc

3.1 Data Types and Dimensions

...goto toc

In [3]:

```
print("Bank Marketing Data Set has \033[4m\033[1m{}\033[0m\033[0m data points with \
```

```
Bank Marketing Data Set has 41188 data points with 21 variables each.
```

In [4]:

```
# check the data types of the features
raw_data.info()
```

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

In [5]:

```
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']

numeric_features = raw_data.select_dtypes(include=numerics).columns.tolist()
categorical_features = raw_data.select_dtypes(exclude=numerics).columns.tolist()

print(f"Number of categorical features are \033[4m\033[1m{len(categorical_features)}
```

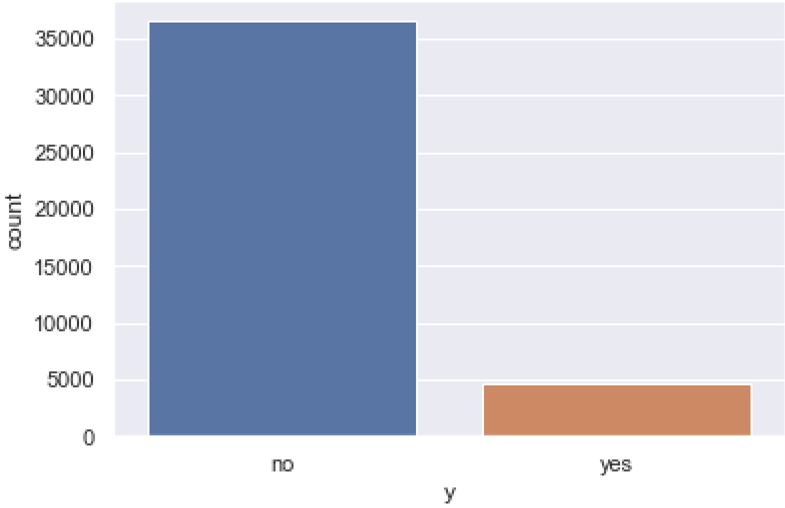
```
Number of categorical features are 11
Number of numeric features are 10
```

In [6]:

```
# Get count of each label
N, Y = raw_data['y'].value_counts()
```

```
print(f'Number of Client placed deposit : \033[4m\033[1m{Y}\033[0m\033[0m')
print(f'Number of Clients not placed deposit  : \033[4m\033[1m{N}\033[0m\033[0m')

# Plot the countplot
ax = sns.countplot(x = raw_data.y,label="count")
```

```
Number of Client placed deposit : 4640
Number of Clients not placed deposit  : 36548
```



Note: The target feature is higly imbalanced we need to make it balanced.

3.2. Data Cleaning

...goto toc

Missing Data Treatment

If the missing values are not handled properly we may end up drawing an inaccurate inference about the data. Due to improper handling, the result obtained will differ from the ones where the missing values are present.

In [7]:

```
# get the count of missing values
missing_values = raw_data.isnull().sum()

# print the count of missing values
print(missing_values)
```

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
```

```
euribor3m        0
nr.employed      0
y                0
dtype: int64
```

Note: There are no missing values in the dataset so we can proceed further

Drop duplicates

An important part of Data analysis is analyzing Duplicate Values and removing them. Pandas drop_duplicates() method helps in removing duplicates from the data frame.

In [8]:

```
# Make the copy of the original dataset
data = raw_data.copy(deep = True)

data.drop_duplicates(inplace = True)
```

In [9]:

```
print("Bank Marketing Data Set has \033[4m\033[1m{}\033[0m\033[0m data points with \
```

```
Bank Marketing Data Set has 41176 data points with 21 variables each.
```

Summary

Number of Instances	Number of Attributes	Numeric Features	Categorical Features	Missing Values
41176	21	10	11	Null

3.3. Exploratory Analysis

The preliminary analysis of data to discover relationships between measures in the data and to gain an insight on the trends, patterns, and relationships among various entities present in the data set with the help of statistics and visualization tools is called Exploratory Data Analysis (EDA).

Exploratory data analysis is cross-classified in two different ways where each method is either graphical or non-graphical. And then, each method is either univariate, bivariate or multivariate.

...goto toc

3.3.1. Numerical Features

Analysis of only numeric features

...goto toc

In [10]:

```
# Function to plot a numeric feature
def plot_numeric_feature(data, feature):

    # Create subplots figure
    fig, axes = plt.subplots(1, 2, figsize=(14, 6))
```

```python
    # Boxplot of given feature
    sns.boxplot(ax=axes[0], data = data[feature])

    # Boxplot of given feature with respect to output variable
    sns.boxplot(ax=axes[1], y = feature, data = data, x = 'y')

    # Displot of given feature with respect to output variable
    sns.displot(data=data, x=feature, hue="y", multiple="stack", kind="kde")
```

In [11]:

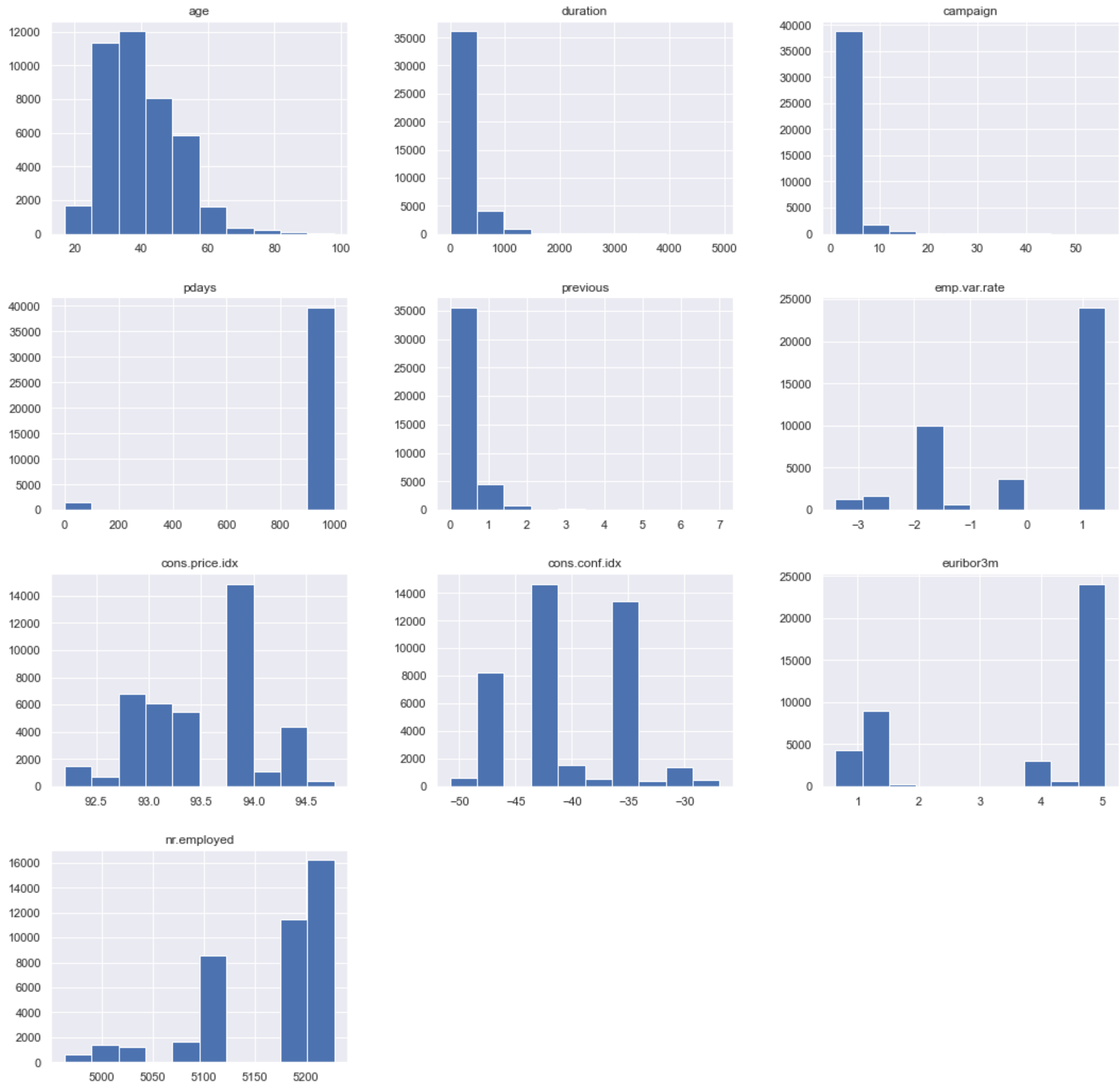
```python
# Import 'mode' function from 'statistics' library
from statistics import mode

# Function to calculate central tendancy i.e mean, median and mode of a feature
def central_tendancy(data, feature):
    print(f"Mean: \033[4m\033[1m{int(data[feature].mean())}\033[0m\033[0m \nMedian:
```

In [12]:

```python
# Function to remove outliears
def removeOutliers(data, features, n):
    outlier_free_features = []
    outliear_index = []
    new_data = pd.DataFrame()
    indexs = []
    for col in features:
        Q3 = np.quantile(data[col], 0.75)
        Q1 = np.quantile(data[col], 0.25)
        IQR = Q3 - Q1

        lower_range = Q1 - 1.5 * IQR
        upper_range = Q3 + 1.5 * IQR
        if (max(data[col]) <= upper_range) and min(data[col]) >= lower_range:
            outlier_free_features.append(col)
            new_data[col] = data[col]
            continue

        outliear_index.extend(data[col][data[col] < lower_range].index.tolist())# or
        outliear_index.extend(data[col][data[col] > upper_range].index.tolist())# or

    outliear_index = set(outliear_index)

    return data.drop(outliear_index, axis = 0)
```

In [13]:

```python
# Probability Distribution Functions of Numeric features
fig = data.hist(figsize = (18,18))
```



1. Age

...goto toc

In [14]:
```python
# Analysis of age
feature = 'age'
```

In [15]:
```python
# Statistical summary of age
data.age.describe()
```

Out[15]:
```
count    41176.00000
mean        40.02380
std         10.42068
min         17.00000
25%         32.00000
50%         38.00000
75%         47.00000
max         98.00000
Name: age, dtype: float64
```

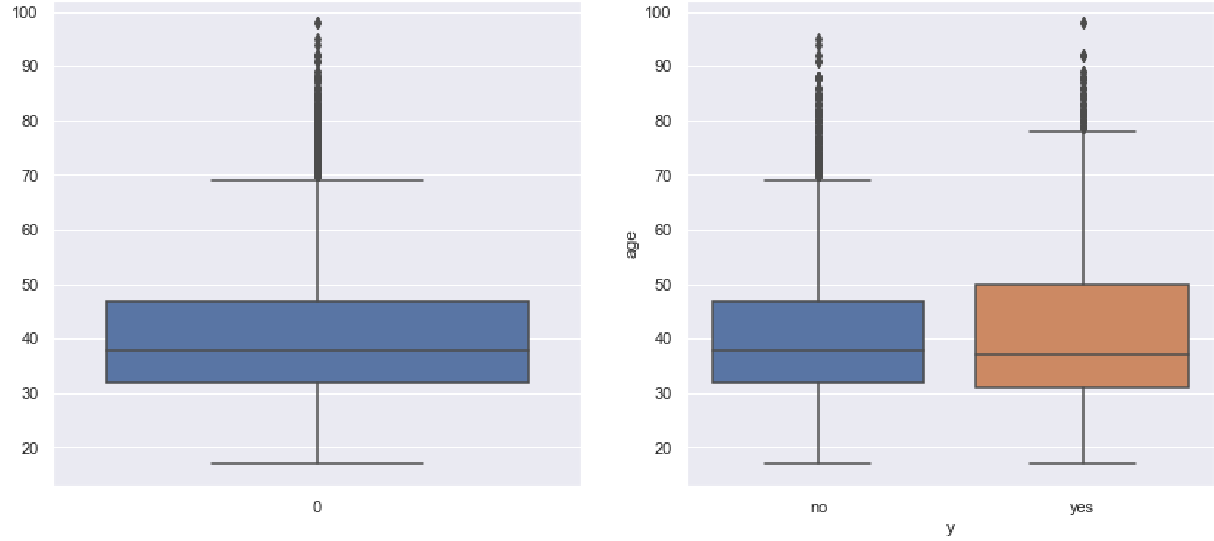
In [16]:
```python
# Calculate central tendancy of age feature
central_tendancy(data, feature)
```

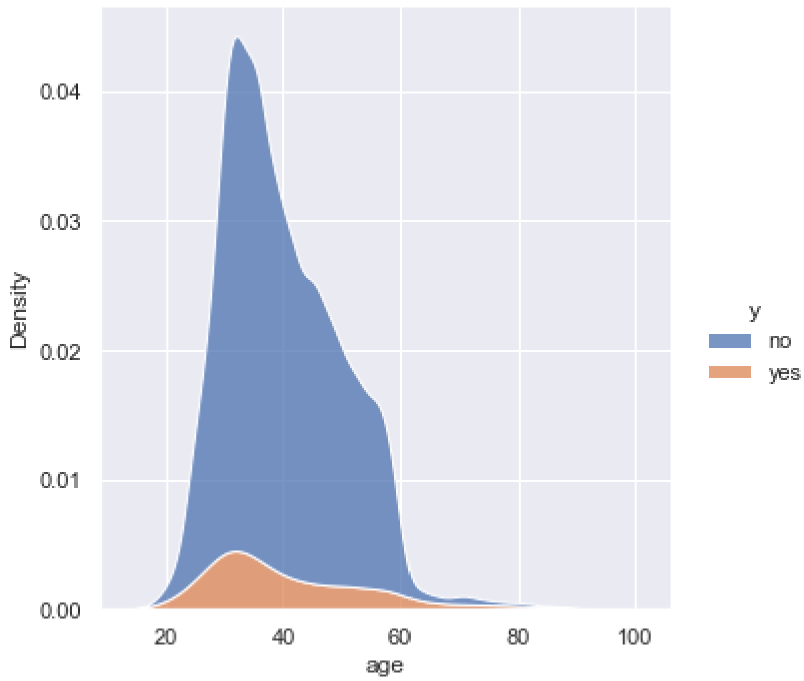
Mean: **40**

```
Median: 38
Mode: 31
```

In [17]:

```python
# Plot basis plots of age
plot_numeric_feature(data, feature)
```





Note: From analysis of age we can see that age is skewed toward right i.e positive skewness because mean > meadian > mode

So to remove skewness we will perform logarithmic transformation

2. Duration

last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y="no"). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

...goto toc

In [18]:
```python
# Analysis of duration
feature = 'duration'
```

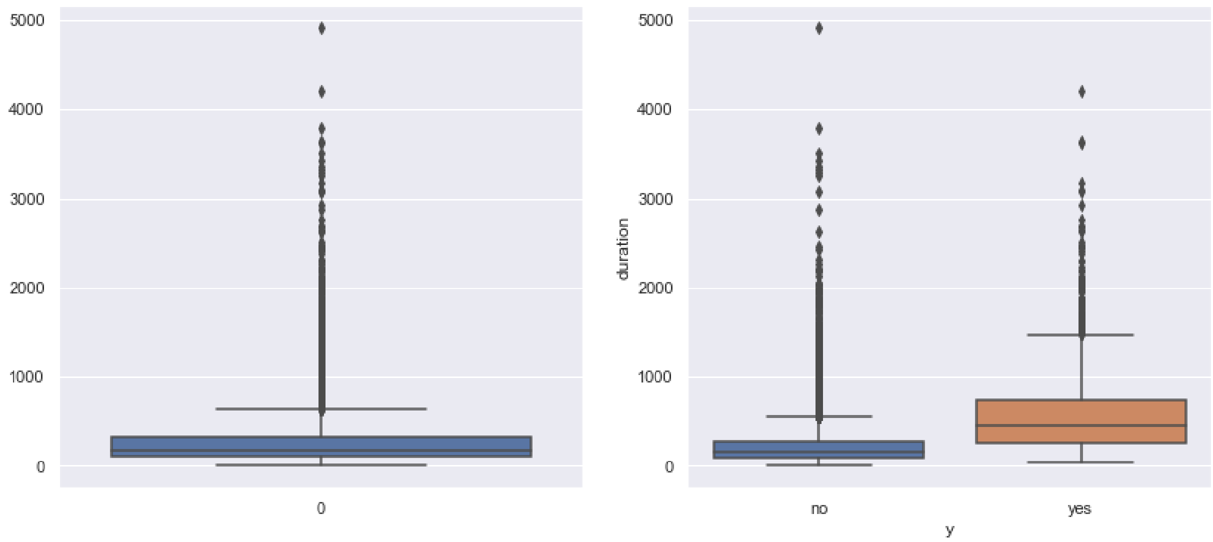
In [19]:
```python
# Statistical summary of duration
data.duration.describe()
```

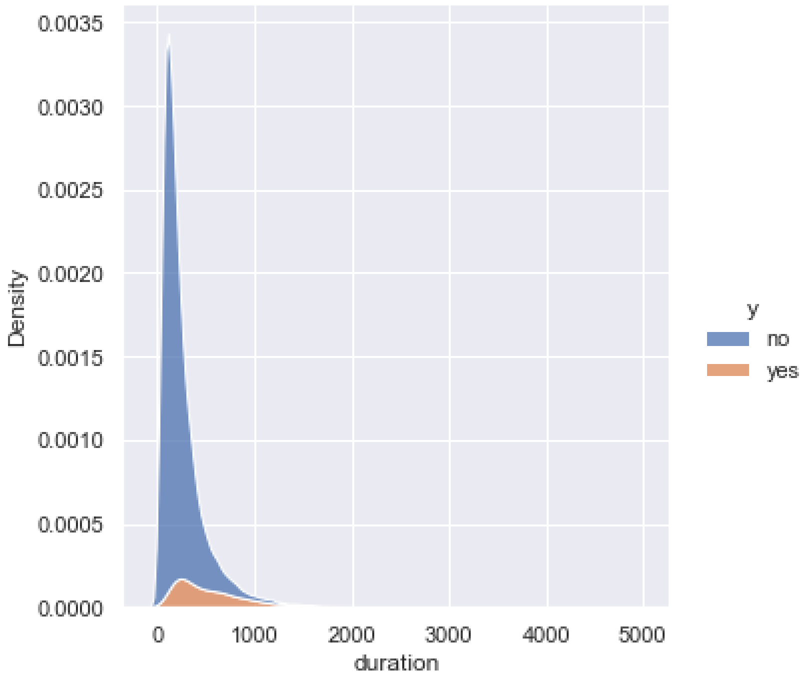
Out[19]:
```
count    41176.000000
mean       258.315815
std        259.305321
min          0.000000
25%        102.000000
50%        180.000000
75%        319.000000
max       4918.000000
Name: duration, dtype: float64
```

In [20]:
```python
# Calculate central tendancy of duration feature
central_tendancy(data, feature)
```

Mean: **258**
Median: **180**
Mode: **85**

In [21]:
```python
# Plot basis plots of duration
plot_numeric_feature(data, feature)
```





Note: We will bin the *duration* feature so as to convert it into intervals.

Binning:

- duration <= 102 : 1
- duration > 102 and duration <= 180 : 2
- duration > 180 and duration <= 319 : 3
- duration > 319 and duration <= 645 : 4
- duration > 645 : 5

3. campaign

number of contacts performed during this campaign and for this client (numeric, includes last contact)

...goto toc

In [156…

```
# Analysis of campaign
feature = 'campaign'
```

In [23]:

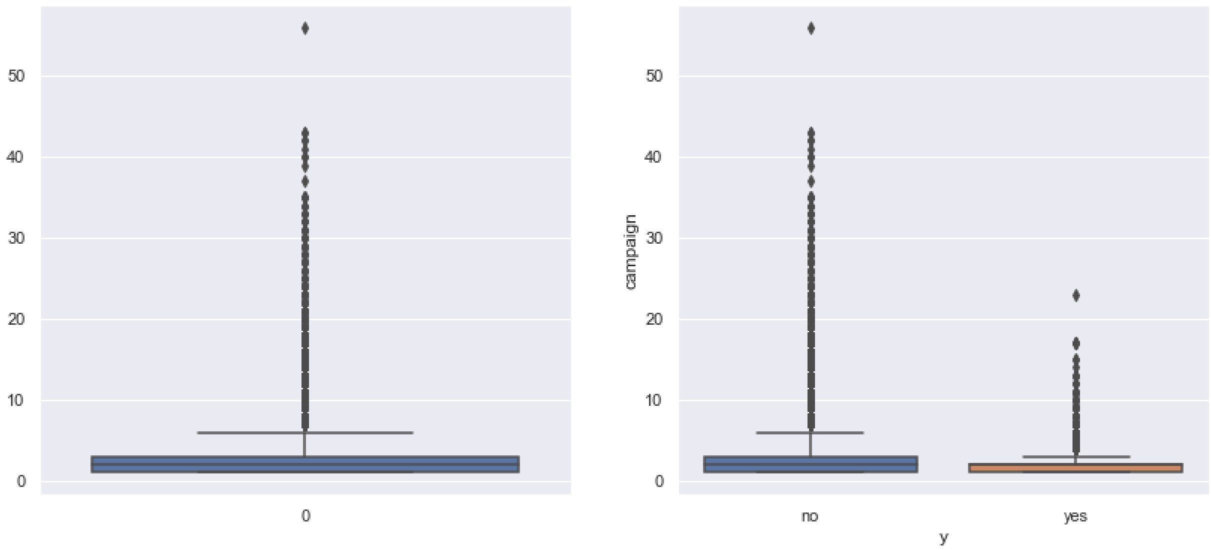
```
# Statistical summary of duration
data.campaign.describe()
```

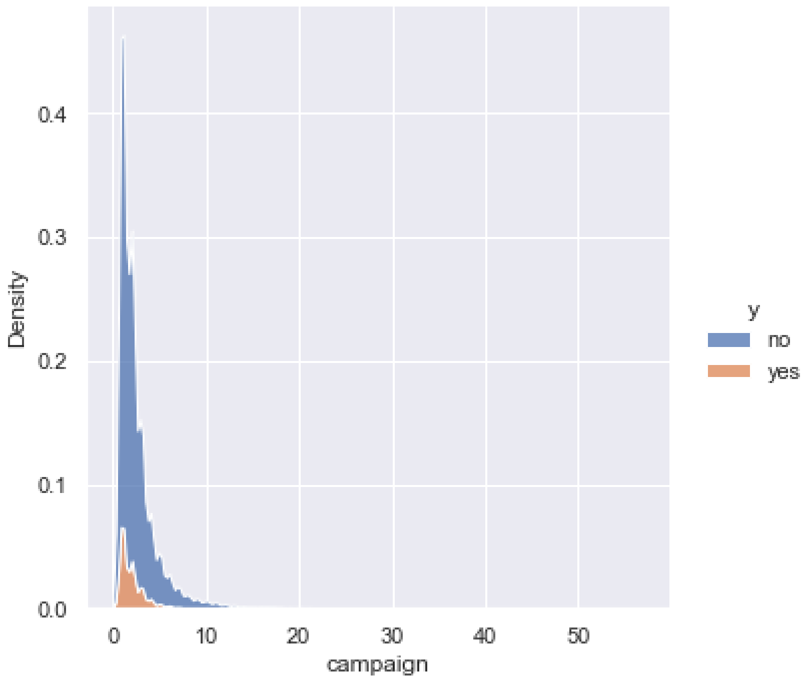
Out[23]:

```
count    41176.000000
mean         2.567879
std          2.770318
min          1.000000
25%          1.000000
50%          2.000000
75%          3.000000
max         56.000000
Name: campaign, dtype: float64
```

In [24]:

```
# Plot basis plots of campaign
plot_numeric_feature(data, feature)
```





4. previous

number of contacts performed before this campaign and for this client (numeric)

...goto toc

In [153…

```
# Analysis of previous
feature = 'previous'
```

In [26]:

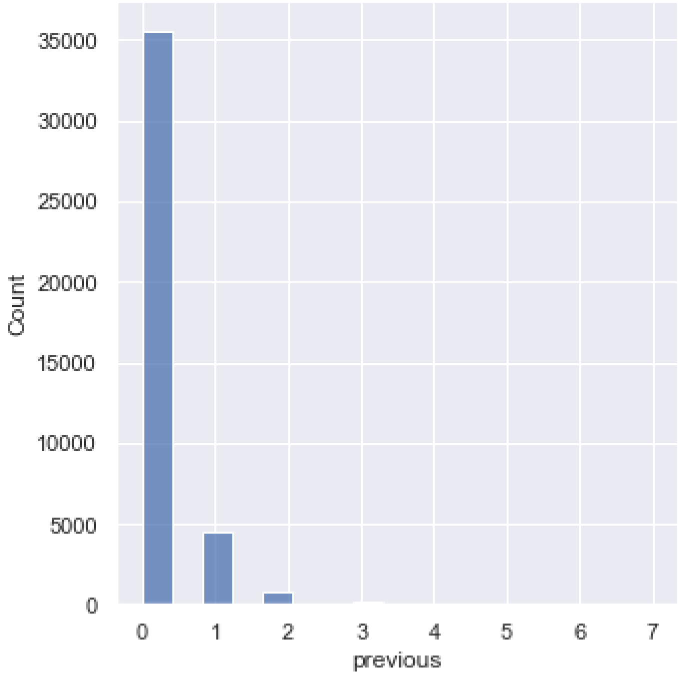
```
# Statistical summary of previous
data.previous.describe()
```

Out[26]:

```
count    41176.000000
mean         0.173013
std          0.494964
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          7.000000
Name: previous, dtype: float64
```

In [154…
```
sns.displot(x = feature, data = data)
```

Out[154…
```
<seaborn.axisgrid.FacetGrid at 0x1ed6bff6340>
```



Note: We will simply encode this feature as 1 if client is contacted else 0

5. pdays

number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

...goto toc

In [150…
```
# Analysis of pdays
feature = 'pdays'
```

In [29]:
```
# Statistical summary of pdays
data.pdays.describe()
```

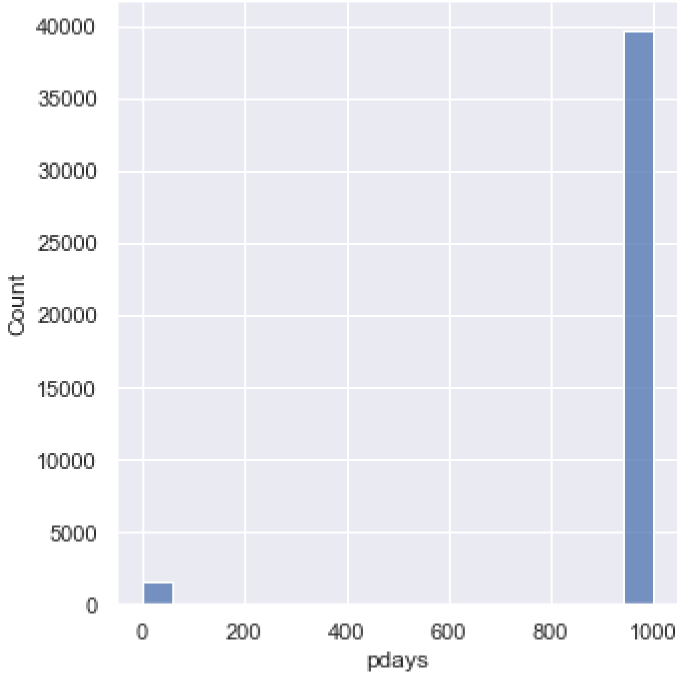
Out[29]:
```
count    41176.000000
mean       962.464810
std        186.937102
min          0.000000
25%        999.000000
50%        999.000000
75%        999.000000
max        999.000000
Name: pdays, dtype: float64
```

In [30]:
```
# Calculate central tendancy of age feature
central_tendancy(data, feature)
```

Mean: **962**
Median: **999**
Mode: **999**

In [151…
```
sns.displot(x=feature, data = data)
```

```
Out[151…  <seaborn.axisgrid.FacetGrid at 0x1ed6c12d040>
```



Note: As *999* means client was not previously contacted so we will encode it as 0 and everything will remain same

6. emp.var.rate

employment variation rate - quarterly indicator (numeric)

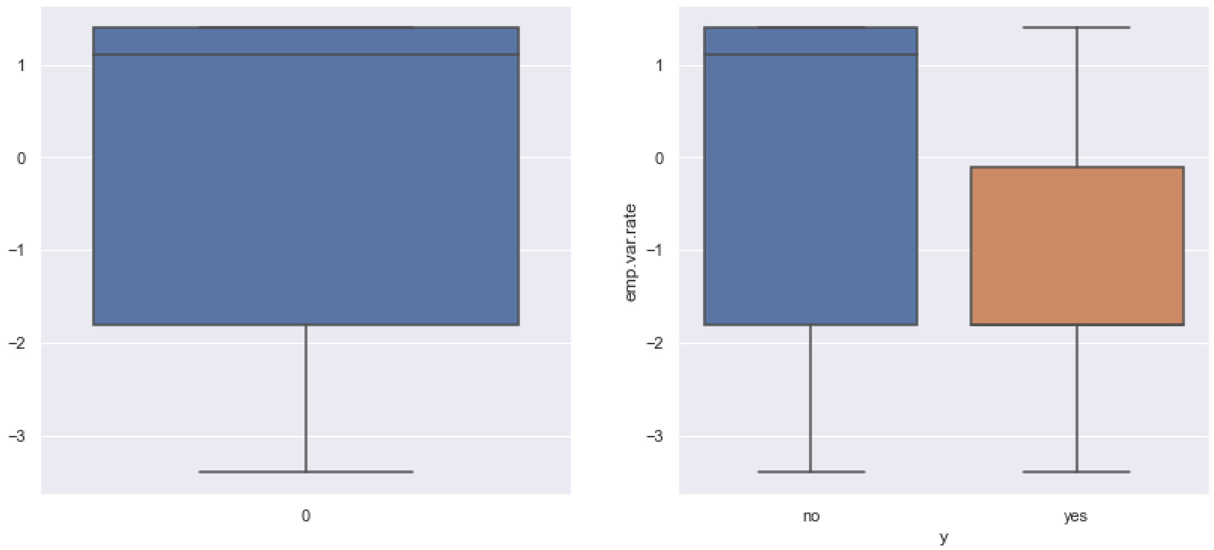
...goto toc

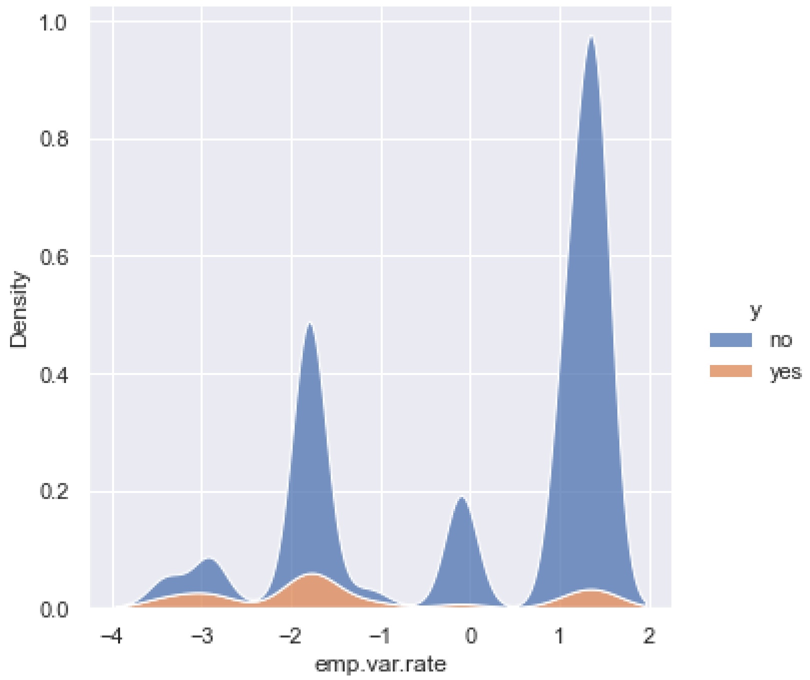
In [32]:
```
# Analysis of emp.var.rate
feature = 'emp.var.rate'
```

In [33]:
```
# Statistical summary of emp.var.rate
data['emp.var.rate'].describe()
```

Out[33]:
```
count    41176.000000
mean         0.081922
std          1.570883
min         -3.400000
25%         -1.800000
50%          1.100000
75%          1.400000
max          1.400000
Name: emp.var.rate, dtype: float64
```

In [34]:
```
# Plot basis plots of age
plot_numeric_feature(data, feature)
```





Note: We will perform logarithmic transformation by taking into consideration the negeative and positives values

7. cons.price.idx:

consumer price index - monthly indicator (numeric)

*The **Consumer Price Index (CPI)** is a measure that examines the weighted average of prices of a basket of consumer goods and services, such as transportation, food, and medical care. It is calculated by taking price changes for each item in the predetermined basket of goods and averaging them.*

...goto toc

In [35]:
```
# Analysis of cons.price.idx
feature = 'cons.price.idx'
```

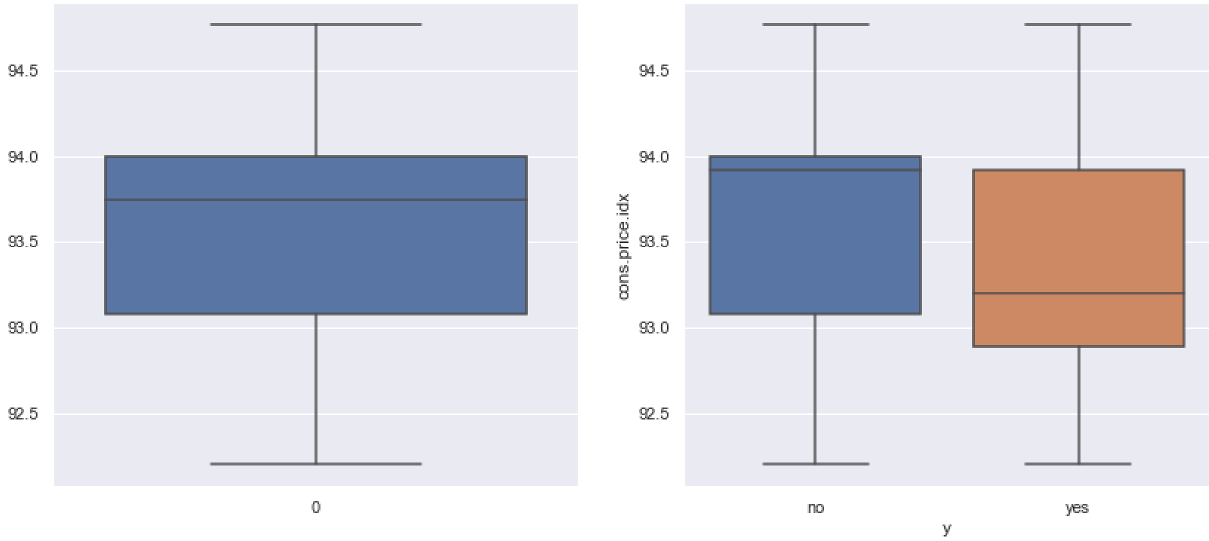
In [36]:
```
# Statistical summary of cons.price.idx
data['cons.price.idx'].describe()
```

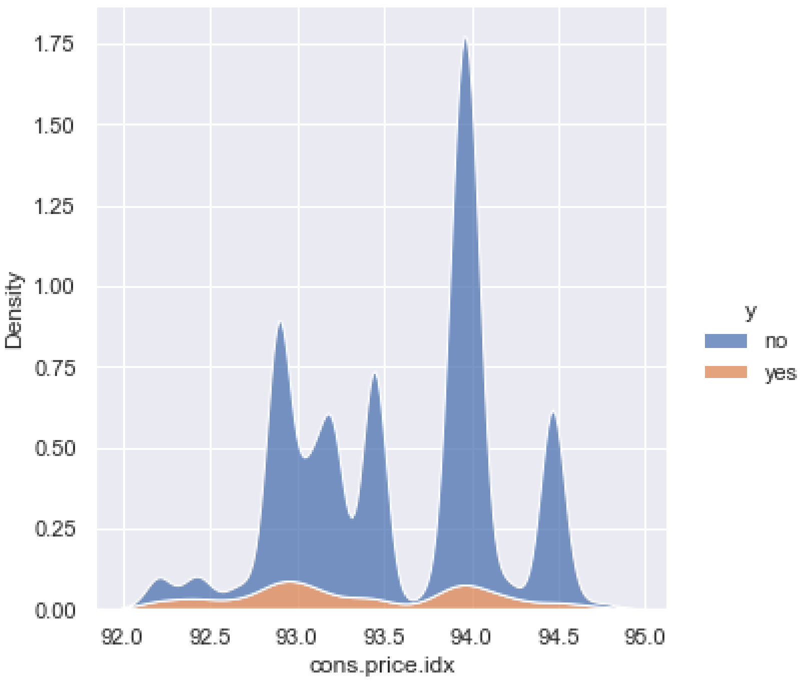
```
Out[36]: count    41176.000000
mean        93.575720
std          0.578839
min         92.201000
25%         93.075000
50%         93.749000
75%         93.994000
max         94.767000
Name: cons.price.idx, dtype: float64
```

```
In [37]: # Calculate central tendancy of cons.price.idx feature
central_tendancy(data, feature)
```

Mean: **93**
Median: **93**
Mode: **93**

```
In [38]: # Plot basis plots of age
plot_numeric_feature(data, feature)
```





8. cons.conf.idx

consumer confidence index - monthly indicator (numeric)

*This **consumer confidence indicator** provides an indication of future developments of households' consumption and saving, based upon answers regarding their expected financial situation, their sentiment about the general economic situation, unemployment and capability of savings.*

...goto toc

In [39]:
```
# Analysis of cons.conf.idx
feature = 'cons.conf.idx'
```

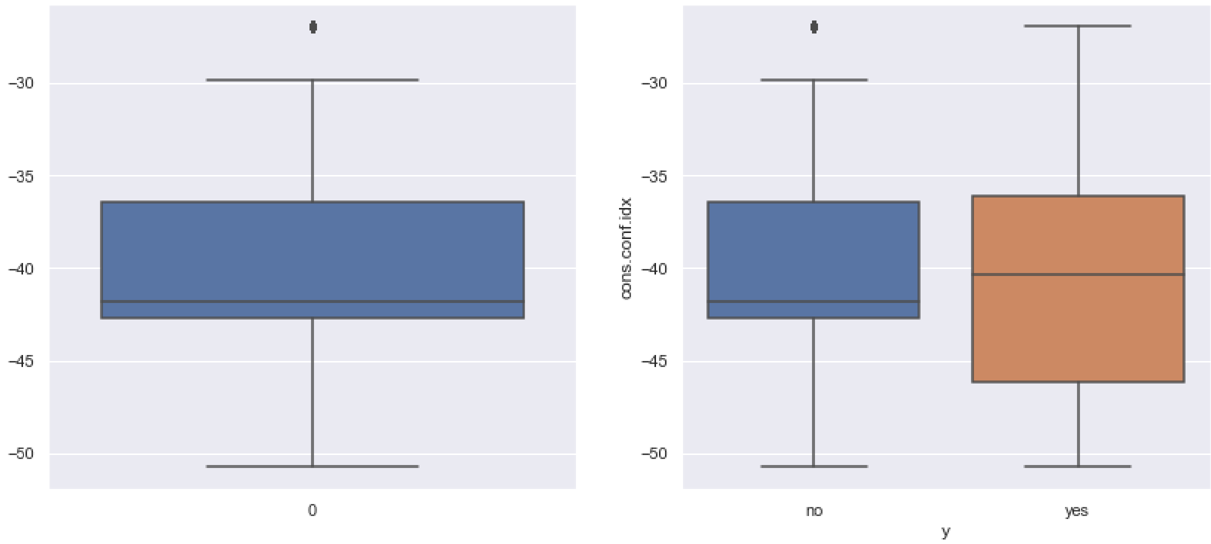
In [40]:
```
# Statistical summary of cons.price.idx
data['cons.conf.idx'].describe()
```

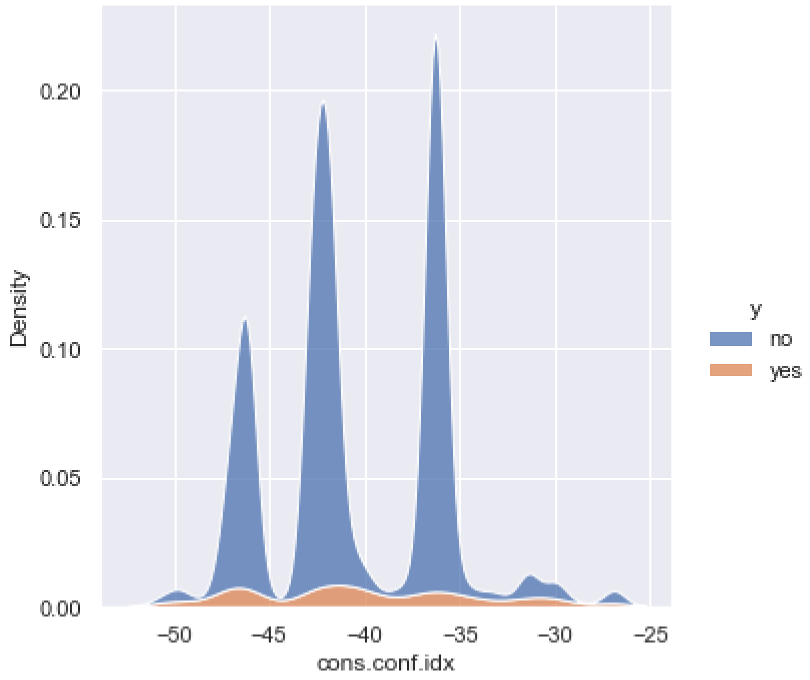
Out[40]:
```
count    41176.000000
mean       -40.502863
std          4.627860
min        -50.800000
25%        -42.700000
50%        -41.800000
75%        -36.400000
max        -26.900000
Name: cons.conf.idx, dtype: float64
```

In [41]:
```
# Calculate central tendancy of age feature
central_tendancy(data, feature)
```

Mean: **-40**
Median: **-41**
Mode: **-36**

In [42]:
```
# Plot basis plots of cons.price.idx
plot_numeric_feature(data, feature)
```





9. euribor3m

euribor 3 month rate - daily indicator (numeric)

Euribor is short for **Euro Interbank Offered Rate**. *The Euribor rates are based on the average interest rates at which a large panel of European banks borrow funds from one another.*

...goto toc

In [43]:
```
# Analysis of euribor3m
feature = 'euribor3m'
```

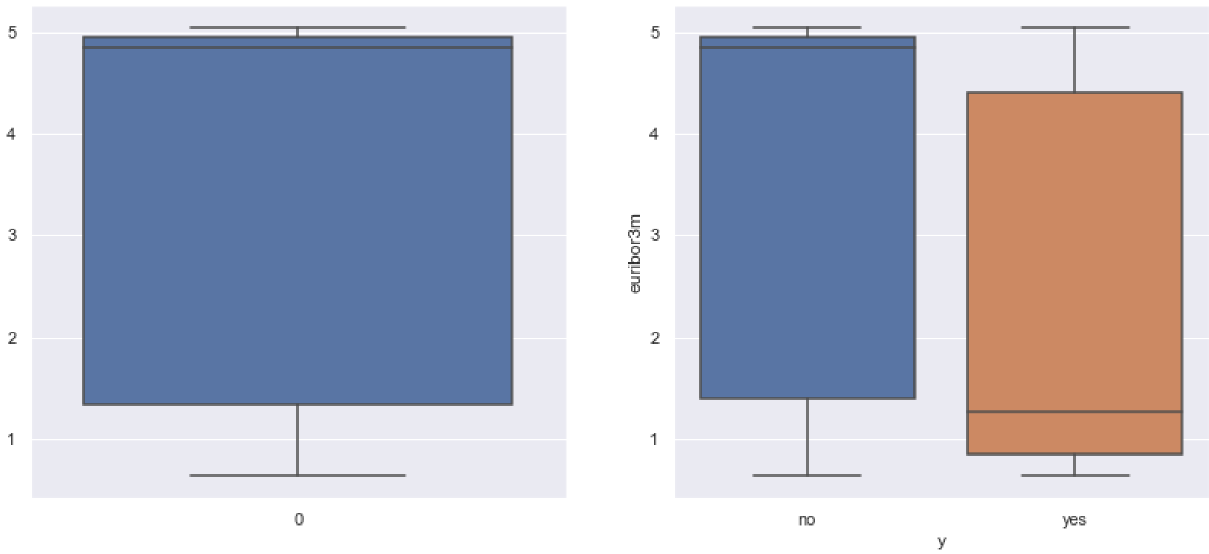
In [44]:
```
# Statistical summary of euribor3m
data.euribor3m.describe()
```

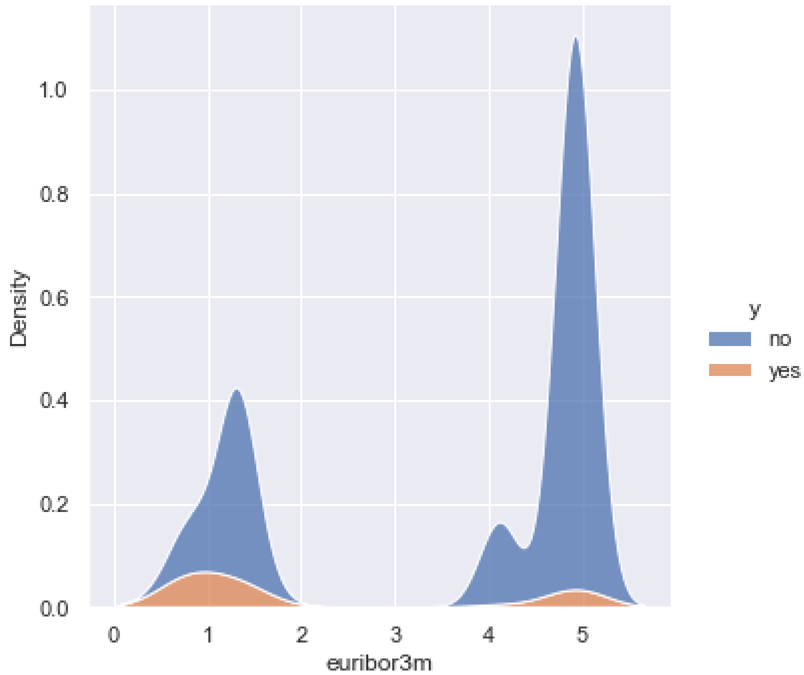
Out[44]:
```
count    41176.000000
mean         3.621293
std          1.734437
min          0.634000
25%          1.344000
50%          4.857000
75%          4.961000
max          5.045000
Name: euribor3m, dtype: float64
```

In [45]:
```
# Calculate central tendancy of euribor3m feature
central_tendancy(data, feature)
```

Mean: 3
Median: 4
Mode: 4

In [46]:
```
# Plot basis plots of euribor3m
plot_numeric_feature(data, feature)
```





Note: There are no outliears in *euribor3m* feature. We will explore it more in further analysis to get clear picture of this feature.

10. nr.employed

number of employees - quarterly indicator (numeric)

...goto toc

In [47]:

```
# Analysis of nr.employed
feature = 'nr.employed'
```

In [48]:

```
# Statistical summary of nr.employed
data['nr.employed'].describe()
```

Out[48]:

```
count    41176.000000
mean      5167.034870
std         72.251364
min       4963.600000
25%       5099.100000
50%       5191.000000
75%       5228.100000
```

```
max        5228.100000
Name: nr.employed, dtype: float64
```

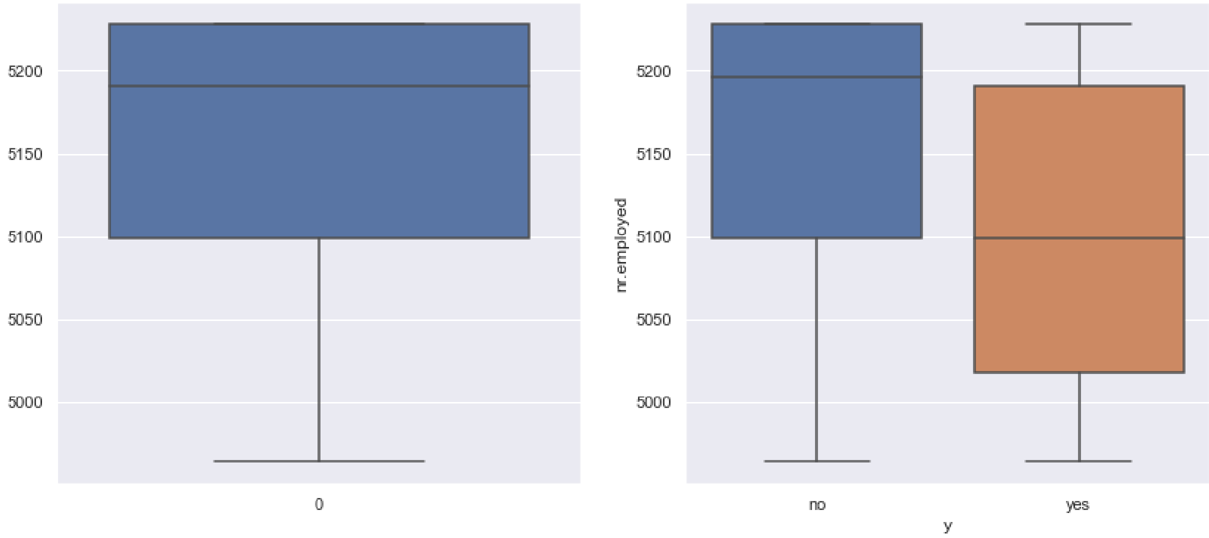
In [49]:

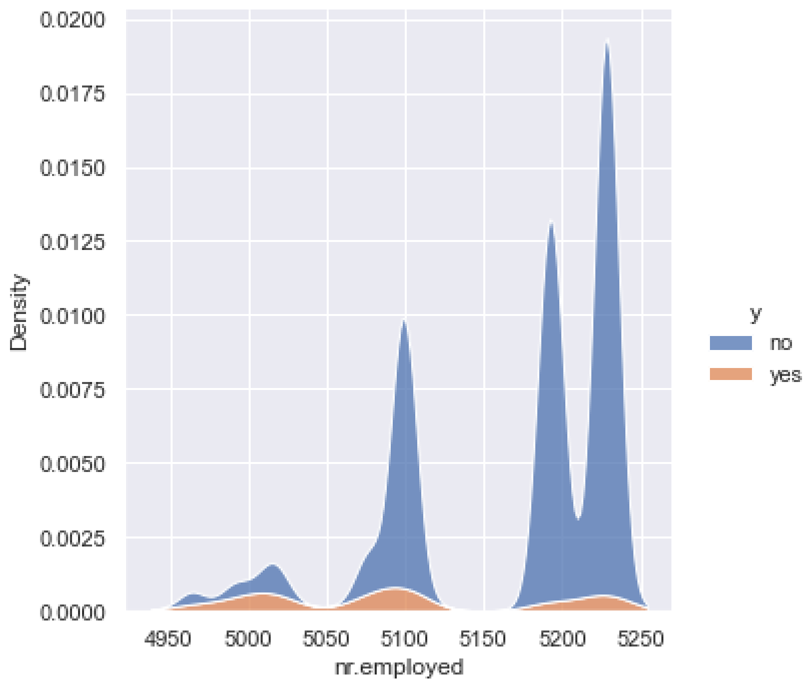
```
# Calculate central tendancy of nr.employed feature
central_tendancy(data, feature)
```

```
Mean: 5167
Median: 5191
Mode: 5228
```

In [50]:

```
# Plot basis plots of nr.employed
plot_numeric_feature(data, feature)
```





Note: There are no outliears in *nr.employed* feature. We will explore it more in further analysis to get clear picture of this feature.

Correlation

In [51]:

```
# check correlation
corr = data.corr(method = 'spearman')
corr
```

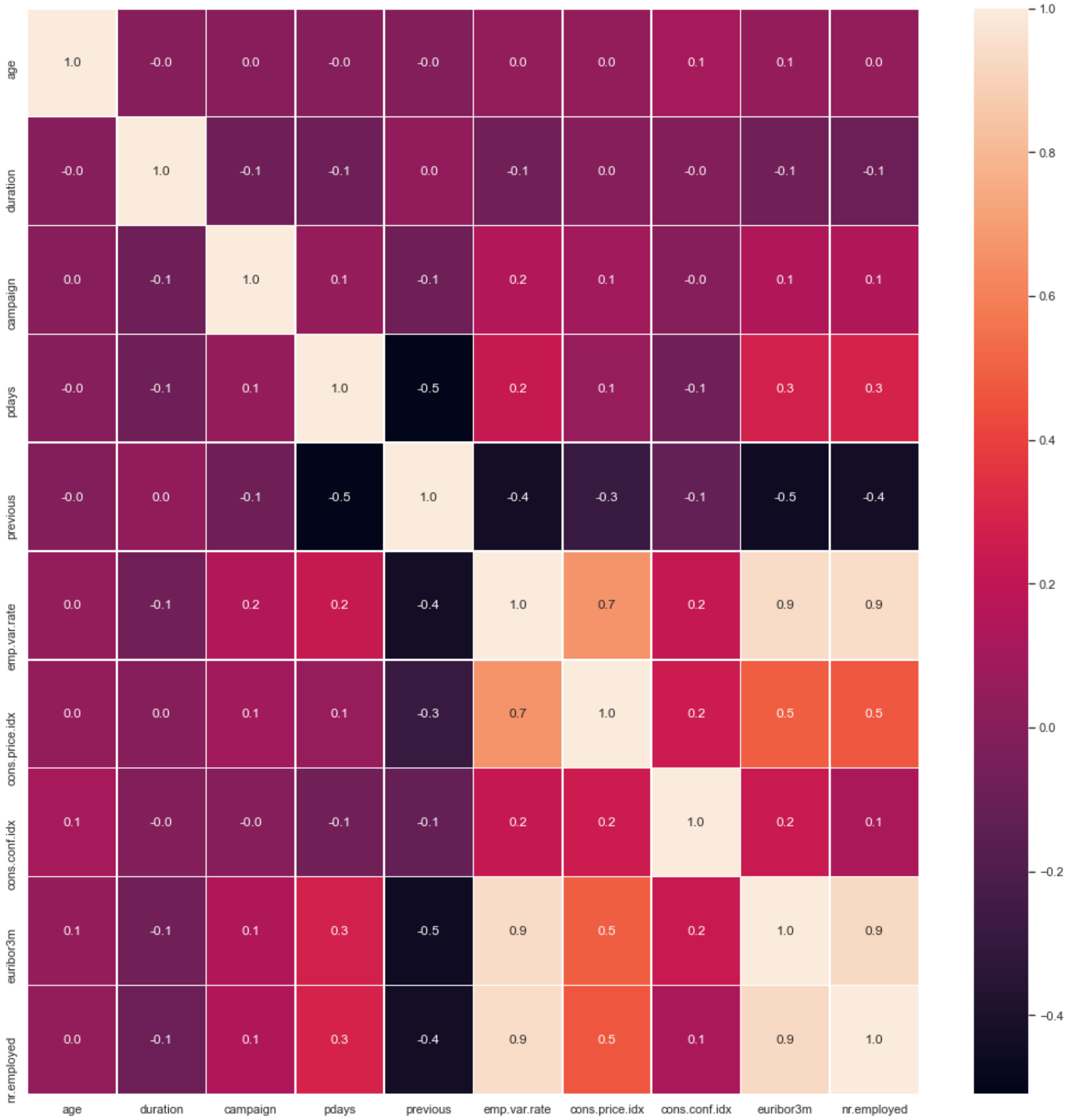
Out[51]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	co
age	1.000000	-0.002017	0.005754	-0.001065	-0.012639	0.045064	0.044871	
duration	-0.002017	1.000000	-0.081101	-0.083056	0.042360	-0.069110	0.002872	
campaign	0.005754	-0.081101	1.000000	0.055551	-0.087491	0.156419	0.096475	
pdays	-0.001065	-0.083056	0.055551	1.000000	-0.509580	0.227741	0.056785	
previous	-0.012639	0.042360	-0.087491	-0.509580	1.000000	-0.435385	-0.282791	
emp.var.rate	0.045064	-0.069110	0.156419	0.227741	-0.435385	1.000000	0.664881	
cons.price.idx	0.044871	0.002872	0.096475	0.056785	-0.282791	0.664881	1.000000	
cons.conf.idx	0.114313	-0.008637	-0.001403	-0.077283	-0.115981	0.224840	0.245771	
euribor3m	0.054460	-0.078221	0.140634	0.278530	-0.454800	0.939915	0.490945	
nr.employed	0.044845	-0.095135	0.144311	0.290714	-0.438791	0.944687	0.464699	

In [52]:

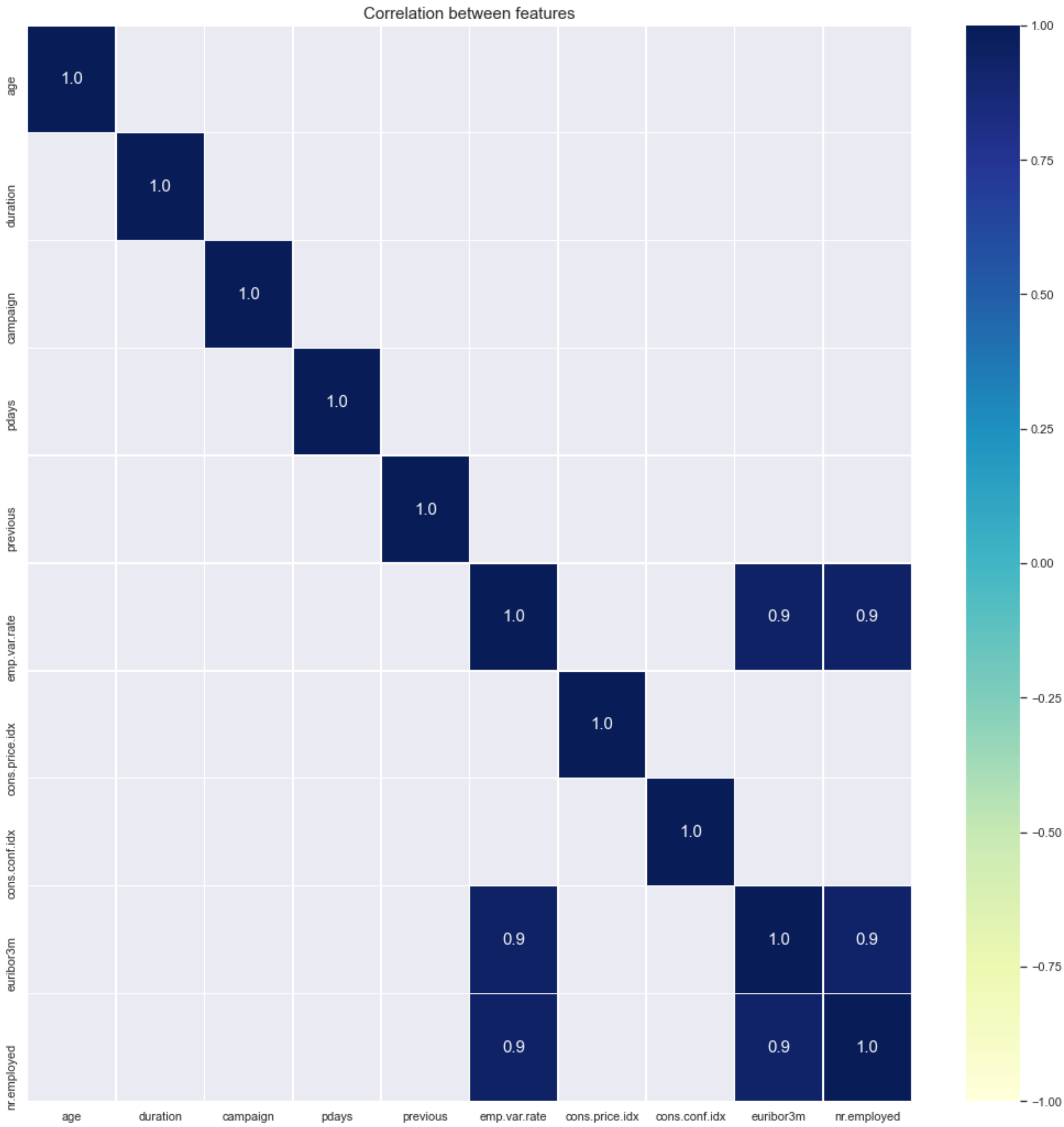
```
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(corr, annot=True, linewidths=.5, fmt= '.1f', ax=ax)
```

Out[52]: <AxesSubplot:>



In [53]:

```
plt.figure(figsize=(18, 18))
sns.heatmap(corr[(corr >= 0.9) | (corr <= -0.9)],
            cmap='YlGnBu', vmax=1.0, vmin=-1.0,
            annot=True, annot_kws={"size": 15}, linewidths=.5, fmt= '.1f')
plt.title('Correlation between features', fontsize=15)
plt.show()
```



Note: Features **emp.var.rate**, **euribor3m** and **nr.employed** are highly correlated

3.3.2. Categorical Features

Analysis of categorical features

...goto toc

In [54]:
```python
# Building a function to visualize categorical features
def plot_categorical_feature(data, feature):

    temp_1 = pd.DataFrame() # temp dataframe

    # count categorical values
    temp_1['No_deposit'] = data[data['y'] == 'no'][feature].value_counts()
    temp_1['Yes_deposit'] = data[data['y'] == 'yes'][feature].value_counts()

    # Plot barplot
    temp_1.plot(kind='bar')
    plt.xlabel(f'{feature}')
    plt.ylabel('Number of clients')
```

```
    plt.title('Distribution of {} and deposit'.format(feature))
    plt.show()
```

In [55]:
```
print(categorical_features)
```

```
['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'd
ay_of_week', 'poutcome', 'y']
```

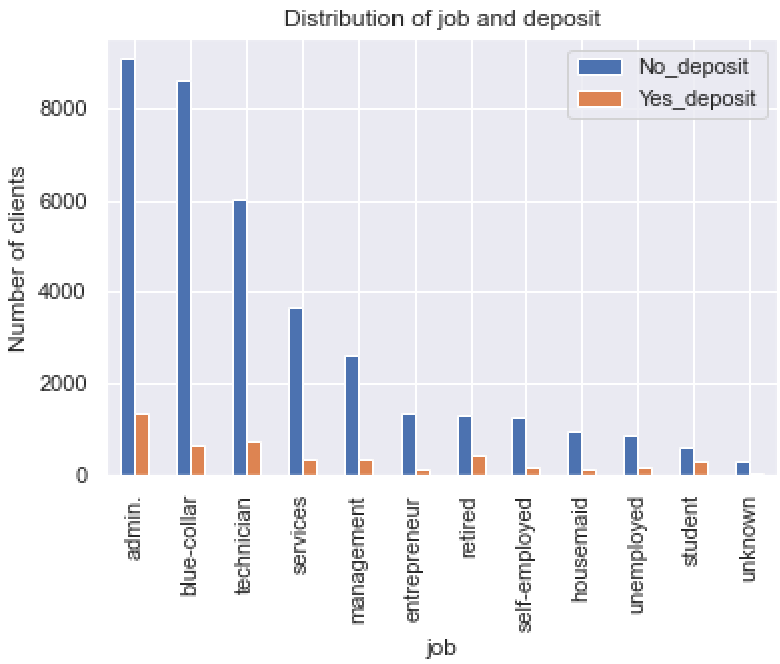
1. Job

type of job(categorical:"admin.","bluecollar","entrepreneur","housemaid","management","retired","self-employed","services","student","technician","unemployed","unknown")

...goto toc

In [56]:
```
# Analysis of job
feature = 'job'
```

In [57]:
```
# Plot barplot of 'job' with respect to 'y' feature
plot_categorical_feature(data,feature)
```



Conclusion:

- Job has 12 different categories
- Additionally they are supposed to be encoded

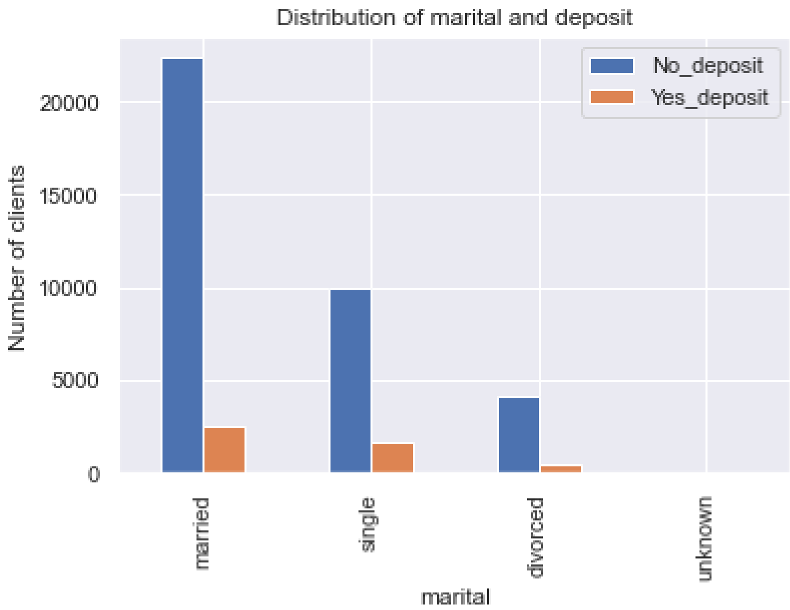
2. marital

marital status (categorical: "divorced","married","single","unknown"; note: "divorced" means divorced or widowed)

...goto toc

In [58]:
```
# Analysis of marital
feature = 'marital'
```

In [59]:
```
# Plot barplot of 'marital' with respect to 'y' feature
plot_categorical_feature(data,feature)
```



Note:

- Martial status should be calssified into 3 categories - married, single and divorced
- *Unknown* is acting as as null value and should be handled

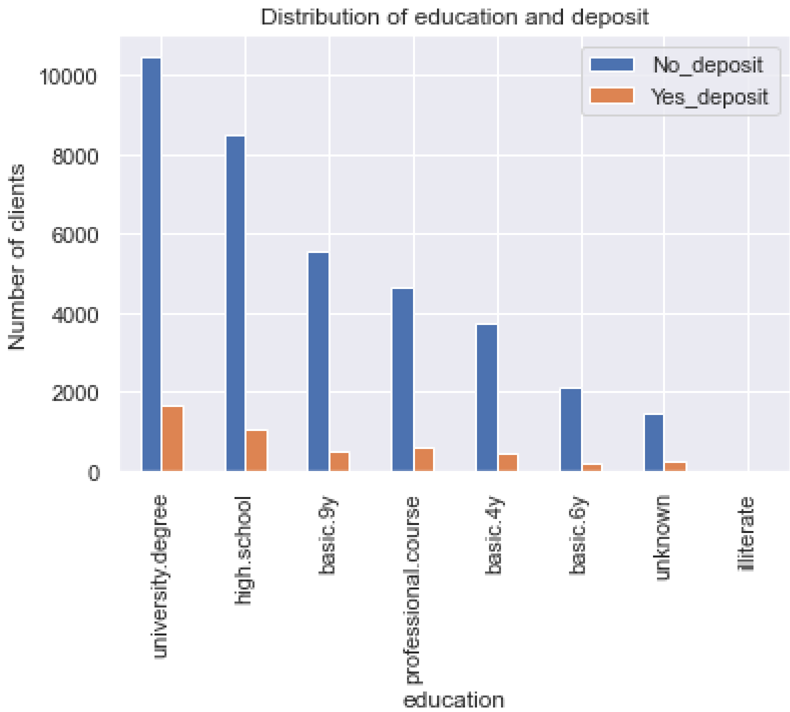
3. education

education of individual (categorical:
"basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree","unk

...goto toc

In [60]:
```
# Analysis of education
feature = 'education'
```

In [61]:
```
# Plot barplot of 'education' with respect to 'y' feature
plot_categorical_feature(data,feature)
```



Note:

Since the education qualification of the customers matters a lot, so it should be encoded properly

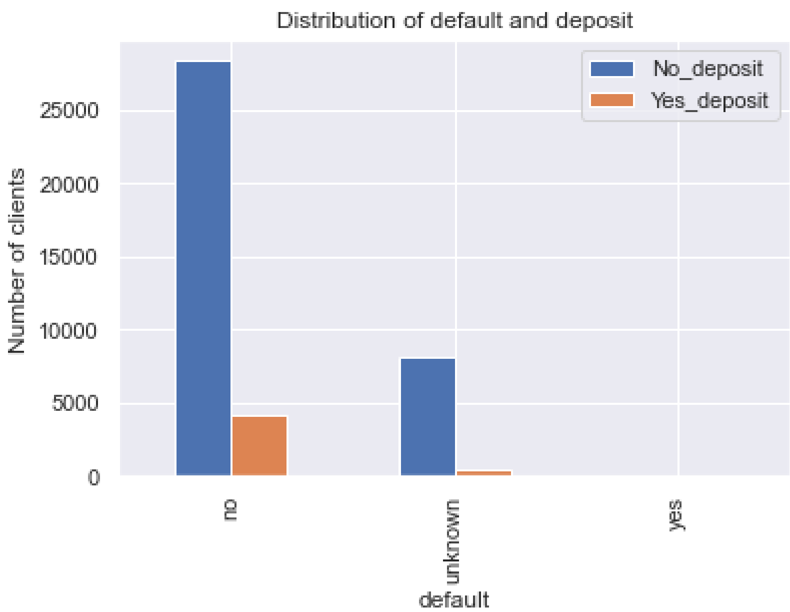
4. default

has credit in default? (categorical: "no","yes","unknown")

...goto toc

In [62]:
```
# Analysis of  default
feature = 'default'
```

In [63]:
```
# Plot barplot of 'default' with respect to 'y' feature
plot_categorical_feature(data,feature)
```



Note: Missing values are encoded as *Unknown* and *default* is binary variable having class *yes* or *no*

5. housing

has housing loan? (categorical: "no","yes","unknown")

...goto toc

In [64]:
```
# Analysis of  housing
feature = 'housing'
```

In [65]:
```
# Plot barplot of 'housing' with respect to 'y' feature
plot_categorical_feature(data,feature)
```

Note: Missing values are encoded as *Unknown* and *housing* is binary variable having class *yes* or *no*

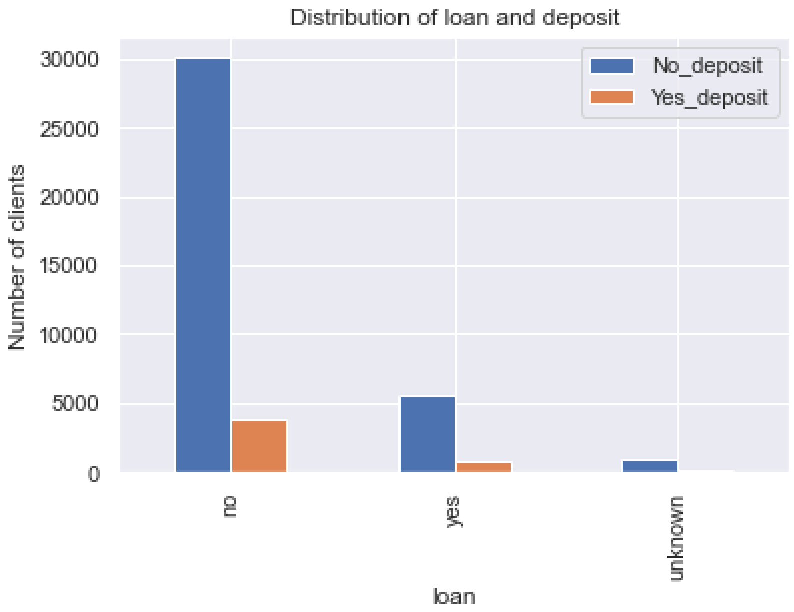
6. loan

has personal loan? (categorical: "no","yes","unknown")

...goto toc

In [66]:
```
# Analysis of loan
feature = 'loan'
```

In [67]:
```
# Plot barplot of 'loan' with respect to 'y' feature
plot_categorical_feature(data,feature)
```



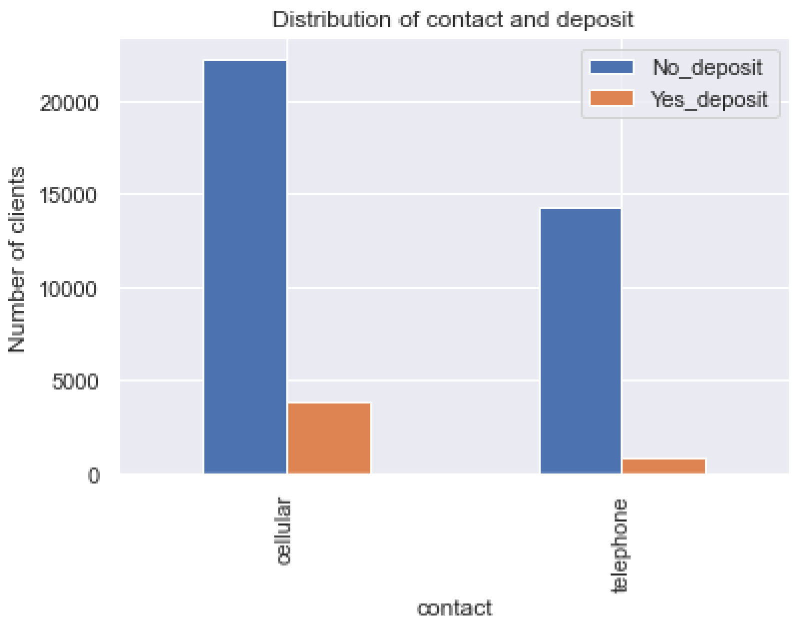
7. contact

contact communication type (categorical: "cellular","telephone")

...goto toc

In [68]:
```
# Analysis of contact
feature = 'contact'
```

In [69]:
```
# Plot barplot of 'contact' with respect to 'y' feature
plot_categorical_feature(data,feature)
```



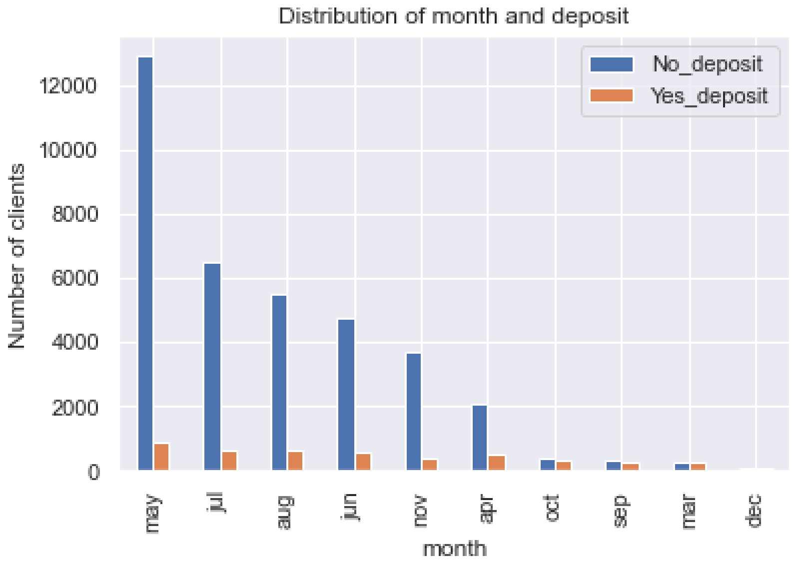
8. month

last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")

...goto toc

In [70]:
```
# Analysis of month
feature = 'month'
```

```
In [71]: # Plot barplot of 'month' with respect to 'y' feature
plot_categorical_feature(data,feature)
```



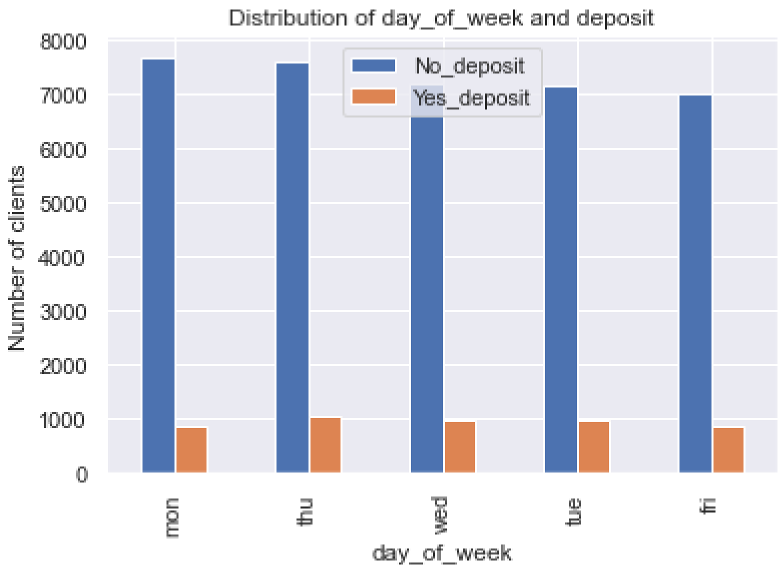
9. day_of_week

last contact day of the week (categorical: "mon","tue","wed","thu","fri")

...goto toc

```
In [72]: # Analysis of day_of_week
feature = 'day_of_week'
```

```
In [73]: # Plot barplot of 'day_of_week' with respect to 'y' feature
plot_categorical_feature(data,feature)
```



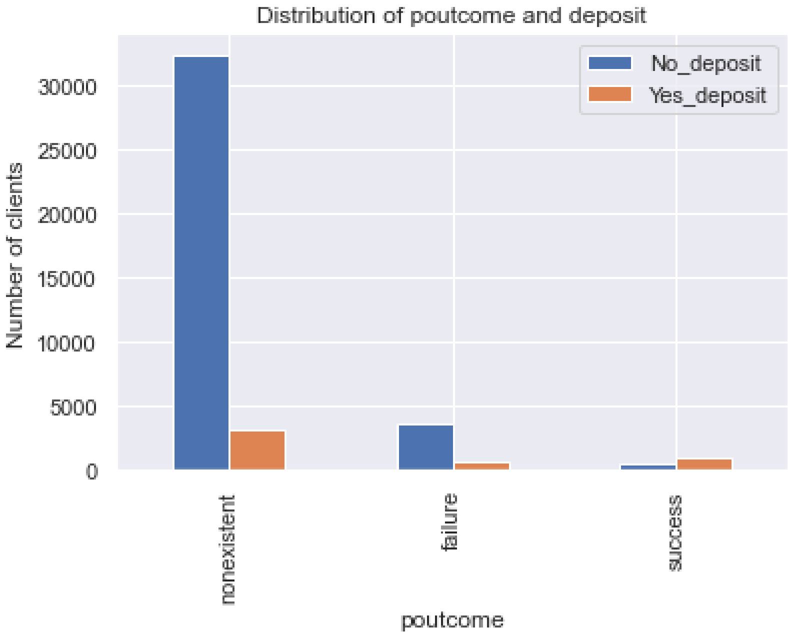
10. poutcome

outcome of the previous marketing campaign (categorical: "failure","nonexistent","success")

...goto toc

```
In [74]: # Analysis of poutcome
feature = 'poutcome'
```

```
In [75]: # Plot barplot of 'poutcome' with respect to 'y' feature
plot_categorical_feature(data,feature)
```



3.3.3. Analysis Report

...goto toc

Exploratory Data Analysis

Number of Instances	Number of Attributes	Numeric Features	Categorical Features	Target Feature	Missing Values
41176	21	10	11	y (binary)	Null

Numeric Features

- **Age** feature is skewed toward right i.e positive skewness. So to remove skewness we should perform *logarithmic transformation*
- **duration** should be *binned* on based on its data distribution as
  - asd
- **previous** feature should be *encoded* as
  - '1' : if contacted to the customer
  - '0' : if no contact made
- In **pdays** feature, we should encode **'999'** as **'0'** which means that client was not previously contacted
- In **emp.var.rate** we should perform *logarithmic transformation* by taking into consideration the negeative and positives values

- For feature **cons.price.idx**, we should first multiply it by **10** and then perform logarithmic transformation
- In feature **cons.conf.idx** all values are negeative so we should first convert them into positive and then should perform logarithmic transformation
- In feature **nr.employed** the values are on higher scale i.e thousand scale, so they should be reduced on lower scale using logarithmic tranformation
- Higly correlated features (**employment rate**, **consumer confidence index**, **consumer price index**) may describe clients state from different social-economic angles. Their variance might support model capacity for generalization.

Categorical Features

- For categories of more than 3 types of possible option (**marital** and **education**) it is proposed to use the encode targeting - it will allow correctly relate the values to the target variable and use indicated categories in numerical form
- Additionally features like 'job', 'month', 'day_of_week' should be encoded using **One-hot encoding** method as these features are nomial in nature
- The feature **poutcome** should be labelled as
  - nonexistent: 0
  - failure : 0
  - success : 1
- The feature **contact** should be labelled as
  - telephone : 0
  - cellular : 1
- Features like **loan** and **housing** should be labelled as
  - unknown: 0
  - no : 0
  - yes : 1
- The feature **default** should be labelled as
  - unknown: 0
  - no : 1
  - yes : 0
- Target feature **y** should be encoded as 0 or 1

In [76]:

```
# Create the copuy of the dataset
data_1 = data.copy(deep = True)
```

In [77]:

```
# Labelling contact, loan, housing, default, poutcome
data_1.contact = data_1.contact.map({'cellular': 1, 'telephone': 0}).astype('uint8')
data_1.loan = data_1.loan.map({'yes': 1, 'unknown': 0, 'no' : 0}).astype('uint8')
data_1.housing = data_1.housing.map({'yes': 1, 'unknown': 0, 'no' : 0}).astype('uint
data_1.default = data_1.default.map({'no': 1, 'unknown': 0, 'yes': 0}).astype('uint8
data_1.poutcome = data_1.poutcome.map({'nonexistent':0, 'failure':0, 'success':1}).a

# Encode duration feature
data_1.loc[data_1['duration'] <= 102, 'duration'] = 1
data_1.loc[(data_1['duration'] > 102) & (data_1['duration'] <= 180)  , 'duration'] =
data_1.loc[(data_1['duration'] > 180) & (data_1['duration'] <= 319)  , 'duration'] =
data_1.loc[(data_1['duration'] > 319) & (data_1['duration'] <= 645), 'duration'] = 4
data_1.loc[data_1['duration']  > 645, 'duration'] = 5
```

In [78]:

```
# replace 999 with 0
data_1.pdays = data_1.pdays.replace(999, 0)

# replace with 0 if not contact
data_1.previous = data_1.previous.apply(lambda x: 1 if x > 0 else 0).astype('uint8')

# change the range of Var Rate
data_1['emp.var.rate'] = data_1['emp.var.rate'].apply(lambda x: x*-0.0001 if x > 0 e
data_1['emp.var.rate'] = data_1['emp.var.rate'] * -1
data_1['emp.var.rate'] = data_1['emp.var.rate'].apply(lambda x: -np.log(x) if x < 1

# Multiply consumer index
data_1['cons.price.idx'] = (data_1['cons.price.idx'] * 10).astype('uint8')

# change the sign (we want all be positive values)
data_1['cons.conf.idx'] = data_1['cons.conf.idx'] * -1

# re-scale variables
data_1['nr.employed'] = np.log2(data_1['nr.employed']).astype('uint8')
data_1['cons.price.idx'] = np.log2(data_1['cons.price.idx']).astype('uint8')
data_1['cons.conf.idx'] = np.log2(data_1['cons.conf.idx']).astype('uint8')
data_1.age = np.log(data_1.age)

# Reduce meemory consumption
data_1.euribor3m = data_1.euribor3m.astype('uint8')
data_1.campaign = data_1.campaign.astype('uint8')
data_1.pdays = data_1.pdays.astype('uint8')
```

In [79]:

```
# fucntion to perform One Hot Encoding
def encode(data, col):
    return pd.concat([data, pd.get_dummies(col, prefix=col.name)], axis=1)
```

In [80]:

```
# One Hot encoding of 3 variable
data_1 = encode(data_1, data_1.job)
data_1 = encode(data_1, data_1.month)
data_1 = encode(data_1, data_1.day_of_week)
```

In [81]:

```
# Drop tranfromed features
data_1.drop(['job', 'month', 'day_of_week'], axis=1, inplace=True)
```

In [82]:

```
# Convert target variable into numeric
data_1.y = data_1.y.map({'no':0, 'yes':1})
```

In [83]:

```
# Target encoder for features - 'marital' and 'education'
import category_encoders as ce

# save target variable before transformation
y = data_1.y

# Create target encoder object and transoform two value
target_encode = ce.target_encoder.TargetEncoder(cols=['marital', 'education']).fit(d
cleaned_data = target_encode.transform(data_1)

# drop target variable
cleaned_data.drop('y', axis=1, inplace=True)
```

```
C:\Users\arun\anaconda3\envs\place_deposit\lib\site-packages\category_encoders\util
s.py:21: FutureWarning: is_categorical is deprecated and will be removed in a future
version.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

In [84]:

```python
cleaned_data.head()
```

Out[84]:

	age	marital	education	default	housing	loan	contact	duration	campaign	pdays	...
0	4.025352	0.101561	0.102490	1	0	0	0	3	1	0	...
1	4.043051	0.101561	0.108389	0	0	0	0	2	1	0	...
2	3.610918	0.101561	0.108389	1	1	0	0	3	1	0	...
3	3.688879	0.101561	0.082060	1	0	0	0	2	1	0	...
4	4.025352	0.101561	0.108389	1	0	1	0	3	1	0	...

5 rows × 44 columns

In [85]:

```python
cleaned_data.shape
```

Out[85]:

```
(41176, 44)
```

In [86]:

```python
y.shape
```

Out[86]:

```
(41176,)
```

3.4. Feature Selection

Since there are all together **44** independent features we need to perform feature selection to eliminate curse of dimensionality

...goto toc

In [87]:

```python
# Import required functions for feature selection
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
```

In [88]:

```python
# Initialize the feature selector in our case Random Forest Classifier
feature_selector = SelectFromModel(RandomForestClassifier(n_estimators = 100))

# Fit the selector of the data
feature_selector.fit(cleaned_data, y)
```

Out[88]:

```
SelectFromModel(estimator=RandomForestClassifier())
```

In [89]:

```python
# Get best features
selected_feature = cleaned_data.columns[(feature_selector.get_support())]
```

In [90]:

```
print(f"Only \033[4m\033[1m{len(selected_feature)}\033[0m\033[0m features are select
print(f"\nSelected features are : {list(selected_feature)}")
```

```
Only 10 features are selected from 44

Selected features are : ['age', 'marital', 'education', 'housing', 'duration', 'camp
aign', 'pdays', 'poutcome', 'emp.var.rate', 'euribor3m']
```

In [91]:

```
# Filter dataset with resepect to selected features
X = cleaned_data[selected_feature]
X.head()
```

Out[91]:

	age	marital	education	housing	duration	campaign	pdays	poutcome	emp.var.rate	eu
0	4.025352	0.101561	0.102490	0	3	1	0	0	9	
1	4.043051	0.101561	0.108389	0	2	1	0	0	9	
2	3.610918	0.101561	0.108389	1	3	1	0	0	9	
3	3.688879	0.101561	0.082060	0	2	1	0	0	9	
4	4.025352	0.101561	0.108389	0	3	1	0	0	9	

3.5. Data Transformation

...goto toc

3.5.1 Handling unbalanced target feature (SMOTE)

SMOTE is an oversampling technique that generates synthetic samples from the minority class. It is used to obtain a synthetically class-balanced or nearly class-balanced training set, which is then used to train the classifier.

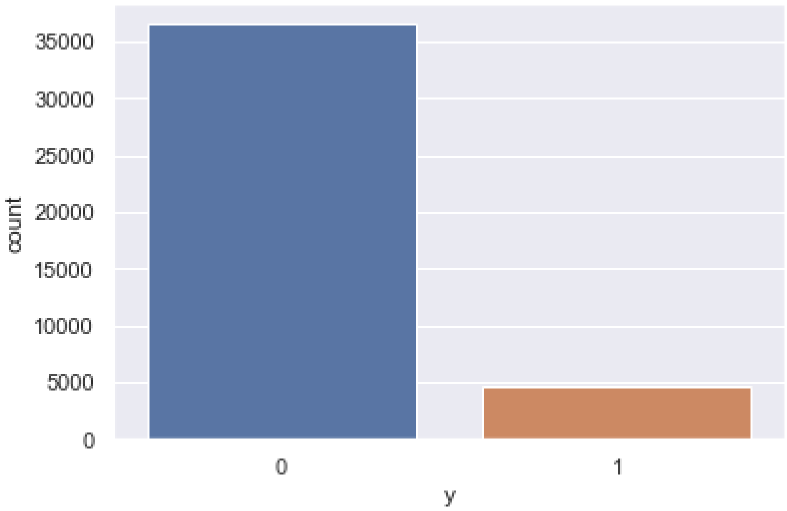
Since there are all together **44** independent features we need to perform feature selection to eliminate curse of dimensionality

...goto toc

In [92]:

```
ax = sns.countplot(x = y,label="Count")
Y, N = y.value_counts()
print('Number of Client subscribed : ', Y)
print('Number of Clients not subscribed  : ', N)
```

```
Number of Client subscribed :  36537
Number of Clients not subscribed  :  4639
```



Note:

As we can see from the plot that data is **highly imbalanced**. And we built model based on this dataset then I will be baised. To avoid this we will apply oversamplying technique **SMOTE**.

In [93]:
```
# Oversample and plot imbalanced dataset with SMOTE
from collections import Counter
from imblearn.over_sampling import SMOTE
```

In [94]:
```
# summarize class distribution
counter = Counter(y)
print(f"Current count of target features: {counter}")
```

```
Current count of target features: Counter({0: 36537, 1: 4639})
```

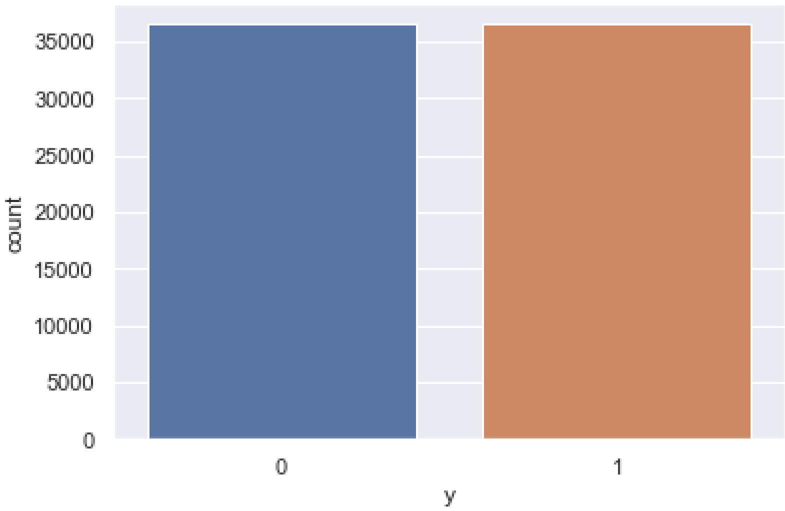
In [95]:
```
# Initalize smote object
oversample = SMOTE()

# Perform fit and resample on target feature
X, y = oversample.fit_resample(X, y)
```

In [96]:
```
counter = Counter(y)
print(f"Count of target feature after resampling : {counter}")
```

```
Count of target feature after resampling : Counter({0: 36537, 1: 36537})
```

In [97]:
```
ax = sns.countplot(x = y, label="Count")
Y, N = y.value_counts()
print('Number of Client subscribed : ', Y)
print('Number of Clients not subscribed  : ', N)
```

```
Number of Client subscribed :  36537
Number of Clients not subscribed  :  36537
```



After applying SMOTE, we can see that target feature is balanced now we can move further

3.5.2 Normalization

Normalization is used to scale the data of an attribute so that it falls in a smaller range, such as -1.0 to 1.0 or 0.0 to 1.0. It is generally useful for classification algorithms.

We will use *Standard Scaler* to perform normalization.

...goto toc

In [98]:

```
# Import the required function
from sklearn.preprocessing import StandardScaler
```

In [100…

```
# Initilize scaler
scaler = StandardScaler()

# fit the scaler
scaler.fit(X)
```

Out[100…

```
StandardScaler()
```

In [101…

```
# Transform the dataset
X = scaler.fit_transform(X)
```

3.5.3 Split dataset

We will be splitting the dataset into train and test set with **70-30** split

...goto toc

In [102…

```
# Import trai test plit function
from sklearn.model_selection import train_test_split
```

In [103…

```
# let us now split the dataset into train & test
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_s

# print the shape of 'x_train'
print("X_train : ",X_train.shape)

# print the shape of 'x_test'
print("X_test : ",X_test.shape)

# print the shape of 'y_train'
print("y_train : ",y_train.shape)

# print the shape of 'y_test'
print("y_test : ",y_test.shape)
```

```
X_train :  (51151, 10)
X_test :  (21923, 10)
y_train :  (51151,)
y_test :  (21923,)
```

4. Model Development

We will be training different classification model and choose the one with best performance

...goto toc

In [113...

```
# Import packages to calculate performance of the models
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

4.1 Logistic Regression

Training a logistic regression classifier

...goto toc

In [104...

```
# Import Logistic regressor
from sklearn.linear_model import LogisticRegression
```

In [105...

```
# Initialize the regressor
logistic = LogisticRegression()
```

In [106...

```
# Fit the model on training set
logistic.fit(X_train,y_train)
```

Out[106...

```
LogisticRegression()
```

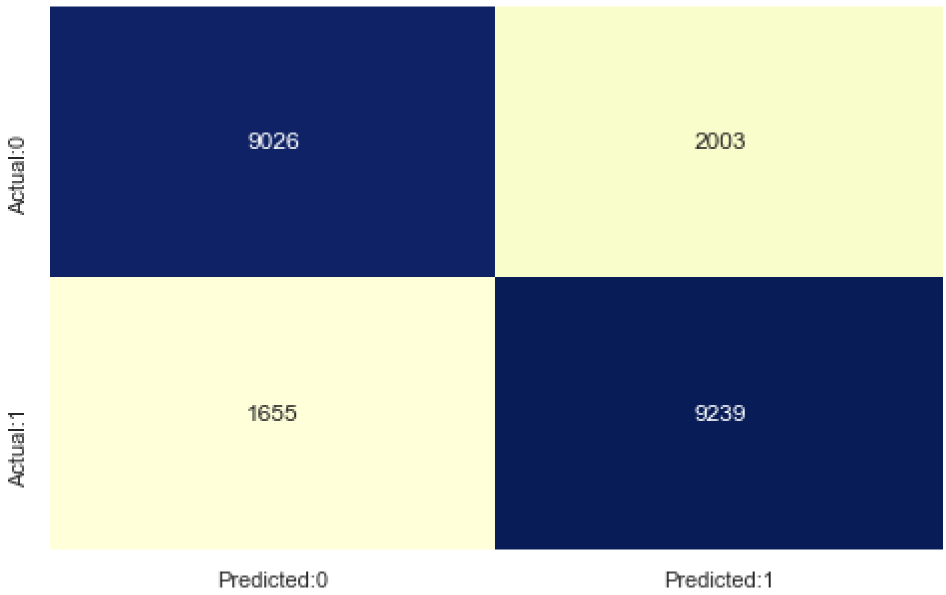
In [107...

```
# predict the values
y_pred = logistic.predict(X_test)
```

In [108...

```
# Compute the accuracy
```

```python
# compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# label the confusion matrix
conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Act

# set sizeof the plot
plt.figure(figsize = (8,5))

# plot a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```



In [109…

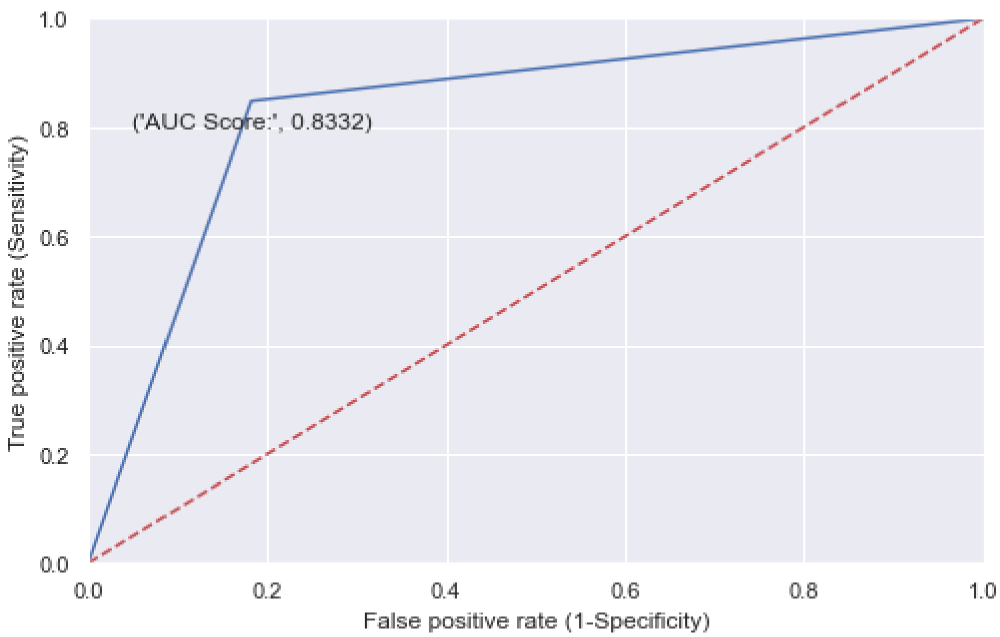
```python
# Generate classiffication report

# accuracy measures by classification_report()
result = classification_report(y_test,y_pred)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.83     11029
           1       0.82      0.85      0.83     10894

    accuracy                           0.83     21923
   macro avg       0.83      0.83      0.83     21923
weighted avg       0.83      0.83      0.83     21923
```

In [111…

```python
# Get and plot roc curve
# set the figure size
plt.rcParams['figure.figsize']=(8,5)

fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# plot the ROC curve
plt.plot(fpr,tpr)

# set limits for x and y axes
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
```

```python
# plot the straight line showing worst prediction for the model
plt.plot([0, 1], [0, 1],'r--')

# add the AUC score
plt.text(x = 0.05, y = 0.8, s =('AUC Score:', round(roc_auc_score(y_test, y_pred),4)

# name the plot, and both axes
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')

# plot the grid
plt.grid(True)
```



In [112…

```python
# Tabulate the result
from sklearn import metrics

# create a list of column names
cols = ['Model', 'AUC Score', 'Precision Score', 'Recall Score','Accuracy Score','f1

# creating an empty dataframe of the colums
result_tabulation = pd.DataFrame(columns = cols)

# compiling the required information
logistic_regression_estimator = pd.Series({'Model': "Logistic Regression",
                     'AUC Score' : metrics.roc_auc_score(y_test, y_pred),
                 'Precision Score': metrics.precision_score(y_test, y_pred),
                 'Recall Score': metrics.recall_score(y_test, y_pred),
                 'Accuracy Score': metrics.accuracy_score(y_test, y_pred),
                  'f1-score':metrics.f1_score(y_test, y_pred)})



# appending our result table
result_tabulation = result_tabulation.append(logistic_regression_estimator , ignore_

# view the result table
result_tabulation
```

Out[112…

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Logistic Regression	0.833235	0.821829	0.848082	0.833143	0.834749

4.2 AdaBoost

...goto toc

In [114...

```
# Import Adaboost classifier
from sklearn.ensemble import AdaBoostClassifier
```

In [115...

```
# build the model
adaboost = AdaBoostClassifier(random_state=10)

# fit the model
adaboost.fit(X_train, y_train)
```

Out[115...

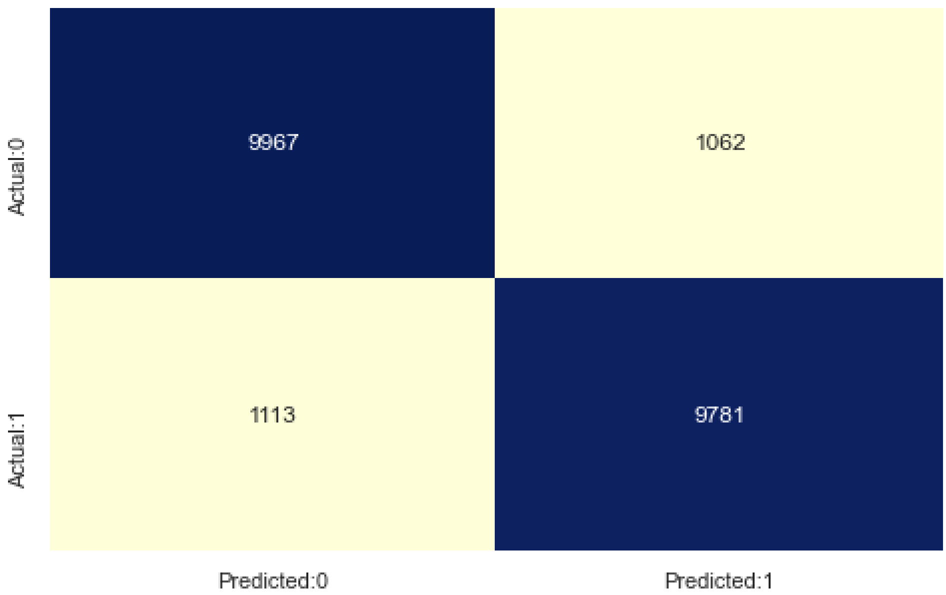
```
AdaBoostClassifier(random_state=10)
```

In [116...

```
# predict the values
y_pred_adaboost  = adaboost.predict(X_test)
```

In [117...

```
# compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_adaboost)

# label the confusion matrix
conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Act

# set sizeof the plot
plt.figure(figsize = (8,5))

# plot a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```



In [118...

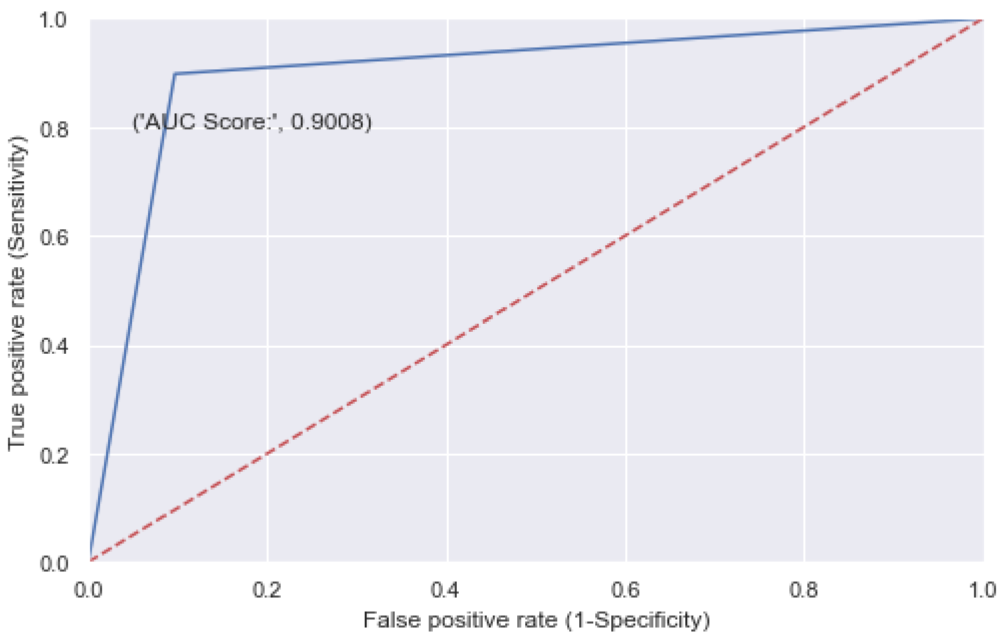
```
# Generate classification report
result = classification_report(y_test, y_pred_adaboost)

# print the result
print(result)
```

```
              precision    recall  f1-score   support
```

0	0.90	0.90	0.90	11029
1	0.90	0.90	0.90	10894
accuracy			0.90	21923
macro avg	0.90	0.90	0.90	21923
weighted avg	0.90	0.90	0.90	21923

In [119…

```
# set the figure size
plt.rcParams['figure.figsize']=(8,5)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_adaboost)

# plot the ROC curve
plt.plot(fpr,tpr)

# set limits for x and y axes
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])

# plot the straight line showing worst prediction for the model
plt.plot([0, 1], [0, 1],'r--')

# add the AUC score
plt.text(x = 0.05, y = 0.8, s =('AUC Score:', round(metrics.roc_auc_score(y_test, y_

# name the plot, and both axes
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')

# plot the grid
plt.grid(True)
```



In [120…

```
# create the result table for all scores
adaboost_metrics = pd.Series({'Model': "AdaBoost",
                     'AUC Score' : metrics.roc_auc_score(y_test, y_pred_adaboost),
                 'Precision Score': metrics.precision_score(y_test, y_pred_adaboost)
                 'Recall Score': metrics.recall_score(y_test, y_pred_adaboost),
                 'Accuracy Score': metrics.accuracy_score(y_test, y_pred_adaboost),
                  'f1-score':metrics.f1_score(y_test, y_pred_adaboost)})
```

```
# appending our result table
result_tabulation = result_tabulation.append(adaboost_metrics , ignore_index = True)

# view the result table
result_tabulation
```

Out[120…

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Logistic Regression	0.833235	0.821829	0.848082	0.833143	0.834749
1	AdaBoost	0.900771	0.902057	0.897834	0.900789	0.899940

4.3 Naive Bayes

...goto toc

In [121…

```
# Import Naive bayes classifier
from sklearn.naive_bayes import GaussianNB

# build the model
GNB = GaussianNB()

# fit the model
GNB.fit(X_train, y_train)
```

Out[121…

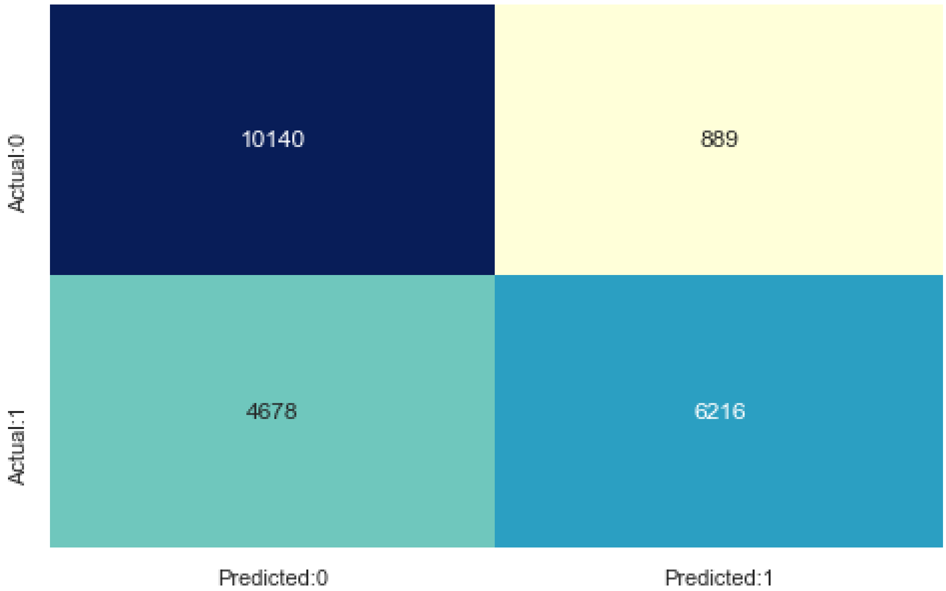
```
GaussianNB()
```

In [122…

```
# predict the values
y_pred_GNB  = GNB.predict(X_test)
```

In [123…

```
# compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_GNB)

# label the confusion matrix
conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Act

# set sizeof the plot
plt.figure(figsize = (8,5))

# plot a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```



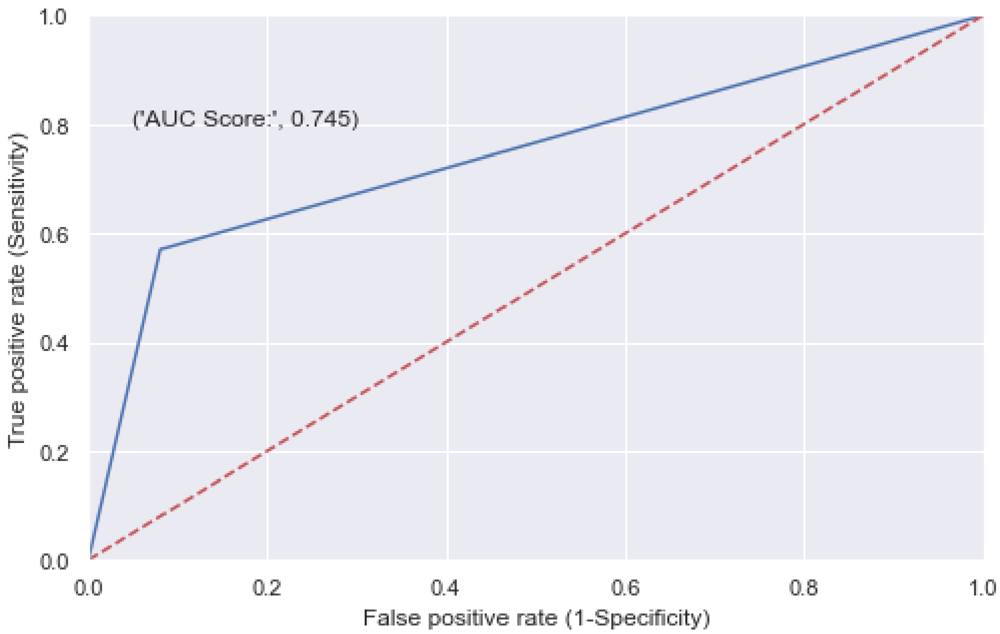
In [124…

```
# Generate classification report
result = classification_report(y_test,y_pred_GNB)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

           0       0.68      0.92      0.78     11029
           1       0.87      0.57      0.69     10894

    accuracy                           0.75     21923
   macro avg       0.78      0.74      0.74     21923
weighted avg       0.78      0.75      0.74     21923
```

In [125…

```
# set the figure size
plt.rcParams['figure.figsize']=(8,5)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_GNB)

# plot the ROC curve
plt.plot(fpr,tpr)

# set limits for x and y axes
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])

# plot the straight line showing worst prediction for the model
plt.plot([0, 1], [0, 1],'r--')

# add the AUC score
plt.text(x = 0.05, y = 0.8, s =('AUC Score:', round(metrics.roc_auc_score(y_test, y_

# name the plot, and both axes
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')

# plot the grid
plt.grid(True)
```



In [126…

```
# create the result table for all scores
GNB_metrices = pd.Series({'Model': "Naive Bayes",
                     'AUC Score' : metrics.roc_auc_score(y_test, y_pred_GNB),
                 'Precision Score': metrics.precision_score(y_test, y_pred_GNB),
                 'Recall Score': metrics.recall_score(y_test, y_pred_GNB),
                 'Accuracy Score': metrics.accuracy_score(y_test, y_pred_GNB),

                  'f1-score':metrics.f1_score(y_test, y_pred_GNB)})



# appending our result table
result_tabulation = result_tabulation.append(GNB_metrices , ignore_index = True)

# view the result table
result_tabulation
```

Out[126…

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Logistic Regression	0.833235	0.821829	0.848082	0.833143	0.834749
1	AdaBoost	0.900771	0.902057	0.897834	0.900789	0.899940
2	Naive Bayes	0.744992	0.874877	0.570589	0.746066	0.690705

4.4 KNN

...goto toc

To find optimal value of **k** we will be performing hyperparameter tuning using **Grid Search Cross Validation**

In [127…

```
# Import KNN classifier
from sklearn.neighbors import KNeighborsClassifier
```

In [129…

```
# Hyperparameter tuning
from sklearn.model_selection import GridSearchCV
```

```python
# Initialize a knn object
knn = KNeighborsClassifier()

# Create a dictionary of all values we want to test for n_neighbors
param_grid = {'n_neighbors': np.arange(2, 6)}
```

In [131…

```python
# Perform gridsearch
knn_gscv = GridSearchCV(knn, param_grid, cv=5)

# fit the data
knn_gscv.fit(X_train, y_train)
```

Out[131…

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([2, 3, 4, 5])})
```

In [132…

```python
# Get the best estimator
knn_gscv.best_estimator_
```

Out[132…

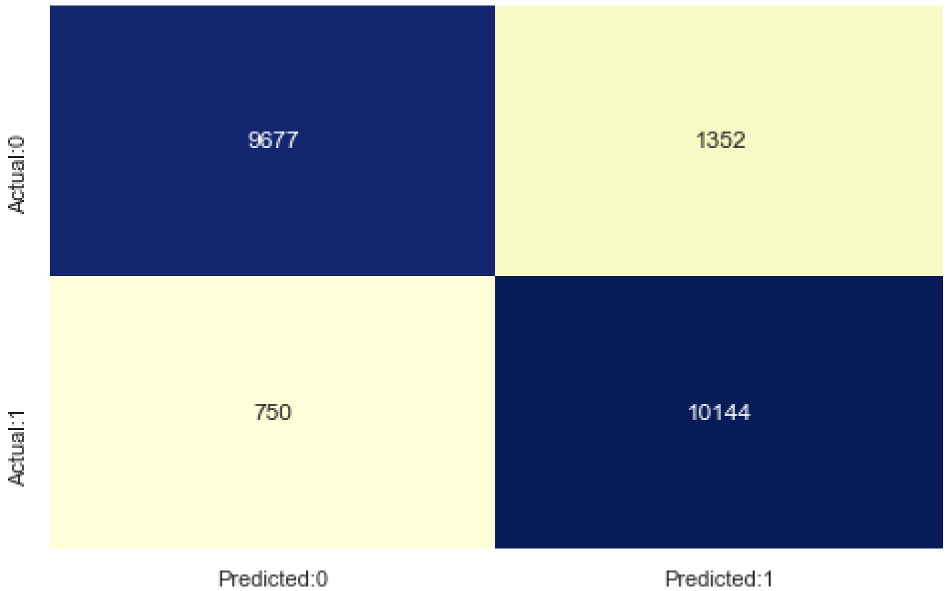
```
KNeighborsClassifier(n_neighbors=3)
```

In [133…

```python
# predict the values
y_pred_knn  = knn_gscv.predict(X_test)
```

In [134…

```python
# compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_knn)

# label the confusion matrix
conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Act

# set sizeof the plot
plt.figure(figsize = (8,5))

# plot a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```



In [135…

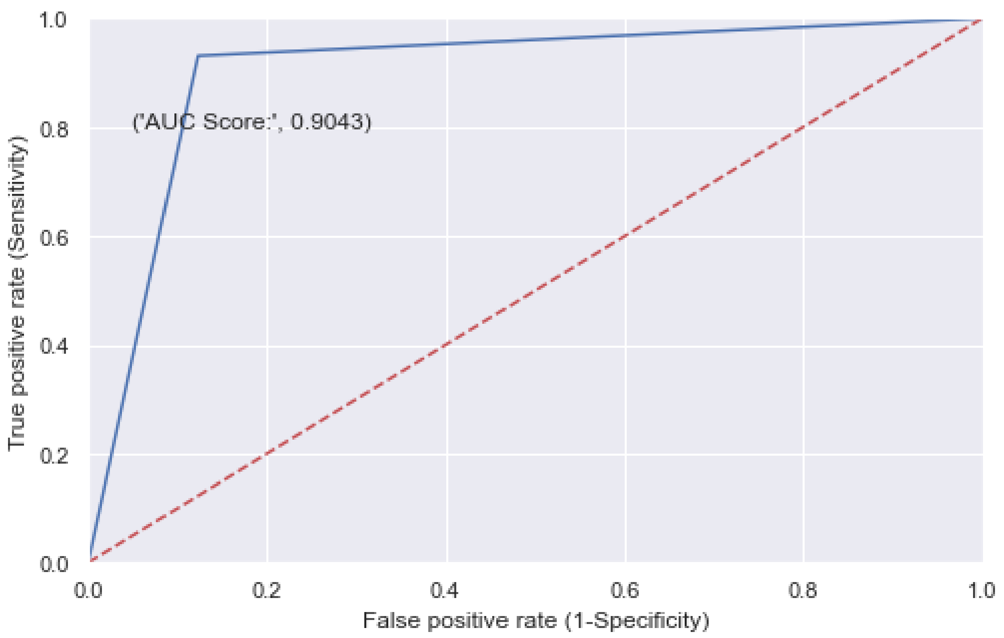
```python
# Generate classification_report
result = classification_report(y_test, y_pred_knn)
```

```
# print the result
print(result)
```

```
              precision    recall  f1-score   support

           0       0.93      0.88      0.90     11029
           1       0.88      0.93      0.91     10894

    accuracy                           0.90     21923
   macro avg       0.91      0.90      0.90     21923
weighted avg       0.91      0.90      0.90     21923
```

In [136…

```
# set the figure size
plt.rcParams['figure.figsize']=(8,5)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_knn)

# plot the ROC curve
plt.plot(fpr,tpr)

# set limits for x and y axes
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])

# plot the straight line showing worst prediction for the model
plt.plot([0, 1], [0, 1],'r--')

# add the AUC score
plt.text(x = 0.05, y = 0.8, s =('AUC Score:', round(metrics.roc_auc_score(y_test, y_

# name the plot, and both axes
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')

# plot the grid
plt.grid(True)
```



In [137…

```
# create the result table for all scores
knn_metrics = pd.Series({'Model': "KNN",
                     'AUC Score' : metrics.roc_auc_score(y_test, y_pred_knn),
                 'Precision Score': metrics.precision_score(y_test, y_pred_knn),
                 'Recall Score': metrics.recall_score(y_test, y_pred_knn),
```

```
                    'Accuracy Score': metrics.accuracy_score(y_test, y_pred_knn),
                     'f1-score':metrics.f1_score(y_test, y_pred_knn)})



# appending our result table
result_tabulation = result_tabulation.append(knn_metrics , ignore_index = True)

# view the result table
result_tabulation
```

Out[137…

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Logistic Regression	0.833235	0.821829	0.848082	0.833143	0.834749
1	AdaBoost	0.900771	0.902057	0.897834	0.900789	0.899940
2	Naive Bayes	0.744992	0.874877	0.570589	0.746066	0.690705
3	KNN	0.904284	0.882394	0.931155	0.904119	0.906119

4.5 Support Vector Machine

...goto toc

In [139…

```
# Import Support Vector Machine class
from sklearn.svm import SVC

# Initialize svm and kernel as linear
svclassifier = SVC(kernel = 'linear')

# fit the model
svclassifier.fit(X_train, y_train)
```

Out[139…

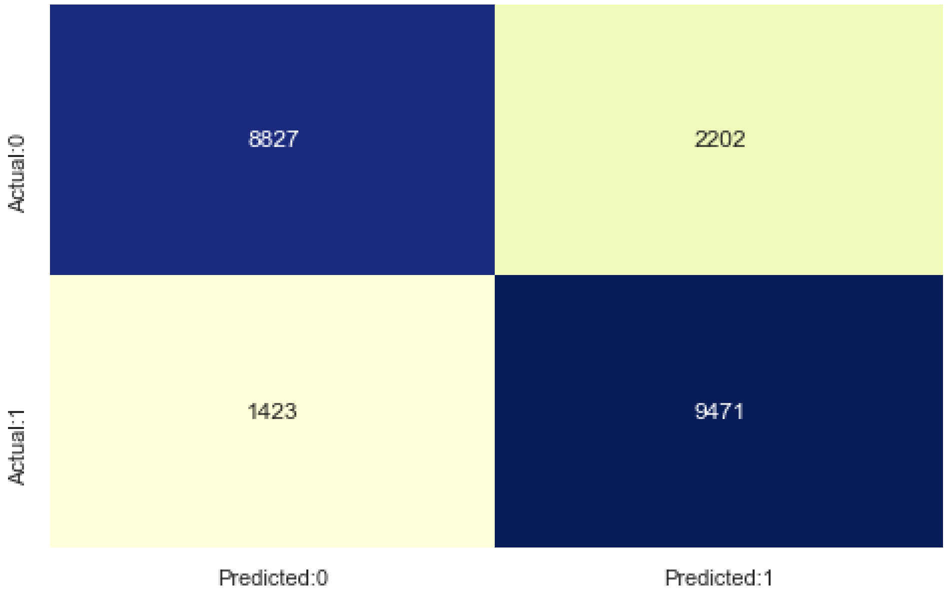
```
SVC(kernel='linear')
```

In [140…

```
# predict the values
y_pred_SVC  = svclassifier.predict(X_test)
```

In [141…

```
# compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_SVC)

# label the confusion matrix
conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Act

# set sizeof the plot
plt.figure(figsize = (8,5))

# plot a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```



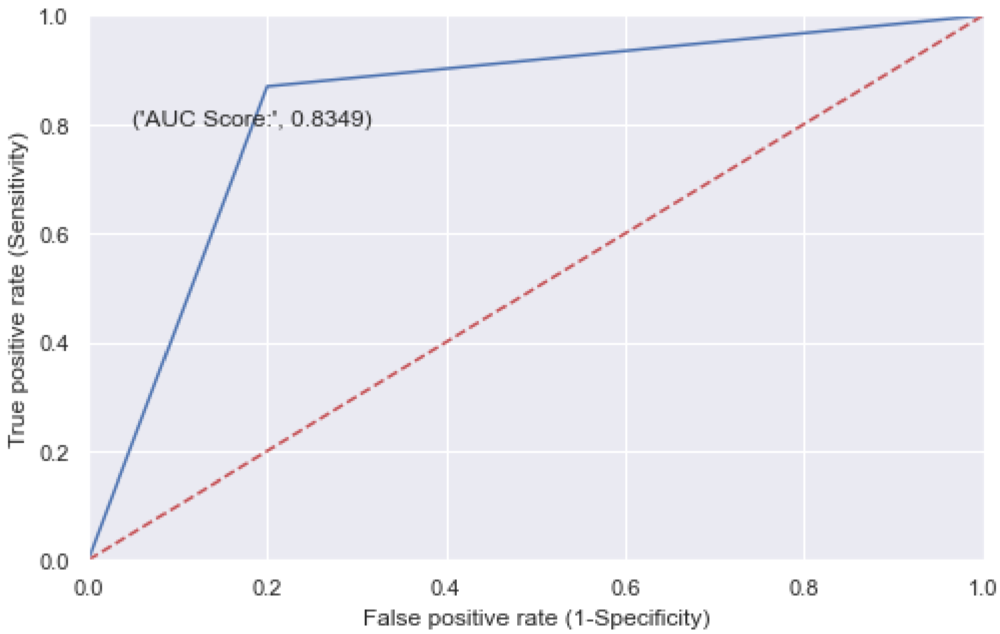
In [142…

```
# Generate classification_report
result = classification_report(y_test,y_pred_SVC)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

           0       0.86      0.80      0.83     11029
           1       0.81      0.87      0.84     10894

    accuracy                           0.83     21923
   macro avg       0.84      0.83      0.83     21923
weighted avg       0.84      0.83      0.83     21923
```

In [143…

```
# set the figure size
plt.rcParams['figure.figsize']=(8,5)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_SVC)

# plot the ROC curve
plt.plot(fpr,tpr)

# set limits for x and y axes
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])

# plot the straight line showing worst prediction for the model
plt.plot([0, 1], [0, 1],'r--')

# add the AUC score
plt.text(x = 0.05, y = 0.8, s =('AUC Score:', round(metrics.roc_auc_score(y_test, y_

# name the plot, and both axes
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')

# plot the grid
plt.grid(True)
```



In [144…

```
# create the result table for all scores
SVM_metrices = pd.Series({'Model': "Support Vector MAchine",
                     'AUC Score' : metrics.roc_auc_score(y_test, y_pred_SVC),
                 'Precision Score': metrics.precision_score(y_test, y_pred_SVC),
                 'Recall Score': metrics.recall_score(y_test, y_pred_SVC),
                 'Accuracy Score': metrics.accuracy_score(y_test, y_pred_SVC),

                  'f1-score':metrics.f1_score(y_test, y_pred_SVC)})



# appending our result table
result_tabulation = result_tabulation.append(SVM_metrices , ignore_index = True)

# view the result table
result_tabulation
```

Out[144…

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Logistic Regression	0.833235	0.821829	0.848082	0.833143	0.834749
1	AdaBoost	0.900771	0.902057	0.897834	0.900789	0.899940
2	Naive Bayes	0.744992	0.874877	0.570589	0.746066	0.690705
3	KNN	0.904284	0.882394	0.931155	0.904119	0.906119
4	Support Vector MAchine	0.834861	0.811360	0.869378	0.834649	0.839367

5. Model Comparision

...goto toc

In [145…

```
result_tabulation
```

Out[145…

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Logistic Regression	0.833235	0.821829	0.848082	0.833143	0.834749

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
1	AdaBoost	0.900771	0.902057	0.897834	0.900789	0.899940
2	Naive Bayes	0.744992	0.874877	0.570589	0.746066	0.690705
3	KNN	0.904284	0.882394	0.931155	0.904119	0.906119
4	Support Vector MAchine	0.834861	0.811360	0.869378	0.834649	0.839367

Best Model

Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
KNN	0.904284	0.882394	0.931155	0.904119	0.906119

In [147…

```
best_model = knn_gscv
```

Save the model

In [146…

```
import pickle
```

In [148…

```
pickle.dump(best_model, open("place_deposit.sav", "wb"))
```

In [ ]:

```

```