

Classification problem of Bank's Telemarketing Campaign: Project #3

- **Student name:** Natalia Edelson
 - **Student pace:** Flex
 - **Scheduled project review date/time:**
 - **Instructor name:** 12/12/22
 - **Blog post URL:**
-
- Importing Libraries
 - Cleaning Data
 - Explanatory Data Analysis
 - Building Classification Models
 - Evaluating Models
 - Conclusions

Importing Libraries

In [27]:

```
# Pandas and Matplotlib

import pandas as pd
import numpy as np
np.random.seed(0)
import seaborn as sns
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt

#Sk-learn

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, accuracy_score
from sklearn.metrics import f1_score, confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import auc, roc_curve, roc_auc_score, precision_recall_curve
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import f1_score,precision_score,recall_score, plot_confusion_matrix
from sklearn.metrics import plot_roc_curve,classification_report
import warnings
import random
from matplotlib import cm
from sklearn.model_selection import GridSearchCV
warnings.filterwarnings('ignore')
import time
import seaborn as sns
import seaborn
%matplotlib inline
```

```

from xgboost import XGBClassifier
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import StandardScaler
import re
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.svm import SVC
from tqdm.notebook import tqdm
import warnings
warnings.filterwarnings('ignore')

```

Data Set - Variables Explanation

- **age** - age of the costumer (numeric)
- **job** type of job (categorical: 'admin','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
- **marital** marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
- **education** -

(categorical:'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university')
- **default** - has credit in default? (categorical: 'no','yes','unknown')
- **housing** has housing loan? (categorical: 'no','yes','unknown')
- **loan** - has personal loan? (categorical: 'no','yes','unknown')
- **contact** - contact communication type (categorical: 'cellular','telephone')
- **month** -last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
- **day_of_week** -last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
- **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.
- **campaign** -number of contacts performed during this campaign and for this client (numeric, includes last contact)

- **pdays:** - number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
- **previous** -number of contacts performed before this campaign and for this client (numeric)
- **poutcome** - outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')
- **emp.var.rate** - employment variation rate - quarterly indicator (numeric)
- **cons.price.idx** - consumer price index - monthly indicator (numeric)
- **cons.conf.idx** - consumer confidence index - monthly indicator (numeric)
- **uribor3m** - euribor 3 month rate - daily indicator (numeric)
- **nr.employed** - number of employees - quarterly indicator (numeric)
- **subscribed** - has the client subscribed a term deposit? (binary: 'yes','no')

In [28]: *# Upload the data and view it*

```
df_bank = pd.read_excel('bank.xlsx')
df_bank.head()
```

Out[28]:

	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

In [29]: *# Replace the name of the target variable from 'y' to 'subscribed' for clarity p*

```
df_bank = df_bank.rename({'y': 'subscribed'}, axis=1) # new method
```

In [30]: *# Check the overall data, data type, and number of entries. Spot null values.*

```
df_bank.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  npv               41188 non-null   float64
 11  emp.var.rate     41188 non-null   float64
 12  cons.price.idx   41188 non-null   float64
 13  cons.conf.idx    41188 non-null   float64
 14  uribor3m         41188 non-null   float64
 15  nr.employed      41188 non-null   float64
 16  previous          41188 non-null   int64 
 17  poutcome          41188 non-null   object 
 18  pdays             41188 non-null   int64 
 19  subscribed        41188 non-null   object 
 20  balance           41188 non-null   float64
 21  nlastmonth        41188 non-null   float64
```

```

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  subscribed    41188 non-null  object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB

```

In [31]: *# Check the shape of the data 21 columns and 41,188 rows*

```
df_bank.shape
```

Out[31]: (41188, 21)

In [32]: *# List all the columns*

```
df_bank.columns
```

Out[32]: Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
 'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
 'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
 'cons.conf.idx', 'euribor3m', 'nr.employed', 'subscribed'],
 dtype='object')

Cleaning the Data

Checking for Null Values

In [33]: *# Using isnull method and sum it up*

```
df_bank.isnull().sum()
```

age	0
job	0
marital	0
education	0
default	0
housing	0
loan	0
contact	0
month	0
day_of_week	0
duration	0
campaign	0
pdays	0
previous	0

```
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
subscribed        0
dtype: int64
```

In [34]:

```
# Checking for object values

cat_df = df_bank.select_dtypes('object')
cat_df.head()
```

Out[34]:

	job	marital	education	default	housing	loan	contact	month	day_of_week	pou
0	housemaid	married	basic.4y	no	no	no	telephone	may	mon	none
1	services	married	high.school	unknown	no	no	telephone	may	mon	none
2	services	married	high.school	no	yes	no	telephone	may	mon	none
3	admin.	married	basic.6y	no	no	no	telephone	may	mon	none
4	services	married	high.school	no	no	yes	telephone	may	mon	none

Unknown Values

There are many unknown values and here I break down the percentage of the unknown values to see which ones would be okay to remove from the data.

In [35]:

```
# Checking for the frequency of values of each column
for colname in cat_df.columns:
    try:
        print(colname, cat_df[colname].value_counts(normalize=True)[:10])
    except:
        print(colname, cat_df[colname].value_counts(normalize=True))

    print('\n')
```

```
job admin.          0.253035
blue-collar        0.224677
technician         0.163713
services           0.096363
management         0.070992
retired            0.041760
entrepreneur       0.035350
self-employed      0.034500
housemaid          0.025736
unemployed         0.024619
Name: job, dtype: float64
```

```
marital married     0.605225
single             0.280859
divorced           0.111974
unknown            0.001942
Name: marital, dtype: float64
```

```
education university.degree   0.295426
```

```
high.school      0.231014
basic.9y        0.146766
professional.course 0.127294
basic.4y        0.101389
basic.6y        0.055647
unknown          0.042027
illiterate       0.000437
Name: education, dtype: float64
```

```
default no      0.791201
unknown      0.208726
yes          0.000073
Name: default, dtype: float64
```

```
housing yes      0.523842
no            0.452122
unknown      0.024036
Name: housing, dtype: float64
```

```
loan no        0.824269
yes          0.151695
unknown      0.024036
Name: loan, dtype: float64
```

```
contact cellular 0.634748
telephone     0.365252
Name: contact, dtype: float64
```

```
month may      0.334296
jul        0.174177
aug        0.149995
jun        0.129115
nov        0.099568
apr        0.063902
oct        0.017432
sep        0.013839
mar        0.013256
dec        0.004419
Name: month, dtype: float64
```

```
day_of_week thu      0.209357
mon        0.206711
wed        0.197485
tue        0.196416
fri        0.190031
Name: day_of_week, dtype: float64
```

```
poutcome nonexistent 0.863431
failure      0.103234
success      0.033335
Name: poutcome, dtype: float64
```

```
subscribed no      0.887346
yes          0.112654
Name: subscribed, dtype: float64
```

marital

**unknown 0.001942

education

**unknown 0.042027

default

**unknown 0.208726

housing yes

**unknown 0.024036

loan

**unknown 0.024036

In [36]:

```
# The unkown values replaced with None and then dropped
unknown_col = ['job', 'marital', 'education', 'loan', 'housing']

for col in unknown_col:
    df_bank[col] = df_bank[col].replace('unknown', None)
    #df_bank[col] = df_bank[col].fillna(df[col].mode())
    df_bank.dropna(axis=0, subset=[col], inplace = True)
```

Handling Duplicates

In [37]:

```
# Check for duplicates
df_bank.duplicated().sum()
```

Out[37]: 12

In [38]:

```
# Remove duplicates
df_bank = df_bank.drop_duplicates()
```

In [39]:

```
# Doble check for duplicates

df_bank.duplicated().sum()
```

Out[39]: 0

Exploratory Data Analysis

Checking if the data is balanced

In [40]:

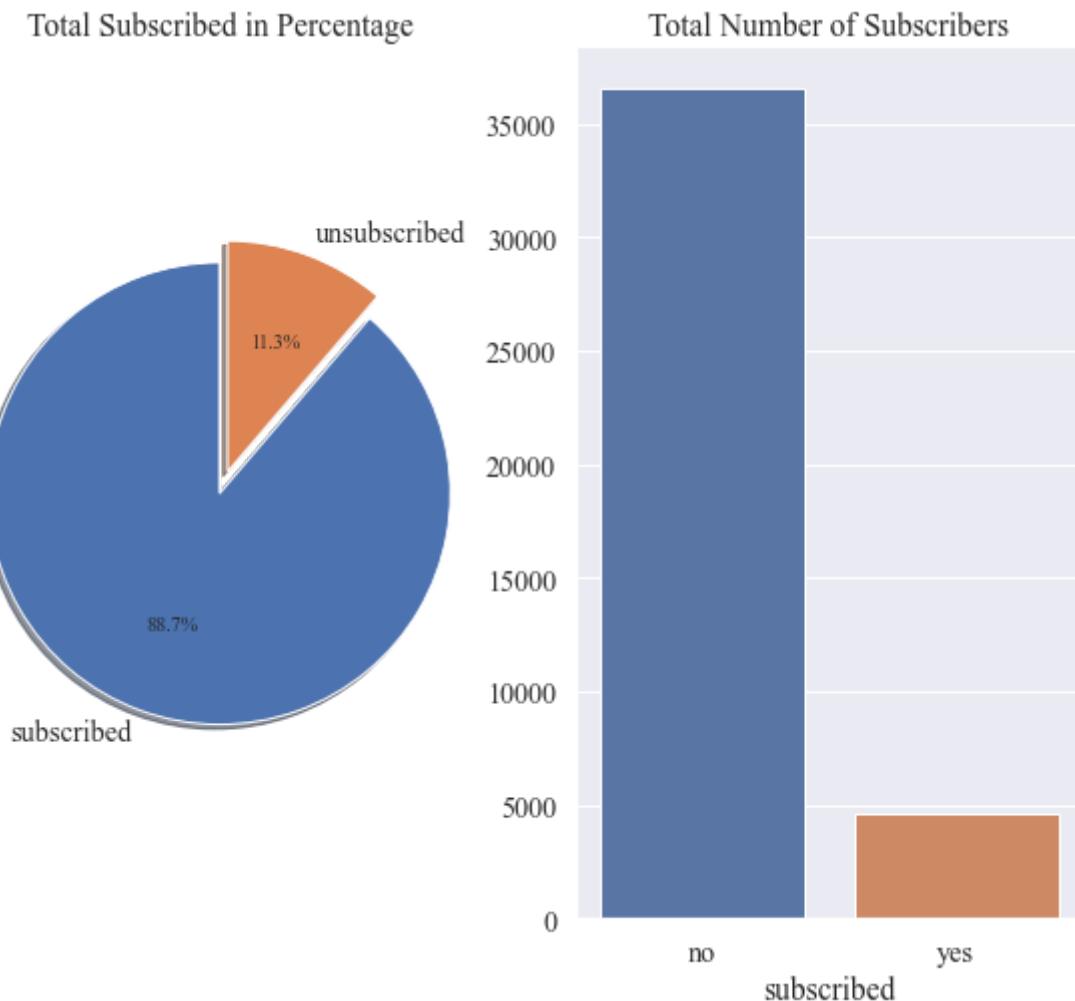
```
# Plot the target variables in order to examine whether the data is imbalanced.
```

```
subscribe_vc = df_bank['subscribed'].value_counts()  
subscribe_vc
```

```
Out[40]: no      36537  
yes      4639  
Name: subscribed, dtype: int64
```

```
In [41]:
```

```
# Pie chart, where the slices will be ordered and plotted counter-clockwise:  
# Reference: https://matplotlib.org/stable/gallery/pie_and_polar_charts/pie_feat  
  
labels = 'subscribed', 'unsubscribed'  
sizes = df_bank['subscribed'].value_counts()  
explode = (0, 0.1) # only "explode" the 2nd slice  
  
fig, ax = plt.subplots(1,2, figsize=(10,8))  
  
ax[0].pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',  
          shadow=True, startangle=90)  
ax[0].axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.  
ax[0].set_title('Total Subscribed in Percentage', fontname='Times New Roman')  
  
sns.countplot('subscribed', data=df_bank, ax=ax[1])  
  
ax[1].set_title('Total Number of Subscribers', fontname='Times New Roman')  
ax[1].set_ylabel('')  
  
plt.show()
```



The data is imbalanced - only 11.3% costumes subscribed to the bank campaign. Later on, when we build the model, it will need to be dealt with. Most models can't produce proper results when the data is imbalanced.

```
In [42]: # Check the distribution of data type
```

```
print("Data types and their frequency\n{}".format(df_bank.dtypes.value_counts()))
```

```
Data types and their frequency
object      11
int64       5
float64     5
dtype: int64
```

```
In [43]: # Check balance of target data
```

```
df_bank[ 'subscribed' ].value_counts()
```

```
Out[43]: no      36537
yes      4639
Name: subscribed, dtype: int64
```

```
In [44]: # Check for unique values for all the columns
```

```
df_bank.nunique()
```

```
Out[44]: age          78
          job           11
          marital        3
          education      7
          default         3
          housing          2
          loan            2
          contact          2
          month          10
          day_of_week      5
          duration       1544
          campaign        42
          pdays           27
          previous         8
          poutcome         3
          emp.var.rate     10
          cons.price.idx   26
          cons.conf.idx    26
          euribor3m       316
          nr.employed      11
          subscribed       2
          dtype: int64
```

Exploring Continuous Variables

```
In [45]: # Checking the distribution for the categorical variables c
df_bank.describe()
```

	age	duration	campaign	pdays	previous	emp.var.rate	cons
count	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000
mean	40.02380	258.315815	2.567879	962.464810	0.173013	0.081922	11.530000
std	10.42068	259.305321	2.770318	186.937102	0.494964	1.570883	11.530000
min	17.000000	0.000000	1.000000	0.000000	0.000000	-3.400000	11.530000
25%	32.000000	102.000000	1.000000	999.000000	0.000000	-1.800000	11.530000
50%	38.000000	180.000000	2.000000	999.000000	0.000000	1.100000	11.530000
75%	47.000000	319.000000	3.000000	999.000000	0.000000	1.400000	11.530000
max	98.000000	4918.000000	56.000000	999.000000	7.000000	1.400000	11.530000

Correlation

```
In [46]: # Create a function that captures the numerical variables.
numerical_var = [feature for feature in df_bank.columns if ((df_bank[feature].dtypes == 'float64') & (feature != 'y'))]
print('Number of continuous variables: ', len(numerical_var))

# visualise the numerical variables
df_bank_num = df_bank[numerical_var].head()

df_bank_num
```

Number of continuous variables: 10

Out[46]:	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor
0	56	261	1	999	0	1.1	93.994	-36.4	4.8
1	57	149	1	999	0	1.1	93.994	-36.4	4.8
2	37	226	1	999	0	1.1	93.994	-36.4	4.8
3	40	151	1	999	0	1.1	93.994	-36.4	4.8
4	56	307	1	999	0	1.1	93.994	-36.4	4.8

In [47]:

```
# Checking the correlation metrics.

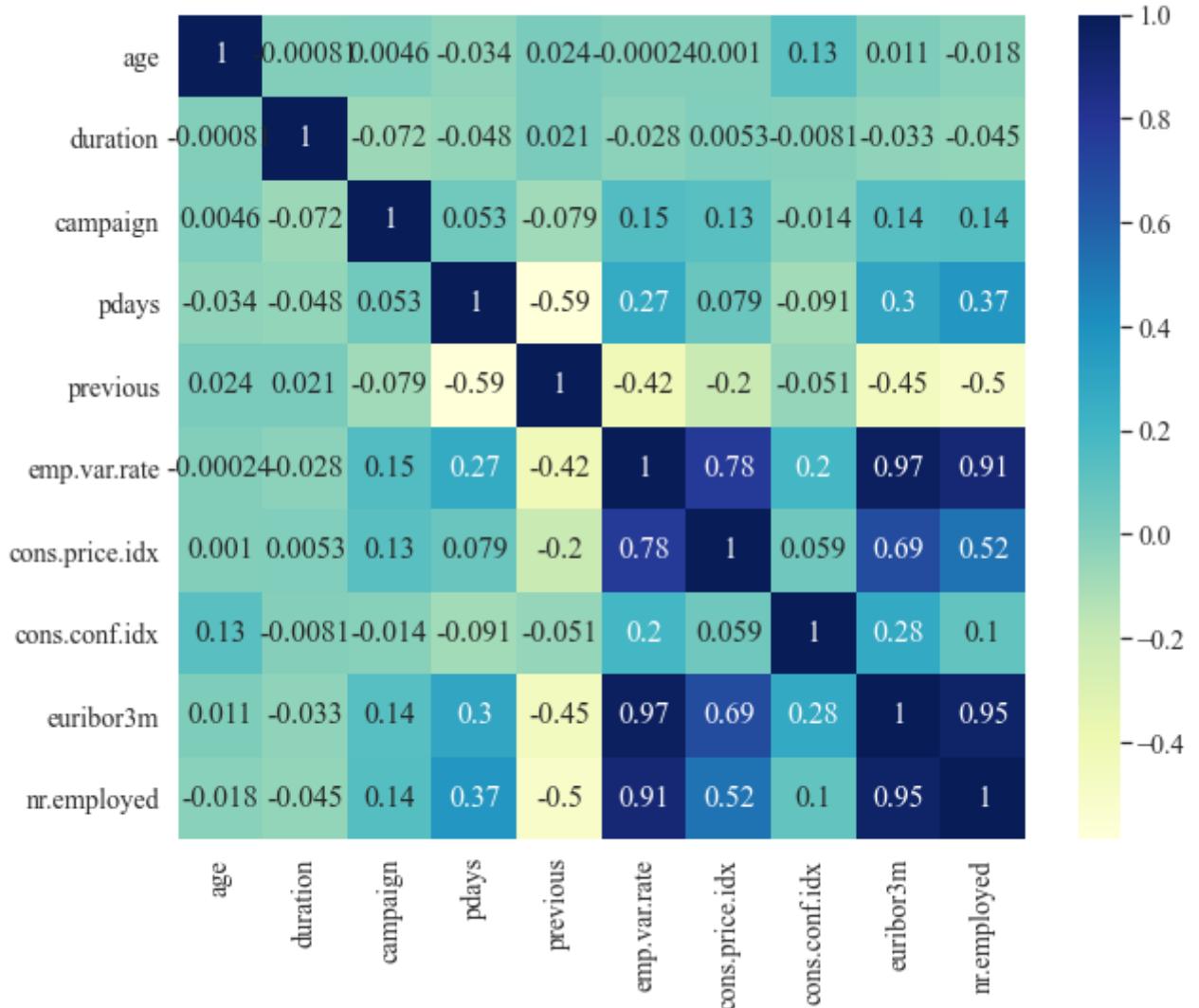
sns.set_theme(context='notebook', style='darkgrid', palette='deep', font='Times
font_scale= 1.3, color_codes=True, rc=None)

plt.figure(figsize=(10,8))

sns.heatmap(df_bank.corr(), cmap='YlGnBu', annot=True)

plt.show
```

Out[47]: <function matplotlib.pyplot.show(close=None, block=None)>



```
In [48]: # Select upper triangle of correlation matrix, removing corr. higher than 0.95
corr_matrix = df_bank.corr()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Using a loop that would scan for high correlation 0.95 and
# would remove the respective variable.

to_drop = [column for column in upper.columns if any(upper[column] > 0.90)]
to_drop
```

Out[48]: ['euribor3m', 'nr.employed']

These are the variables suggested 'euribor3m', 'nr.employed' - there are multiple strong relationships. 'emp.var.rate' and euribor3m as well as emp.car.rate with 'nr.employed'.

I will remove 'nr.employed' and 'emp.var.rate' since 'euribor3m' is likely to be an important part of decision making for individuals when they decide to Subscribe or not to Subscribe.

```
In [49]: # Drop the respective features
```

```
to_drop = ['nr.employed', 'emp.var.rate']

df_bank.drop(to_drop, axis=1, inplace=True)
```

In [50]:

```
# Scan the data again to verify that the variables were removed.
df_bank.head()
```

Out[50]:

	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

In [51]:

```
# Checking again the correlation metrics.
```

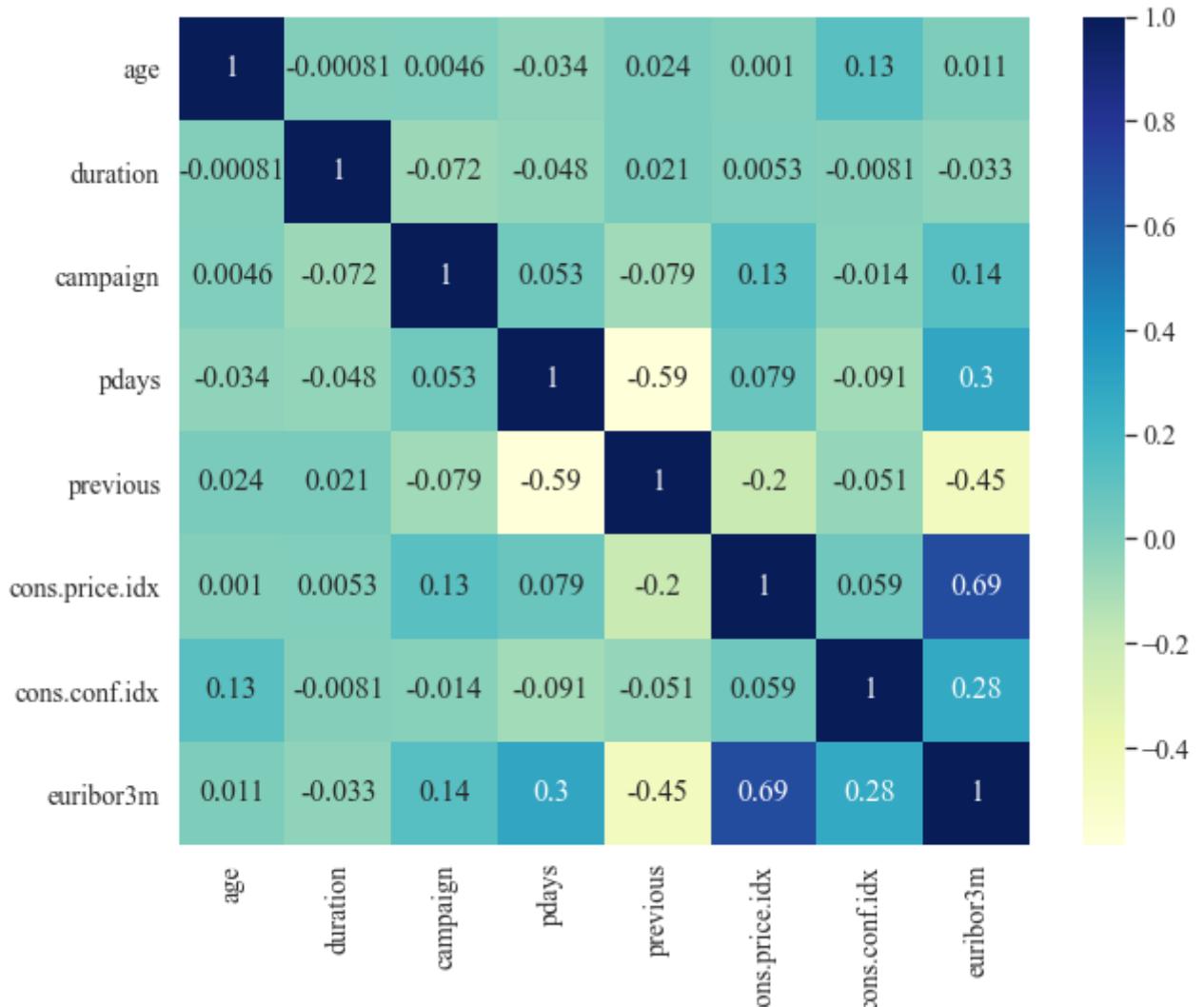
```
sns.set_theme(context='notebook', style='darkgrid', palette='deep', font='Times
font_scale= 1.3, color_codes=True, rc=None)

plt.figure(figsize=(10,8))

sns.heatmap(df_bank.corr(), cmap='YlGnBu', annot=True)

plt.show
```

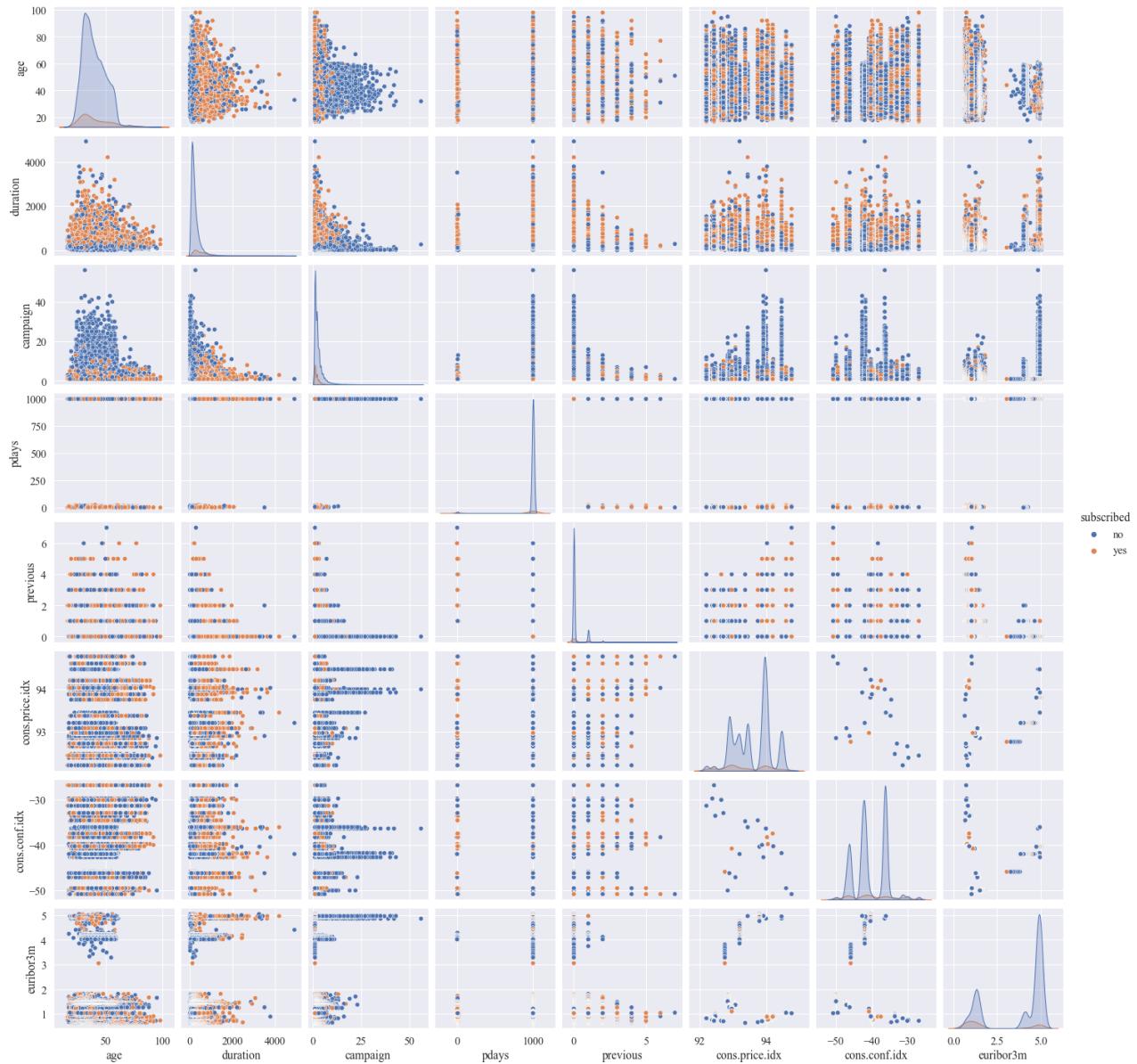
Out[51]: <function matplotlib.pyplot.show(close=None, block=None)>



Pairplot

```
In [53]: # I will plot pairplot - using the hue 'subscribed' in order to spot any pattern
plt.figure(figsize=(50,40))
sns.pairplot(df_bank, hue='subscribed')
#ax.legend()
```

Out[53]: <seaborn.axisgrid.PairGrid at 0x7fac8acc1df0>
<Figure size 3600x2880 with 0 Axes>



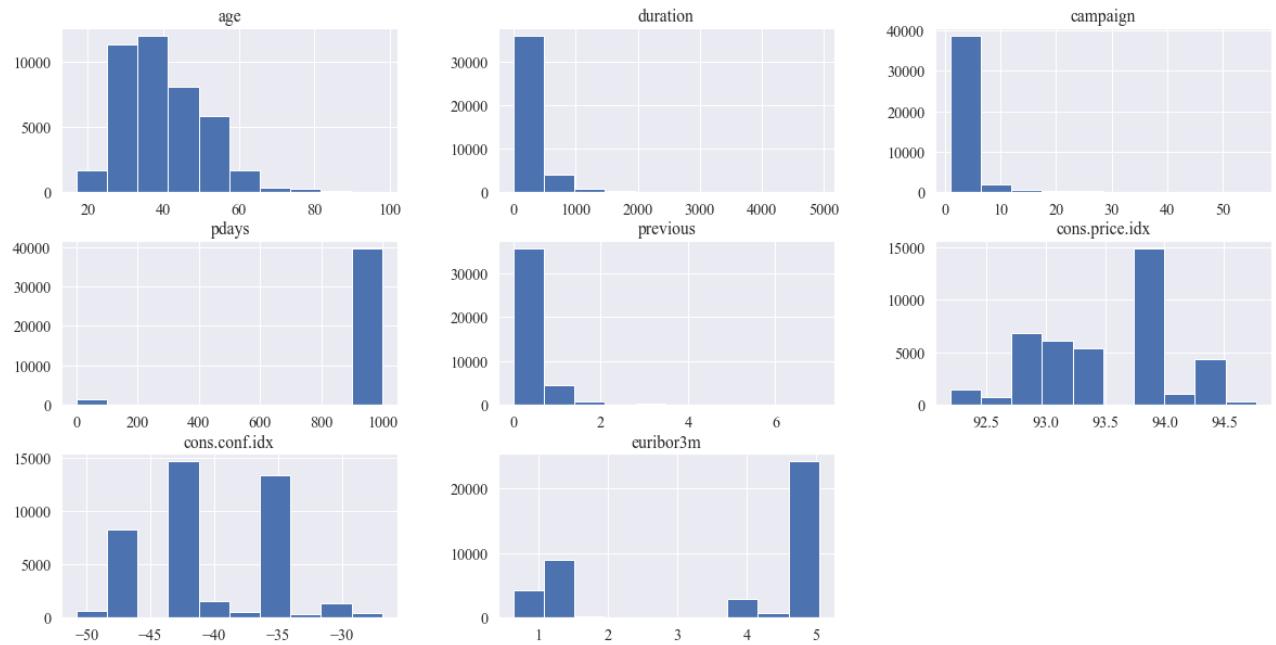
Notes from pairplot - most variables are categorical. The continuous variables show various types of distribution.

Histogram

In [54]:

```
# Plot a histogram to take a closer look at the distribution.
```

```
df_bank.hist(figsize= [20,10])
plt.show()
```



Visualize Numerical Variables

In [55]:

```
# Here is another way to get the numerical data using _get_numeric_data function
numeric_data = df_bank._get_numeric_data()
numeric_data.head()
```

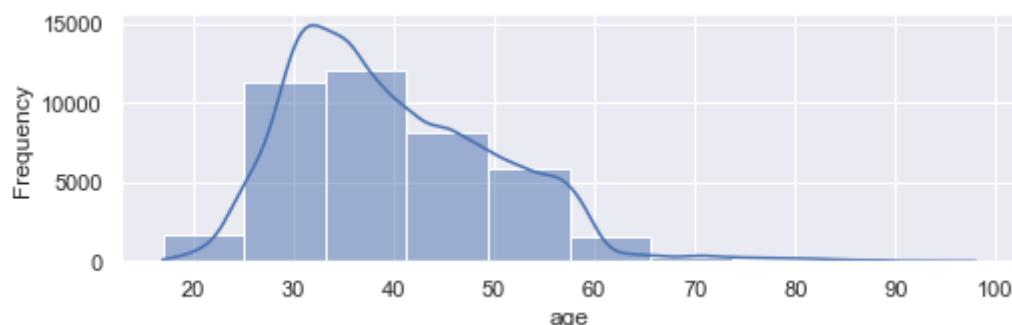
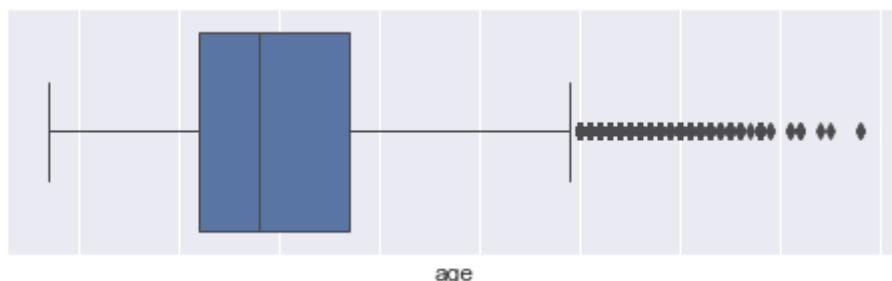
Out[55]:

	age	duration	campaign	pdays	previous	cons.price.idx	cons.conf.idx	euribor3m
0	56	261	1	999	0	93.994	-36.4	4.857
1	57	149	1	999	0	93.994	-36.4	4.857
2	37	226	1	999	0	93.994	-36.4	4.857
3	40	151	1	999	0	93.994	-36.4	4.857
4	56	307	1	999	0	93.994	-36.4	4.857

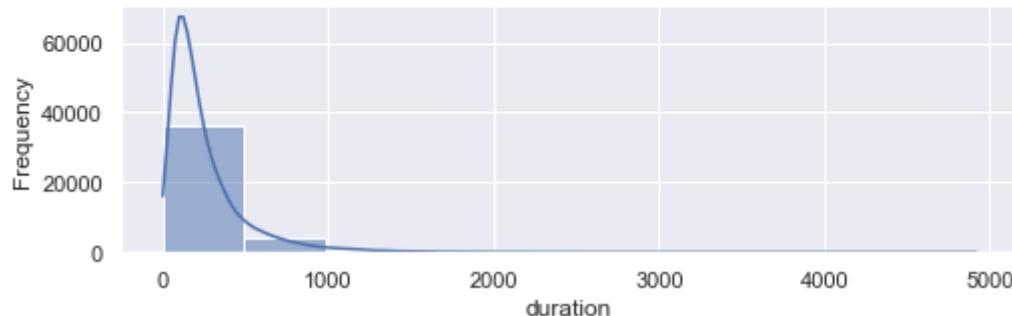
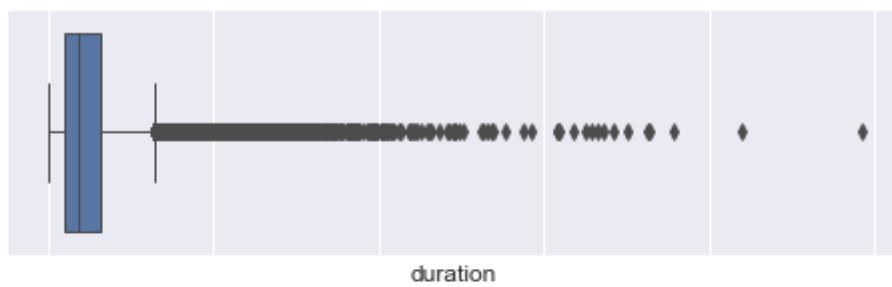
In [56]:

```
# In this plot, we dive a little closer into the distribution of the numerical columns
for i in numeric_data.columns:
    plt.figure()
    plt.tight_layout()
    sns.set(rc={"figure.figsize":(8, 5)})
    f, (ax_box, ax_hist) = plt.subplots(2, sharex=True)
    plt.gca().set(xlabel= i,ylabel='Frequency')
    sns.boxplot(numeric_data[i], ax=ax_box , linewidth= 1.0)
    sns.histplot(numeric_data[i], ax=ax_hist , bins = 10,kde=True)
```

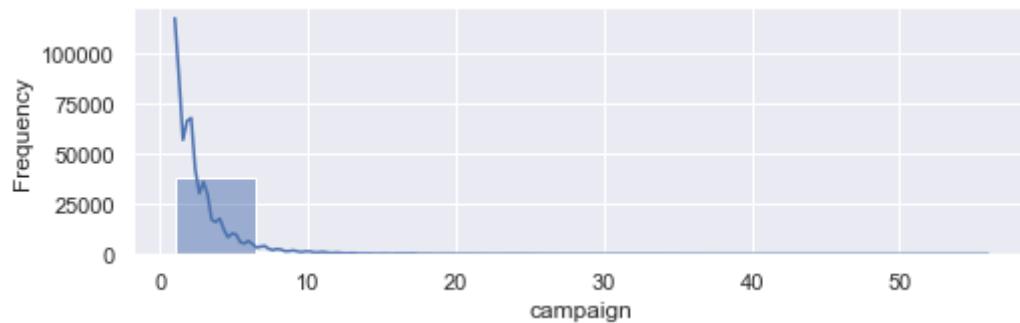
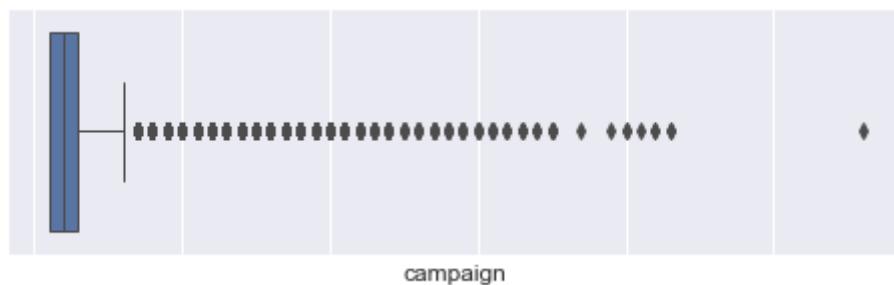
<Figure size 432x288 with 0 Axes>



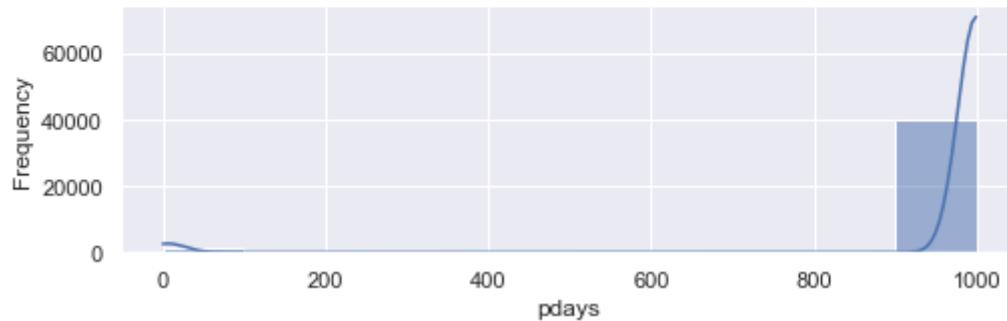
<Figure size 576x360 with 0 Axes>



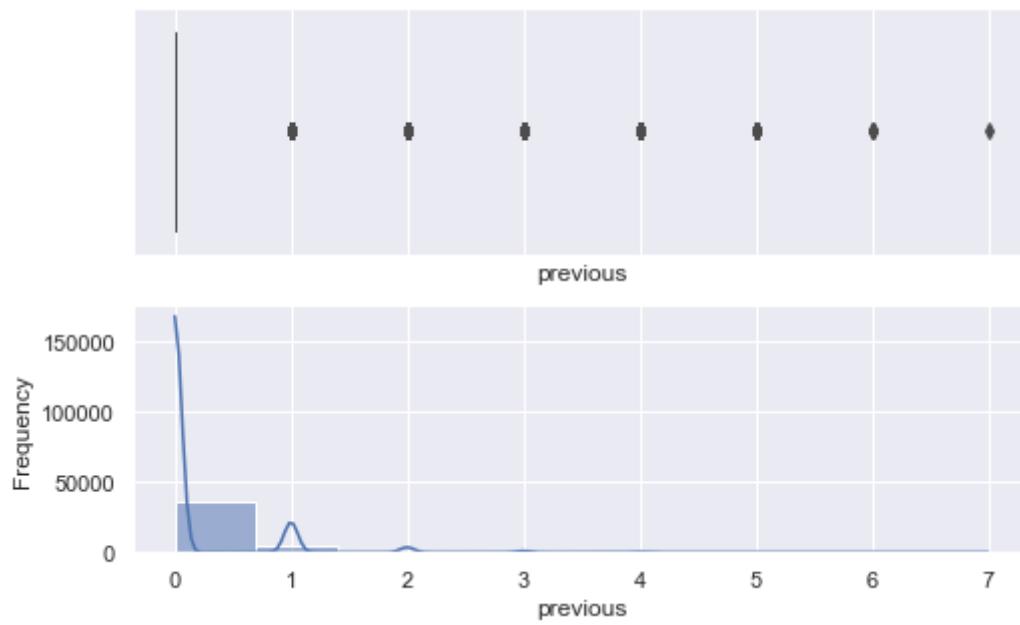
<Figure size 576x360 with 0 Axes>



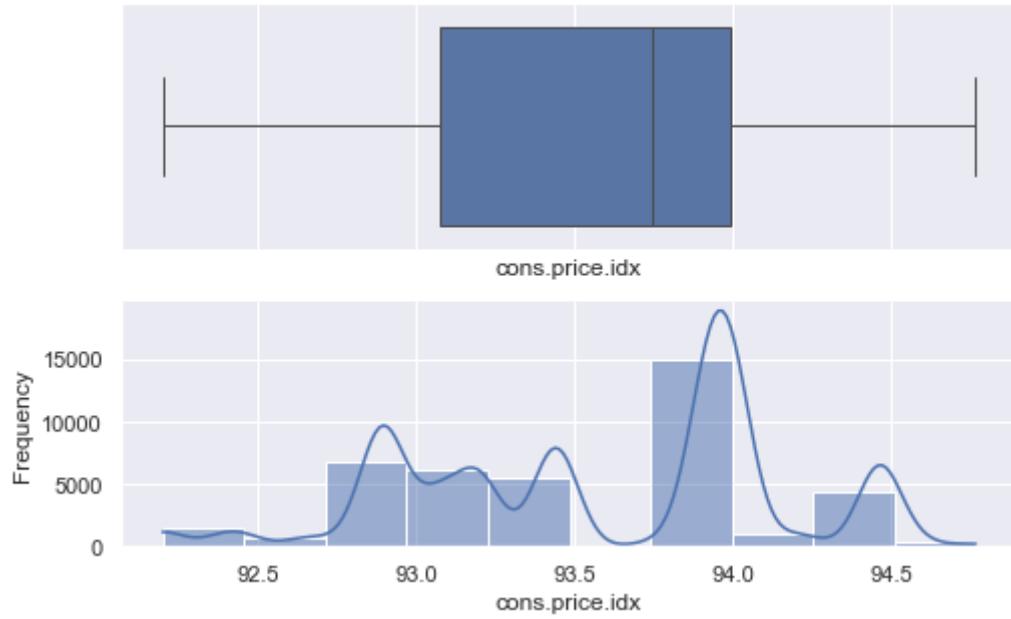
<Figure size 576x360 with 0 Axes>



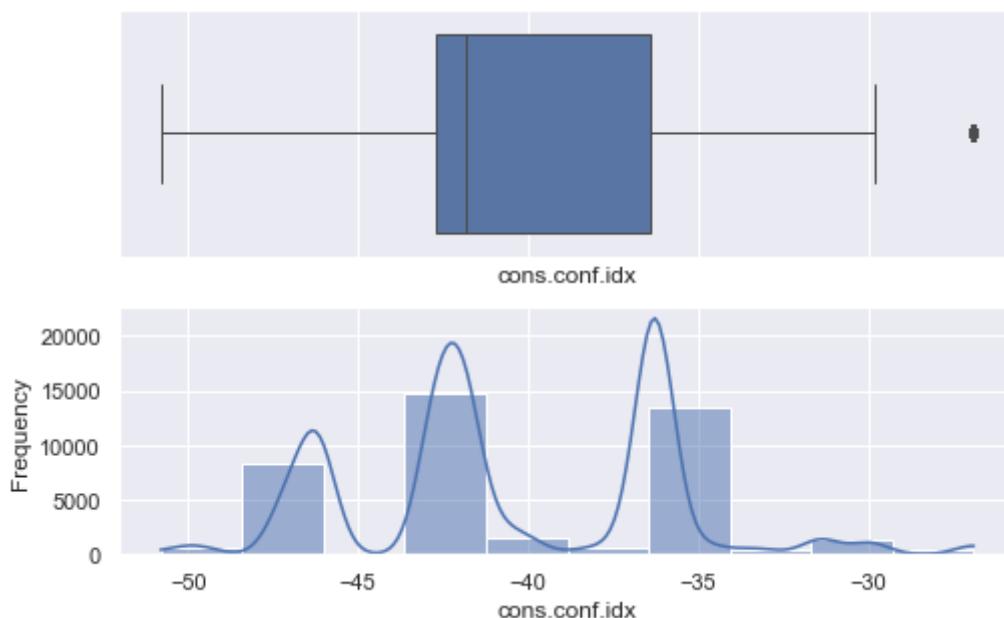
<Figure size 576x360 with 0 Axes>



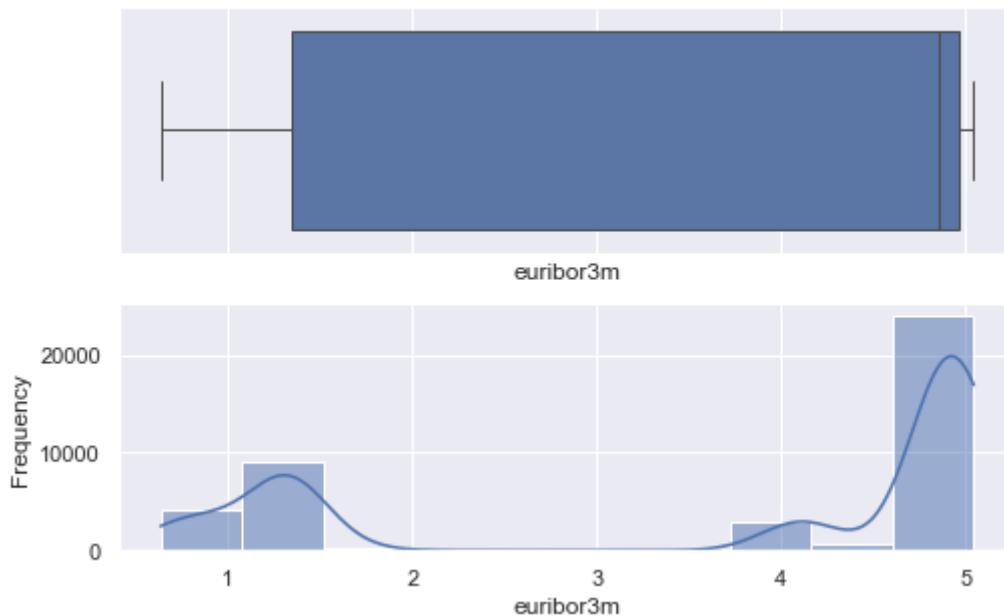
<Figure size 576x360 with 0 Axes>



<Figure size 576x360 with 0 Axes>



<Figure size 576x360 with 0 Axes>



Age: most of the population was from 30 to 50 years old.

Duration: most calls ranged from a minute to 8.5 minutes.

In [57]:

```
# Using a lambda function - I replace all the dots with the underlying symbol.

df_bank.columns = list(map(lambda x: x.replace('.','_'),
                           df_bank.columns))
```

In [58]:

```
# Plotting the numerical variables in relationship to subscribers versus
# people who did not subscribe.
print('Statistical Summary ')
print('\n')
print('Subscribed')
print(df_bank[df_bank['subscribed']=='yes'][['age','duration','pdays','previous']]
```

```

'cons_price_idx',
'cons_conf_idx']].describe().transp

# print('--'*40)
# print('')

# Subplots of Numeric Features
sns.set_style('darkgrid')
fig = plt.figure(figsize = (18,15))
fig.subplots_adjust(hspace = .50)

ax1 = fig.add_subplot(421)
ax1.hist(df_bank[df_bank['subscribed']=='no'].age, bins = 25,
          label = 'Did Not Subscribed', alpha = .50, edgecolor= 'black', color = 'blue')
ax1.hist(df_bank[df_bank['subscribed']=='yes'].age, bins = 25,
          label = 'Subscribed', alpha = .50, edgecolor = 'black', color = 'lightgreen')
ax1.set_title('Costumer Age: Subscribed vs Non-Subscribed')
ax1.set_xlabel('Age')
ax1.set_ylabel('# Costumers')
ax1.legend(loc = 'upper right')

ax2 = fig.add_subplot(422)
ax2.hist(df_bank[df_bank['subscribed']=='no'].duration,
          bins = 25, label = 'Did Not Subscribed', alpha = .50,
          edgecolor = 'black', color = 'blue')
ax2.hist(df_bank[df_bank['subscribed']=='yes'].duration,
          bins = 25, label = 'Subscribed', alpha = .50,
          edgecolor = 'black', color = 'lightgreen')
ax2.set_title('Duration: Subscribed vs Non-Subscribed')
ax2.set_xlabel('Duration')
ax2.set_ylabel('# Costumers')
ax2.legend(loc = 'upper right')

ax3 = fig.add_subplot(423)

ax3.hist(df_bank[df_bank['subscribed']=='no'].pdays, bins = 50,
          label = 'Did Not Subscribed', alpha = .50, edgecolor = 'black', color = 'blue')
ax3.hist(df_bank[df_bank['subscribed']=='yes'].pdays, bins = 50,
          label = 'Subscribed', alpha = .50, edgecolor = 'black', color = 'lightgreen')
ax3.set_title('pdays: Subscribed vs Non-Subscribed')
ax3.set_xlabel('pdays')
ax3.set_ylabel('# Costumers')
ax3.legend(loc = 'upper right')

ax4 = fig.add_subplot(424)

ax4.hist(df_bank[df_bank['subscribed']=='no'].previous,
          bins = 50, label = 'Did Not Subscribed', alpha = .50, edgecolor = 'black',
          color = 'blue')
ax4.hist(df_bank[df_bank['subscribed']=='yes'].previous,
          bins = 50, label = 'Subscribed', alpha = .50, edgecolor = 'black',
          color = 'lightgreen')
ax4.set_title('previous: Subscribed vs Non-Subscribed')
ax4.set_xlabel('previous')
ax4.set_ylabel('# Costumers')
ax4.legend(loc = 'upper right')

```

```

ax5 = fig.add_subplot(425)

ax5.hist(df_bank[df_bank['subscribed']=='no'].cons_price_idx,
          bins = 50, label = 'Did Not Subscribed', alpha = .50,
          edgecolor ='black', color = 'blue')
ax5.hist(df_bank[df_bank['subscribed']=='yes'].cons_price_idx,
          bins = 50, label = 'Subscribed', alpha = .50, edgecolor = 'black',
          color = 'lightgreen')
ax5.set_title('cons.price.idx: Subscribed vs Non-Subscribed')
ax5.set_xlabel('cons.price.idx')
ax5.set_ylabel('# Costumers')
ax5.legend(loc = 'upper right')

ax6 = fig.add_subplot(426)

ax6.hist(df_bank[df_bank['subscribed']=='no'].cons_conf_idx, bins = 50,
          label = 'Did Not Subscribed', alpha = .50, edgecolor ='black', color =
ax6.hist(df_bank[df_bank['subscribed']=='yes'].cons_conf_idx, bins = 50,
          label = 'Subscribed', alpha = .50, edgecolor = 'black', color = 'lightgre
ax6.set_title('euribor3m: Subscribed vs Non-Subscribed')
ax6.set_xlabel('euribor3m')
ax6.set_ylabel('# Costumers')
ax6.legend(loc = 'upper right')

ax3.legend()

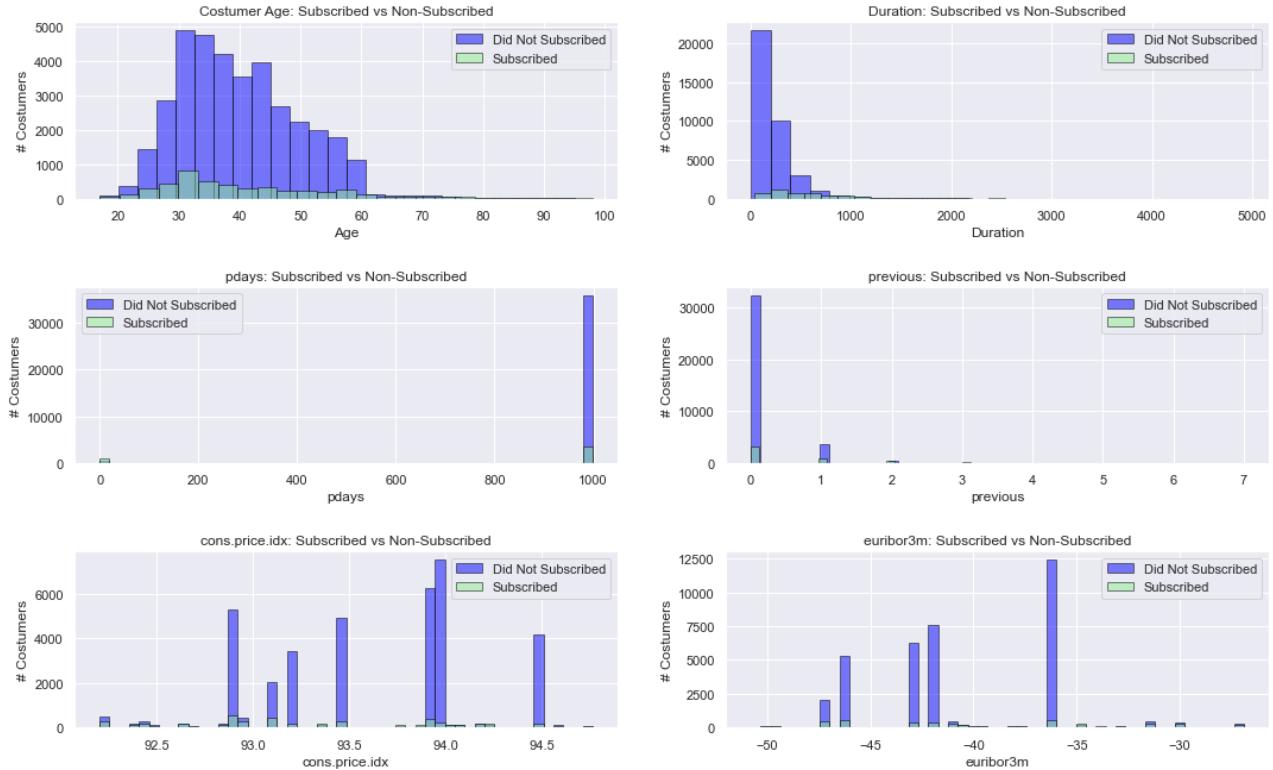
plt.show()

```

Statistical Summary

Subscribed

	count	mean	std	min	25%	50%	\
age	4639.0	40.912266	13.838838	17.000	31.000	37.0	
duration	4639.0	553.256090	401.190736	37.000	253.500	449.0	
pdays	4639.0	791.990946	403.439220	0.000	999.000	999.0	
previous	4639.0	0.492779	0.860406	0.000	0.000	0.0	
cons_price_idx	4639.0	93.354577	0.676592	92.201	92.893	93.2	
cons_conf_idx	4639.0	-39.791119	6.139657	-50.800	-46.200	-40.4	
		75%	max				
age		50.000	98.000				
duration		741.500	4199.000				
pdays		999.000	999.000				
previous		1.000	6.000				
cons_price_idx		93.918	94.767				
cons_conf_idx		-36.100	-26.900				



Certain elements are evident -

- * A prospective client is more likely to subscribe without having any previous contacts.
- * People in their 30s are more likely to subscribe.
- * Certain indices are showing a small increase in likelihood.

Age

In [59]:

```
# Plotting specific ages to see which ones subscribed the most.

def myplot(colname):

    df_plot = df_bank.groupby(['subscribed',
                               colname]).size().reset_index().pivot(columns='subscribed', index=colname)
    df_plot['total'] = df_plot.sum(axis=1)
    # df_plot['no'] = df_plot['no']*100/df_plot['total']
    # df_plot['yes'] = df_plot['yes']*100/df_plot['total']
    df_plot.drop('total', axis=1, inplace=True)
    # df_plot.sort_values('yes', ascending=False, inplace=True)

    # fig.set_size_inches(20, 8)

    df_plot.plot(kind='bar', stacked=True, figsize=(30,12))

    plt.xlabel(colname, fontsize=32)
```

```

plt.ylabel('Count', fontsize=32)
plt.xticks(rotation = 90, fontsize=26)
plt.yticks(rotation = 0, fontsize=24)
plt.grid(True, color = "white", linewidth = "1.4", linestyle = "-")
plt.legend(loc=2, prop={'size': 22})

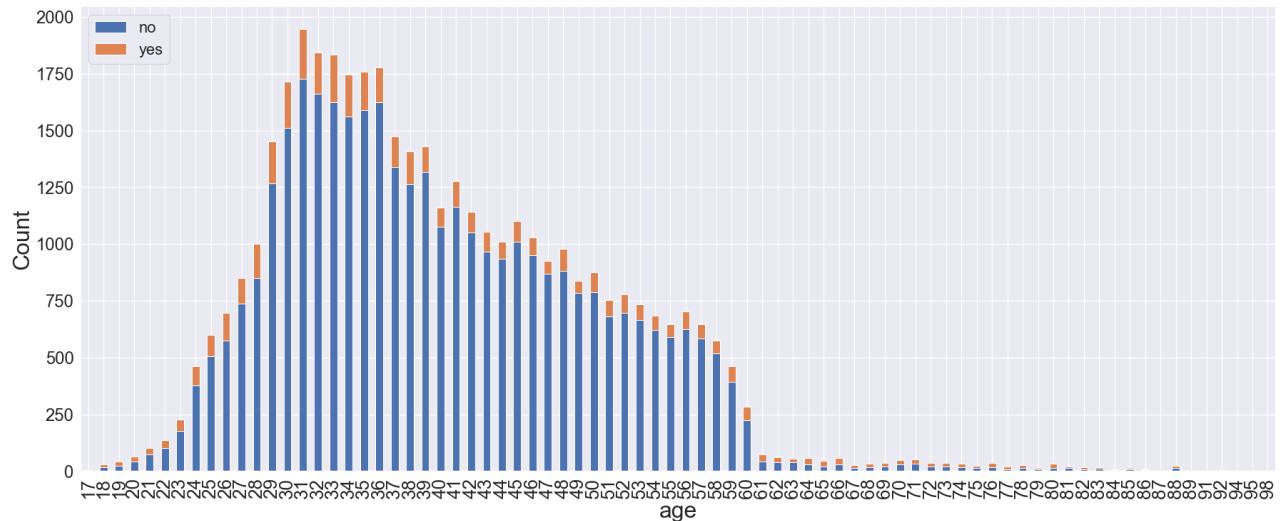
#ax.set_title('Age Count Distribution', fontsize=28)

sns.despine()

```

In [60]:

myplot('age')



I decided to create ranges of ages since it will give a better sense of what group to target.

In [61]:

```

# Here are the bins based on the values observed above.
# 5 values will result in 4 bins

bins = [17, 35, 36, 55, 56, 65, 66, 98]

#We'll pd.cut method to separate data into bins.
df_bank['bins_age'] = pd.cut(df_bank['age'], bins)

# I will use .cat.as_unordered() method transforming the data to
# ordered categories.
# bins_age_builtin = bins_yr_builtin.cat.as_unordered()
# bins_yr_builtin.head()

```

In [62]:

```

# Recheck data set for bins column.

df_bank.head()

```

Out[62]:

	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

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

In [63]:

```
# Assigning a name for a demographic age group for each bin.

df_bank = df_bank.assign(age_group=pd.cut(df_bank['age'], bins=[17, 35, 55, 65
                    right=False, labels = ['Young_Adults','Middle_Aged
                    'Older_Adults','Seniors']))
```

In [64]:

```
# Make an independent copy so I won't need to run the notebook from the top.
df_bank_copy = df_bank.copy()
```

Information about the contacted prospects

I will create a function that will calculate each variable's distribution and plot the information. It is important to see the sample of people who have been contacted. In instances where the sample is low (e.g. less than 100), I will omit to calculate the percentage change since it will be insignificantnt.

In [65]:

```
# Building a "countplot" function that will plot the distribution for each categ

def countplot(colname):
    fig, ax = plt.subplots()
    fig.set_size_inches(6, 4)
    sns.countplot(x = colname, data = df_bank, order = df_bank[colname].value_counts()
                  palette = "coolwarm_r")
    ax.set_xlabel(colname, fontsize=16)
    ax.set_ylabel('Count', fontsize=16)
    plt.xticks(rotation = 50,fontsize=16)
    plt.yticks(rotation = 0,fontsize=16)
    ax.set_title('Distribution', fontsize=16)
    ax.tick_params(labelsize=14)
    sns.despine()
```

In [66]:

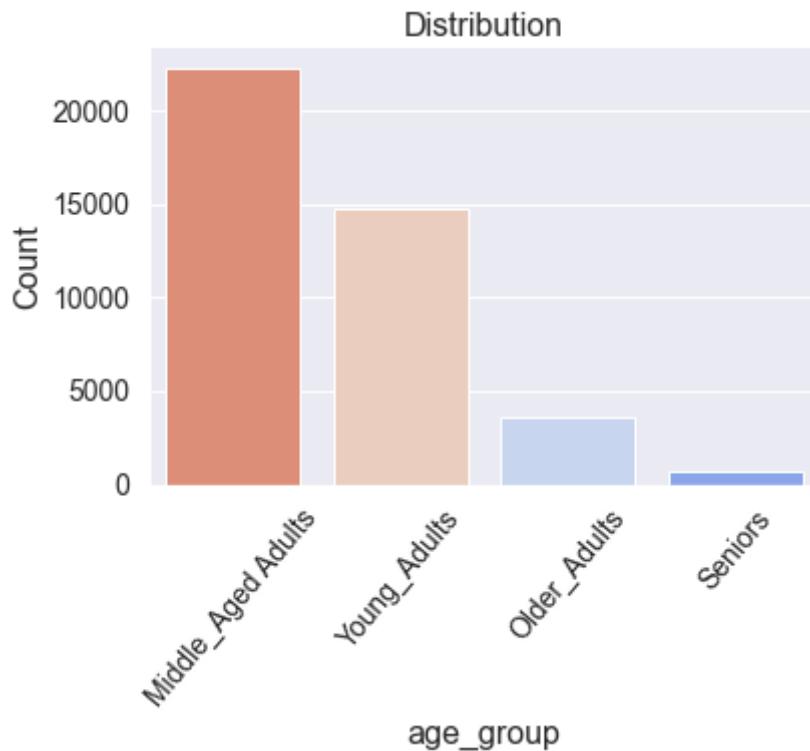
```
# Count the age groups
df_bank["age_group"].value_counts()
```

Out[66]:

Middle_Aged	Adults	22197
Young_Adults		14751
Older_Adults		3566
Seniors		660
Name:	age_group	dtype: int64

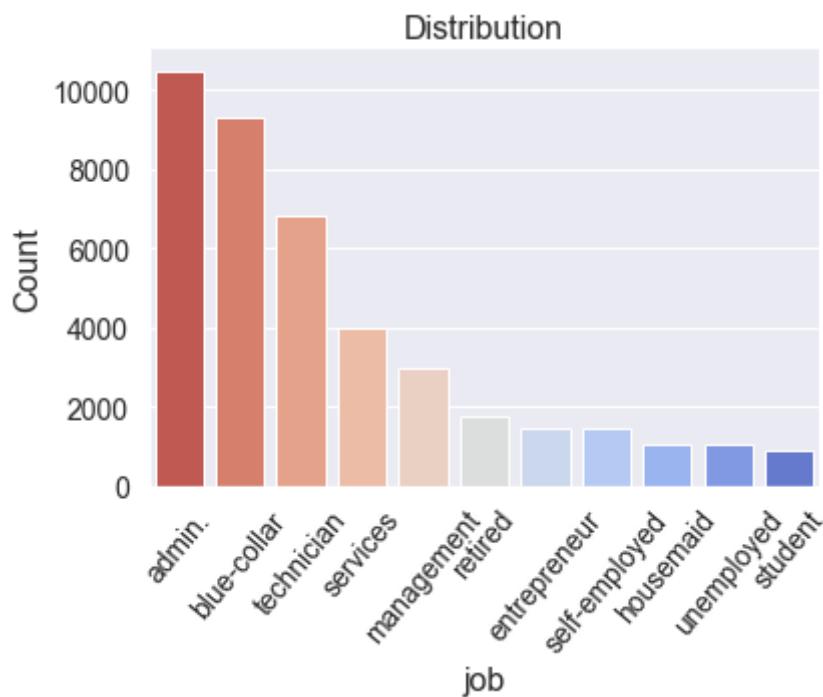
In [67]:

```
bcountplot('age_group')
```



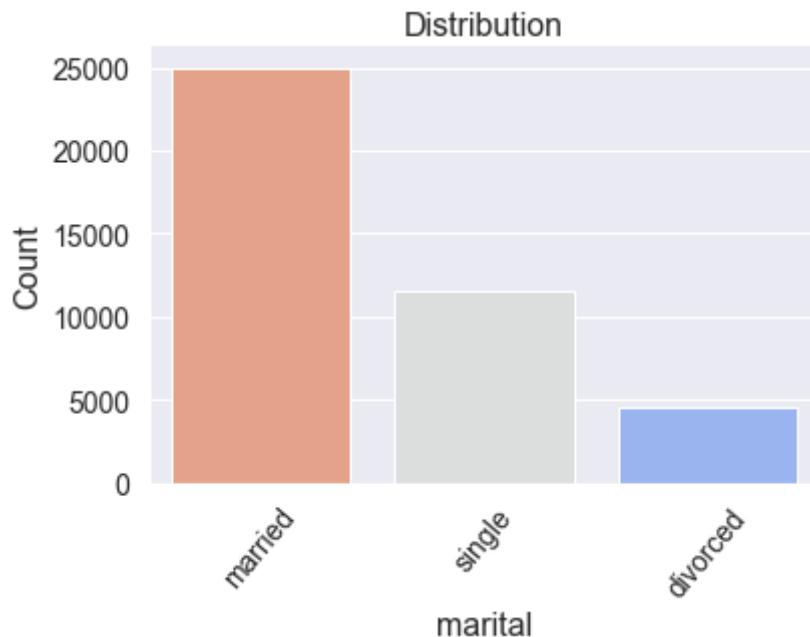
The majority of the people contacted were middle-aged and young adults. Even though only 660 senior people were contacted, there are still enough sample of people to draw information about their behavior.

```
In [68]: countplot('job')
```



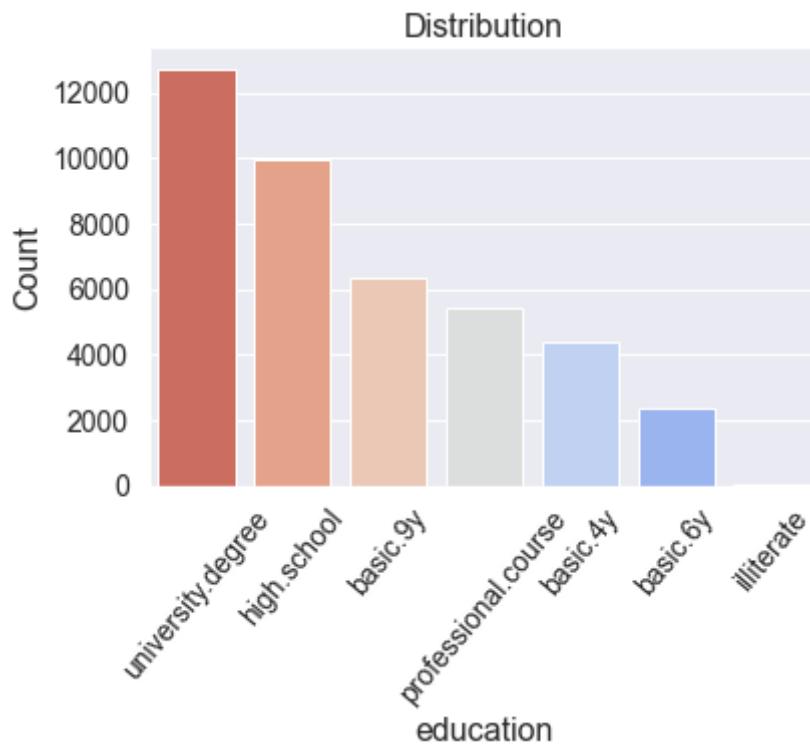
The sample of the population that was mainly included in the job category was: Admin, Blue-color Technician.

```
In [69]: countplot('marital')
```



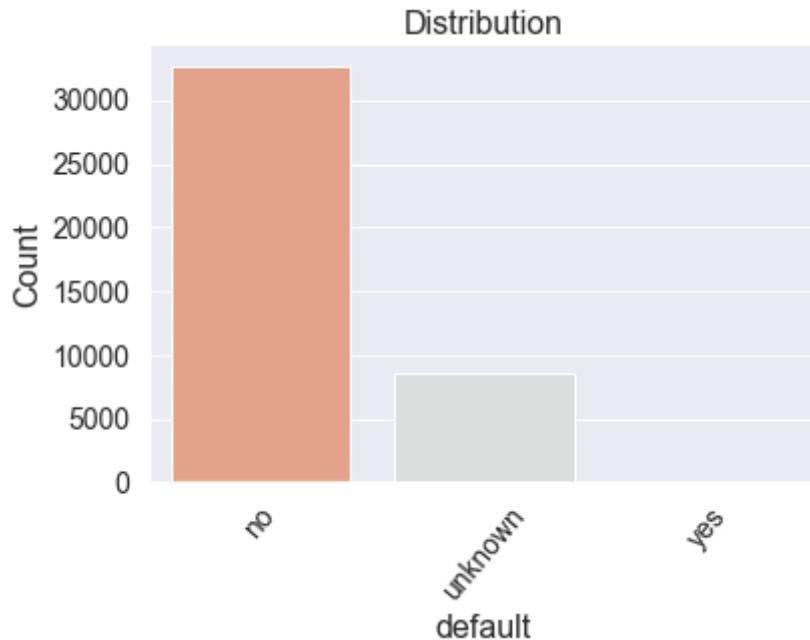
In terms of marital status, there were twice as many married people as single and one-fourth divorced.

```
In [70]: countplot('education')
```



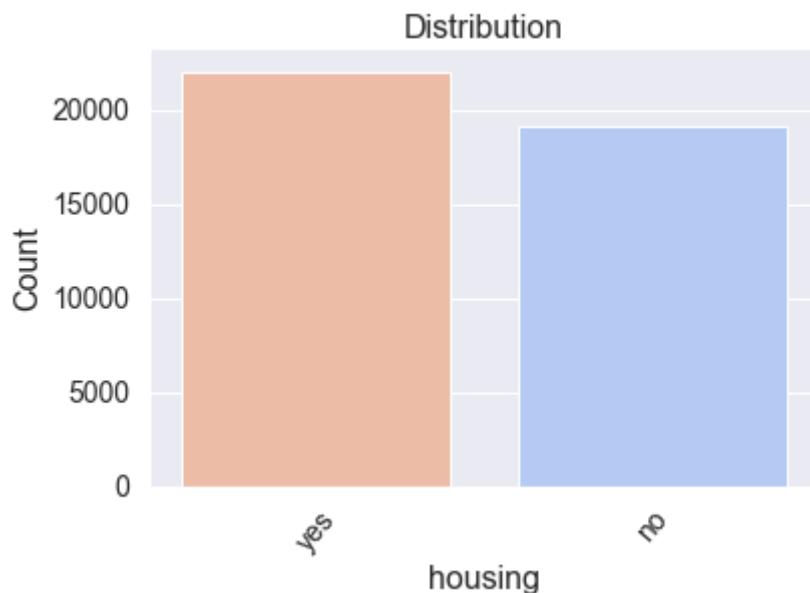
There were twice as many individuals with a university degree who were contacted than individuals who had 9 years of schooling.

```
In [71]: countplot('default')
```



Most individuals who were contacted did not default on their credit.

```
In [72]: countplot('housing')
```



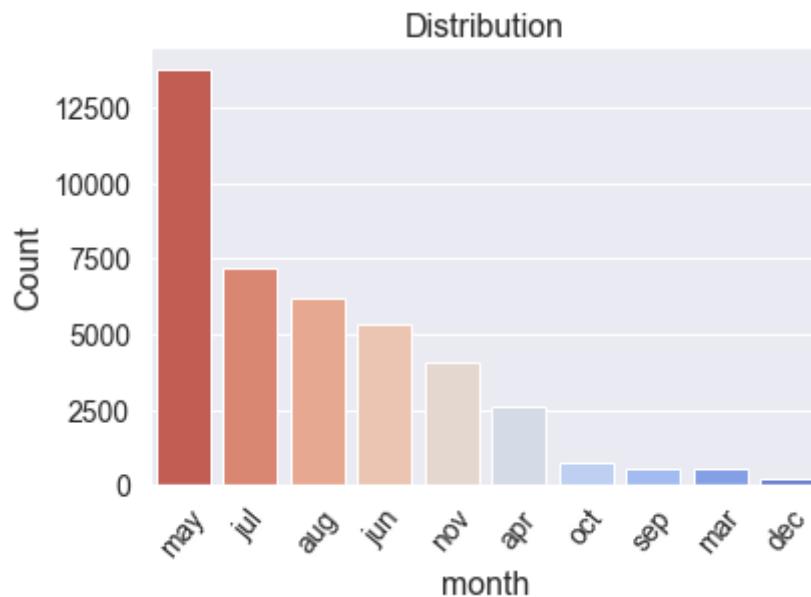
Individuals with or without housing loans were contacted in a similar population size.

```
In [73]: df_bank['month'].value_counts()
```

```
Out[73]: may      13767
       jul      7169
       aug      6176
       jun      5318
       nov      4100
       apr      2631
       oct      717
       sep      570
```

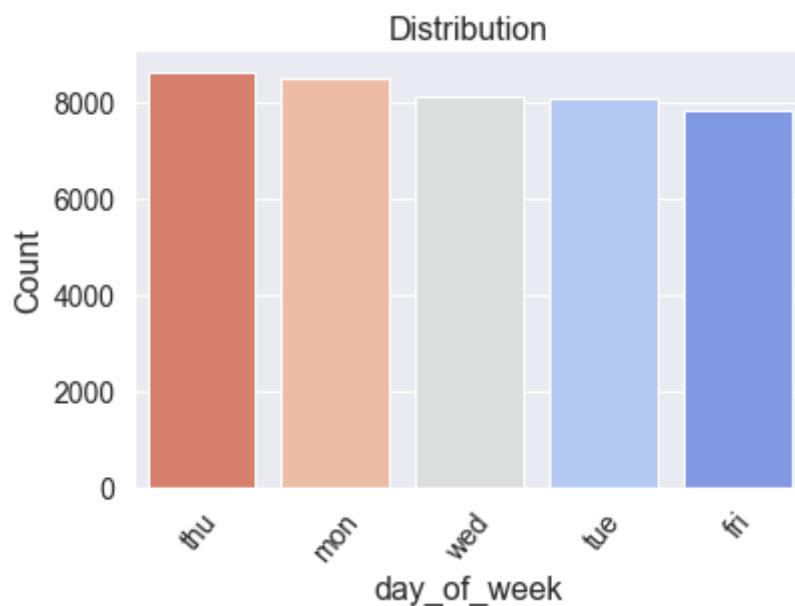
```
mar      546
dec     182
Name: month, dtype: int64
```

In [74]: `countplot('month')`



While May has been the month where most of the calls were made (twice as much as Jan, Aug, and June) the days of the week were evenly distributed.

In [75]: `countplot('day_of_week')`



Individuals were contacted during the week, (roughly equaliy distributed Mon. to Fri.)

Individuals subscribe from each segment (in terms of percentage)

In [76]: `# Building a function that first extracts a data set with a given column and`

```
# a subscribed column.

def statPlot(colname):

    ##### carte df #####
    df = df_bank[[colname, 'subscribed']]

    df = df.groupby([colname,
                    'subscribed']).size().reset_index().pivot(columns=
                                                    index=co
                                                    values=0)

    df['percentage_of_subscribers'] = df['yes']*100/(df['no']+ df['yes'])
    # df_job['percentage_of_unsubscribers']= df_job['no']/(df_job['no']+ df_job[
    df.dropna(inplace=True)
    df= df.sort_values('percentage_of_subscribers').reset_index()
    df[colname] = df[colname].astype('str')

    ##### PLOT #####
    plt.figure(figsize=(15,5))
    ax = sns.barplot(df[colname],df['percentage_of_subscribers'],
                      palette="vlag")
    for bars in ax.containers:
        ax.bar_label(bars, fmt=".2f%%")
    plt.xticks(rotation = 50)
    plt.show()
```

In [77]:

```
# list all the columns
list(df_bank)
```

Out[77]:

```
['age',
 'job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'poutcome',
 'cons_price_idx',
 'cons_conf_idx',
 'euribor3m',
 'subscribed',
 'bins_age',
 'age_group']
```

In [78]:

```
# Creating a list so it would be easy to grab to run it in the function's loop.

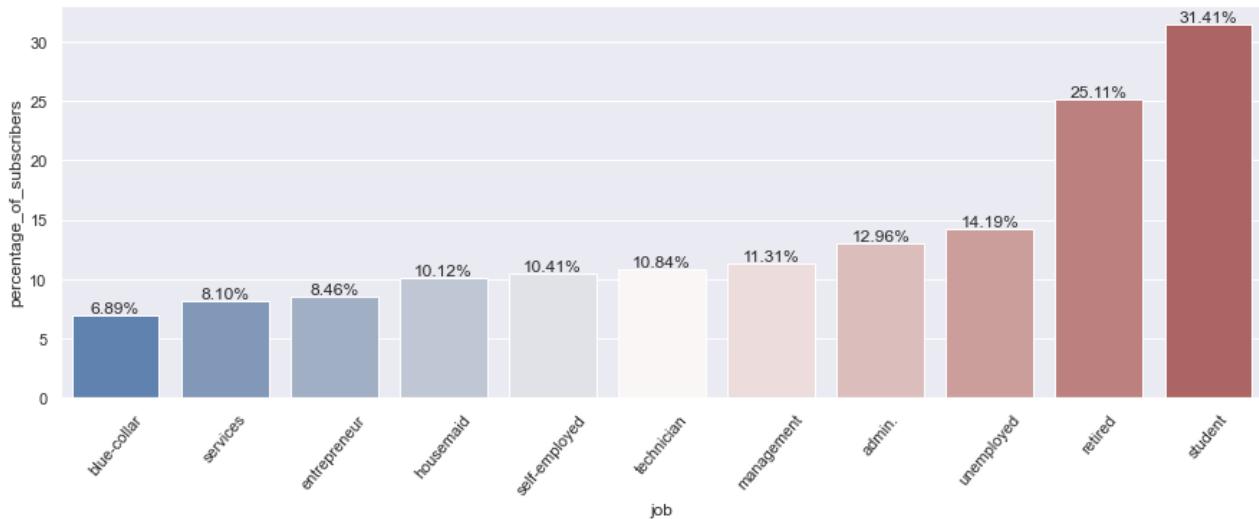
imp_col= ['job',
          'marital',
          'education',
```

```
'default',
'housing',
'loan',
'contact',
'month',
'day_of_week',
'campaign',
'age_group']
```

In [79]: df_bank[['job', 'subscribed']].value_counts()

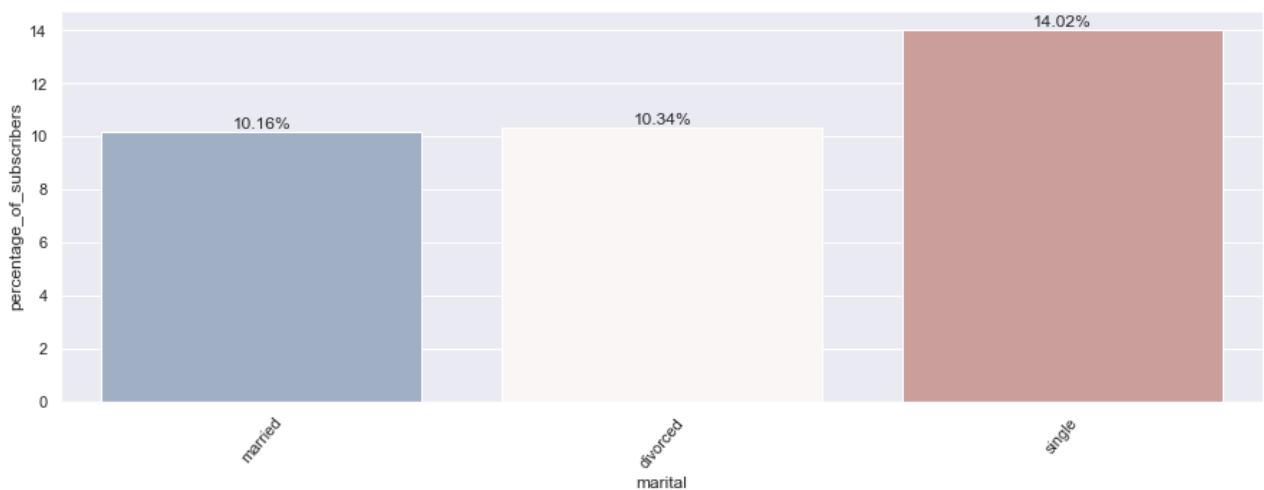
```
Out[79]: job      subscribed
admin.      no          9138
blue-collar no          8670
technician   no          6065
services    no          3674
management  no          2618
admin.      yes         1361
entrepreneur no         1342
retired     no          1306
self-employed no         1283
housemaid   no          959
unemployed  no          877
technician   yes         737
blue-collar yes         642
student     no          605
retired     yes         438
management  yes         334
services    yes         324
student     yes         277
self-employed yes        149
unemployed  yes         145
entrepreneur yes        124
housemaid   yes         108
dtype: int64
```

In [80]: *# Running function for the 'job' column*
statPlot('job')

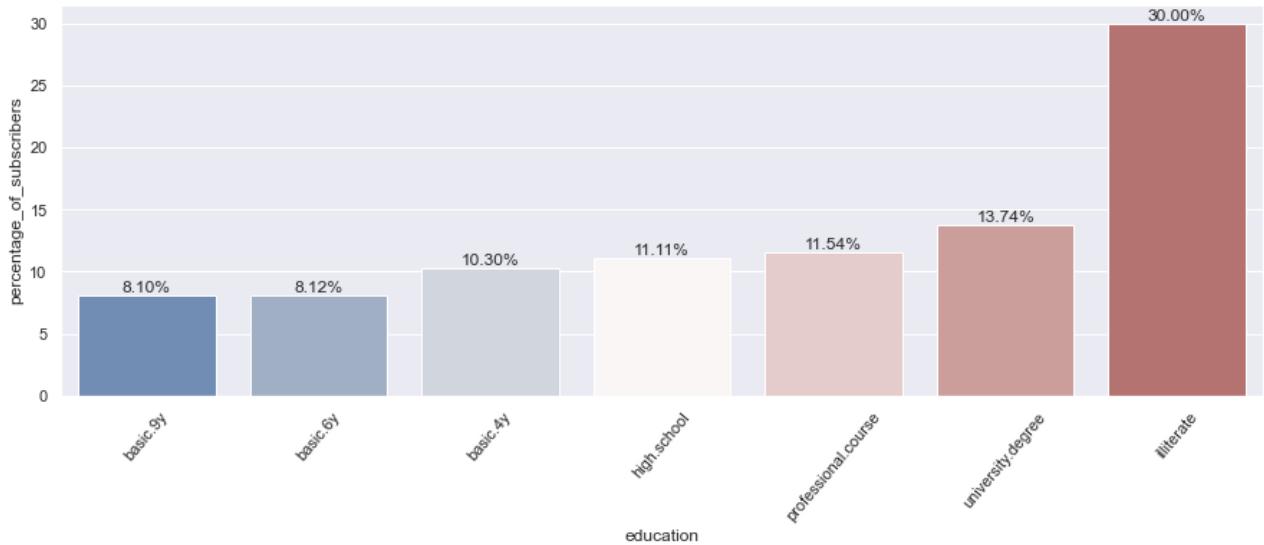


In [81]: *# Running function for the 'marital' column*

```
statPlot('marital')
```



```
In [82]:  
# Running function for the 'education' column  
  
statPlot('education')
```



```
In [83]:  
# String to be searched in start of string  
# https://www.geeksforgeeks.org/python-pandas-series-str-count/  
  
search = "illiterate"  
  
# count of occurrence of a and creating new column  
df_bank["count"] = df_bank["education"].str.count(search, re.I)  
  
# display  
df_bank["count"].value_counts()
```

```
Out[83]: 0    41156  
1      20  
Name: count, dtype: int64
```

```
In [84]: print("of Illiterate:")
```

```
#20/41156 = 0.00048
20 *.3
```

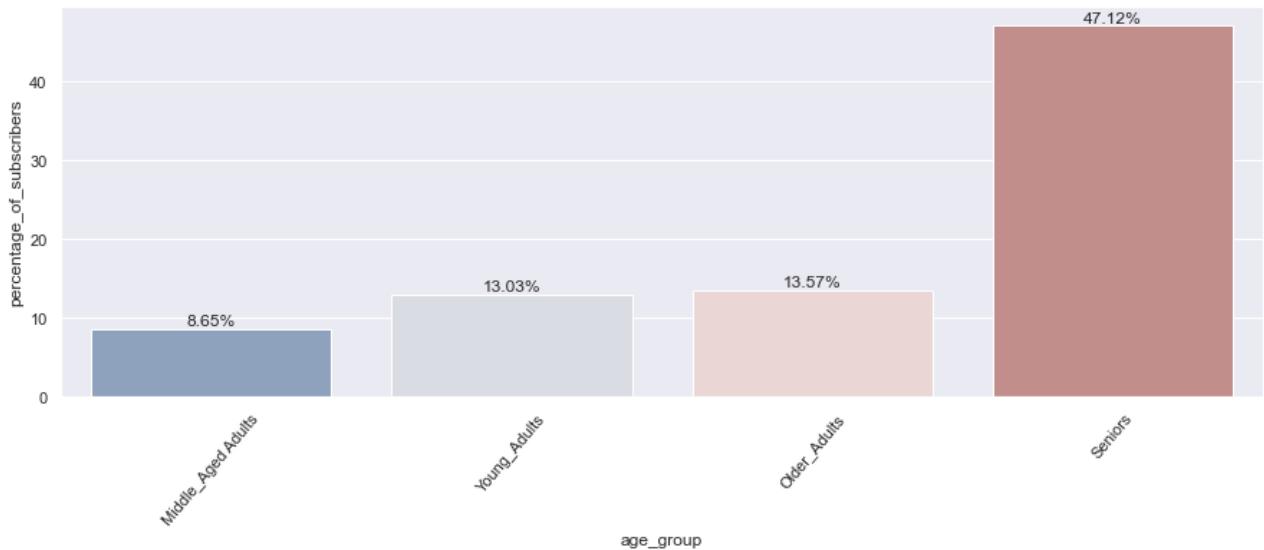
of Illiterate:

Out[84]: 6.0

In [85]: df_bank['education']

```
0          basic.4y
1      high.school
2      high.school
3          basic.6y
4      high.school
...
41183  professional.course
41184  professional.course
41185    university.degree
41186  professional.course
41187  professional.course
Name: education, Length: 41176, dtype: object
```

In [86]:
b# Running function for the 'age_group' column
 statPlot('age_group')

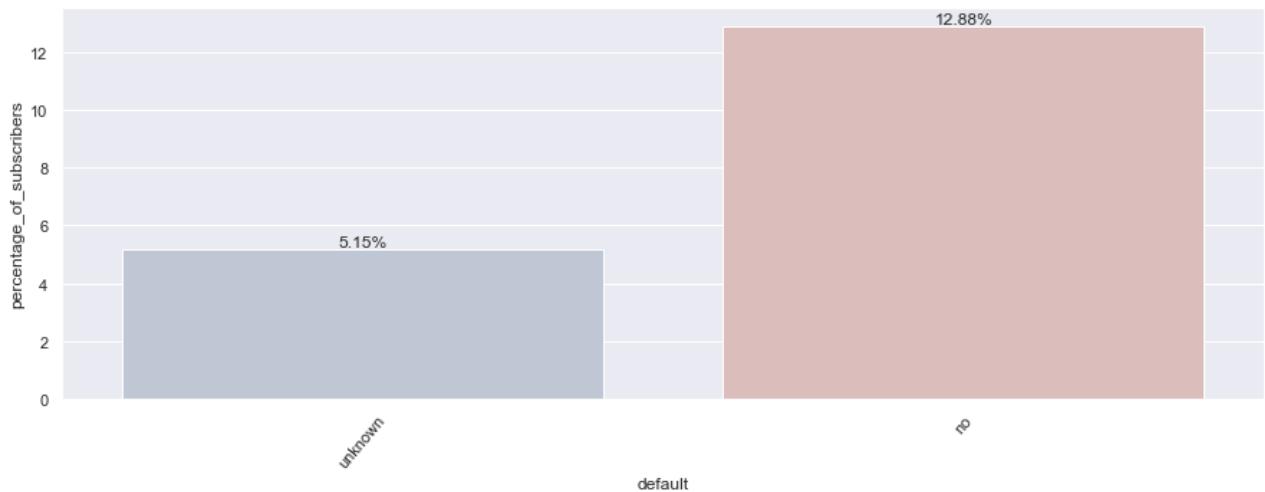


In [87]: df_bank[['age_group', 'subscribed']].value_counts()

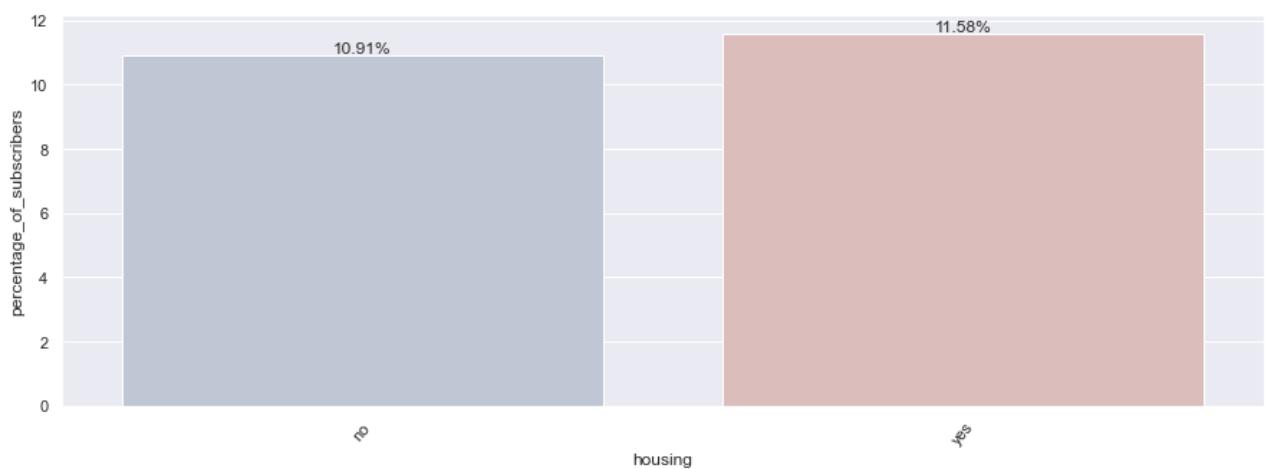
```
Out[87]:   age_group      subscribed
Middle_Aged_Adults  no        20277
Young_Adults        no        12829
Older_Adults        no         3082
Young_Adults        yes        1922
Middle_Aged_Adults yes        1920
Older_Adults        yes         484
Seniors             no          349
                           yes         311
dtype: int64
```

In [88]:
Running function for the 'default' column

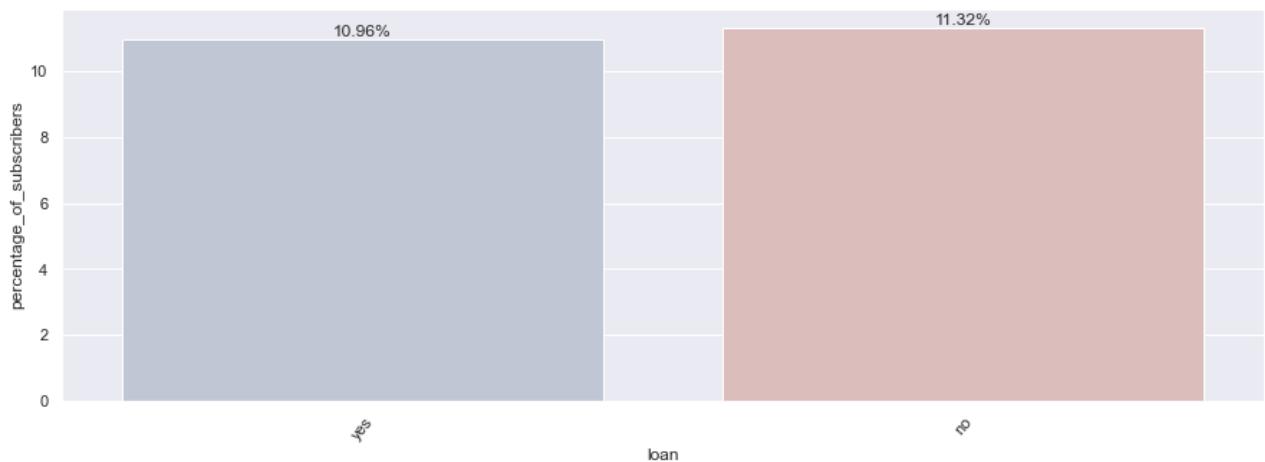
```
statPlot('default')
```



```
In [89]: statPlot('housing')
```



```
In [90]:  
# Running function for the 'loan' column  
statPlot('loan')
```



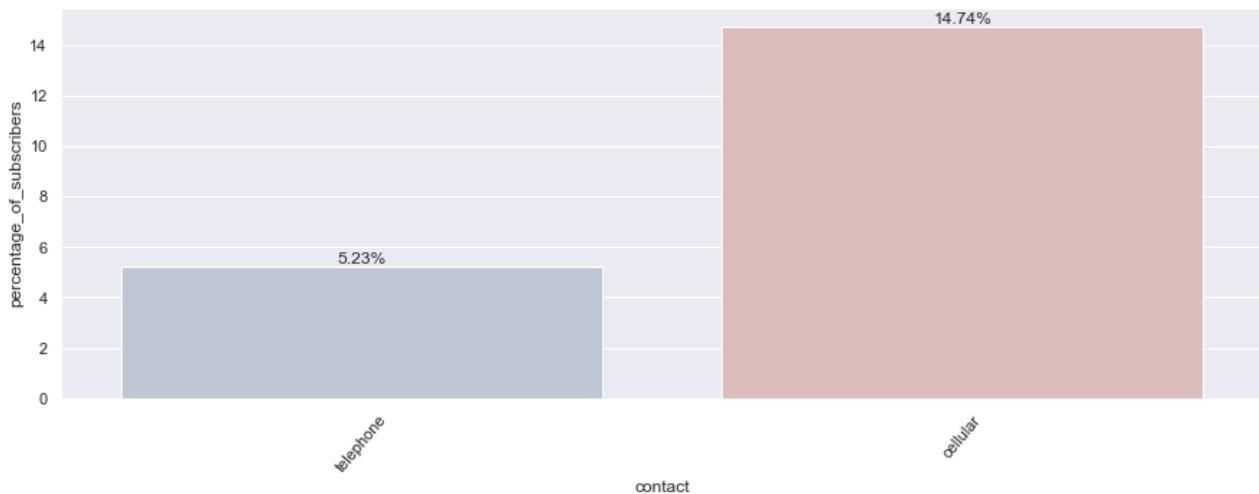
```
In [91]: df_bank[['contact', 'subscribed']].value_counts()
```

```
Out[91]: contact      subscribed
cellular    no           22283
telephone  no           14254
cellular    yes          3852
telephone  yes          787
dtype: int64
```

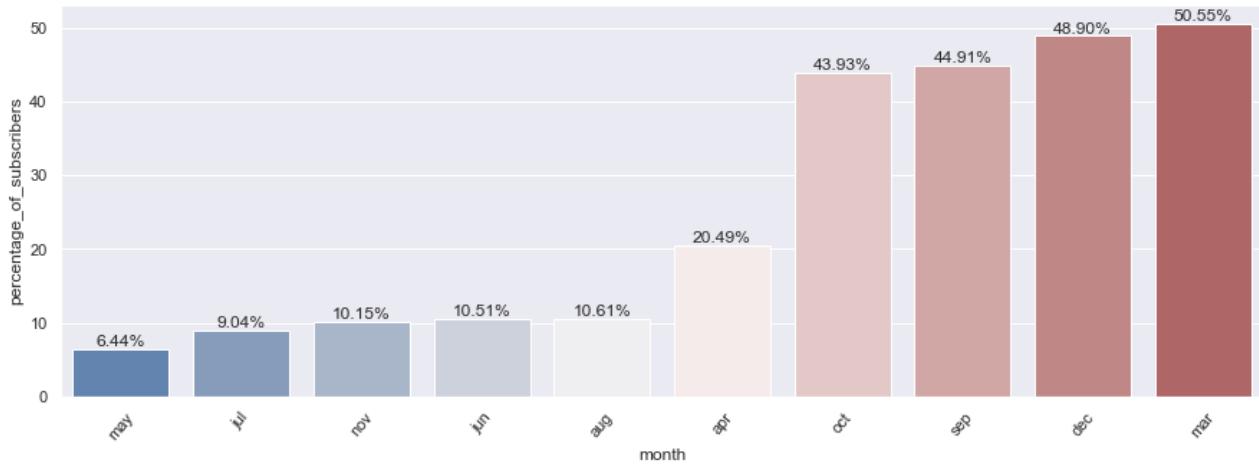
```
In [92]: df_bank[ 'contact' ].value_counts()
```

```
Out[92]: cellular     26135
telephone   15041
Name: contact, dtype: int64
```

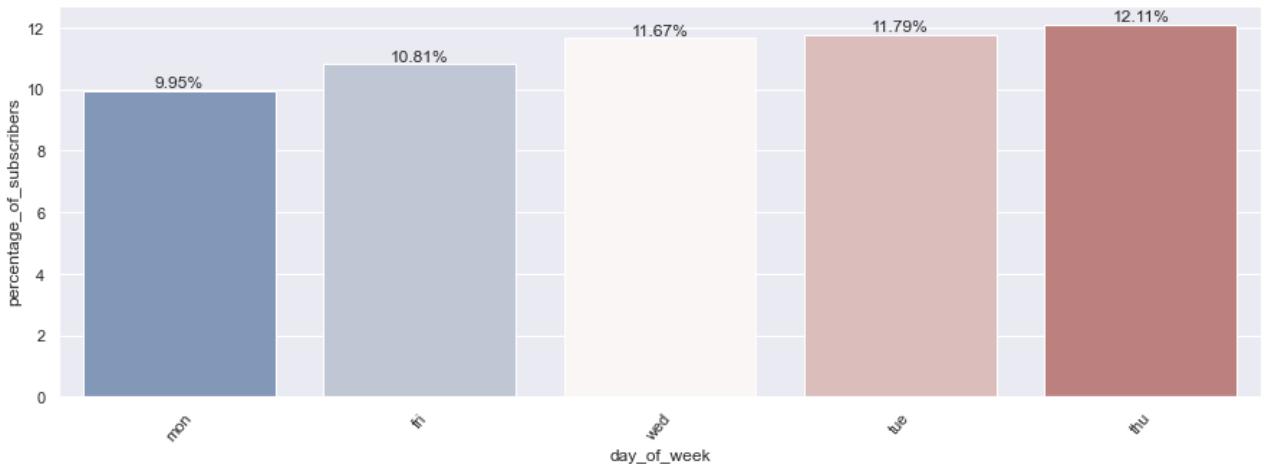
```
In [93]: # Running function for the 'contact' column
statPlot('contact')
```



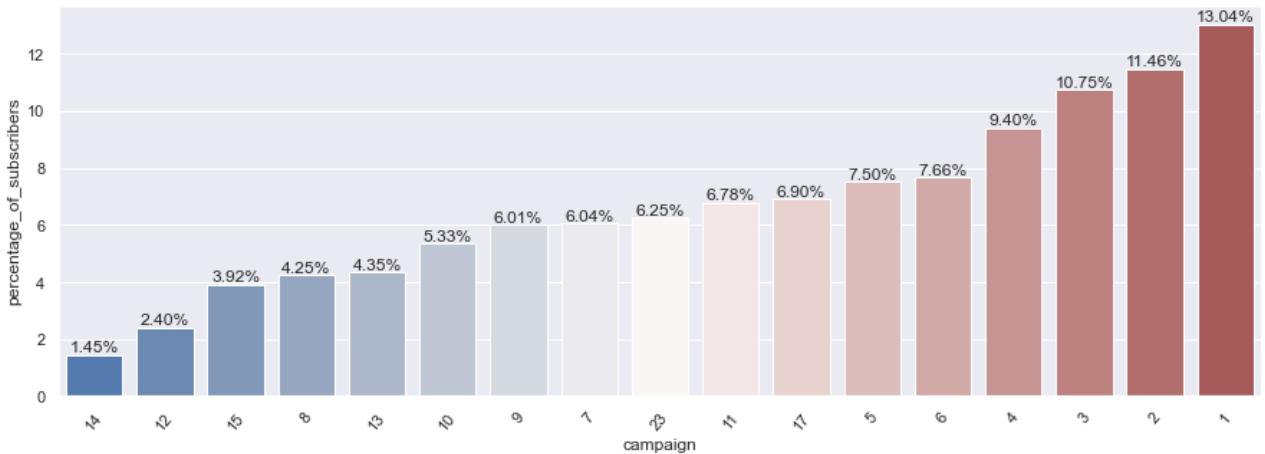
```
In [94]: # Running function for the 'month' column
statPlot('month')
```



```
In [95]: # Running function for the 'day_of_week' column
statPlot('day_of_week')
```



```
In [96]: # Running function for the 'campaign' column  
statPlot('campaign')
```



Notes on the graphs:

- Job: Top three subscribers were: Students 31.43%, Retired Individuals: 25.23% and unemployed 14.2%. Students were twice as likely to subscribe than unemployed individuals.
 - Marital Status: Didn't play a major role. There was not a significant difference among the individuals who were single, married, or divorced.
 - Education: There are six illiterate people included in this analysis. Therefore, there is not enough data to make any conclusion for that category.
 - The highest percentage was by individuals those holding a university degree of 13.72%, and people who have taken professional courses 11.35%.
 - Default: People who did not default on their loans were 12% more likely to subscribe than the individuals who did default on their loan.
 - Loan: Having a loan did not make a major difference.

- Contact: Contacting individuals through a cellular phone was twice as likely to lead to a subscription.
- Month: October, September, December, and March were the most successful months of the year. Prospects are three times more likely to subscribe to the bank campaign of term deposits.
- Day of the week: There was no major difference among the days of the week in terms of the likelihood of subscribing.
- Campaign: the highest percentage of subscribers came from individuals who were contacted 1-4 times.

Adding bins for the 'campaign' column - for the presentation

```
In [97]: df_bank[ "campaign" ].describe()
```

```
Out[97]: 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
```

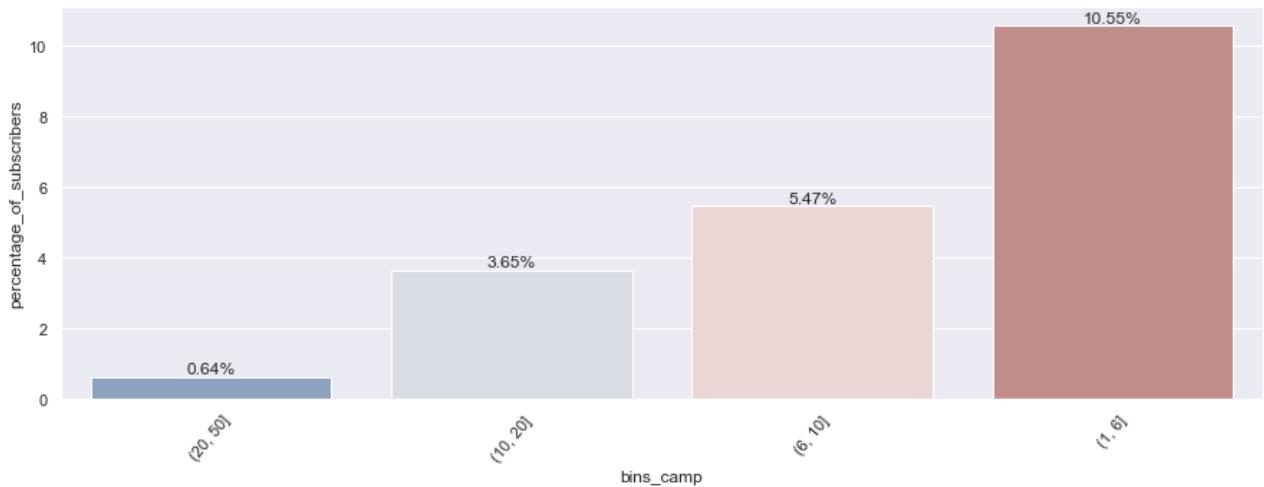
```
In [98]: #Here are the bins based on the values observed above.
# 5 values will result in 4 bins

bins = [1, 6, 10, 20, 50]

#We'll pd.cut method to separate data into bins.
df_bank[ 'bins_camp' ] = pd.cut(df_bank[ 'campaign' ], bins)

# I will use .cat.as_unordered() method transforming the data to
# ordered categories.
# bins_age_built = bins_yr_built.cat.as_unordered()
# bins_yr_built.head()
```

```
In [99]: statPlot( 'bins_camp' )
```



Building Classification Models

I will drop off the 'duration' column since the time of the call will not be known before contacting the prospect, and therefore including it in the model would create an unrealistic prediction. The variables included need to be known prior to the campaign call to assist businesses in targeting certain segments of the population.

In [100...]

```
df_bank.drop(['duration', 'bins_age', 'bins_camp'], axis=1, inplace=True)
```

In [101...]

```
# Changing 'Yes' to '1' and 'No' to '0' in order to have numerical values for
# our target variable.
dic_sub = {'yes':1, 'no':0}

df_bank['subscribed'] = df_bank['subscribed'].map(dic_sub)

df_bank['subscribed']
```

Out[101...]

```
0      0
1      0
2      0
3      0
4      0
 ..
41183    1
41184    0
41185    0
41186    1
41187    0
Name: subscribed, Length: 41176, dtype: int64
```

In [102...]

```
# Splitting my data frame into X( independent variables) and y (dependent/target

X = df_bank.drop("subscribed", axis=1)
y = df_bank["subscribed"]
```

In [103...]

```
# Look into object data
```

```
X.select_dtypes('object')
```

Out[103...]

	job	marital	education	default	housing	loan	contact	month	day_of_
0	housemaid	married	basic.4y	no	no	no	telephone	may	
1	services	married	high.school	unknown	no	no	telephone	may	
2	services	married	high.school	no	yes	no	telephone	may	
3	admin.	married	basic.6y	no	no	no	telephone	may	
4	services	married	high.school	no	no	yes	telephone	may	
...
41183	retired	married	professional.course	no	yes	no	cellular	nov	
41184	blue-collar	married	professional.course	no	no	no	cellular	nov	
41185	retired	married	university.degree	no	yes	no	cellular	nov	
41186	technician	married	professional.course	no	no	no	cellular	nov	
41187	retired	married	professional.course	no	yes	no	cellular	nov	

41176 rows × 10 columns

In [104...]

```
# converting the data type into sting
df_bank['age_group'] = df_bank['age_group'].astype('str')
```

In [105...]

```
# Double checking for Null values.
X.isna().sum()
```

Out[105...]

age	0
job	0
marital	0
education	0
default	0
housing	0
loan	0
contact	0
month	0
day_of_week	0
campaign	0
pdays	0
previous	0
poutcome	0
cons_price_idx	0
cons_conf_idx	0
euribor3m	0
age_group	2
count	0
dtype:	int64

In [106...]

```
# Drop Null values from the 'age_group' column
df_bank = df_bank.dropna(axis=0, subset=['age_group'])
```

```
In [107...]: # Double check that there are no Null values
df_bank.isna().sum()
```

```
Out[107...]: age          0
job           0
marital        0
education      0
default         0
housing         0
loan            0
contact         0
month           0
day_of_week     0
campaign        0
pdays           0
previous        0
poutcome        0
cons_price_idx  0
cons_conf_idx   0
euribor3m       0
subscribed      0
age_group       0
count           0
dtype: int64
```

Score Function

We built a function that captures the classified models, trains them, and produces the scoring results (Accuracy, Precious Recall, and F1_Score). We added an “if statement” with a display element to have the option to not display when unnecessary.

```
In [108...]: # Building a function that will fit the model and then fit it to produce predict
def Train_Test_Scores(model, display=False):

    model.fit(X_train,y_train)
    y_preds = model.predict(X_test)

    # Store the score for later evaluation of the model.

    train_acc = model.score(X_train,y_train)
    test_acc = model.score(X_test,y_test)
    precision = precision_score(y_test,y_preds)
    recall = recall_score(y_test,y_preds)
    f1 = f1_score(y_test,y_preds)

    # Allowing the display to switch off for later on when I just need to functi
    # for other purposes like creating a data frame of the scoring.

    if display:
        print('Training_Accuracy:', train_acc)
        print('Test_Accuracy:', test_acc)

        print('Precision:', precision)
        print('Recall:', recall)
        print('F1_Score:', f1)
```

```
plot_confusion_matrix(model, X_test, y_test, cmap="Blues")
```

```
return train_acc,test_acc,precision,recall,f1
```

Note on Scaling:

We are scaling the data using the Standard Scaler method. Standardize the data by making the mean of the distribution zero and the majority of the data will be between -1 and 1.

In [109...]

```
# Splitting the data into X and y - y is the target variable.

X = df_bank.drop("subscribed", axis=1)
y = df_bank["subscribed"]

# Extracting the index in order to know which ones are the categorical variables
# Testing if it equals to '0' - object.
cat_indx = [indx for indx, tp in enumerate(X.dtypes) if tp=='O']

# Creating a separate list for numerical and categorical variables.

cat_col = list(X.select_dtypes('object'))
num_col = list(set(X) - set(cat_col))
```

SMOTE

Given that our data is unbalanced, where only 11% of the individuals subscribed and not even equality represented with the group that unsubscribed, it can lead to inaccurate results when we employ a predictive model. In such scenarios of class imbalance, it is recommended to use Synthetic Minority Over - Sampling Technique. We implemented SMOTE- Nominal and Continuous ('SMOTE-NC') since our data contains categorical and numerical data.

In [110...]

```
# Applying Smote

from imblearn.over_sampling import SMOTENC #importing SMOTENC

smt = SMOTENC(categorical_features=cat_indx, random_state=2)
X_smote, y_smote = smt.fit_resample(X, y)
```

In [111...]

```
y_smote.value_counts()
```

Out[111...]

0	36537
1	36537
Name: subscribed, dtype: int64	

In [187...]

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size
```

Focus on Accuracy

In this project, false positives and false negatives are both equally important.

Reaching out to a prospect incorrectly labeled as a potential subscriber will lead to the bank losing money on marketing efforts. At the same time, if a potential subscriber is labeled incorrectly as a non-subscribers and the bank will not contact them – the bank would lose business with an individual that would otherwise subscribe to a term deposit.

Given that the data is now balanced, I can rely on the accuracy score for model valuation.

Pipeline

We inserted the pipeline as part of a function so that it can loop around when we employed the various classified models.

```
In [113...]: # Pipeline function

def custom_pipeline(clf, display=False):

    num_attribs = num_col # Continuous columns
    cat_attribs = cat_col # Categorical columns

    # When the data passed through the pipeline,
    # it went through the first step of StandardScaler() and OneHotEncoder().
    preprocessor = ColumnTransformer([
        ("num", StandardScaler(), num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

    model = Pipeline(steps=[("preprocessor", preprocessor), ('model', clf)])
    model.fit(X_train, y_train)

    # The adjusted data fit the model/classifier and ultimately produced
    # predictive values for the testing and training data and reverted back to
    # the model scoring.

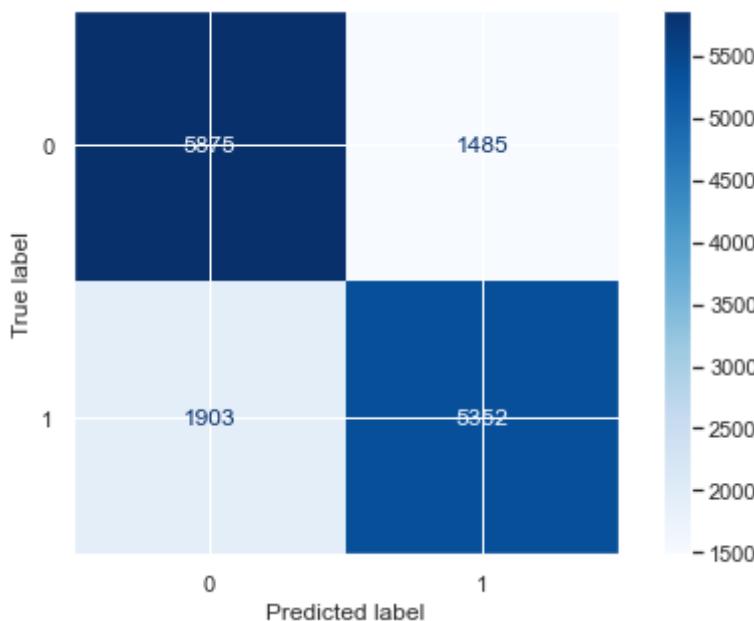
    #####
    y_pred = model.predict(X_test)
    X_probs = model.predict_proba(X_test)
    train_acc, test_acc, precision, recall, f1 = Train_Test_Scores(model, display)

    print("")
    return train_acc, test_acc, precision, recall, f1, y_pred, X_probs[:, 1], model
```

Logistic Regression Model

```
In [114...]: # Running logistic regression
lr = custom_pipeline(LogisticRegression(), display=True)
```

```
Training_Accuracy: 0.7674780615474093
Test_Accuracy: 0.7681833732466644
Precision: 0.782799473453269
Recall: 0.7376981392143349
F1_Score: 0.7595799034913425
```

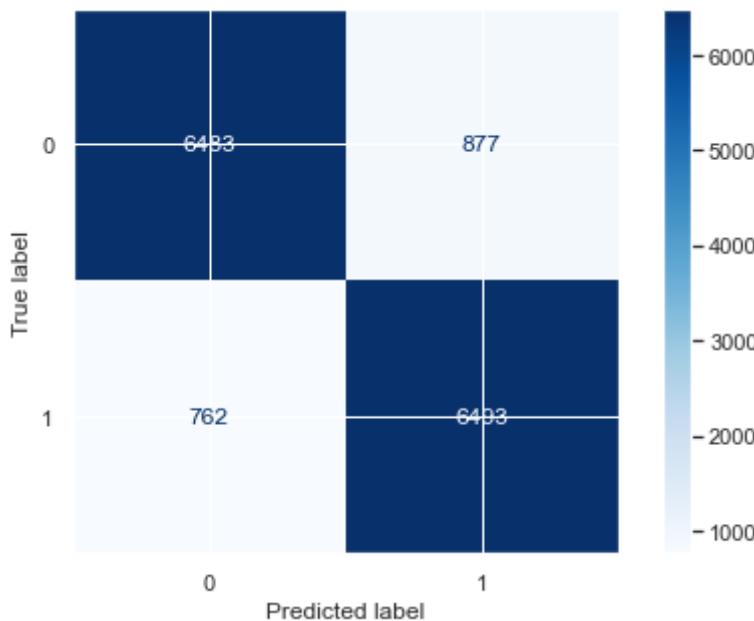


```
In [115]: # print(classification_report(y_test, val[-1]))
```

Decision Tree Model

```
In [116]: # Running Decision Tree
dt = custom_pipeline(DecisionTreeClassifier(), display= True)
```

Training_Accuracy: 0.9958090285499239
 Test_Accuracy: 0.8878549435511461
 Precision: 0.8810040705563094
 Recall: 0.8949689869055824
 F1_Score: 0.887931623931624

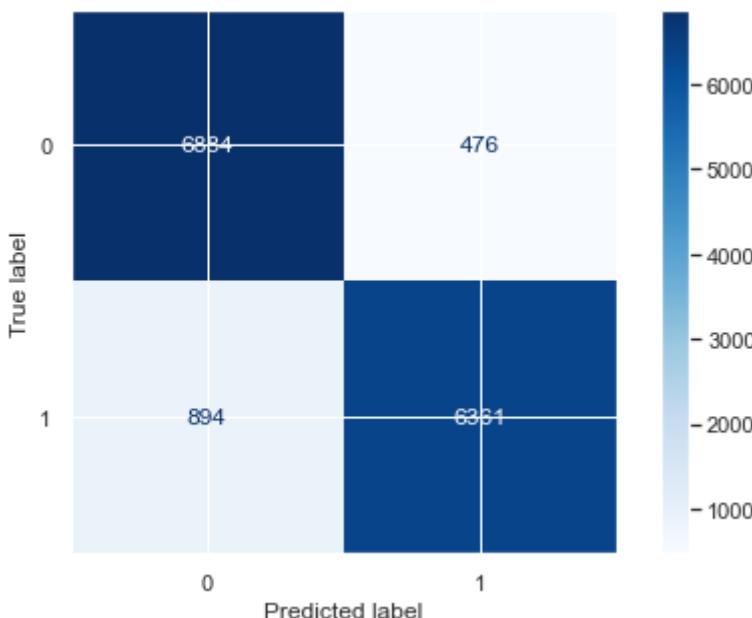


XG Boost Model

```
In [117]:
```

```
# Running XG Boost
xgboost = custom_pipeline(XGBClassifier(n_jobs=-1), display= True)
```

```
[12:23:52] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-4
3e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Startin
g in XGBoost 1.3.0, the default evaluation metric used with the objective 'binar
y:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if
you'd like to restore the old behavior.
[12:23:56] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-4
3e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Startin
g in XGBoost 1.3.0, the default evaluation metric used with the objective 'binar
y:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if
you'd like to restore the old behavior.
Training_Accuracy: 0.9206623445491712
Test_Accuracy: 0.9062606910708176
Precision: 0.9303788211203744
Recall: 0.8767746381805651
F1_Score: 0.9027817201248935
```

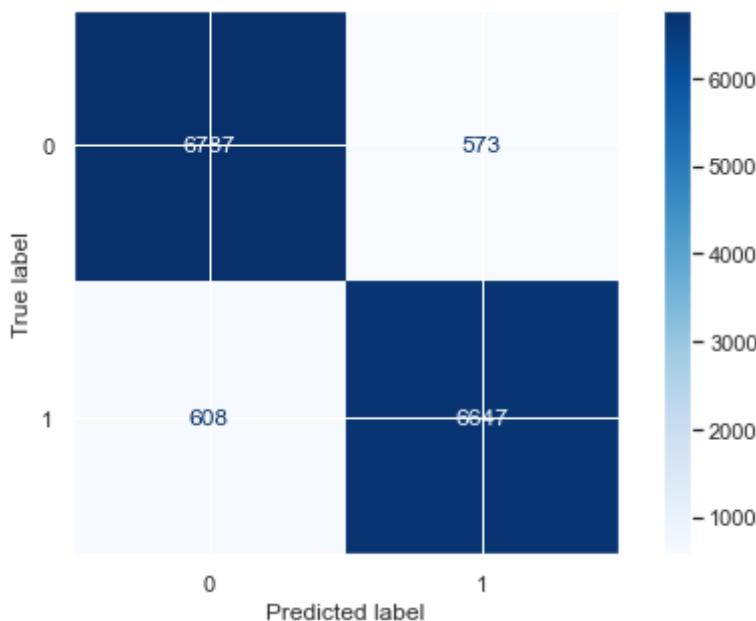


Random Forest Model

In [118...]

```
# Running Random Forest
rf = custom_pipeline(RandomForestClassifier(), display= True)
```

```
Training_Accuracy: 0.9958090285499239
Test_Accuracy: 0.9191926103318508
Precision: 0.9206371191135734
Recall: 0.9161957270847692
F1_Score: 0.9184110535405872
```



In [119...]

rf

Out[119...]

```
(0.9958090285499239,
 0.9191926103318508,
 0.9206371191135734,
 0.9161957270847692,
 0.9184110535405872,
 array([0, 0, 1, ..., 0, 1, 0]),
 array([0.04 , 0.39 , 0.91 , ..., 0.125, 0.73 , 0.04 ]),
 Pipeline(steps=[('preprocessor',
                  ColumnTransformer(transformers=[('num', StandardScaler(),
                                                     ['count', 'euribor3m',
                                                      'campaign', 'pdays',
                                                      'previous', 'cons_price_id
x'],
                                                     'cons_conf_idx', 'age']),
                           ['cat', OneHotEncoder(),
                            ['job', 'marital',
                             'education', 'default',
                             'housing', 'loan', 'contac
t'],
                            'month', 'day_of_week',
                            'poutcome',
                            'age_group'])])),
 ('model', RandomForestClassifier()))]))
```

In [120...]

pipe_models[-2]

function-pipeline

Evaluating Models

In [121...]

```
# Creating a data frame to collect all the results and evaluate them

models_DataFrame = pd.DataFrame(columns=['Model', 'Train_Accuracy', 'Test_Accuracy',
                                         'Precision', 'Recall', 'F1_score'])
```

```
list_models = [LogisticRegression(), DecisionTreeClassifier(),
               RandomForestClassifier(n_jobs=-1), XGBClassifier(n_jobs=-1)]

model_names = 'Lositic_Regression  Decision_Tree Random_Forest XGboost'.split()

from tqdm import tqdm
x_probs = []
pipe_models = []

for model, model_name in tqdm(zip(list_models, model_names)):
    train_acc, test_acc, precision, recall, f1, _, x_prob, pipe_model = custom_pipeline
    x_probs.append(x_prob)
    pipe_models.append(pipe_model)
    models_DataFrame.loc[len(models_DataFrame)] = [model_name, train_acc, test_acc]
```

1it [00:01, 1.80s/it]

2it [00:02, 1.44s/it]

3it [00:05, 2.14s/it]

[12:24:17] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

[12:24:21] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

4it [00:13, 3.45s/it]

In [122...]

```
# Display data frame
models_DataFrame
```

Out[122...]

	Model	Train_Accuracy	Test_Accuracy	Precision	Recall	F1_score
0	Lositic_Regression	0.767478	0.768183	0.782799	0.737698	0.759580
1	Decision_Tree	0.995809	0.887444	0.881840	0.892901	0.887336
2	Random_Forest	0.995792	0.918235	0.919319	0.915644	0.917478
3	XGboost	0.920662	0.906261	0.930379	0.876775	0.902782

In [123...]

```
# Checking which one was the top performer
models_DataFrame.sort_values('Test_Accuracy', ascending=False).values[0][0]
```

Out[123...]

'Random_Forest'

In [124...]

```
# Preparing for plotting x and y axis
x_plot = models_DataFrame["Model"]
y_plot = models_DataFrame["Test_Accuracy"]
```

In [125...]

```
# Plotting model results

fig, ax = plt.subplots(figsize=(12,6))

sns.barplot(x=x_plot, y=y_plot)

# Label and define fontsize for main and axis titles.

plt.xlabel('Model', fontsize=18)
plt.ylabel('Accuracy Score', fontsize=22)
plt.title('Model Comparison', fontsize=35)
plt.tick_params(axis='both', which='major', labelsize=18)

for bars in ax.containers:
    ax.bar_label(bars, fmt=".2f%%")
#Set x-axis tick labels.

plt.tight_layout()
plt.show()
```



Hyperparameter Tuning using GridSearch

Xgboost - Hyperparameter Tuning

In [126...]

```
# Checking the location of the model and its keys

pipeline_xgboost = pipe_models[-1]
pipeline_xgboost .get_params().keys()
```

```
Out[126... dict_keys(['memory', 'steps', 'verbose', 'preprocessor', 'model', 'preprocessor_n_jobs', 'preprocessor_remainder', 'preprocessor_sparse_threshold', 'preprocessor_transformer_weights', 'preprocessor_transformers', 'preprocessor_verbose', 'preprocessor_num', 'preprocessor_cat', 'preprocessor_num_copy', 'preprocessor_num_with_mean', 'preprocessor_num_with_std', 'preprocessor_cat_cate'])
```

```
gories', 'preprocessor_cat_drop', 'preprocessor_cat_dtype', 'preprocessor_cat_handle_unknown', 'preprocessor_cat_sparse', 'model_objective', 'model_us_e_label_encoder', 'model_base_score', 'model_booster', 'model_colsample_bylevel', 'model_colsample_bynode', 'model_colsample_bytree', 'model_gamma', 'mode_l_gpu_id', 'model_importance_type', 'model_interaction_constraints', 'model_learning_rate', 'model_max_delta_step', 'model_max_depth', 'model_min_child_weight', 'model_missing', 'model_monotone_constraints', 'model_n_estimators', 'model_n_jobs', 'model_num_parallel_tree', 'model_random_state', 'model_reg_alpha', 'model_reg_lambda', 'model_scale_pos_weight', 'model_subsample', 'mod el_tree_method', 'model_validate_parameters', 'model_verbosity'])
```

In [127...]

```
# The below cells are commented since it takes a couple of hours to runthrough a

# params_xgBoost = {
#     'model_min_child_weight': [1, 5, 10, 15, 20],
#     'model_gamma': [0.5, 1, 1.5, 2, 5, 6, 7, 8],
#     'model_subsample': [0.6, 0.8, 1.0],
#     'model_colsample_bytree': [0.6, 0.8, 1.0],
#     'model_max_depth': [3, 4, 5, 6, 7, 8]
# }
```

In [128...]

```
# Using GridSearch to find the best parameters

# GriffSearch

# grid_clf_xgboost = GridSearchCV(pipeline_xgboost, params_xgBoost, scoring='acc
#                                     cv=3, n_jobs=-1, verbose=3)
```

In [129...]

```
# Fitting the model

# xgboost_tuned = grid_clf_xgboost.fit(X_train, y_train)
```

In [130...]

```
# Obtaining best parameters

# best_param = xgboost_tuned.best_params_
# best_param
```

In [131...]

```
best_param = {'model_max_depth': 8, 'model_min_child_weight': 1, 'model_subsa
```

In [132...]

```
# Storing the best parameters
# {'model_max_depth': 8, 'model_min_child_weight': 1, 'model_subsample': 0.8}
```

In [133...]

```
# Checking the tuned result for XG boost

model_tuned_pipeline_xgboost = custom_pipeline(XGBClassifier(**best_param), displ
```

```
[12:24:25] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-4
3e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:541:
Parameters: { model_max_depth, model_min_child_weight, model_subsample } might
not be used.
```

This may not be accurate due to some parameters are only used in language bindings but

passed down to XGBoost core. Or some parameters are not used but slip through this verification. Please open an issue if you find above cases.

[12:24:25] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

[12:24:29] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:541: Parameters: { model_max_depth, model_min_child_weight, model_subsample } might not be used.

This may not be accurate due to some parameters are only used in language bindings but

passed down to XGBoost core. Or some parameters are not used but slip through this verification. Please open an issue if you find above cases.

[12:24:29] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

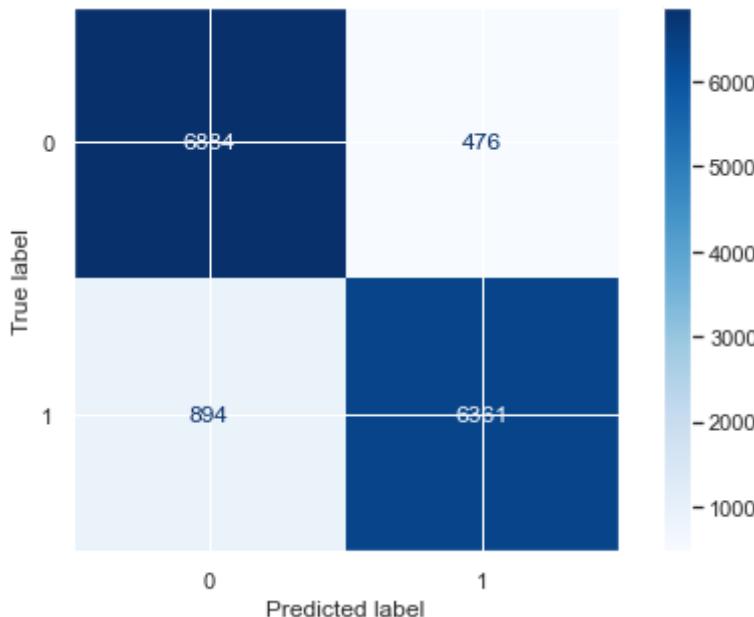
Training_Accuracy: 0.9206623445491712

Test_Accuracy: 0.9062606910708176

Precision: 0.9303788211203744

Recall: 0.8767746381805651

F1_Score: 0.9027817201248935



In [134...]

```
# Named_steps['model'] allows us to see the details of all the parameters,
model_tuned_pipeline_xgboost[-1].named_steps['model']
```

Out[134...]

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
```

```
min_child_weight=1, missing=nan, model__max_depth=8,
model__min_child_weight=1, model__subsample=0.8,
monotone_constraints='()', n_estimators=100, n_jobs=8,
num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
scale_pos_weight=1, subsample=1, tree_method='exact',
validate_parameters=1, verbosity=None)
```

In [135...]

```
# Looking at the steps
model_tuned_pipeline_xgboost
```

```
Out[135...](0.9206623445491712,
0.9062606910708176,
0.9303788211203744,
0.8767746381805651,
0.9027817201248935,
array([0, 1, 1, ..., 0, 1, 0]),
array([0.07020947, 0.5874478 , 0.8636114 , ..., 0.030653 , 0.8801508 ,
       0.09180609], dtype=float32),
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num', StandardScaler(),
                                                 ['count', 'euribor3m',
                                                 'campaign', 'pdays',
                                                 'previous', 'cons_price_id']),
                ('cat', OneHotEncoder(),
                 ['job', 'marital',
                  'education', 'default',
                  'housing', 'loan', 'contact',
                  'month', 'day_of_week',
                  'poutcome',
                  'age_group']))),
               ('model',
                 XGBClassifier(base_sco...
                           learning_rate=0.300000012, max_delta_step=0,
                           max_depth=6, min_child_weight=1, missing=nan,
                           model__max_depth=8, model__min_child_weight=1,
                           model__subsample=0.8, monotone_constraints='()', n_estimators=100, n_jobs=8, num_parallel_tree=1,
                           random_state=0, reg_alpha=0, reg_lambda=1,
                           scale_pos_weight=1, subsample=1,
                           tree_method='exact', validate_parameters=1,
                           verbosity=None))))
```

In [136...]

```
# Obtaining the probabilities for XG Boost
```

```
y_probs_xgboost_t = model_tuned_pipeline_xgboost[-2]
y_probs_xgboost_t
```

```
Out[136...](0.07020947, 0.5874478 , 0.8636114 , ..., 0.030653 , 0.8801508 ,
0.09180609], dtype=float32)
```

Random Forest - Hyperparameter Tuning

In [137...]

```
# Verifying that this is where Random Forest is stored.
pipe_models[-2]
```

```
Out[137...](Pipeline(steps=[('preprocessor',
                            ColumnTransformer(transformers=[('num', StandardScaler(),
```

```
[ 'count', 'euribor3m',
  'campaign', 'pdays',
  'previous', 'cons_price_idx',
  'cons_conf_idx', 'age']),
('cat', OneHotEncoder(),
 [ 'job', 'marital',
  'education', 'default',
  'housing', 'loan', 'contact',
  'month', 'day_of_week',
  'poutcome',
  'age_group'))]),
('model', RandomForestClassifier(n_jobs=-1))])
```

In [138...]: # Checking the location of the model and its keys

```
pipeline_rf = pipe_models[-2]
pipeline_rf.get_params().keys() # Checking the keys for the parameters
```

Out[138...]: dict_keys(['memory', 'steps', 'verbose', 'preprocessor', 'model', 'preprocessor_n_jobs', 'preprocessor_remainder', 'preprocessor_sparse_threshold', 'preprocessor_transformer_weights', 'preprocessor_transformers', 'preprocessor_verbose', 'preprocessor_num', 'preprocessor_cat', 'preprocessor_num_copy', 'preprocessor_num_with_mean', 'preprocessor_num_with_std', 'preprocessor_cat_categories', 'preprocessor_cat_drop', 'preprocessor_cat_dtype', 'preprocessor_cat_handle_unknown', 'preprocessor_cat_sparse', 'model_bootstrap', 'model_cc_p_alpha', 'model_class_weight', 'model_criterion', 'model_max_depth', 'model_max_features', 'model_max_leaf_nodes', 'model_max_samples', 'model_min_impurity_decrease', 'model_min_impurity_split', 'model_min_samples_leaf', 'model_min_samples_split', 'model_min_weight_fraction_leaf', 'model_n_estimators', 'model_n_jobs', 'model_oob_score', 'model_random_state', 'model_verbose', 'model_warm_start'])

In [139...]: # The below cells are commented since it takes a couple of hours to runthrough a

```
# param_grid_rf = {
#     'model_n_estimators': [100, 300, 600, 900, 1200],
#     'model_max_features': [2, 4, 6, 8, 10, 12, 14],
#     'model_max_depth' : [50, 100, 200],
#     'model_criterion' :['gini', 'entropy']
# }
```

In [140...]: # # GriffSearch

```
# grid_clf_rf = GridSearchCV(pipeline_rf, param_grid_rf, scoring='accuracy',
#                             cv=3, n_jobs=-1, verbose=2)
```

In [141...]: # grid_clf_rf.fit(X_train, y_train)

In [142...]: # best_params_rf = grid_clf_rf.best_params_

In [143...]: # Using the best parameters

```
best_params_rf = {'model_criterion': 'gini',
  'model_max_depth': 200,
  'model_max_features': 14,
  'model_n_estimators': 900}
```

```
In [144...]: # Checking Parameters  
best_params_rf
```

```
Out[144...]: {'model__criterion': 'gini',  
              'model__max_depth': 200,  
              'model__max_features': 14,  
              'model__n_estimators': 900}
```

```
In [145...]: # Removing '_' - to match the code out side of the pipeline  
best_params_rf = {k.split('__')[1]:best_params_rf[k] for k in best_params_rf}
```

```
In [146...]: # RandomForestClassifier(criterion='gini', max_depth=200, )  
  
# {'model__criterion': 'gini',  
#  'model__max_depth': 200,  
#  'model__max_features': 14,  
#  'model__n_estimators': 900}
```

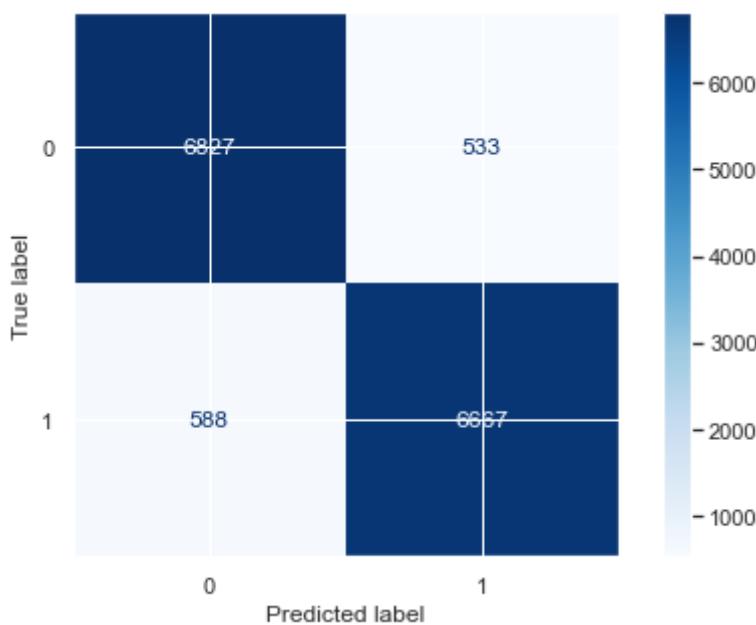
```
In [147...]: # Applying best parameters  
clf_rf = RandomForestClassifier(**best_params_rf)
```

```
In [148...]: # Checking if worked  
clf_rf
```

```
Out[148...]: RandomForestClassifier(max_depth=200, max_features=14, n_estimators=900)
```

```
In [149...]: # Model tuned results  
model_tuned_pipeline_rf = custom_pipeline(clf_rf, display=True)
```

```
Training_Accuracy: 0.9958090285499239  
Test_Accuracy: 0.9232979815258296  
Precision: 0.9259722222222222  
Recall: 0.9189524465885596  
F1_Score: 0.9224489795918368
```



Before and After Hyper Tuning - Receiver Operating Characteristic ("ROC")

```
In [150...]: # Displaying the results step by step
model_tuned_pipeline_rf
```

```
Out[150...]: (0.9958090285499239,
 0.9232979815258296,
 0.9259722222222222,
 0.9189524465885596,
 0.9224489795918368,
 array([0, 0, 1, ..., 0, 1, 0]),
 array([0.05888889, 0.25      , 0.89444444, ..., 0.07525397, 0.72768519,
        0.04      ]),
 Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num', StandardScaler(),
                                                     ['count', 'euribor3m',
                                                      'campaign', 'pdays',
                                                      'previous', 'cons_price_id'],
                                                     'cons_conf_idx', 'age']),
                           ['cat', OneHotEncoder(),
                            ['job', 'marital',
                             'education', 'default',
                             'housing', 'loan', 'contact',
                             'month', 'day_of_week',
                             'poutcome',
                             'age_group'])])),
              ('model',
               RandomForestClassifier(max_depth=200, max_features=14,
                                      n_estimators=900))]))
```

```
In [151...]: list_models
```

```
Out[151...]: [LogisticRegression(),
```

```
DecisionTreeClassifier(),
RandomForestClassifier(n_jobs=-1),
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=-1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [152...]: # Random Forest - Tuned with best parameters
x_probs[2]

Out[152...]: array([0.1 , 0.26 , 0.93 , ..., 0.06333333, 0.73333333,
 0.02])

In [153...]: #NEED TO CHANE THE MODLES ONLY rf_recall_t is correct

In [154...]: # Checking x_probs for Randon Forest
x_probs

Out[154...]: [array([0.11788034, 0.80084101, 0.43681956, ..., 0.09284231, 0.88515423,
 0.166515]),
 array([0., 1., 1., ..., 0., 0., 0.]),
 array([0.1 , 0.26 , 0.93 , ..., 0.06333333, 0.73333333,
 0.02]),
 array([0.07020947, 0.5874478 , 0.8636114 , ..., 0.030653 , 0.8801508 ,
 0.09180609], dtype=float32)]

ROC PLOT

In [155...]: # Locating Random Forest without tuning
x_probs[3]

Out[155...]: array([0.07020947, 0.5874478 , 0.8636114 , ..., 0.030653 , 0.8801508 ,
 0.09180609], dtype=float32)

In [156...]: # Locating Random Forest with tuning
model_tuned_pipeline_rf[6]

Out[156...]: array([0.05888889, 0.25 , 0.89444444, ..., 0.07525397, 0.72768519,
 0.04])

In [157...]: x_probs[-3]

Out[157...]: array([0., 1., 1., ..., 0., 0., 0.])

In [158...]: model_tuned_pipeline_xgboost[6]

Out[158...]: array([0.07020947, 0.5874478 , 0.8636114 , ..., 0.030653 , 0.8801508 ,
 0.09180609], dtype=float32)

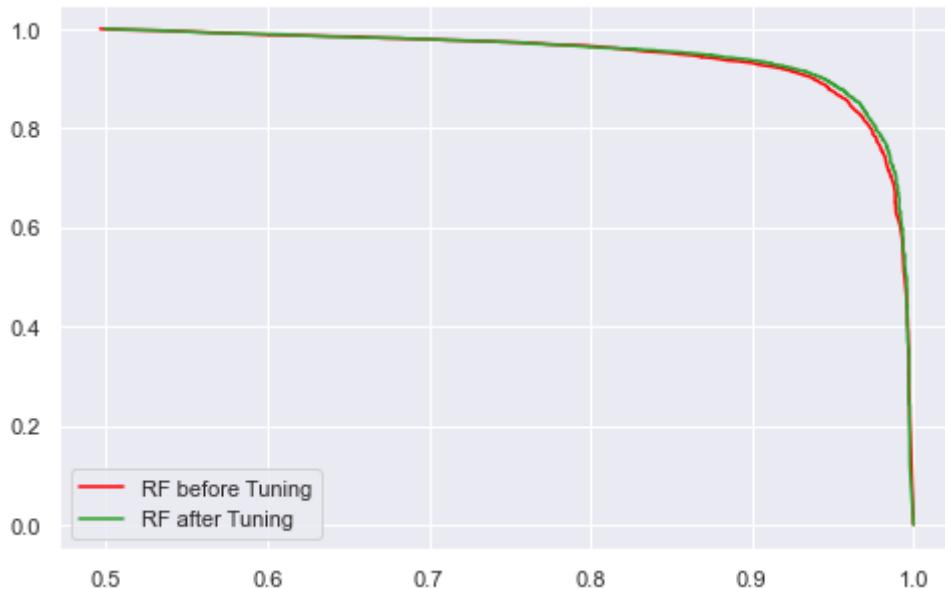
```
In [159... # Generate probabilities to plot for Random Forest
rf_precision, rf_recall, rf_thresholds = precision_recall_curve(y_test, x_probs[2])
rf_precision_t, rf_recall_t, rf_threshold_t = precision_recall_curve(y_test, mod
```

```
In [160... # Generate probabilities to plot for XgBoost
xg_precision, xg_recall, xg_thresholds = precision_recall_curve(y_test, x_probs[1])
xg_precision_t, xg_recall_t, xg_threshold_t = precision_recall_curve(y_test, mod
```

```
In [161... print(' ROC for Random Forest Before and After Hyper Tunning ')
# Subplot for ROC
#fig, axs = (figsize=(15, 5))
#fig, axs = plt.subplots(1, 2, figsize=(15, 5))
# Random Forest Before Tuning Plot
plt.plot(rf_precision, rf_recall, color='red', label="RF before Tuning")

#Random Forest After Tuning Plot
plt.plot(rf_precision_t, rf_recall_t, 'tab:green', label="RF after Tuning")
plt.legend()
plt.show()
```

ROC for Random Forest Before and After Hyper Tunning



```
In [162... x_probs[2]
```

```
Out[162... array([0.1      , 0.26      , 0.93      , ..., 0.06333333, 0.73333333,
```

```
In [163...]: model_tuned_pipeline_rf[6]
```

```
Out[163...]: array([0.05888889, 0.25      , 0.89444444, ..., 0.07525397, 0.72768519,
       0.04      ])
```

Feature of Importance

Feature of Importance for Random Forst

```
In [164...]: # Step 5 - predict values
pipe_models[-2]
```

```
Out[164...]: Pipeline(steps=[('preprocessor',
                           ColumnTransformer(transformers=[('num', StandardScaler(),
                                                             ['count', 'euribor3m',
                                                              'campaign', 'pdays',
                                                              'previous', 'cons_price_idx',
                                                              'cons_conf_idx', 'age']),
                                                       ('cat', OneHotEncoder(),
                                                             ['job', 'marital',
                                                               'education', 'default',
                                                               'housing', 'loan', 'contact',
                                                               'month', 'day_of_week',
                                                               'poutcome',
                                                               'age_group']))]),
                           ('model', RandomForestClassifier(n_jobs=-1))])
```

```
In [165...]: # In order to obtain the column names, including the columns that have been dummied
# I would need to go back to X-train and use the get_dummies feature.
pipeline_col = num_col + list(pd.get_dummies(X_train[cat_col]))
pipeline_col
```

```
Out[165...]: ['count',
              'euribor3m',
              'campaign',
              'pdays',
              'previous',
              'cons_price_idx',
              'cons_conf_idx',
              'age',
              'job_admin.',
              'job_blue-collar',
              'job_entrepreneur',
              'job_housemaid',
              'job_management',
              'job_retired',
              'job_self-employed',
              'job_services',
              'job_student',
              'job_technician',
              'job_unemployed',
              'marital_divorced',
              'marital_married',
              'marital_single',
              'education_basic.4y',
              'education_basic.6y',
              'education_basic.9y',
              'education_high.school',
              'education_illiterate',
              'education_professional.course',
```

```
'education_university.degree',
'default_no',
'default_unknown',
'default_yes',
'housing_no',
'housing_yes',
'loan_no',
'loan_yes',
'contact_cellular',
'contact_telephone',
'month_apr',
'month_aug',
'month_dec',
'month_jul',
'month_jun',
'month_mar',
'month_may',
'month_nov',
'month_oct',
'month_sep',
'day_of_week_fri',
'day_of_week_mon',
'day_of_week_thu',
'day_of_week_tue',
'day_of_week_wed',
'poutcome_failure',
'poutcome_nonexistent',
'poutcome_success',
'age_group_Middle_Aged_Adults',
'age_group_Older_Adults',
'age_group_Seniors',
'age_group_Young_Adults']
```

In [166...]

#Double check x_train variables

x_train

Out[166...]

	age	job	marital	education	default	housing	loan	contact	month
26596	50	technician	married	high.school	no	yes	no	telephone	nov
20458	41	technician	married	professional.course	no	no	no	cellular	aug
11003	41	services	married	basic.9y	no	yes	no	telephone	jun
18253	44	admin.	divorced	high.school	no	yes	no	cellular	jul
14277	36	admin.	single	high.school	unknown	yes	yes	cellular	jul
...
37194	44	admin.	married	university.degree	no	yes	no	cellular	aug
6265	34	blue-collar	married	basic.9y	no	no	yes	telephone	may
54886	35	technician	married	professional.course	no	no	no	telephone	may

	age	job	marital	education	default	housing	loan	contact	month
860	40	management	married	university.degree	no	yes	no	telephone	may
15795	19	student	single	basic.9y	unknown	yes	no	cellular	jul

58459 rows × 19 columns

```
In [167...]: # Locate where model is stored in -2
rf_model = pipe_models[-2].named_steps['model']
rf_model
```

```
Out[167...]: RandomForestClassifier(n_jobs=-1)
```

```
In [168...]: # Another way to locate where model is stored in -2
pipe_models[-2].steps[1][1]
```

```
Out[168...]: RandomForestClassifier(n_jobs=-1)
```

```
In [169...]: #pipe_models[-2][-2].named_steps['model']
```

```
In [170...]: # Full pipeline overview
pipe_models
```

```
Out[170...]: Pipeline(steps=[('preprocessor',
                           ColumnTransformer(transformers=[('num', StandardScaler(),
                                                               ['count', 'euribor3m',
                                                                'campaign', 'pdays',
                                                                'previous', 'cons_price_id
                                                               x'],
                                                               'cons_conf_idx', 'age']),
                                         ('cat', OneHotEncoder(),
                                          ['job', 'marital',
                                           'education', 'default',
                                           'housing', 'loan', 'contac
                                         t'],
                                           'month', 'day_of_week',
                                           'poutcome',
                                           'age_group'))]),
                           ('model', LogisticRegression()))],
Pipeline(steps=[('preprocessor',
                           ColumnTransformer(transformers=[('num', StandardScaler(),
                                                               ['count', 'euribor3m',
                                                                'campaign', 'pdays',
                                                                'previous', 'cons_price_id
                                                               x'],
                                                               'cons_conf_idx', 'age']),
                                         ('cat', OneHotEncoder(),
                                          ['job', 'marital',
                                           'education', 'default',
                                           'housing', 'loan', 'contac
                                         t'],
                                           'month', 'day_of_week',
                                           'poutcome',
                                           'age_group'))]),
                           ('model', LogisticRegression())]))
```

```

'month', 'day_of_week',
'poutcome',
'age_group']))),
('model', DecisionTreeClassifier()))]),
Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('num', StandardScaler(),
['count', 'euribor3m',
'campaign', 'pdays',
'previous', 'cons_price_id
x',
'cons_conf_idx', 'age']),
('cat', OneHotEncoder(),
['job', 'marital',
'education', 'default',
'housing', 'loan', 'contac
t',
'month', 'day_of_week',
'poutcome',
'age_group'))])),
('model', RandomForestClassifier(n_jobs=-1))]),
Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('num', StandardScaler(),
['count', 'euribor3m',
'campaign', 'pdays',
'previous', 'cons_price_id
x',
'cons_conf_idx', 'age']),
('cat', OneHotEncoder(),
['job', 'marital',
'education', 'default',
'housing', 'loan', 'contac
t',
'month', 'day_of_week',
'poutcome',
'age_group'))])),
('model',
XGBClassifier(base_sco...
colsample_bytree=1, gamma=0, gpu_id=-1,
importance_type='gain',
interaction_constraints='',
learning_rate=0.300000012, max_delta_step=0,
max_depth=6, min_child_weight=1, missing=nan,
monotone_constraints='()', n_estimators=100,
n_jobs=-1, num_parallel_tree=1, random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
subsample=1, tree_method='exact',
validate_parameters=1, verbosity=None))))]

```

In [171... # Random Forest check for prediction
model_tuned_pipeline_rf[5]

Out[171... array([0, 0, 1, ..., 0, 1, 0])

In [172... # Store predictions
y_pred_rf = model_tuned_pipeline_rf[5]

In [173... # Store features
f_imp = clf_rf.feature_importances_

```
In [174...]: #f_imp = val[-1].steps[1][1].feature_importances_
```

```
In [175...]: # Create a data frame for the important features from the tuned Random Forest model

f_imp_df = pd.DataFrame({'imp':f_imp,'col':pipeline_col})
```

```
In [176...]: # Sort values

f_imp_df = f_imp_df.sort_values('imp',ascending=False).head(10)
```

```
In [177...]: # Display results for the important features

f_imp_df
```

```
Out[177...]:
```

	imp	col
1	0.249701	euribor3m
7	0.105155	age
2	0.051120	campaign
6	0.049933	cons_conf_idx
5	0.041324	cons_price_idx
37	0.032082	contact_telephone
36	0.031510	contact_cellular
3	0.021474	pdays
55	0.019058	poutcome_success
29	0.017027	default_no

```
In [178...]: # Display results for the important features

f_imp_df['col'].to_list()
```

```
Out[178...]: ['euribor3m',
 'age',
 'campaign',
 'cons_conf_idx',
 'cons_price_idx',
 'contact_telephone',
 'contact_cellular',
 'pdays',
 'poutcome_success',
 'default_no']
```

```
In [179...]: # Plot important features

plt.figure(figsize=(10,5))

sns.barplot(data=f_imp_df, x="col", y="imp")
```

```
plt.xlabel("Features", fontsize=22)

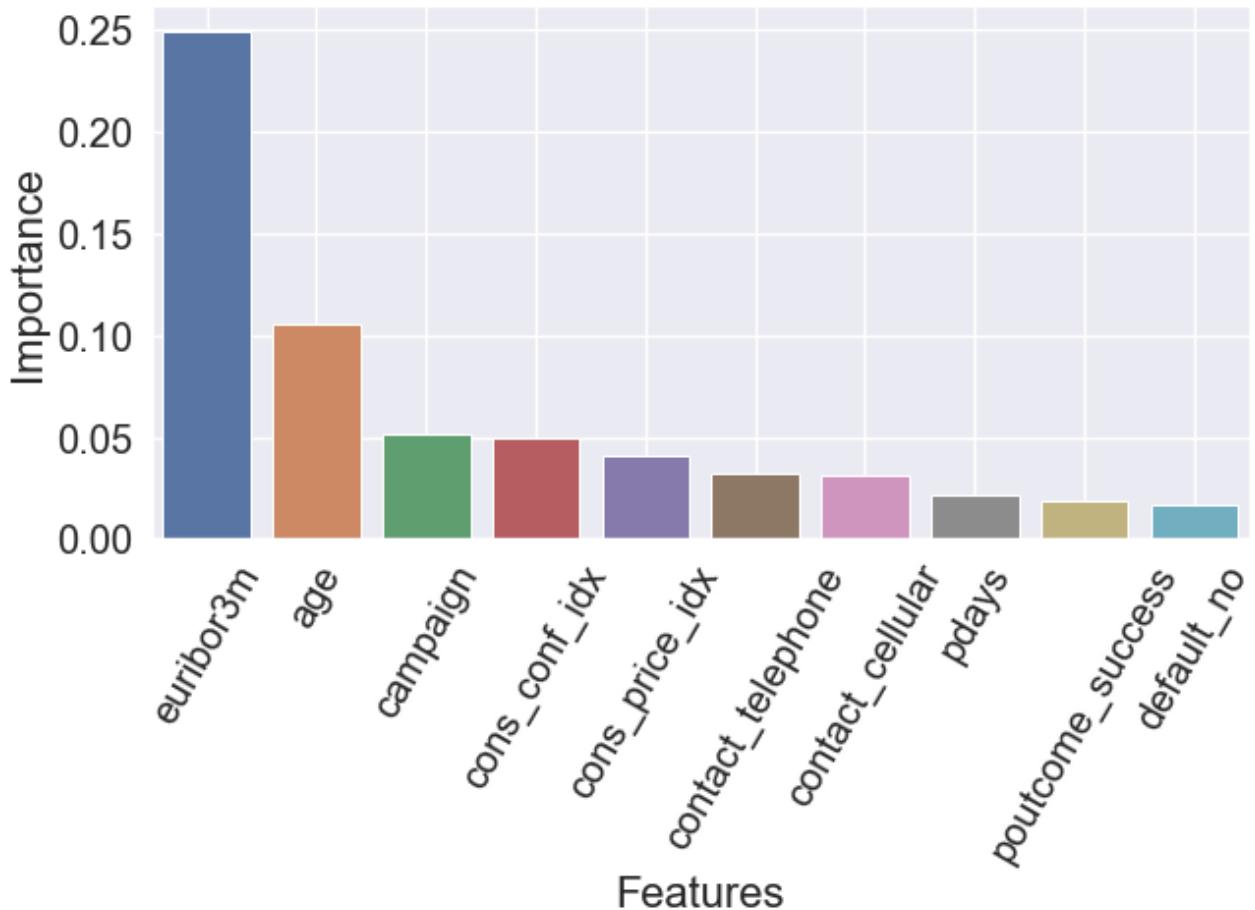
plt.ylabel('Importance', fontsize=22)

plt.xticks(rotation = 60, fontsize=20)

plt.yticks(rotation = 0, fontsize=20)

plt.grid(True, color = "white", linewidth = "1.4", linestyle = "-")

plt.show()
```



In [180]:

```
# Look into the classification report

report = classification_report(y_test, y_pred_rf)

print(report)
```

	precision	recall	f1-score	support
0	0.92	0.93	0.92	7360
1	0.93	0.92	0.92	7255
accuracy			0.92	14615
macro avg	0.92	0.92	0.92	14615
weighted avg	0.92	0.92	0.92	14615

Confusion Matrix

```
In [181... # Build a confutation matrix with percentage and storing
# "True Neg", "False Pos", "False Neg", "True Pos"

tn, fp, fn, tp = confusion_matrix(y_test,y_pred_rf).ravel()
```

```
In [182... # Display

tn, fp, fn, tp

Out[182... (6834, 526, 587, 6668)
```

```
In [183... # Confusion matrix

cf_matrix = confusion_matrix(y_test,y_pred_rf)
cf_matrix
```

```
Out[183... array([[6834, 526],
 [ 587, 6668]])
```

```
In [184... # Storing results

tn, fp, fn, tp = cf_matrix.ravel()
```

```
In [186... # Customizing the matrix

# Adding names
group_names = ["True Neg", "False Pos", "False Neg", "True Pos"]
group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]

# Formatting and adding percentages
group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()/np

# Connecting the names with the numbers

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names, group_counts, gro

# Apply heat map
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt=' ', cmap='Accent')
plt.show()
```



Let's look closer into how well the model predicted subscribers versus non-subscribers.

- We predicted correctly 93% of the time that an individual would not subscribe.
- We predicted correctly 92% of the time that an individual would subscribe.
- We predicted incorrectly 7% of the time that an individual would not subscribe.
- We predicted incorrectly 8% of the time that an individual would subscribe.

Conclusions:

In this project, we implemented data analysis and predictive classified models to predict term deposit subscriptions at a Portuguese bank institution. With the accuracy of 93% using Random Forest, the following business recommendations are:

- Pay close attention to socioeconomic data: 3 Month Euribor rate - The Euribor rate is based on the average interest rates at which Eurozone banks lend funds to other banks. Ramp up the campaign when rates are high. A prospect is more likely to invest in a term deposit knowing that they will receive a high interest rate. Consumer Confidence Index - Individuals are more likely to invest if their financial situation is good and if the country's outlook is optimistic.
- Target telemarketing calls toward select the individuals who belong to a specific age group and occupation. Seniors, and Students, and Retired Individuals are more likely to subscribe to a term deposit.
- Conduct the campaign during specific months. March, September, and October have been shown to have a higher rate of subscriptions.

