Name : *Saurabh Mirchandani*
Roll No. : *20BCM073*
Subject : *Data Mining*
Subject Code : *CSI0803*
Assignment- 1
Date : *27-12-2022*

Assignment-1

Aim:- Preprocessing of dataset

Banking Dataset - Marketing Targets

About Dataset

Context

Term deposits are a major source of income for a bank. A term deposit is a cash investment held at a financial institution. Your money is invested for an agreed rate of interest over a fixed amount of time, or term. The bank has various outreach plans to sell term deposits to their customers such as email marketing, advertisements, telephonic marketing, and digital marketing.

Telephonic marketing campaigns still remain one of the most effective way to reach out to people. However, they require huge investment as large call centers are hired to actually execute these campaigns. Hence, it is crucial to identify the customers most likely to convert beforehand so that they can be specifically targeted via call.

The data is related to direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe to a term deposit (variable y).

Content

The data is related to the 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 by the customer or not. The data folder contains two datasets:-

- train.csv: 45,211 rows and 18 columns ordered by date (from May 2008 to November 2010)
- test.csv: 4521 rows and 18 columns with 10% of the examples (4521), randomly selected from train.csv

Data Pre-processing and analysis

DESCRIPTION OF THE TASKS

CONTEXT (based from what's written in the dataset):

Term deposits are a major source of income for a bank. A term deposit is a cash investment held at a financial institution. Your money is invested for an agreed rate of interest over a fixed amount of time, or term. The bank has various outreach plans to sell term deposits to their customers such as email marketing, advertisements, telephonic marketing, and digital marketing.

Telephonic marketing campaigns still remain one of the most effective way to reach out to people. However, they require huge investment as large call centers are hired to actually execute these campaigns. Hence, it is crucial to identify the customers most likely to convert beforehand so that they can be specifically targeted via call.

TASK/S:

The classification goal is to predict if the client will subscribe to a term deposit.

DATASET:

The data is related to direct marketing campaigns (phone calls) 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 subscribed ('yes') or not ('no') subscribed by the customer or not. The data folder contains two datasets:

- train.csv: 45,211 rows and 18 columns ordered by date (from May 2008 to November 2010)
- test.csv: 4,521 rows and 18 columns with 10% of the examples (4,521), randomly selected from train.csv

Specifically, these are the following features within the datasets:

1. age (numerical)
2. job: type of job (categorical: "admin.", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")
3. marital: marital status (categorical: "married", "divorced", "single"; note: "divorced" means divorced or widowed)
4. education (categorical: "unknown", "secondary", "primary", "tertiary")
5. default: has credit in default? (categorical: "yes", "no")
6. balance: average yearly balance, in euros (numerical)
7. housing: has housing loan? (categorical: "yes", "no")
8. loan: has personal loan? (categorical: "yes", "no")
9. contact: contact communication type (categorical: "unknown", "telephone", "cellular")
10. day: last contact day of the month (numerical)

11. month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
12. duration: last contact duration, in seconds (numerical)
13. campaign: number of contacts performed during this campaign and for this client; includes last contact (numerical)
14. pdays: number of days that passed by after the client was last contacted from a previous campaign; -1 means client was not previously contacted (numerical)
15. previous: number of contacts performed before this campaign and for this client (numerical)
16. poutcome: outcome of the previous marketing campaign (categorical: "unknown", "other", "failure", "success").
17. y **(TARGET)** - has the client subscribed a term deposit? (categorical: "yes", "no")

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-py
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # data visualization
import matplotlib.pyplot as plt # data visualization

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all fi

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets prese
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside
```

```
/kaggle/input/banking-dataset-marketing-targets/train.csv
/kaggle/input/banking-dataset-marketing-targets/test.csv
```

0. Importing other necessary packages

In [2]:
```
# Encoding
from sklearn import preprocessing

# Splitting and scaling
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler

# Balancing data
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTEENN
from imblearn.combine import SMOTETomek

# Estimators
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb

# Pipelines
from imblearn.pipeline import Pipeline

# Stratified K-Fold
from sklearn.model_selection import StratifiedKFold

# Metrics and cross validation
from sklearn.metrics import confusion_matrix, classification_report
```

```
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, r
from sklearn.metrics import roc_curve, auc
```

I. DATA CLEANING

Loading data

In [3]:
```
df = pd.read_csv('../input/banking-dataset-marketing-targets/train.csv', sep = ';')
```

In [4]:
```
# Checking if data was loaded properly
df.head(3)
```

Out[4]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	du
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	

In [5]:
```
df.tail(3)
```

Out[5]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month
45208	72	retired	married	secondary	no	5715	no	no	cellular	17	no
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	no
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	no

In [6]:
```
# Checking the summary of our main dataset (train.csv)
df.info()
```

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

In [7]:
```
# Checking summary statistics for numerical features
df.describe().T
```

Out[7]:

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

We can see here that the minimum value for `duration`, which talks about the duration of the last contact, is `0`. However, the minimum value for `campaign`, which is the number of contacts made during this campaign (including the last contact), is `1`. This might indicate that the telephone call was dropped after connecting if these were a small amount of datapoints.

In [8]:
```
# Further checking the number of datapoints that had 0 seconds for 'duration'
print("The number of clients that had a call duration of 0:", df[df['duration'] == 0].
```

```
The number of clients that had a call duration of 0: 3
```

This can indicate the possibility that the phone call was instantly dropped, which can happen.

Additionally, we verify the validity of the features `previous` and `pdays`. It should be noted that all clients with `pdays = -1` indicates that the client was not previously contacted, which implies that `previous = 0`.

In [9]:
```
# Verifying 'previous' and 'pdays'
df[df['pdays'] == -1].previous.sum()
```

Out[9]:
```
0
```

The `previous` and `pdays` features check out.

In [10]:
```
# Checking summary statistics for categorical features
df.describe(include = 'object').T
```

Out[10]:

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

Checking missing values

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

There are no missing values

Checking for duplicates

In [12]:
```
df.duplicated().value_counts()
```

Out[12]:
```
False    45211
dtype: int64
```

There are no duplicated data.

Checking misspelled columns / improving column names

In [13]:
```
df.columns
```

Out[13]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

The column names are okay, but they can be improved for clarity. We can change them into the following:

- `marital` > `marital_status`
- `education` > `educational_attainment`
- `default` > `credit_default`
- `balance` > `yearly_balance_ave`
- `housing` > `housing_loan`
- `loan` > `personal_loan`
- `contact` > `contact_type`

- `day` > `last_contact_day`
- `month` > `last_contact_month`
- `duration` > `last_contact_duration`
- `campaign` > `current_contact_count`
- `pdays` > `previous_idle_days`
- `previous` > `previous_contact_count`
- `poutcome` > `previous_outcome`
- `y` > `current_outcome`

In [14]:
```
# Renaming columns
df.rename(columns = {'marital':'marital_status', 'education':'educational_attainment',
                     'default':'credit_default', 'balance':'yearly_balance_ave',
                     'housing':'housing_loan', 'loan':'personal_loan',
                     'contact':'contact_type', 'day':'last_contact_day',
                     'month':'last_contact_month', 'duration':'last_contact_duration',
                     'campaign':'current_contact_count', 'pdays':'previous_idle_days',
                     'previous':'previous_contact_count', 'poutcome':'previous_outcome
                     'y':'current_outcome'}, inplace = True)

#df_test.rename(columns = {'marital':'marital_status', 'education':'educational_attain
#                          'default':'credit_default', 'balance':'yearly_balance_ave',
#                          'housing':'housing_loan', 'loan':'personal_loan',
#                          'contact':'contact_type', 'day':'last_contact_day',
#                          'month':'last_contact_month', 'duration':'last_contact_dura
#                          'campaign':'current_contact_count', 'pdays':'previous_idle_
#                          'previous':'previous_contact_count', 'poutcome':'previous_o
#                          'y':'current_outcome'}, inplace = True)

# Checking updated column names
df.columns
```

Out[14]:
```
Index(['age', 'job', 'marital_status', 'educational_attainment',
       'credit_default', 'yearly_balance_ave', 'housing_loan', 'personal_loan',
       'contact_type', 'last_contact_day', 'last_contact_month',
       'last_contact_duration', 'current_contact_count', 'previous_idle_days',
       'previous_contact_count', 'previous_outcome', 'current_outcome'],
      dtype='object')
```

Checking misspelled column values

In [15]:
```
df_cat = df.select_dtypes('object').columns
for col in df_cat:
    print(df[col].value_counts())
    print('===============')
```

```
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264
self-employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student           938
unknown           288
Name: job, dtype: int64
===============
married     27214
single      12790
divorced     5207
Name: marital_status, dtype: int64
===============
secondary    23202
tertiary     13301
primary       6851
unknown       1857
Name: educational_attainment, dtype: int64
===============
no     44396
yes      815
Name: credit_default, dtype: int64
===============
yes    25130
no     20081
Name: housing_loan, dtype: int64
===============
no     37967
yes     7244
Name: personal_loan, dtype: int64
===============
cellular     29285
unknown      13020
telephone     2906
Name: contact_type, dtype: int64
===============
may    13766
jul     6895
aug     6247
jun     5341
nov     3970
apr     2932
feb     2649
jan     1403
oct      738
sep      579
mar      477
dec      214
Name: last_contact_month, dtype: int64
===============
unknown    36959
```

```
failure     4901
other       1840
success     1511
Name: previous_outcome, dtype: int64
===============
no     39922
yes     5289
Name: current_outcome, dtype: int64
===============
```

There are no misspelled values for our categorical features. However, there are `unknown` values for various features, but we'll consider them as is for this version of the dataset.

Checking column data types

In [16]:
```
df.info()
```

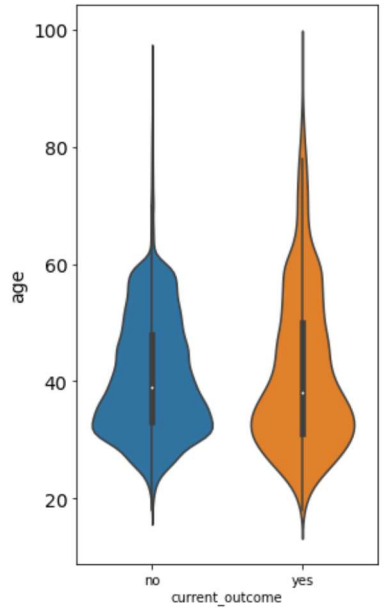
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   age                     45211 non-null  int64 
 1   job                     45211 non-null  object
 2   marital_status          45211 non-null  object
 3   educational_attainment  45211 non-null  object
 4   credit_default          45211 non-null  object
 5   yearly_balance_ave      45211 non-null  int64 
 6   housing_loan            45211 non-null  object
 7   personal_loan           45211 non-null  object
 8   contact_type            45211 non-null  object
 9   last_contact_day        45211 non-null  int64 
 10  last_contact_month      45211 non-null  object
 11  last_contact_duration   45211 non-null  int64 
 12  current_contact_count   45211 non-null  int64 
 13  previous_idle_days      45211 non-null  int64 
 14  previous_contact_count  45211 non-null  int64 
 15  previous_outcome        45211 non-null  object
 16  current_outcome         45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

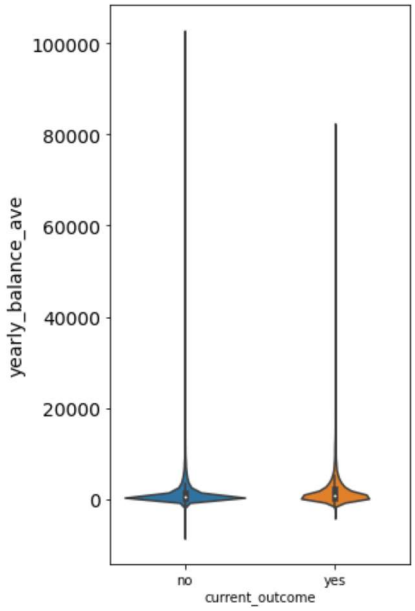
All columns have proper data types.

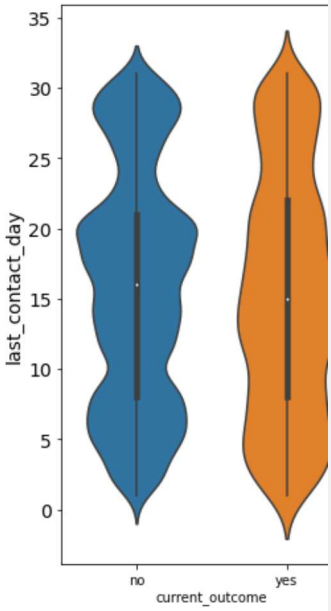
Checking outliers

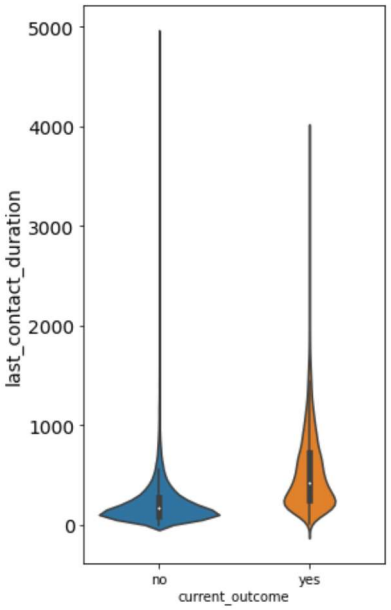
In [17]:
```
df_num = df.select_dtypes('int').columns
plt.figure(figsize = (15, 25))
for idx, col in enumerate(df_num):
    plt.subplot(3, 3, idx + 1)
    if col == 'previous_idle_days':
        ax = sns.violinplot(data = df, y = df[df[col] > -1][col], x = df.current_outcc
    elif col == 'previous_contact_count':
        ax = sns.violinplot(data = df, y = df[df[col] > 0][col], x = df.current_outcom
```

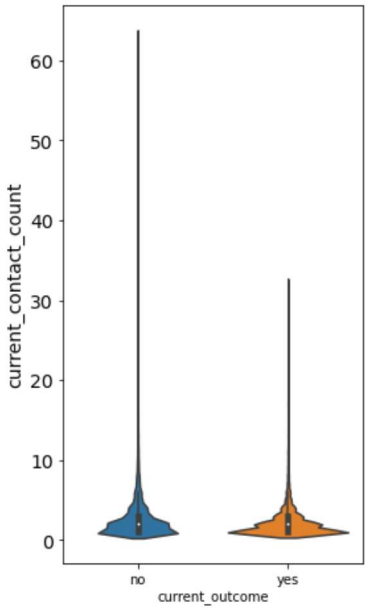
```
    else:
        ax = sns.violinplot(data = df, y = df[col], x = df.current_outcome)
    #plt.axhline(df[col].mean(), color='red', linewidth=3)
    #plt.axhline(df[col].median(), color='green', linewidth=3)
    plt.ylabel(col, fontsize = 14)
    plt.yticks(fontsize = 14)
    plt.subplots_adjust(left=0.1,
                        bottom=0.1,
                        right=0.9,
                        top=0.9,
                        wspace=0.4,
                        hspace=0.2)
```

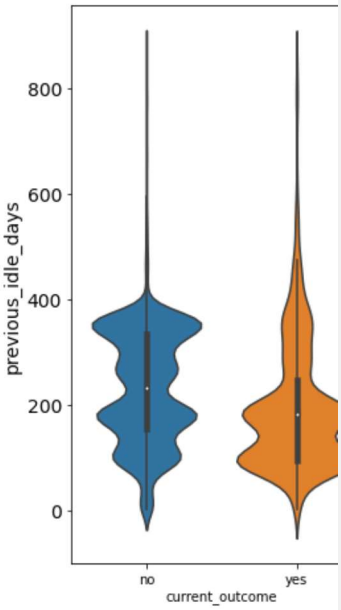

age
100
80
60
40
20
no
yes
current_outcome

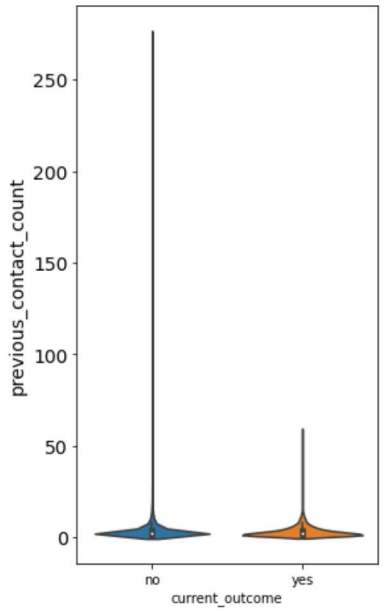

yearly_balance_ave
100000
80000
60000
40000
20000
0
no
yes
current_outcome


last_contact_day
35
30
25
20
15
10
5
0
no
yes
current_outcome


last_contact_duration
5000
4000
3000
2000
1000
0
no
yes
current_outcome


current_contact_count
60
50
40
30
20
10
0
no
yes
current_outcome


previous_idle_days
800
600
400
200
0
no
yes
current_outcome


previous_contact_count
250
200
150
100
50
0
no
yes
current_outcome

It can already be observed that clients that subscribed to a term deposit:

- had `longer durations` for their `last contact` during the `current campaign`, and
- had `shorter number of days that passed` by after the client was `last contacted from a previous campaign`.

In [18]:
```
#Can remove outliers using the IQR method where any value below Q1-1.5*IQR or above Q3

def outlier_density(df,col_name):
    #print("Orig DF Size:"+ str(df.shape))
    if col_name == 'previous_idle_days':
        Q1 = np.quantile(df[df[col_name] > -1][col_name],0.25)
        Q3 = np.quantile(df[df[col_name] > -1][col_name],0.75)
    elif col_name == 'previous_contact_count':
        Q1 = np.quantile(df[df[col_name] > 0][col_name],0.25)
        Q3 = np.quantile(df[df[col_name] > 0][col_name],0.75)
    else:
        Q1 = np.quantile(df[col_name],0.25)
        Q3 = np.quantile(df[col_name],0.75)

    IQR = Q3 - Q1

    lower_limit = Q1 - (1.5*IQR)
    upper_limit = Q3 + (1.5*IQR)

    print("Lower Limit: %.2f" % lower_limit)
    print("Upper Limit: %.2f" % upper_limit)

    df_new = df[(df[col_name] > lower_limit) & (df[col_name] < upper_limit)]
    #print("New DF Size:"+ str(df_new.shape))
    print('Outlier count for ' + col_name + ': ' + str(df.shape[0] - df_new.shape[0]))
    print('Outlier density for ' + col_name + ': ' + str(round((1 - df_new.shape[0] /
    print('========')
```

In [19]:
```
for col in df_num:
    outlier_density(df, col)
```

```
Lower Limit: 10.50
Upper Limit: 70.50
Outlier count for age: 487
Outlier density for age: 1.0772%
========
Lower Limit: -1962.00
Upper Limit: 3462.00
Outlier count for yearly_balance_ave: 4731
Outlier density for yearly_balance_ave: 10.4643%
========
Lower Limit: -11.50
Upper Limit: 40.50
Outlier count for last_contact_day: 0
Outlier density for last_contact_day: 0.0%
========
Lower Limit: -221.00
Upper Limit: 643.00
Outlier count for last_contact_duration: 3247
Outlier density for last_contact_duration: 7.1819%
========
Lower Limit: -2.00
Upper Limit: 6.00
Outlier count for current_contact_count: 4355
Outlier density for current_contact_count: 9.6326%
========
Lower Limit: -158.00
Upper Limit: 618.00
Outlier count for previous_idle_days: 49
Outlier density for previous_idle_days: 0.1084%
========
Lower Limit: -3.50
Upper Limit: 8.50
Outlier count for previous_contact_count: 453
Outlier density for previous_contact_count: 1.002%
========
```

The `yearly_balance_ave`, `last_contact_duration`, and `current_contact_count` have a good amount of outliers. These outliers will not be removed for this version of the dataset, but further analysis will be done to see if these outliers can be removed in succeeding versions.

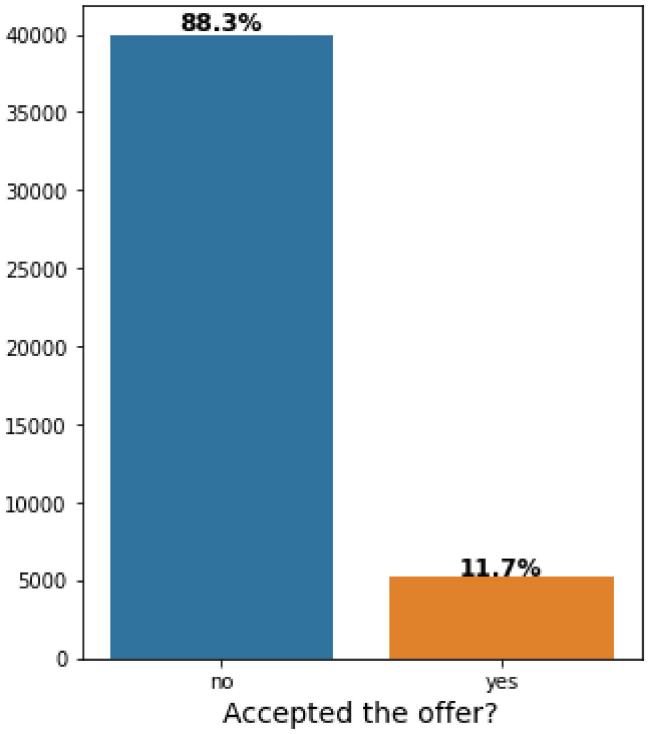
Exploration of these outliers will be done in the next Section II.

II. EXPLORATORY DATA ANALYSIS

Balance of target `y`

In [20]:
```
plt.figure(figsize=(5,6))
ax = sns.countplot(data = df, x = df.current_outcome)
for p in ax.patches:
    width = p.get_width()
    height = p.get_height()
```

```
    x, y = p.get_xy()
    ax.annotate(f'{round(height * 100 / df.shape[0], 2)}%', (x + width/2, y + height *
plt.xlabel(r'Accepted the offer?', fontsize = 14)
plt.ylabel(r'')
plt.show()
```



In [22]:
```
df[df['previous_idle_days'] == -1]['previous_outcome'].value_counts()
```

Out[22]:
```
unknown    36954
Name: previous_outcome, dtype: int64
```

In [23]:
```
# Creating a was_contacted_previously feature based on the previous_idle_days feature
df['was_contacted_previously'] = np.where(df['previous_idle_days'] > -1 , '1', '0')
```

Some observations:

- `Clients` the have ages `around 60 and above` have `lesser contact counts` during the `current campaign`.
- `Generally in all ages`, `clients` that `accepted to pay a term deposit` tend to have `lower current campaign contacts`.
- `Clients` of `lower yearly average balance` got `contacted more`.

In [27]:
```
df_cat = df.select_dtypes('object')
df_cat.drop('current_outcome', axis = 1, inplace = True)
```

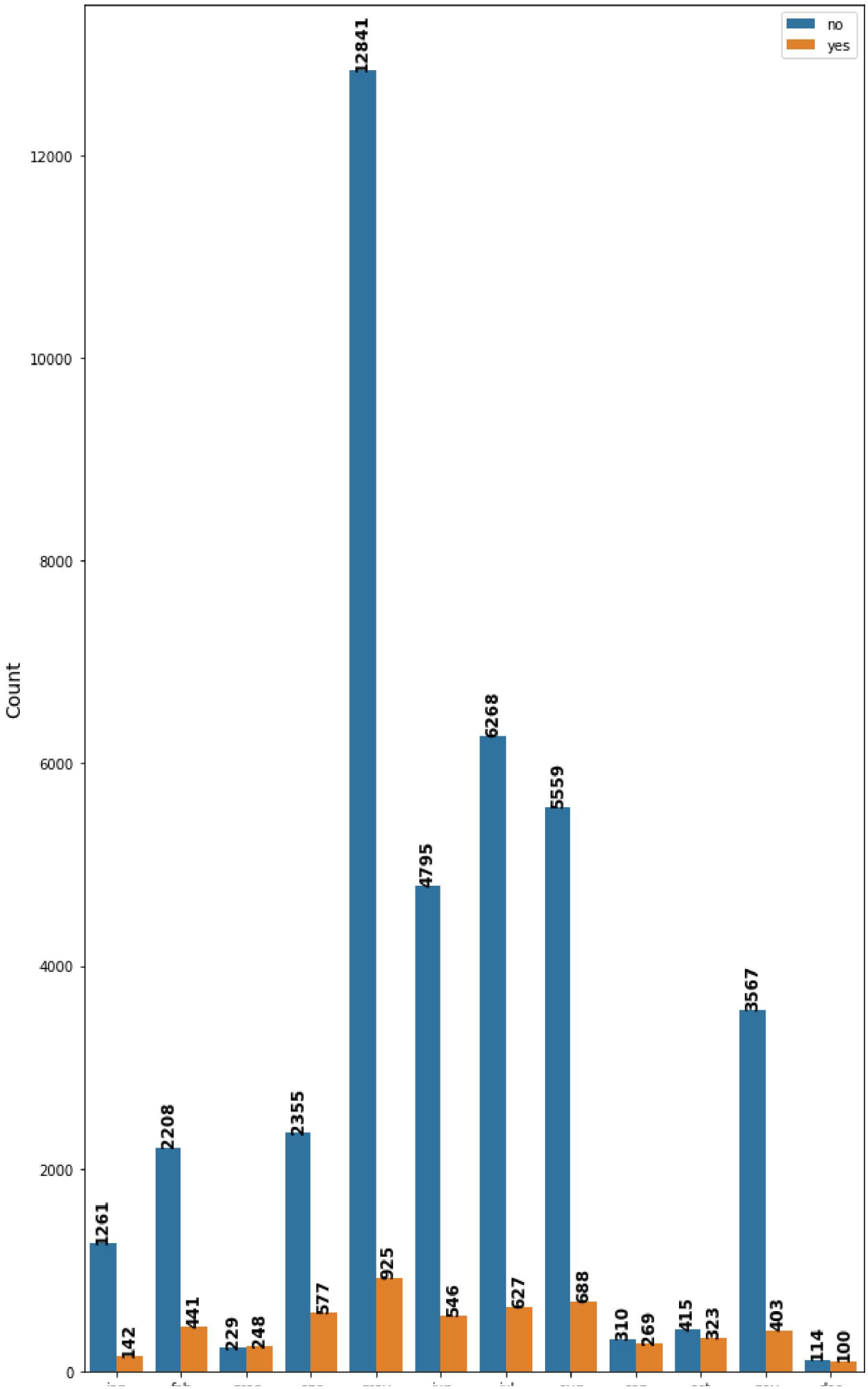
```
/opt/conda/lib/python3.7/site-packages/pandas/core/frame.py:4913: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

Some observations:

- `Most calls` attend to clients with `blue-collar jobs`, but with `most successes` being in clients with `managerial roles`.
- `Most calls` attend to `married` clients, with the `most number of successes as well`. However, the `best ratio of successes` relative to the total number of client belongs to `single` clients.
- `Most calls` attend to clients that finished `secondary education`, with the `most number of successes as well`. However, the `best ratio of successes` relative to the total number of client belongs to `clients that finished college`.
- `Most clients` have `default credit`.
- `More calls` attend to clients `with housing loans`, however `more successes` in terms of number and ratio relative to the total number of clients come from `clients without housing loans`.
- `More calls` attend to clients `without personal loans`, however `more successes` in terms of number and ratio relative to the total number of clients come from `clients with personal loans`.
- `Most calls` are through `cellular contact type`. There are also a `lot of calls` made from `unknown` contact types.
- `Most calls` were made `during summer to fall` (May to August). However, the `best ratio` of successful term deposits in terms of the number of clients within a particular month belong to `October, September, December, and March`. `Maybe` these dates correspond to certain times where `clients are more accessible` / there are `certain situations` where these dates let `clients accept to pay term deposits more`.
- `Most clients` were not previously contacted, but `more clients` that `paid` a term deposit during the `previous` campaign `also paid` for a term deposit `during the current campaign`.
- `A lot more clients` were `not contacted previously`, and these clients `tend to reject the current campaign` as well. However, a `better ratio` of successful term deposits came from `clients who were contacted previously`.

Just a better representation of the `last_contact_month` feature:

In [29]:
```
plt.figure(figsize=(10,18))
ax = sns.countplot(data = df, x = df['last_contact_month'], hue = df['current_outcome'
                   order = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 's
                            'oct', 'nov', 'dec'])
plt.xlabel('Last contact month', fontsize = 14)
plt.ylabel('Count', fontsize = 14)
for col in ax.containers:     #to set a label on top of the bars.
    ax.bar_label(col, weight = 'bold', fontsize = 12, rotation = 90)
plt.legend(loc = 'upper right')
plt.show()
```


no
yes
Count
0
2000
4000
6000
8000
10000
12000
1261
142
2208
441
229
248
2355
577
12841
925
4795
546
6268
627
5559
688
310
269
415
323
3567
403
114
100

The following features will undergo labeled encoding: `job`, `educational_attainment`, `last_contact_month`, `last_contact_quarter`, and `current_outcome` before we undergo OHE.

In [34]:
```
# We can also check for the correlations of feature pairs using the function below
def correlated_pairs(df, threshold):
    # Create correlation matrix
    corr_matrix = df.corr().abs()

    # Reshape the Matrix
    correlated = corr_matrix.unstack()

    # Reset index from multi-index to single index
    correlated = correlated.reset_index(level = 0).reset_index()

    # Rename Columns
    correlated.columns = ["Feature1", "Feature2", "Correlation"]

    # sort by correlation value
    corr_sorted = correlated.sort_values("Correlation", ascending = False)

    # remove self-correlated rows
    corr_sorted_pairs = corr_sorted[corr_sorted['Feature1'].values != corr_sorted['Fea

    # reset index to easily filter out duplicate rows
    corr_sorted_pairs.reset_index(drop = True, inplace = True)

    # skip odd rows to remove redundant pairs
    corr_sorted_final = corr_sorted_pairs.iloc[::2]

    # select rows based on threshold
    corr_sorted_final_ver = corr_sorted_final[corr_sorted_final.Correlation > threshol

    return corr_sorted_final_ver
```

In [35]:
```
# Obtaining highest correlated pairs
corr_sorted_final_ver = correlated_pairs(df, 0)
corr_sorted_final_ver.head()
```

Out[35]:

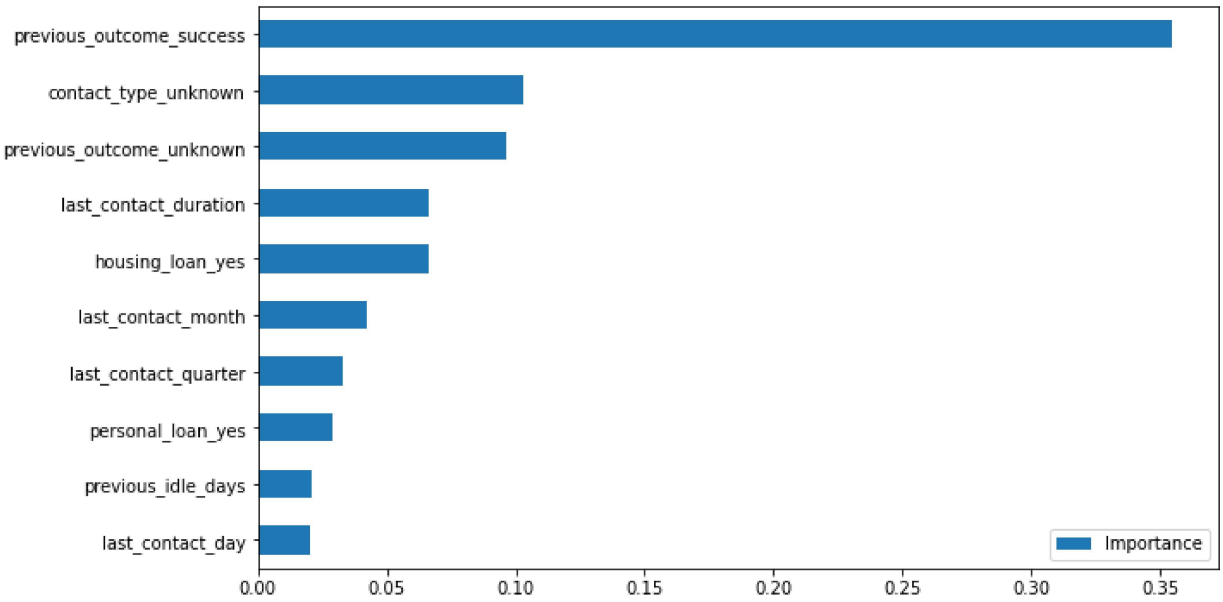
	Feature1	Feature2	Correlation
0	previous_outcome_unknown	was_contacted_previously_1	0.999630
2	was_contacted_previously_1	previous_idle_days	0.870442
4	previous_idle_days	previous_outcome_unknown	0.869962
6	marital_status_married	marital_status_single	0.772357
8	previous_contact_count	was_contacted_previously_1	0.532988

In [39]:
```
# Separating our features and target
features = df.drop('current_outcome', axis = 1)
```

```
target = df['current_outcome']
```

Feature importances

In [53]:
```
# Extracting the feature importance
f_importance = xgbm_pipe_sm.steps[2][1].feature_importances_

# Putting into a DataFrame along with corresponding features.
f_list = features.columns
df_feature_importance = pd.DataFrame(f_importance, index=f_list, columns=["Importance"

# Showing the 10 most important features
df_feature_importance_asc = df_feature_importance.sort_values(["Importance"],ascending

#Visualize importance using a plot, values should be ascending for plotting purposes
df_feature_importance_asc.plot(kind='barh', figsize=(10,6))
plt.show()
```



It can be seen that the `main factor` in predicing whether a client will invest a term deposit when called in the current campaign is the `outcome of the previous telemarketing campaign` for those clients.

`Other important factors` revolve mostly on the `contact type`, the `previous outcome`, the `information regarding the last contact`, and information on whether the clients have `housing/personal loans`.

Conclusions

We explored a dataset involving information regarding telephone marketing campaigns over clients with indication of whether or not they were contacted in the previous campaign, and whether or not the previous and current campaigns were successful. Different models were created over different versions of the dataset, since the dataset

also underwent multiple balancing techniques. It was found that models under the non-balanced data set produced a decent value for precision but poor values for recall, and applying different balancing techniques produced models with improved recall for the price of precision.

Metrics can possible be further improved through using different models or doing hyperparameter optimization.