About The Dataset

This is a business use case for a Portuguese bank that has experienced a decline in revenue due to clients not depositing as frequently as before. The bank wants to identify clients who are likely to subscribe to a term deposit and focus their marketing efforts on them. The goal is to predict whether a client will subscribe to a term deposit or not using client data such as age, job type, marital status, etc., and call details such as duration, month, and day of the call. The dataset contains two files: train.csv for model training and test.csv for predicting whether new clients will subscribe to the term deposit.

In [1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib.pyplot as plt
```

In [5]:
```
df_tr=pd.read_csv('termdeposit_train.csv')
df_tr
```

Out[5]:

	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	pr
0	26110	56	admin.	married	unknown	no	1933	no	no	telephone	19	nov	44	2	-1	
1	40576	31	unknown	married	secondary	no	3	no	no	cellular	20	jul	91	2	-1	
2	15320	27	services	married	secondary	no	891	yes	no	cellular	18	jul	240	1	-1	
3	43962	57	management	divorced	tertiary	no	3287	no	no	cellular	22	jun	867	1	84	
4	29842	31	technician	married	secondary	no	119	yes	no	cellular	4	feb	380	1	-1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
31642	36483	29	management	single	tertiary	no	0	yes	no	cellular	12	may	116	2	-1	
31643	40178	53	management	divorced	tertiary	no	380	no	yes	cellular	5	jun	438	2	-1	
31644	19710	32	management	single	tertiary	no	312	no	no	cellular	7	aug	37	3	-1	
31645	38556	57	technician	married	secondary	no	225	yes	no	telephone	15	may	22	7	337	
31646	14156	55	management	divorced	secondary	no	204	yes	no	cellular	11	jul	1973	2	-1	

31647 rows × 18 columns

In [6]:
```
df_tr.shape
```

Out[6]:
```
(31647, 18)
```

As we can see, there are 18 columns and 31647 rows in our dataset.

Now determining whether this data set contains any null values.

In [7]:
```
df_tr.isnull()
```

Out[7]:

	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
31642	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
31643	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
31644	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
31645	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
31646	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

31647 rows × 18 columns

We examined it in the form of a DataFrame.

Now, we conduct a sum check.

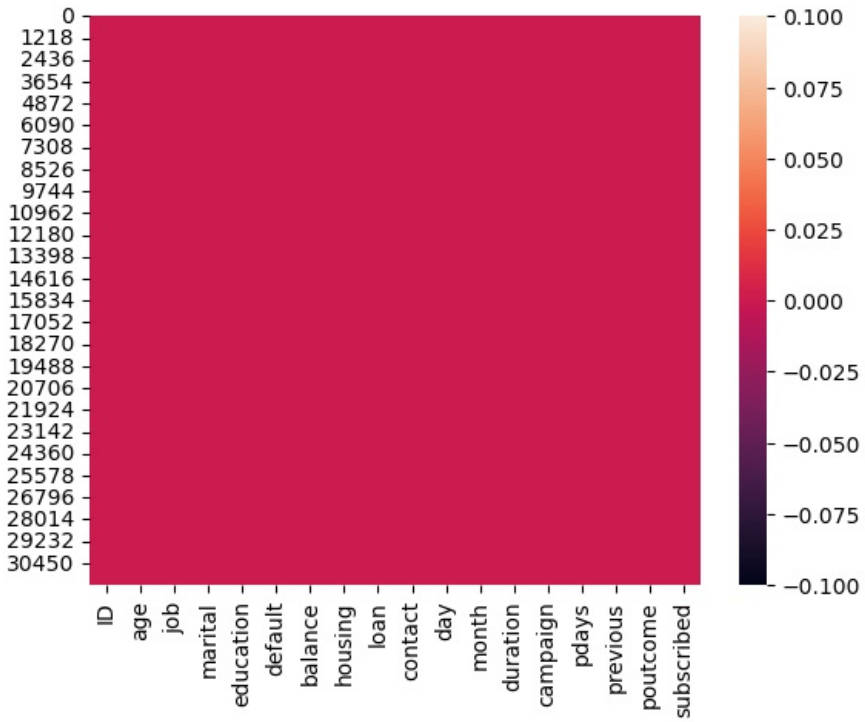
In [8]:
```
df_tr.isnull().sum()
```

Out[8]:
```
ID           0
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
subscribed   0
dtype: int64
```

As we can see, this dataset has no null values.

We ought to represent it graphically using a heatmap.

In [9]:
```
sns.heatmap(df_tr.isnull())
plt.show()
```



The graph shows that there are no null values.

Verifying whether or not there is a na value.

In [10]:
```
df_tr.isna().sum()
```

Out[10]:
```
ID           0
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
subscribed   0
dtype: int64
```

As we can see, this dataset has no na values.

Currently, we are checking the names of the columns.

In [11]:
```
df_tr.columns
```

Out[11]:
```
Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome', 'subscribed'],
      dtype='object')
```

columns names are as above.

Checking the details in our data now.

In [12]:
```
df_tr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31647 entries, 0 to 31646
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   ID          31647 non-null  int64 
 1   age         31647 non-null  int64 
 2   job         31647 non-null  object
 3   marital     31647 non-null  object
 4   education   31647 non-null  object
 5   default     31647 non-null  object
 6   balance     31647 non-null  int64 
 7   housing     31647 non-null  object
 8   loan        31647 non-null  object
 9   contact     31647 non-null  object
 10  day         31647 non-null  int64 
 11  month       31647 non-null  object
 12  duration    31647 non-null  int64 
 13  campaign    31647 non-null  int64 
 14  pdays       31647 non-null  int64 
 15  previous    31647 non-null  int64 
 16  poutcome    31647 non-null  object
 17  subscribed  31647 non-null  object
dtypes: int64(8), object(10)
memory usage: 4.3+ MB
```

As we can see, there are 31647 non-null values, total columns are 18, and 17 of those columns are features.

Now, we examine the dataset's data types.

In [13]:
```
df_tr.dtypes
```

Out[13]:
```
ID             int64
age            int64
job           object
marital       object
education     object
default       object
balance        int64
housing       object
loan          object
contact       object
day            int64
month         object
duration       int64
campaign       int64
pdays          int64
previous       int64
poutcome      object
subscribed    object
dtype: object
```

Our aim is the object data type since there are only two sorts of data types: object and int64.

Since the ID column is no longer necessary, we should get rid of it.

In [14]:
```
df_tr.drop(['ID'],axis=1,inplace=True)
df_tr.head()
```

Out[14]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	pout
0	56	admin.	married	unknown	no	1933	no	no	telephone	19	nov	44	2	-1	0	unl
1	31	unknown	married	secondary	no	3	no	no	cellular	20	jul	91	2	-1	0	unl
2	27	services	married	secondary	no	891	yes	no	cellular	18	jul	240	1	-1	0	unl
3	57	management	divorced	tertiary	no	3287	no	no	cellular	22	jun	867	1	84	3	su
4	31	technician	married	secondary	no	119	yes	no	cellular	4	feb	380	1	-1	0	unl

As shown, the ID column has been deleted.

Now, we examine each column's unique values.

In [15]:
```
df_tr.columns
```

Out[15]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'subscribed'],
      dtype='object')
```

In [16]:
```
df_tr['age'].unique()
```

Out[16]:
```
array([56, 31, 27, 57, 33, 50, 45, 35, 36, 44, 39, 43, 41, 53, 64, 48, 60,
       29, 30, 42, 46, 32, 63, 47, 49, 52, 68, 26, 58, 38, 54, 40, 59, 24,
       37, 34, 61, 55, 20, 69, 25, 62, 22, 51, 28, 70, 66, 23, 87, 71, 74,
       81, 73, 76, 72, 84, 19, 79, 75, 21, 83, 80, 67, 18, 77, 65, 82, 86,
       78, 88, 92, 95, 93, 89, 94, 90], dtype=int64)
```

Age has several distinct values, as we can see.

In [17]:
```
df_tr['job'].unique()
```

Out[17]:
```
array(['admin.', 'unknown', 'services', 'management', 'technician',
       'retired', 'blue-collar', 'housemaid', 'self-employed', 'student',
       'entrepreneur', 'unemployed'], dtype=object)
```

As can be seen, there are a total of 12 distinct values.

In [18]:
```
df_tr['marital'].unique()
```

Out[18]:
```
array(['married', 'divorced', 'single'], dtype=object)
```

As can be seen, there are a total of 3 distinct values.

In [19]:
```
df_tr['education'].unique()
```

Out[19]:
```
array(['unknown', 'secondary', 'tertiary', 'primary'], dtype=object)
```

As can be seen, there are a total of 4 distinct values.

In [20]:
```
df_tr['default'].unique()
```

Out[20]:
```
array(['no', 'yes'], dtype=object)
```

As can be seen, there are a total of 2 distinct values.

In [21]:
```
df_tr['balance'].unique()
```

Out[21]:
```
array([1933,    3,  891, ..., 2787, 8741, 2968], dtype=int64)
```

As we can see, there are lots of distinctive values.

In [22]:
```
df_tr['housing'].unique()
```

Out[22]:
```
array(['no', 'yes'], dtype=object)
```

As can be seen, there are a total of 2 distinct values.

In [23]:
```
df_tr['loan'].unique()
```

Out[23]:
```
array(['no', 'yes'], dtype=object)
```

There are, as we can see, a total of 2 distinct values.

In [24]:
```
df_tr['contact'].unique()
```

Out[24]:
```
array(['telephone', 'cellular', 'unknown'], dtype=object)
```

As can be seen, there are a total of 3 distinct values.

In [25]:
```
df_tr['day'].unique()
```

Out[25]:
```
array([19, 20, 18, 22,  4,  2,  3,  8, 15,  5, 28,  6, 14,  7, 24, 13,  9,
       11, 21, 12, 30, 27, 17, 16, 25, 10,  1, 29, 26, 31, 23],
      dtype=int64)
```

As we can see, there are lots of distinctive values.

In [26]:
```
df_tr['month'].unique()
```

Out[26]:
```
array(['nov', 'jul', 'jun', 'feb', 'sep', 'jan', 'may', 'aug', 'apr',
       'oct', 'mar', 'dec'], dtype=object)
```

As can be seen, there are a total of 12 distinct values.

In [27]:
```
df_tr['duration'].unique()
```

Out[27]:
```
array([  44,   91,  240, ...,  939,  839, 1973], dtype=int64)
```

As we can see, there are lots of distinctive values.

In [28]:
```
df_tr['campaign'].unique()
```

Out[28]:
```
array([ 2,  1,  3,  4,  7,  5, 33, 12,  8,  9,  6, 24, 17, 11, 20, 25, 19,
       29, 21, 10, 27, 38, 16, 18, 14, 30, 13, 15, 63, 23, 31, 43, 35, 22,
       34, 28, 26, 41, 37, 50, 55, 32, 44, 36, 39], dtype=int64)
```

There are numerous distinctive values, as we can see.

In [29]:
```
df_tr['pdays'].unique()
```

Out[29]:
```
array([ -1,  84, 251,   9, 456, 120,  92, 347, 154, 291, 344, 196, 324,
       332, 304, 297, 149, 102, 330, 301, 182,  26, 112, 457, 104, 256,
        90,  94, 135, 113, 360, 224,  98, 153,  82,  18, 343, 337, 365,
        93, 174,  96, 193,  83,  87, 272, 261,  91, 156, 195, 181, 151,
       336, 323, 342,  99, 187, 141, 329, 253, 352, 172, 177, 771, 555,
       119, 190, 362, 189, 351, 254, 169, 357, 111, 317, 367, 160, 116,
       356,  77, 225, 206, 331, 100, 136, 346, 260, 484, 176,  95, 230,
       366, 319, 521, 152, 179, 215, 430, 287, 842, 349, 180, 259, 283,
       275, 348, 140, 202, 370,  89, 184, 213, 308, 255, 188,  86, 161,
       252, 129, 271, 334, 279, 358, 103,   6, 110, 316, 270, 363, 374,
       197, 106, 198, 322, 131, 338, 235, 146, 295,  35, 459, 274, 273,
         4, 105, 143, 326, 183, 229, 369, 241, 280, 318, 173, 132, 201,
       368, 266, 340,  97, 364, 167,   5, 288, 276, 335, 493,  50, 134,
       145,  21, 147, 312, 269, 148, 305, 265,   2, 792, 246, 371, 186,
       245, 192, 518, 164, 264, 124,  13, 278, 299, 109, 381, 262, 478,
       307,   8, 310, 227, 361, 170, 292, 263, 185, 258, 359, 341, 313,
       446, 294, 350, 435, 325, 238, 415, 155, 293, 166,  88, 303, 345,
       115, 150, 101, 205, 247, 315, 165, 321, 320, 203, 178, 300, 244,
       168, 199, 107, 211, 281, 233, 191, 433,  81, 133, 302, 355, 130,
       557, 200, 389,  54, 223, 214,  14, 208, 175,  71,   7,  45, 298,
       240,  64, 285, 121, 163, 328, 354, 547, 125, 250,  53, 209, 449,
        36, 377, 403, 171,  12, 231,  56, 159, 339, 234, 126, 249, 239,
       127,  70,  10, 139, 268, 243, 414,  80, 194, 137,  85, 686, 142,
        57, 391, 474,  31,  78, 237, 207,  79, 162,   1,  60,  40, 144,
       314, 467, 410, 248, 210, 772, 472, 309, 158, 277, 592, 375, 114,
       455, 286, 553, 353, 232, 267, 157, 219,  75,  72, 306, 327, 204,
       500, 412, 333, 257, 390, 372, 458,  28,  67, 561, 394, 626,  47,
       117, 284, 216, 520, 108, 212, 123, 217, 475,  58, 529, 690, 683,
       384, 633, 373, 228, 586, 452, 426, 296, 138, 392,  59, 118, 769,
       378, 242,  30,  49, 687, 603,  62, 535, 526,  25, 282, 376,  20,
        51,  48,  66, 221, 420,  63, 717, 838, 122,  37, 427, 558, 128,
       470, 422, 461, 405, 311, 495, 290, 774, 387, 804,  27,  42, 508,
       594, 222, 465,  29, 385,  76, 745, 413,  15, 463, 524, 579, 515,
       440,  69, 543, 226, 477, 379, 386, 419,  33, 469, 220, 504,  52,
       218, 578, 416, 289, 536, 756, 442, 236, 616, 428,  41, 404, 761,
        68,  74, 775, 667,  46, 805, 397,  73,  19, 481, 871, 434, 393,
       388,  44, 437, 485, 511, 760, 530,  17,  24, 595,  61,  43, 528,
        39,  34, 749, 585, 701, 587, 460, 784, 491, 398, 544, 450, 417,
       462,  38, 854, 791, 651, 778, 782,  32, 439,  55,  22, 532, 409,
       531, 382], dtype=int64)
```

There are a lot of distinct values, as we can see.

In [30]:
```
df_tr['previous'].unique()
```

Out[30]:
```
array([  0,   3,   2,   4,   1,   5,   9,   6,   8,  11,  16,  10,  14,
         7,  12,  23,  13,  18,  30,  27, 275,  20,  15,  17,  19,  22,
        25,  26,  28,  29,  32,  21,  24,  38,  58,  35,  41,  37],
      dtype=int64)
```

As we can see, there are a lot of distinctive values.

In [31]:
```
df_tr['poutcome'].unique()
```

Out[31]:
```
array(['unknown', 'success', 'failure', 'other'], dtype=object)
```

As can be seen, there are a total of 4 distinct values.

In [32]:
```
df_tr['subscribed'].unique()
```

Out[32]:
```
array(['no', 'yes'], dtype=object)
```

As can be seen, there are a total of 2 distinct values.

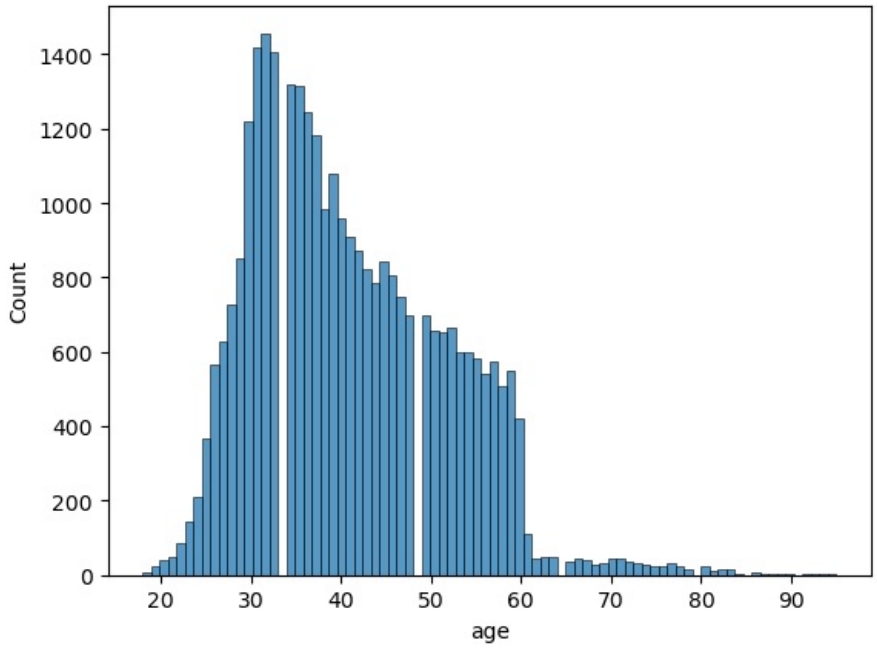
We have now completed checking for unique column values.

We are currently visualising data.

Univariabe Analysis

As needed for columns by their unique values, we can utilise count plot and hist plot for univariate analysis.

In [33]:
```
sns.histplot(df_tr['age'])
plt.show()
```



The bulk of people, as shown in the graph, are between the ages of 32 and 37.

In [34]:
```
df_tr.columns
```

Out[34]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'subscribed'],
      dtype='object')
```

In [35]:
```
print(df_tr['job'].value_counts(),'\n')
print(df_tr['job'].value_counts(normalize=True)*100)
```

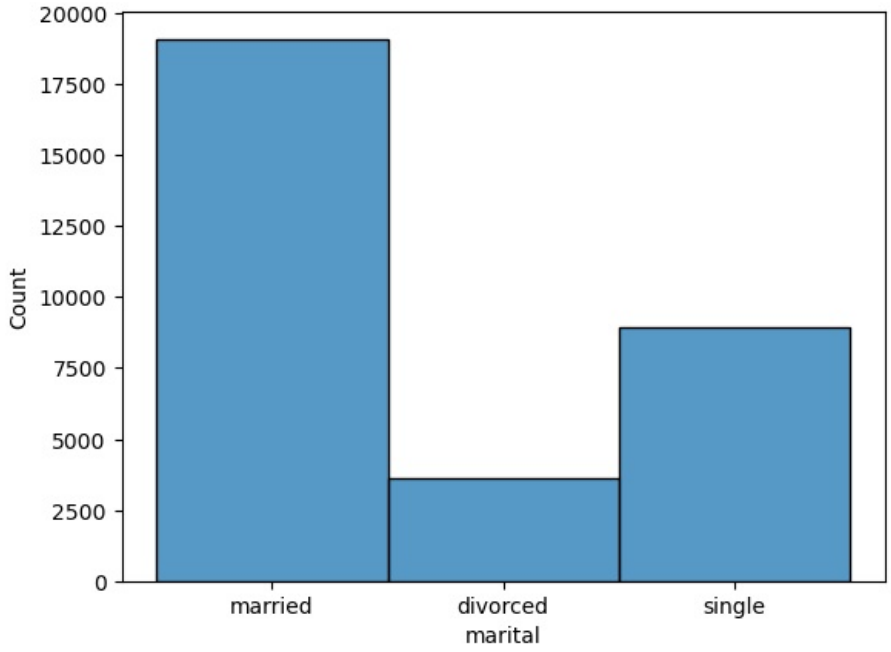
```
blue-collar      6842
management       6639
technician       5307
admin.           3631
services         2903
retired          1574
self-employed    1123
entrepreneur     1008
unemployed        905
housemaid         874
student           635
unknown           206
Name: job, dtype: int64

blue-collar      21.619743
management       20.978292
technician       16.769362
admin.           11.473441
services          9.173065
retired           4.973615
self-employed     3.548520
entrepreneur      3.185136
unemployed        2.859671
housemaid         2.761715
student           2.006509
unknown           0.650931
Name: job, dtype: float64
```

According to the data above, 42% of people are employed in managerial and blue-collar jobs.

In [36]:
```
sns.histplot(df_tr['marital'])
print(df_tr['marital'].value_counts())
print(df_tr['marital'].value_counts(normalize=True)*100)
```

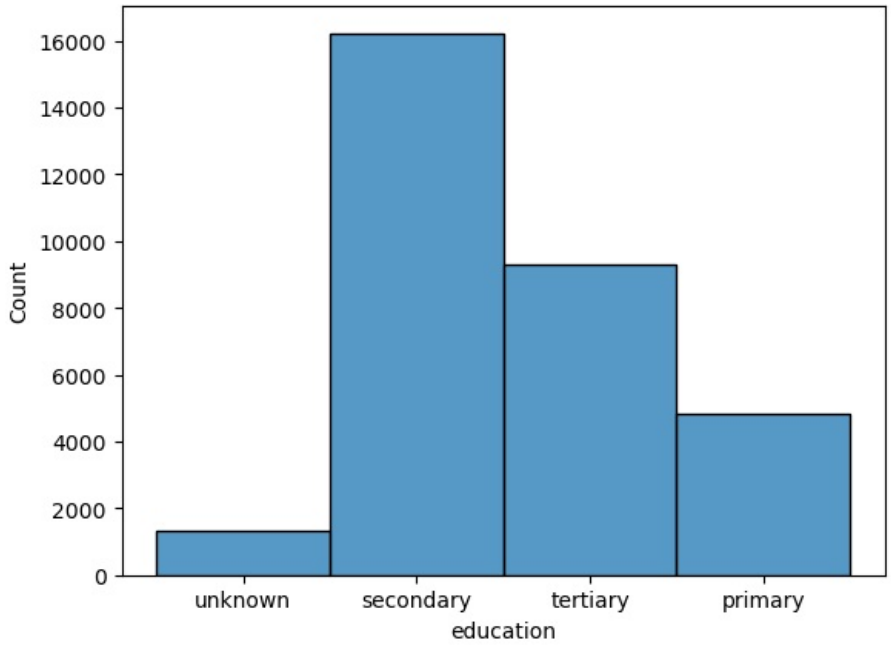
```
married     19095
single       8922
divorced     3630
Name: marital, dtype: int64
married     60.337473
single      28.192246
divorced    11.470282
Name: marital, dtype: float64
```



considering that 60% of people are married.

In [37]:
```
sns.histplot(df_tr['education'])
print(df_tr['education'].value_counts())
print(df_tr['education'].value_counts(normalize=True)*100)
```

```
secondary    16224
tertiary      9301
primary       4808
unknown       1314
Name: education, dtype: int64
secondary    51.265523
tertiary     29.389832
primary      15.192593
unknown       4.152052
Name: education, dtype: float64
```

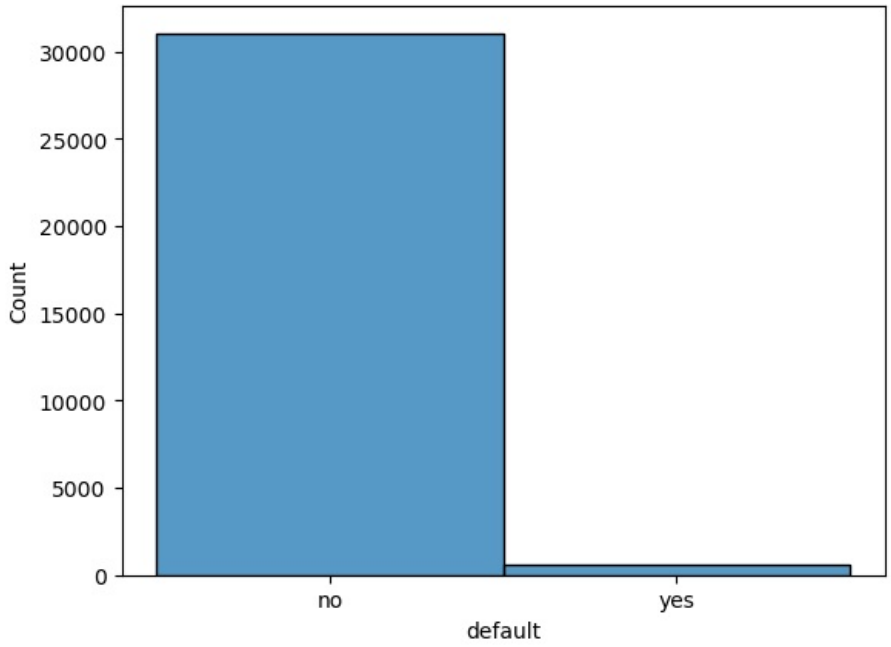


As we can see, 52% of the population has a secondary education.

In [38]:
```
sns.histplot(df_tr['default'])
```

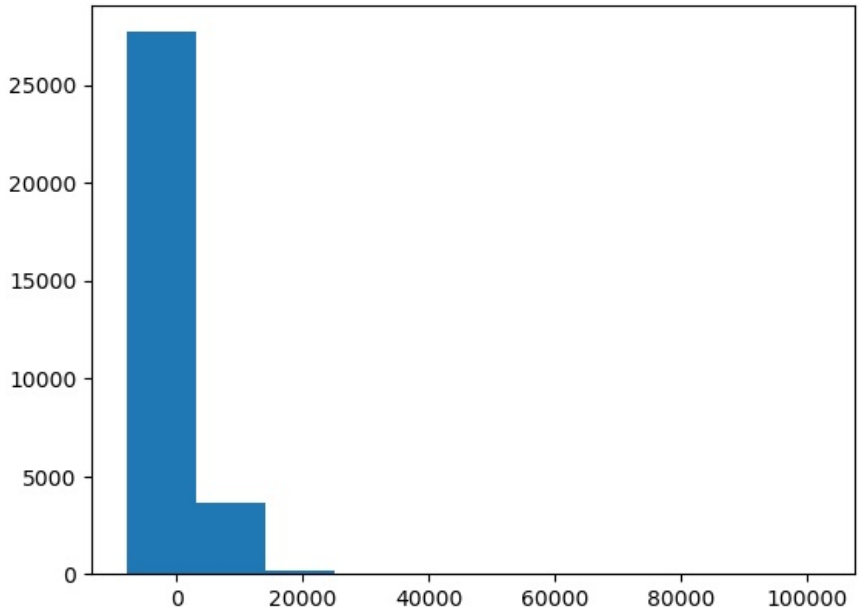
```
print(df_tr['default'].value_counts())
print(df_tr['default'].value_counts(normalize=True)*100)
```

```
no     31062
yes      585
Name: default, dtype: int64
no     98.151484
yes     1.848516
Name: default, dtype: float64
```



As we can see, 98% of them are deemed to be non-defaulters.

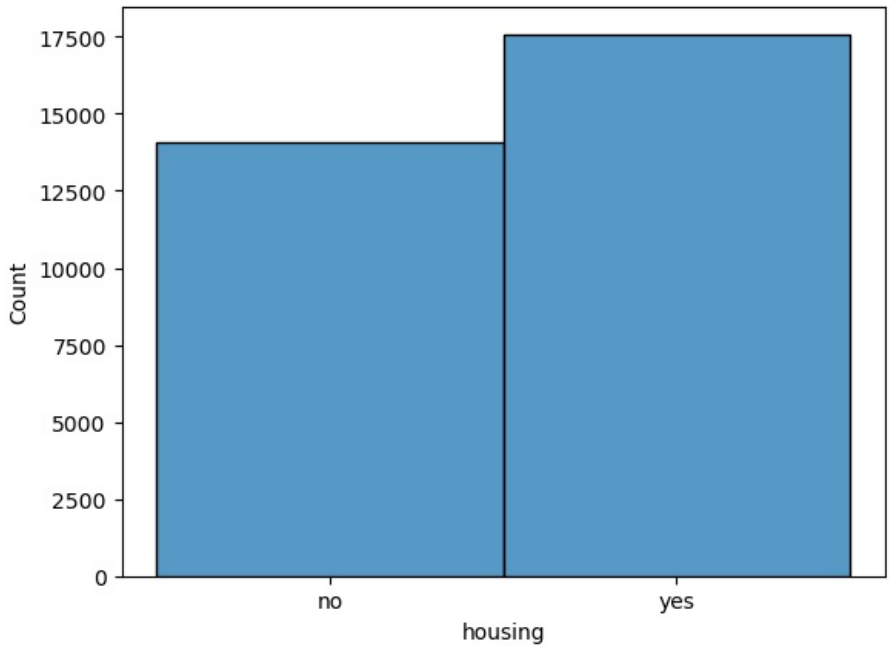
In [39]:
```
plt.plot(figsize=(7,9))
plt.hist(df_tr['balance'])
plt.show()
```



As can be seen, the bulk of people come from incomes under $10,000.

In [40]:
```
sns.histplot(df_tr['housing'])
print(df_tr['housing'].value_counts())
print(df_tr['housing'].value_counts(normalize=True)*100)
```

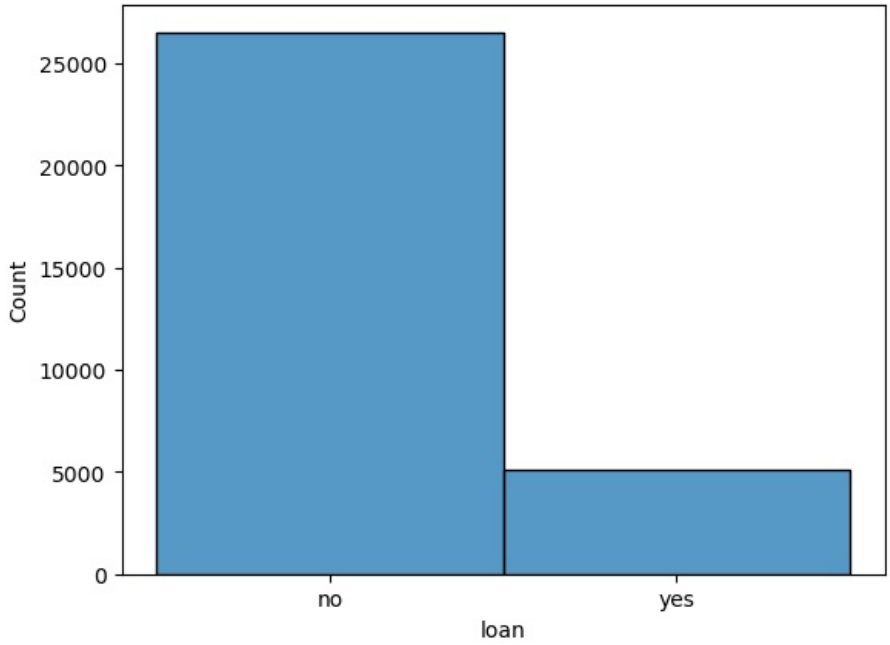
```
yes    17584
no     14063
Name: housing, dtype: int64
yes    55.562929
no     44.437071
Name: housing, dtype: float64
```



when we see that 56% of people have a mortgage.

In [41]:
```
sns.histplot(df_tr['loan'])
print(df_tr['loan'].value_counts())
print(df_tr['loan'].value_counts(normalize=True)*100)
```

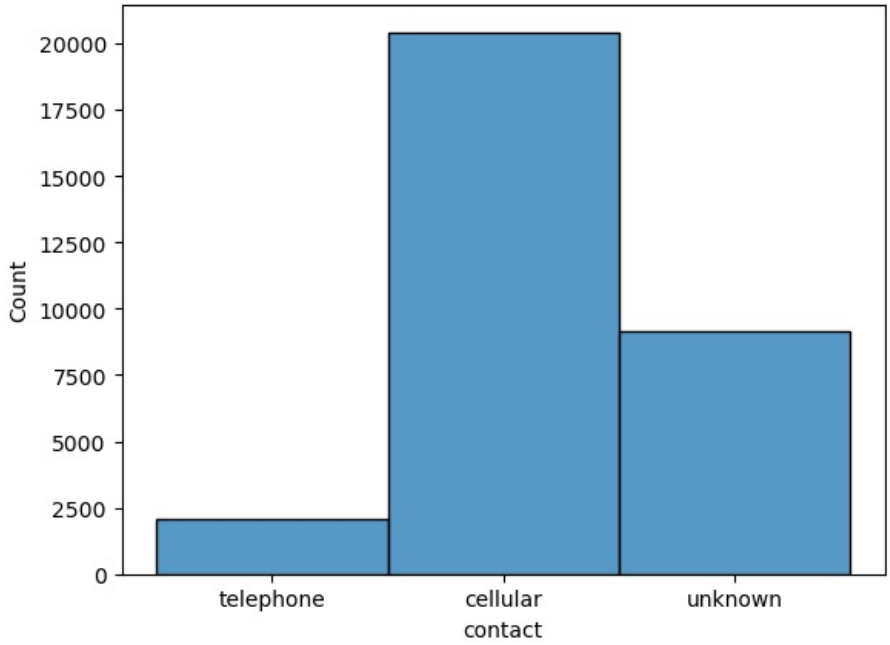
```
no     26516
yes     5131
Name: loan, dtype: int64
no     83.786773
yes    16.213227
Name: loan, dtype: float64
```



As we can see, 84% of people do not have personal loans.

In [42]:
```
sns.histplot(df_tr['contact'])
print(df_tr['contact'].value_counts())
print(df_tr['contact'].value_counts(normalize=True)*100)
```

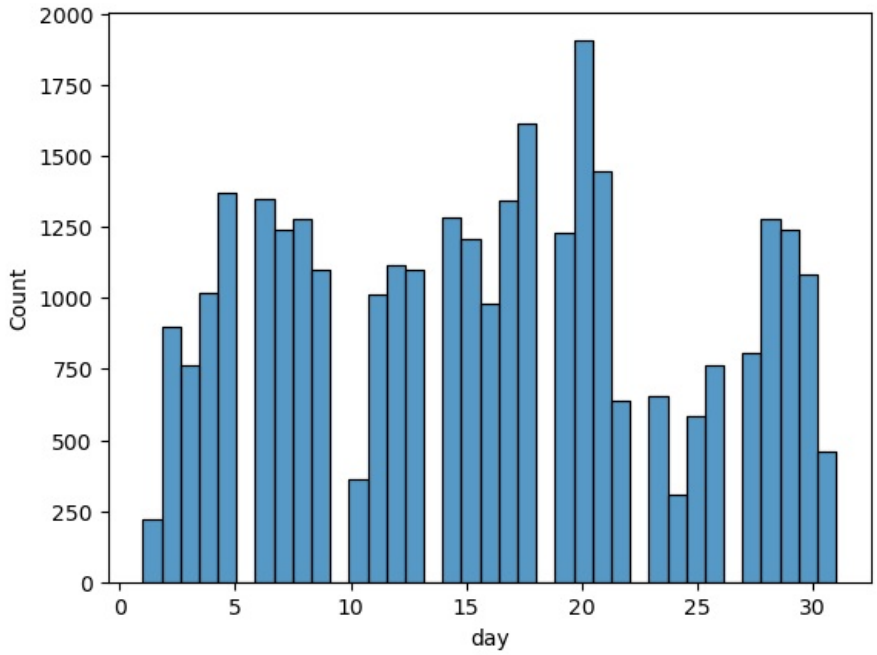
```
cellular     20423
unknown       9177
telephone     2047
Name: contact, dtype: int64
cellular     64.533763
unknown      28.998009
telephone     6.468228
Name: contact, dtype: float64
```



As we can see, 64% of contacts are made using a mobile. is the majority.

In [43]:
```
sns.histplot(df_tr['day'])
```

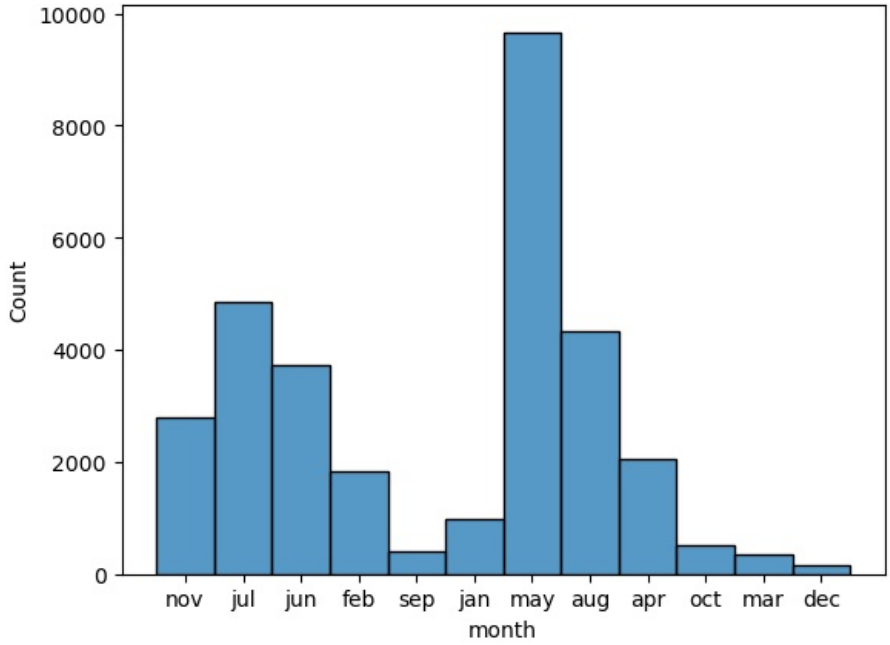
Out[43]:
```
<AxesSubplot:xlabel='day', ylabel='Count'>
```



As we observe that 20 hours make up the majority of the day, we discuss connecting a column in the name.

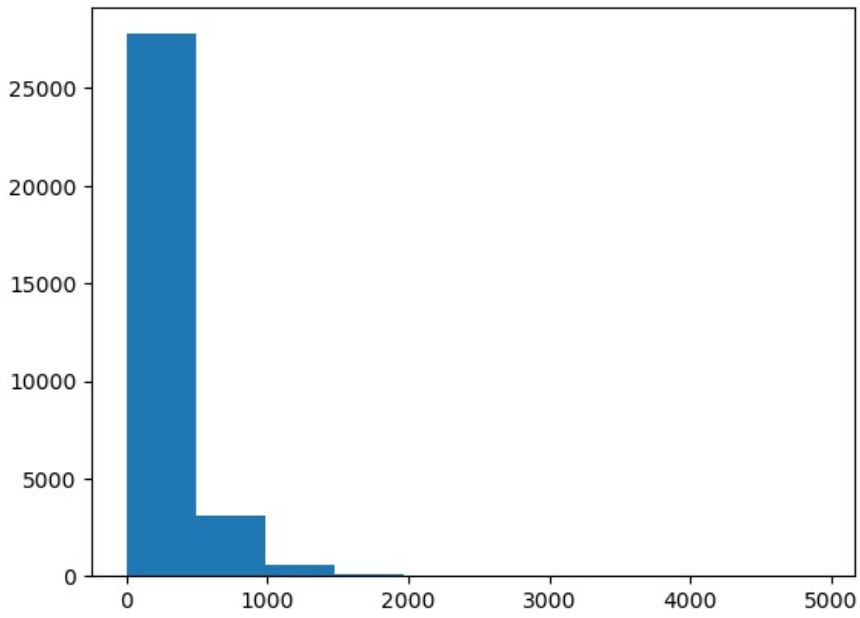
In [44]:
```
sns.histplot(df_tr['month'])
print(df_tr['month'].value_counts())
print(df_tr['month'].value_counts(normalize=True)*100)
```

```
may    9669
jul    4844
aug    4333
jun    3738
nov    2783
apr    2055
feb    1827
jan     977
oct     512
sep     410
mar     342
dec     157
Name: month, dtype: int64
may    30.552659
jul    15.306348
aug    13.691661
jun    11.811546
nov     8.793883
apr     6.493506
feb     5.773059
jan     3.087180
oct     1.617847
sep     1.295541
mar     1.080671
dec     0.496098
Name: month, dtype: float64
```



According to our observations, 30% of contact months occur in may.

In [45]:
```
plt.hist(df_tr['duration'])
plt.show()
```

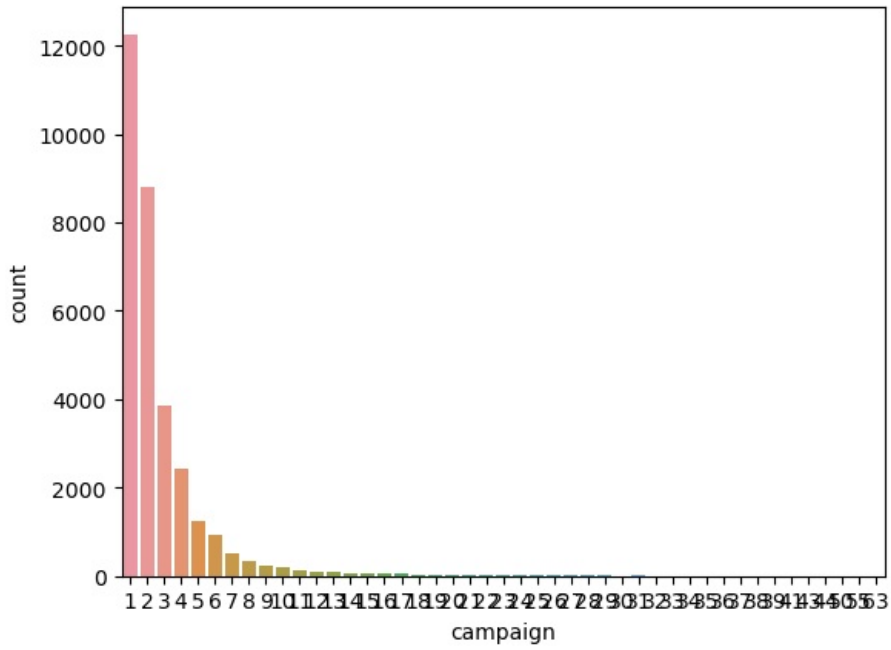


As we can see, the majority of duration falls between 0 and 500.

In [46]:
```
sns.countplot(df_tr['campaign'])
print(df_tr['campaign'].value_counts())
print(df_tr['campaign'].value_counts(normalize=True)*100)
```

```
1     12262
2      8798
3      3858
4      2442
5      1245
6       916
7       518
8       356
9       236
10      184
11      126
12      102
13       93
14       68
15       61
16       54
17       52
20       37
18       37
19       30
21       19
22       18
25       17
28       14
23       13
24       13
29       12
26        9
31        9
27        8
30        6
32        6
33        5
34        5
43        3
35        3
38        3
37        2
63        1
41        1
50        1
55        1
44        1
36        1
39        1
Name: campaign, dtype: int64
1     38.746169
2     27.800423
3     12.190729
4      7.716371
5      3.934022
6      2.894429
7      1.636806
8      1.124909
9      0.745726
10     0.581414
11     0.398142
12     0.322305
13     0.293867
14     0.214870
15     0.192751
16     0.170632
17     0.164313
20     0.116915
18     0.116915
19     0.094796
21     0.060037
22     0.056877
25     0.053718
28     0.044238
23     0.041078
24     0.041078
29     0.037918
26     0.028439
31     0.028439
27     0.025279
30     0.018959
32     0.018959
33     0.015799
34     0.015799
43     0.009480
35     0.009480
38     0.009480
37     0.006320
63     0.003160
```

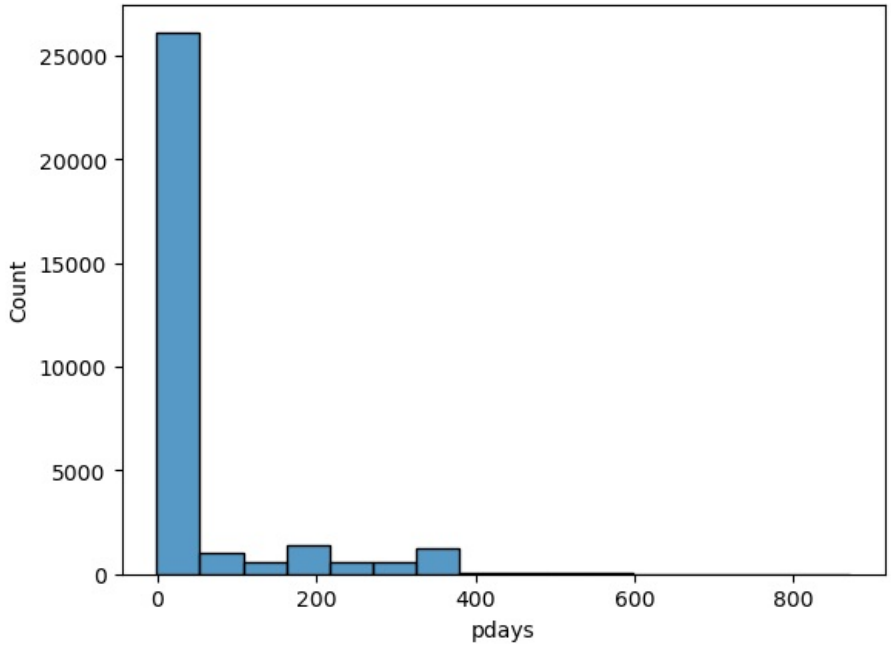
```
41     0.003160
50     0.003160
55     0.003160
44     0.003160
36     0.003160
39     0.003160
Name: campaign, dtype: float64
```



As can be seen, 66% of people are in contact with or connected to the campaigns of 1 and 2.

In [47]:
```
sns.histplot(df_tr['pdays'])
```

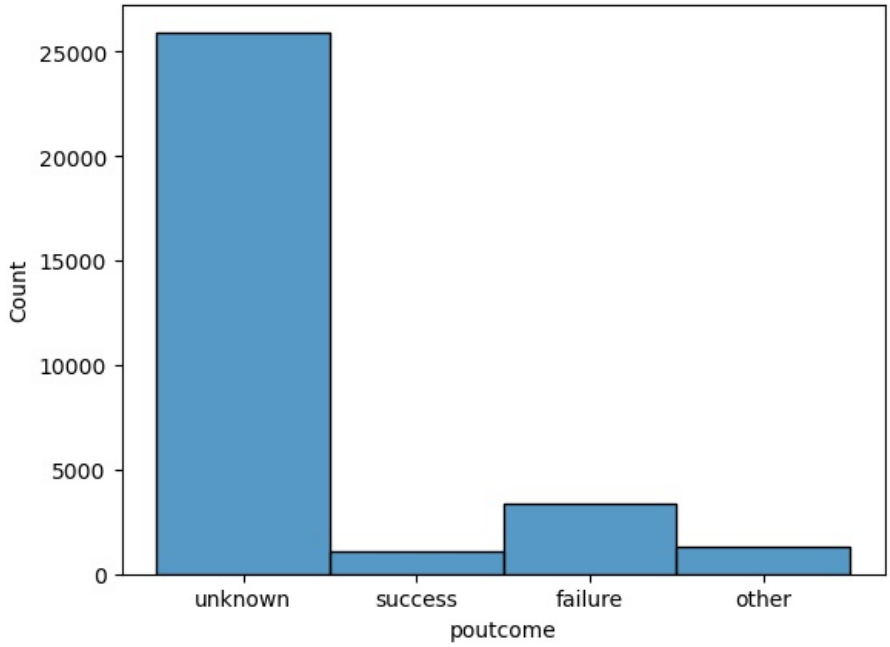
Out[47]:
```
<AxesSubplot:xlabel='pdays', ylabel='Count'>
```



As we can see from the graph, the bulk of values range from 0 to 50.

In [48]:
```
sns.histplot(df_tr['poutcome'])
print(df_tr['poutcome'].value_counts())
print(df_tr['poutcome'].value_counts(normalize=True)*100)
```

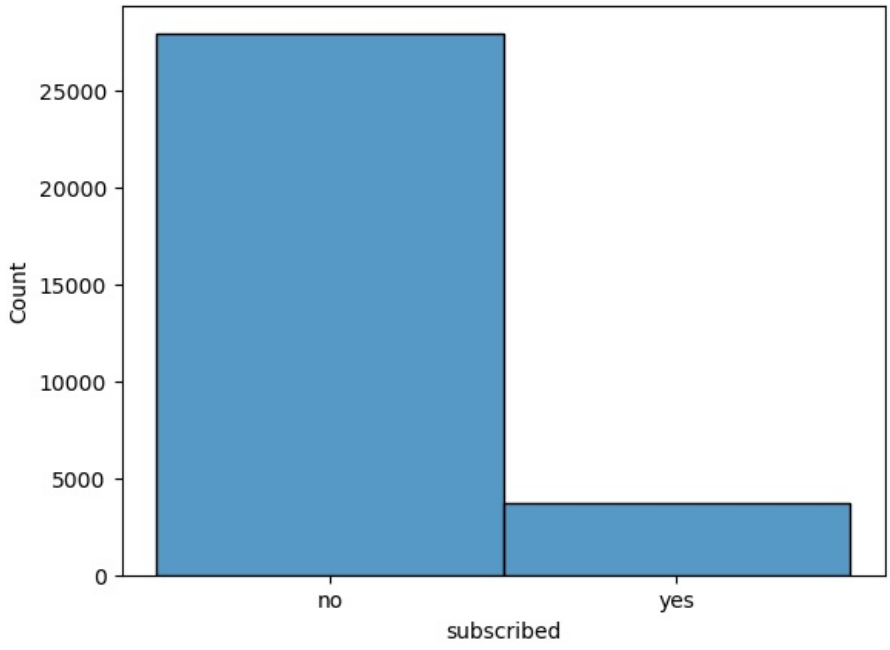
```
unknown    25929
failure     3362
other       1288
success     1068
Name: poutcome, dtype: int64
unknown    81.931937
failure    10.623440
other       4.069896
success     3.374727
Name: poutcome, dtype: float64
```



According to poutcome, 81% of respondents are from the unknown category.

In [49]:
```
sns.histplot(df_tr['subscribed'])
print(df_tr['subscribed'].value_counts())
print(df_tr['subscribed'].value_counts(normalize=True)*100)
```

```
no     27932
yes     3715
Name: subscribed, dtype: int64
no     88.261131
yes    11.738869
Name: subscribed, dtype: float64
```



We should balance it because 88% of people are not subscribed, which is a majority.

As we proceed with bivariate analysis, we must first change the object data type to encrypt the data.

In [50]:
```
from sklearn.preprocessing import LabelEncoder # importing
```

In [51]:
```
df_tr.dtypes
```

Out[51]:
```
age            int64
job           object
marital       object
education     object
default       object
balance        int64
housing       object
loan          object
contact       object
day            int64
month         object
duration       int64
campaign       int64
pdays          int64
previous       int64
poutcome      object
subscribed    object
dtype: object
```

We must use LableEncoder on ['job','marital', 'education', 'default', 'housing', 'loan', 'contact','month', 'poutcome','subscribed'] as we can see.

In [52]:
```
le=LabelEncoder()
for i in df_tr.drop(['subscribed'],axis=1):
    df_tr[i]=le.fit_transform(df_tr[i])
df_tr
```

Out[52]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
0	38	0	1	3	0	2719	0	0	1	18	9	43	1	0	0	3
1	13	11	1	1	0	822	0	0	0	19	5	90	1	0	0	3
2	9	7	1	1	0	1709	1	0	0	17	5	239	0	0	0	3
3	39	4	0	2	0	3815	0	0	0	21	6	864	0	79	3	2
4	13	9	1	1	0	938	1	0	0	3	3	379	0	0	0	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
31642	11	4	2	2	0	819	1	0	0	11	8	115	1	0	0	3
31643	35	4	0	2	0	1199	0	1	0	4	6	437	1	0	0	3
31644	14	4	2	2	0	1131	0	0	0	6	1	36	2	0	0	3
31645	39	9	1	1	0	1044	1	0	1	14	8	21	6	332	12	0
31646	37	4	0	1	0	1023	1	0	0	10	5	1410	1	0	0	3

31647 rows × 17 columns

In [53]:
```
df_tr.head()
```

Out[53]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	subs
0	38	0	1	3	0	2719	0	0	1	18	9	43	1	0	0	3	
1	13	11	1	1	0	822	0	0	0	19	5	90	1	0	0	3	
2	9	7	1	1	0	1709	1	0	0	17	5	239	0	0	0	3	
3	39	4	0	2	0	3815	0	0	0	21	6	864	0	79	3	2	
4	13	9	1	1	0	938	1	0	0	3	3	379	0	0	0	3	

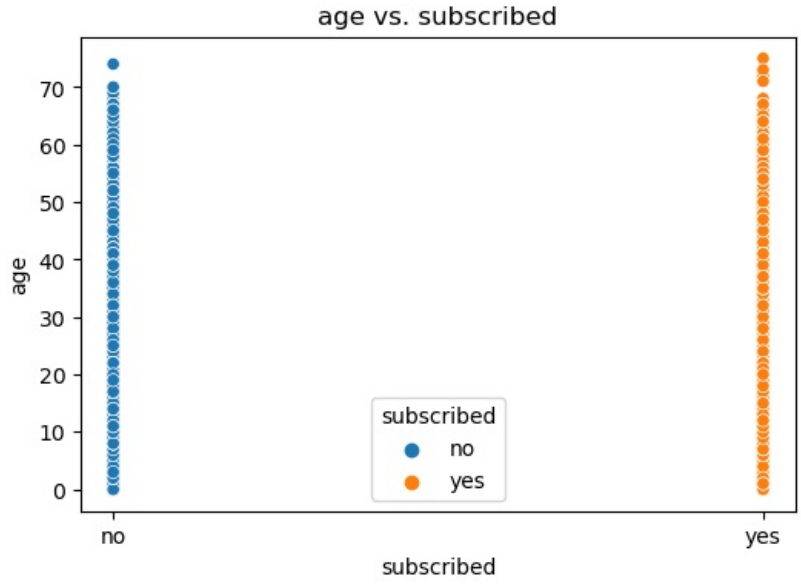
In [54]:
```
df_tr.dtypes
```

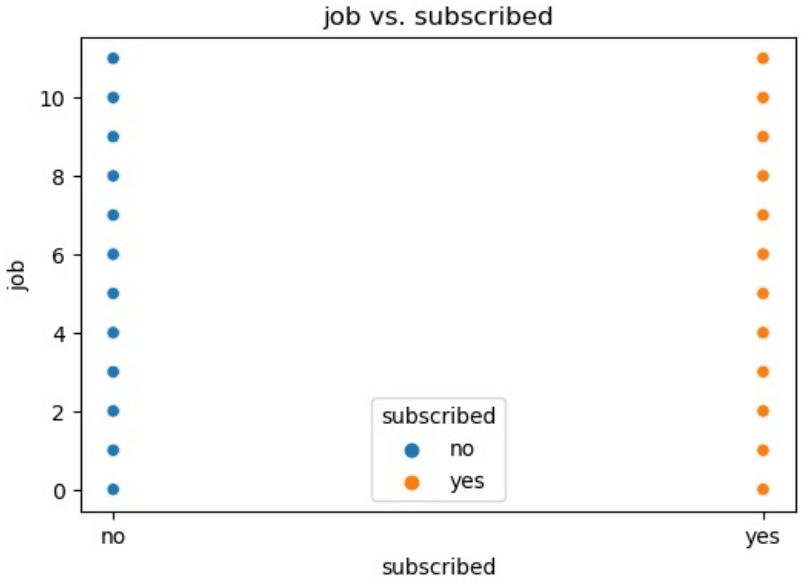
Out[54]:
```
age            int64
job            int32
marital        int32
education      int32
default        int32
balance        int64
housing        int32
loan           int32
contact        int32
day            int64
month          int32
duration       int64
campaign       int64
pdays          int64
previous       int64
poutcome       int32
subscribed    object
dtype: object
```

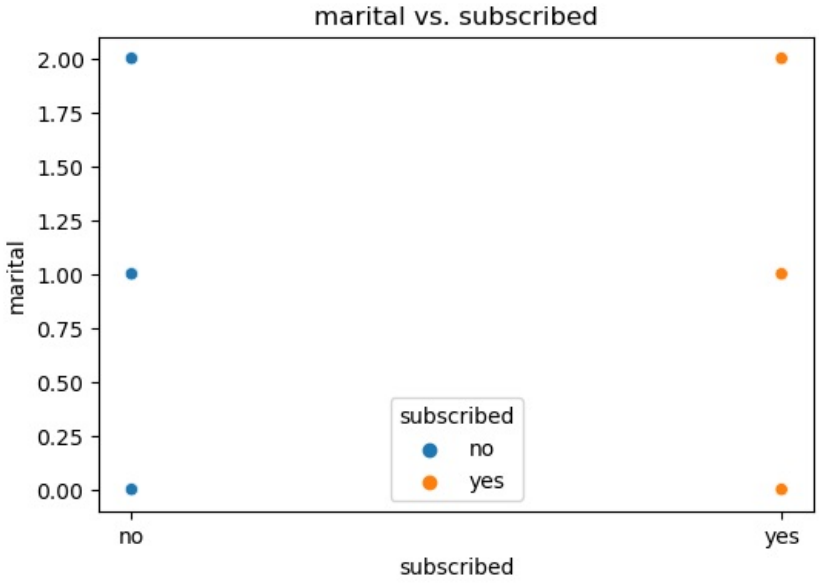
As of right now, all columns have been transformed into numerical data.

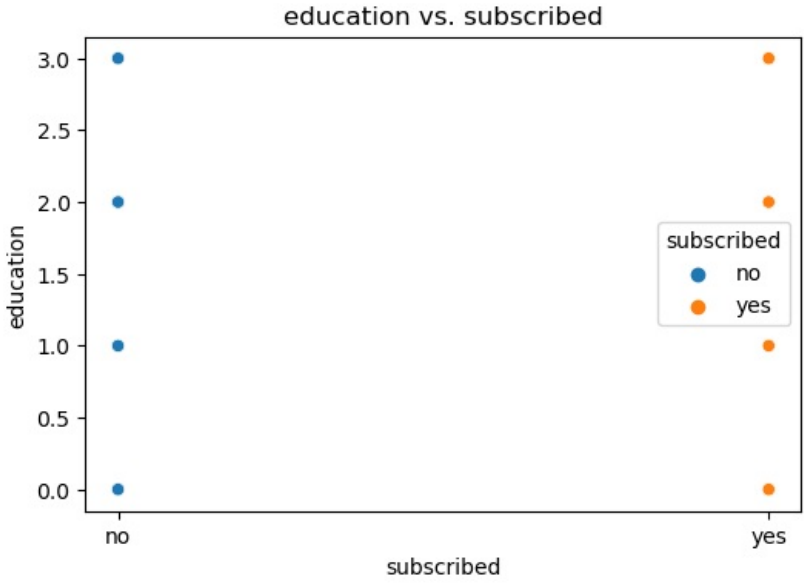
Analysis of Variance

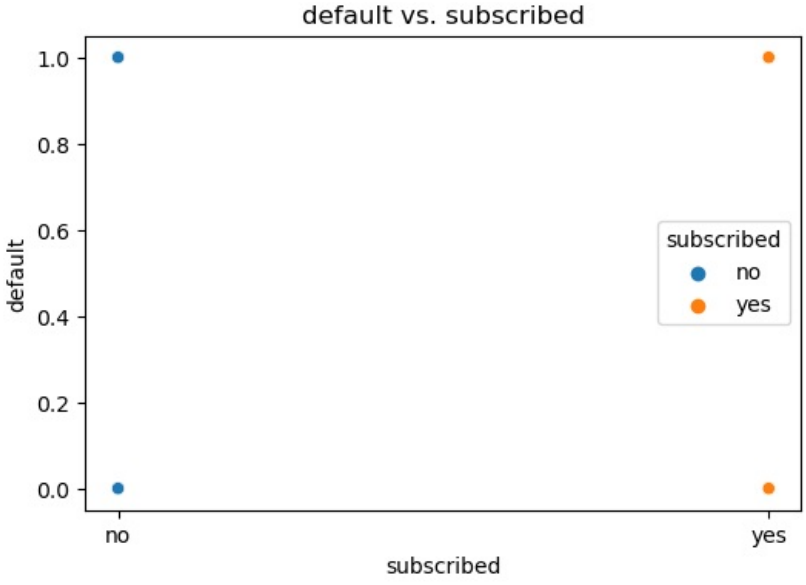
In [55]:
```
# Bivariate Analysis: Co-relation Graph Plot
import seaborn as sns
for col in df_tr.drop(['subscribed'],axis=1):
    plt.figure(figsize=(6,4))
    plt.title(f'{col} vs. subscribed')
    sns.scatterplot(y=df_tr[col],x=df_tr['subscribed'],hue=df_tr['subscribed'])
    plt.show()
```

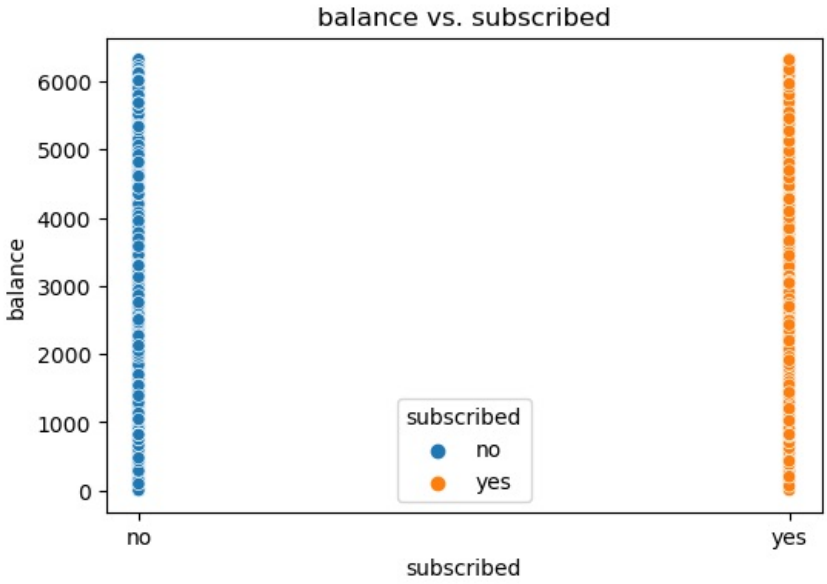


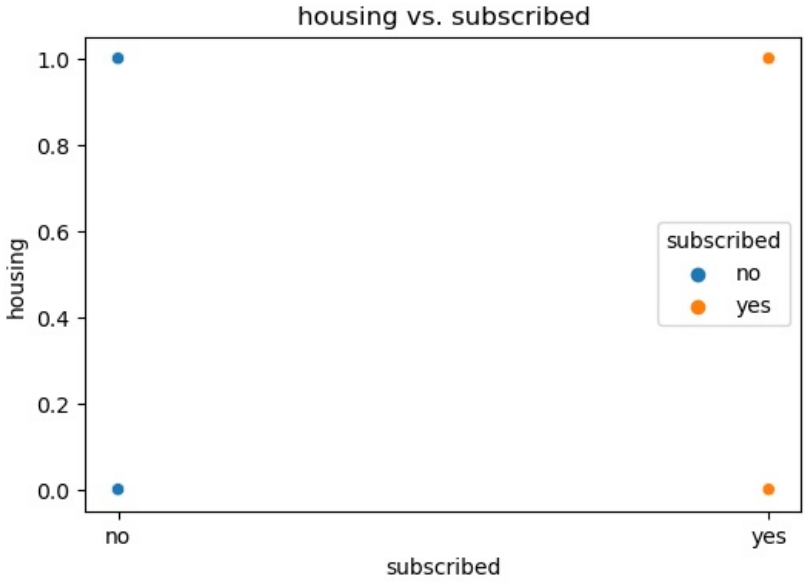


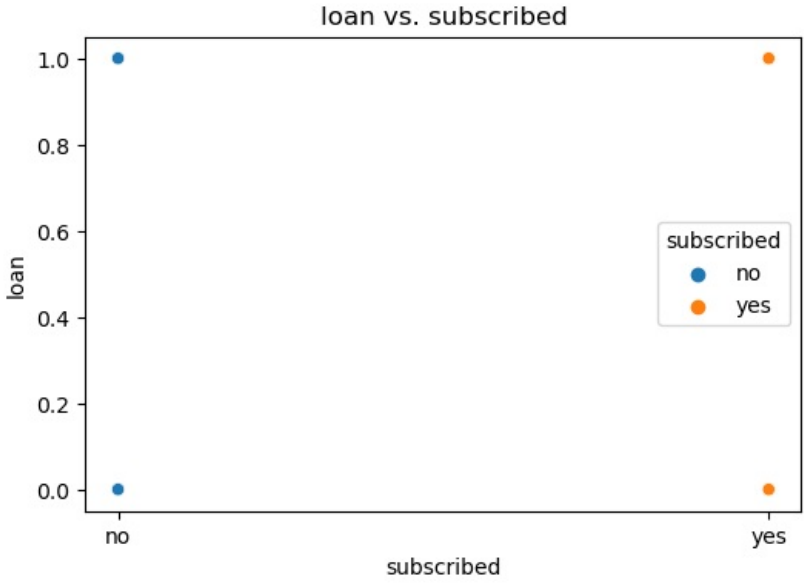

marital vs. subscribed
marital
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
subscribed
no
yes
no
yes
subscribed

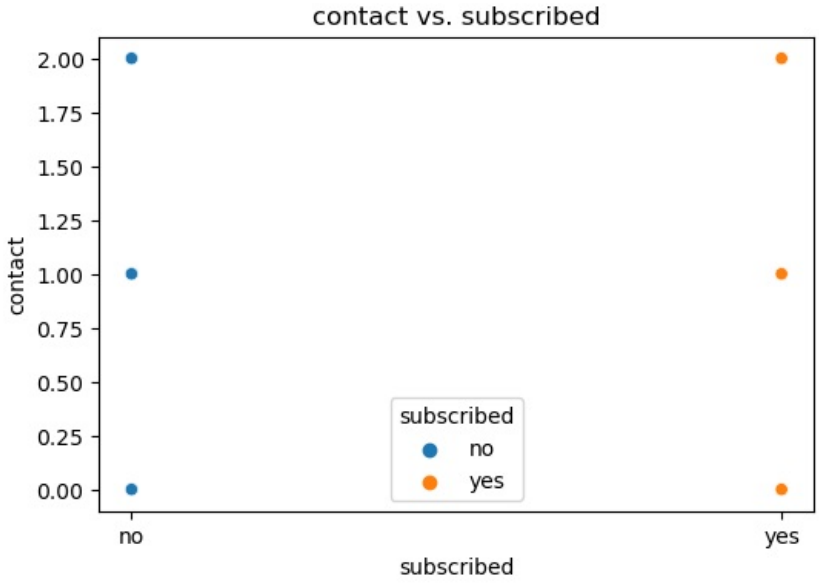

education vs. subscribed
education
3.0
2.5
2.0
1.5
1.0
0.5
0.0
subscribed
no
yes
no
yes
subscribed

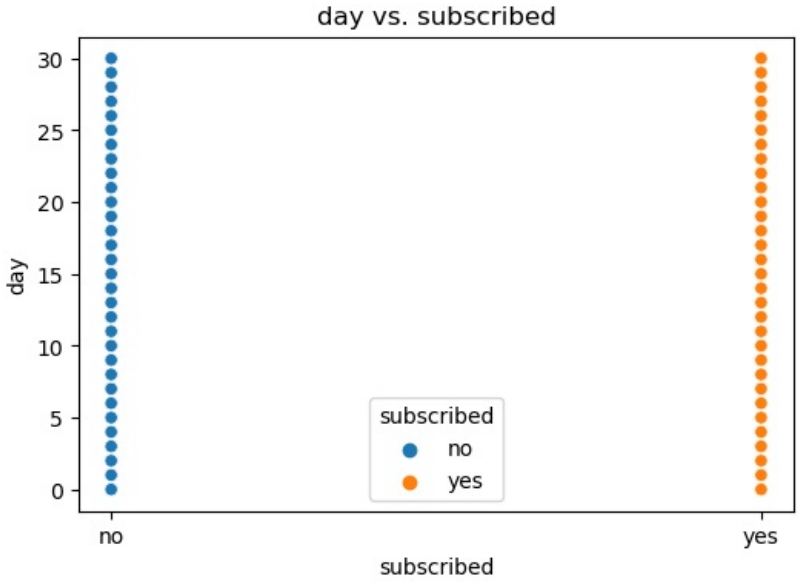

default vs. subscribed
default
1.0
0.8
0.6
0.4
0.2
0.0
subscribed
no
yes
no
yes
subscribed

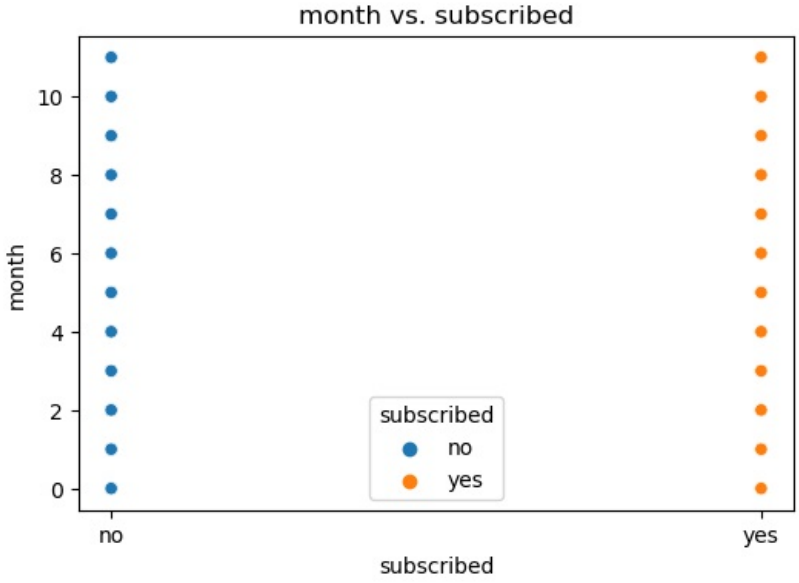

balance vs. subscribed
balance
6000
5000
4000
3000
2000
1000
0
subscribed
no
yes
no
yes
subscribed

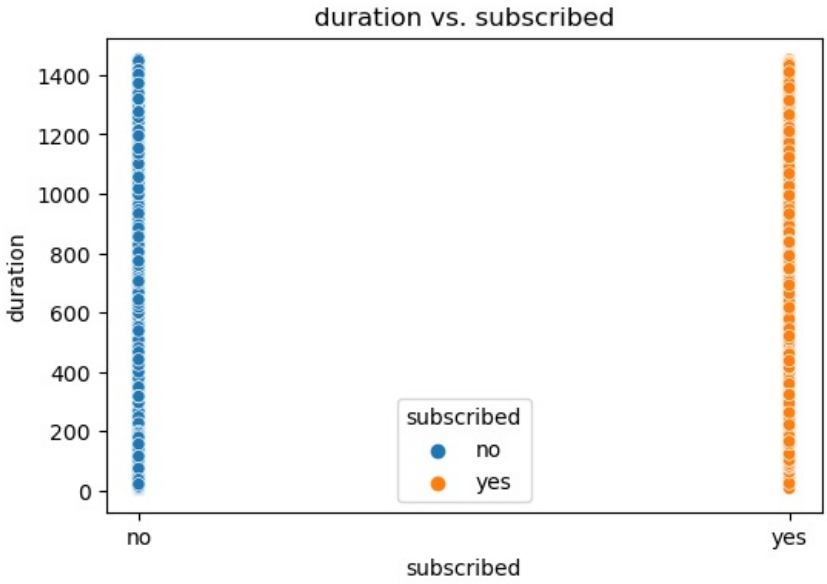

housing vs. subscribed
housing
1.0
0.8
0.6
0.4
0.2
0.0
subscribed
no
yes
no
yes
subscribed

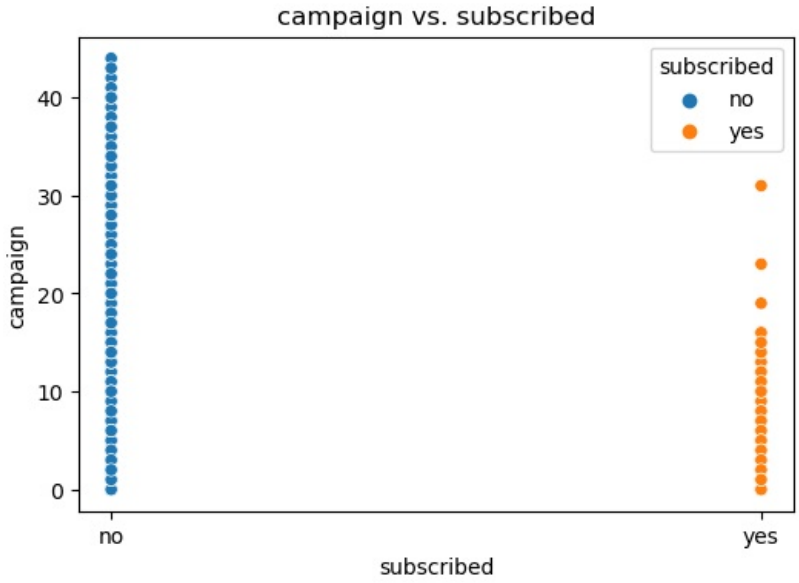

loan vs. subscribed
loan
1.0
0.8
0.6
0.4
0.2
0.0
subscribed
no
yes
no
yes
subscribed

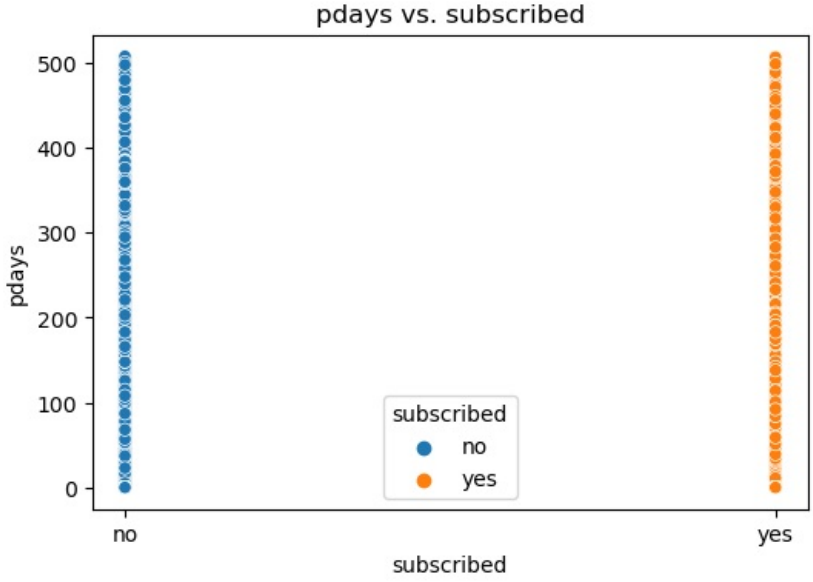

contact vs. subscribed
contact
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
subscribed
no
yes
no
yes
subscribed

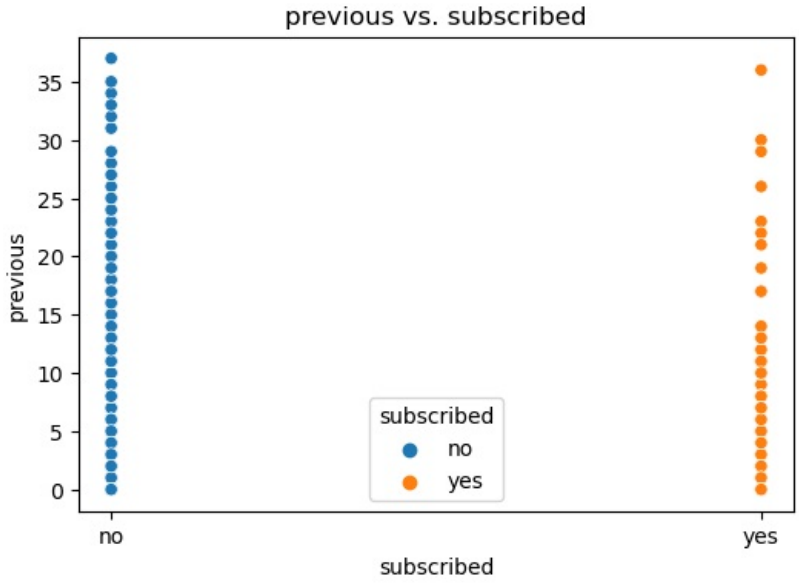

day vs. subscribed
day
30
25
20
15
10
5
0
subscribed
no
yes
no
yes
subscribed

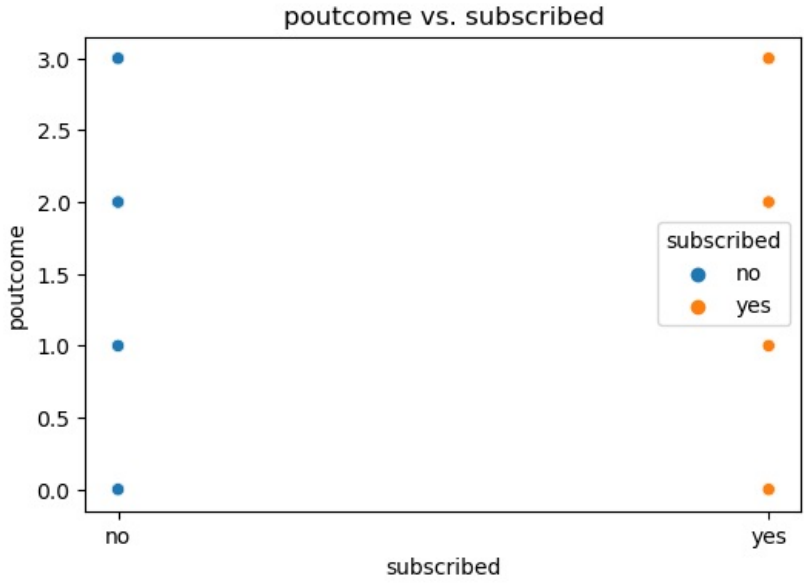

month vs. subscribed
month
10
8
6
4
2
0
subscribed
no
yes
no
yes
subscribed


duration vs. subscribed
duration
1400
1200
1000
800
600
400
200
0
subscribed
no
yes
no
yes
subscribed


campaign vs. subscribed
campaign
40
30
20
10
0
subscribed
no
yes
no
yes
subscribed


pdays vs. subscribed
pdays
500
400
300
200
100
0
subscribed
no
yes
no
yes
subscribed

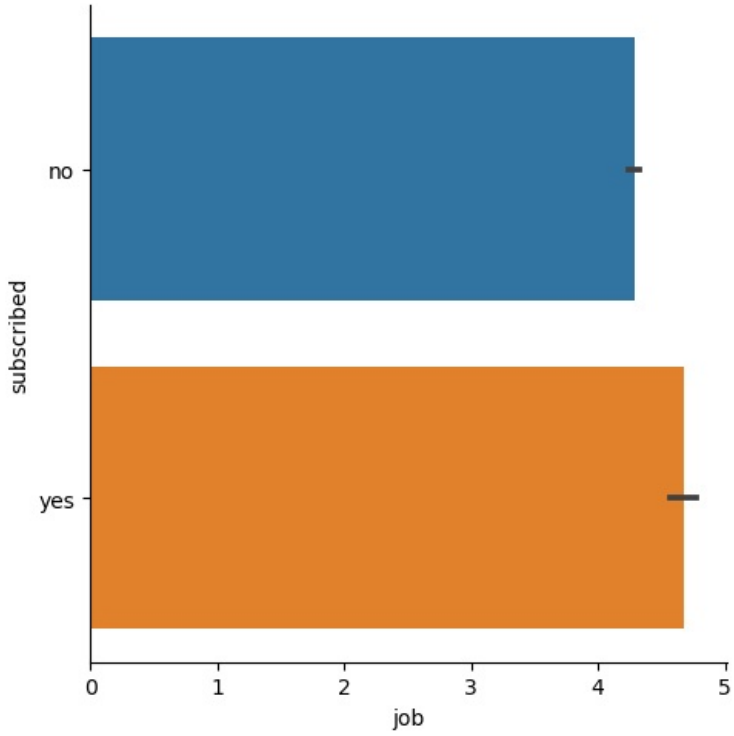




As we can see from the scatter plot, we were unable to obtain all the information necessary for the cat plot.

In [56]:
```
plt.figure(figsize=(6,4))
sns.catplot(x='job',y='subscribed',data=df_tr,kind='bar')
plt.show()
```

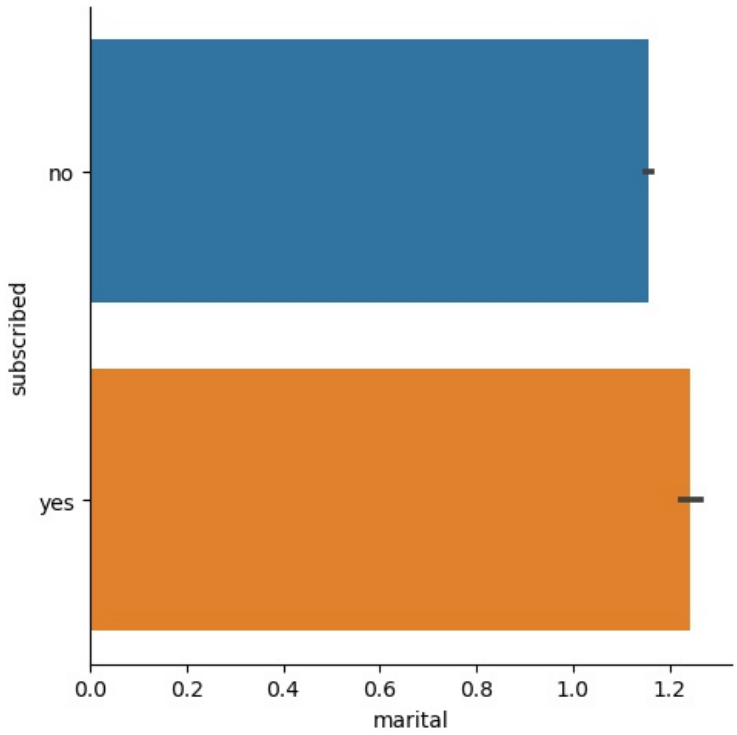
```
<Figure size 600x400 with 0 Axes>
```



We notice that the bulk of subscribers work 8 and 5 type jobs.

In [57]:
```
plt.figure(figsize=(6,4))
sns.catplot(x='marital',y='subscribed',data=df_tr,kind='bar')
plt.show()
```

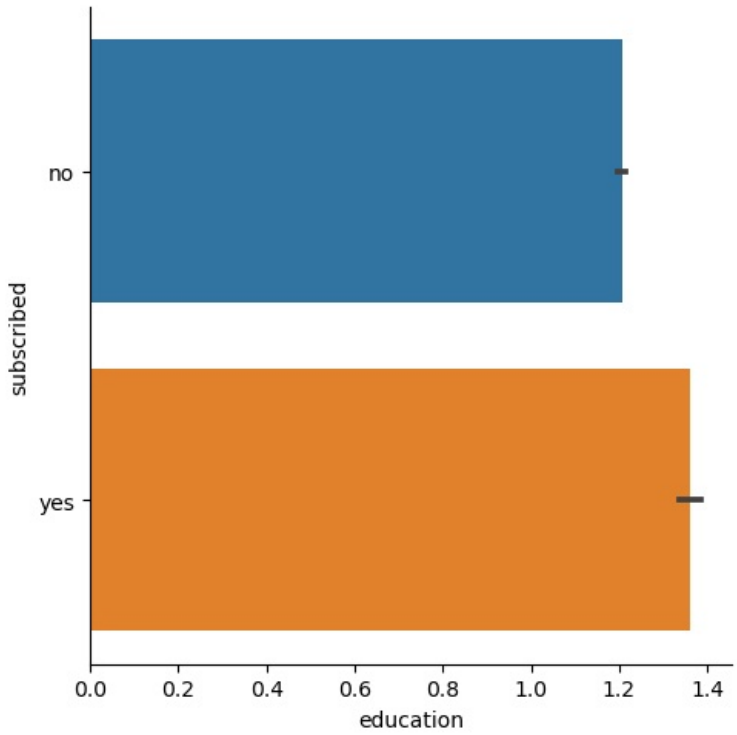
```
<Figure size 600x400 with 0 Axes>
```



As we look at marital data, we see that type 2 subscriptions are the majority.

In [58]:
```
plt.figure(figsize=(6,4))
sns.catplot(x='education',y='subscribed',data=df_tr,kind='bar')
plt.show()
```

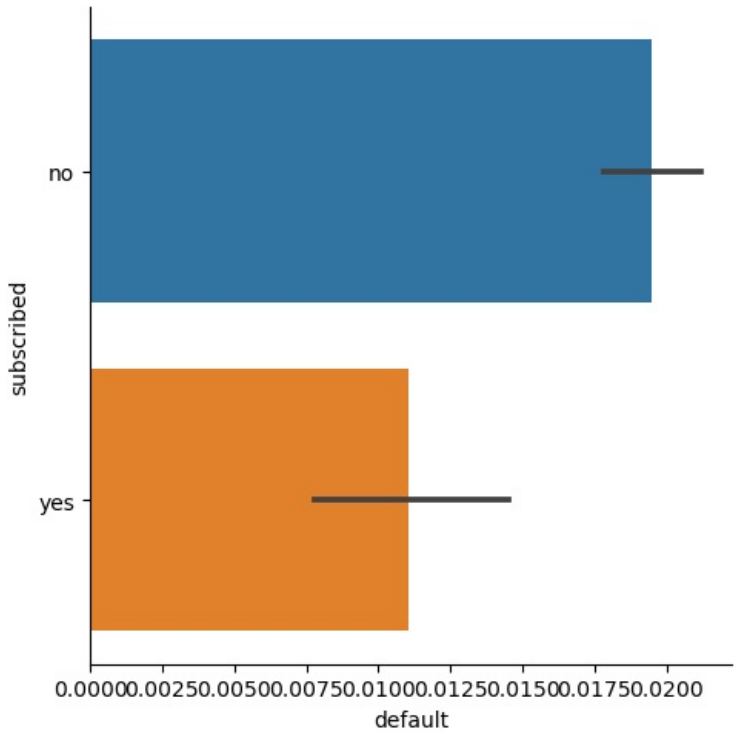
```
<Figure size 600x400 with 0 Axes>
```



When it comes to schooling, the bulk of subscribers come from levels 2 and 3.

In [59]:
```
plt.figure(figsize=(6,4))
sns.catplot(x='default',y='subscribed',data=df_tr,kind='bar')
plt.show()
```

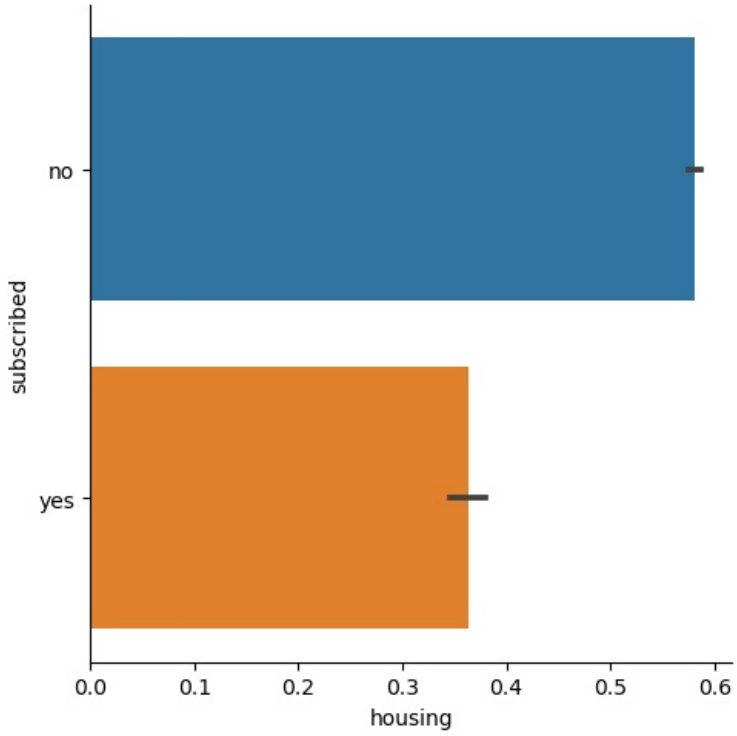
```
<Figure size 600x400 with 0 Axes>
```



As we watch for defaults, we see that non-default subscriptions are in the majority.

In [60]:
```
plt.figure(figsize=(6,4))
sns.catplot(x='housing',y='subscribed',data=df_tr,kind='bar')
plt.show()
```

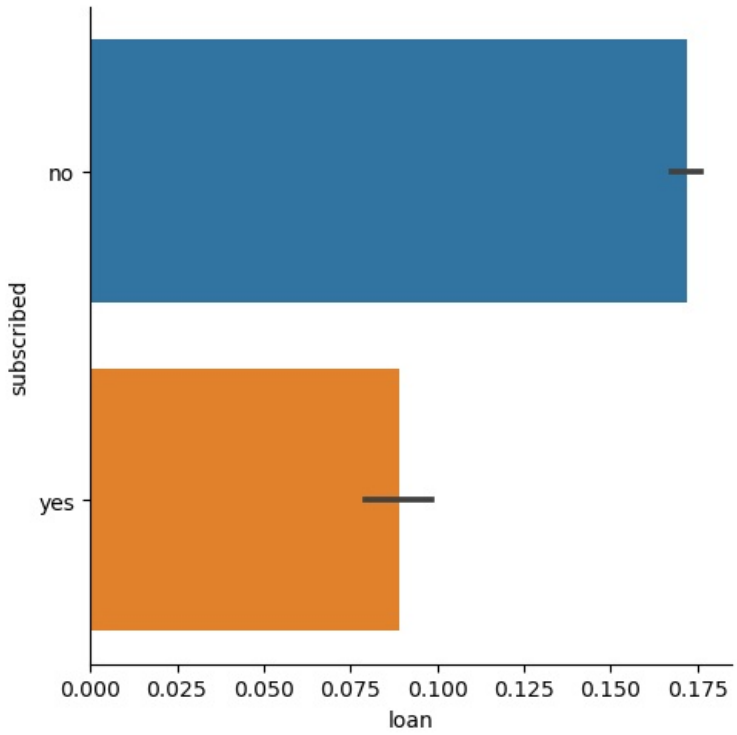
```
<Figure size 600x400 with 0 Axes>
```



As far as housing is concerned, the majority of subscribers do not take out loans.

In [61]:
```
plt.figure(figsize=(6,4))
sns.catplot(x='loan',y='subscribed',data=df_tr,kind='bar')
plt.show()
```

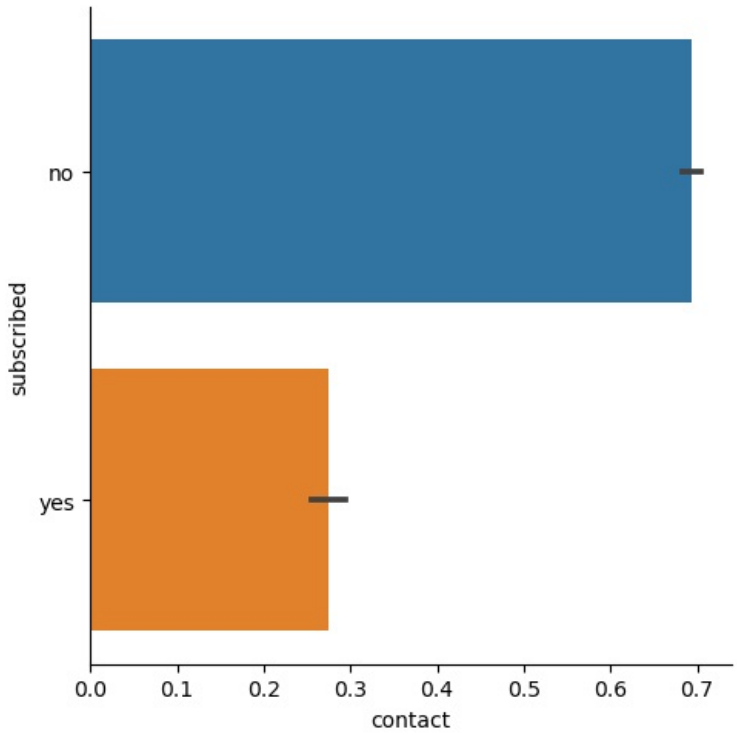
```
<Figure size 600x400 with 0 Axes>
```



According to our observations, the majority of people who have not taken out a personal loan are subscribers.

In [62]:
```
plt.figure(figsize=(6,4))
sns.catplot(x='contact',y='subscribed',data=df_tr,kind='bar')
plt.show()
```

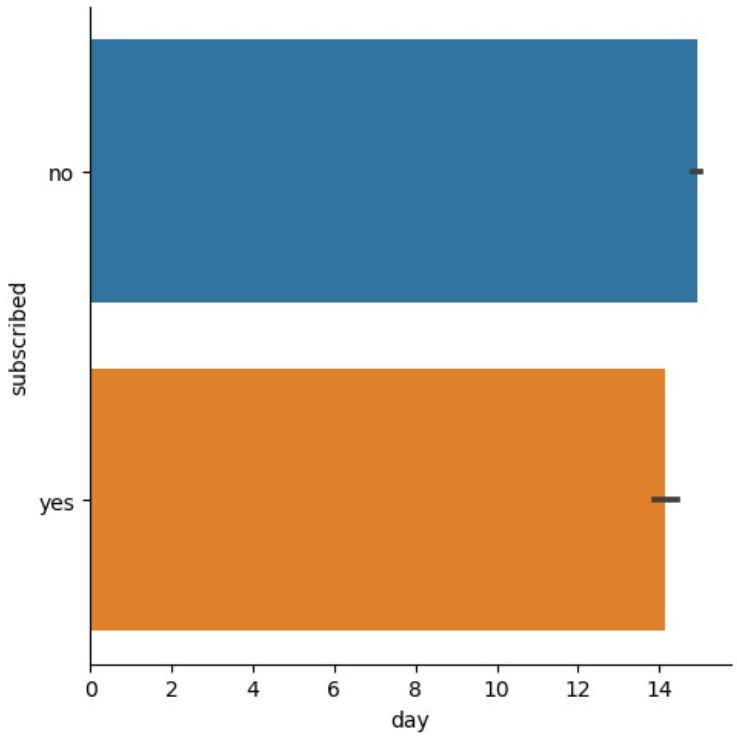
```
<Figure size 600x400 with 0 Axes>
```



As we look at subscribed contacts, we see that the majority are type 0 contacts.

In [63]:
```
plt.figure(figsize=(6,4))
sns.catplot(x='day',y='subscribed',data=df_tr,kind='bar')
plt.show()
```

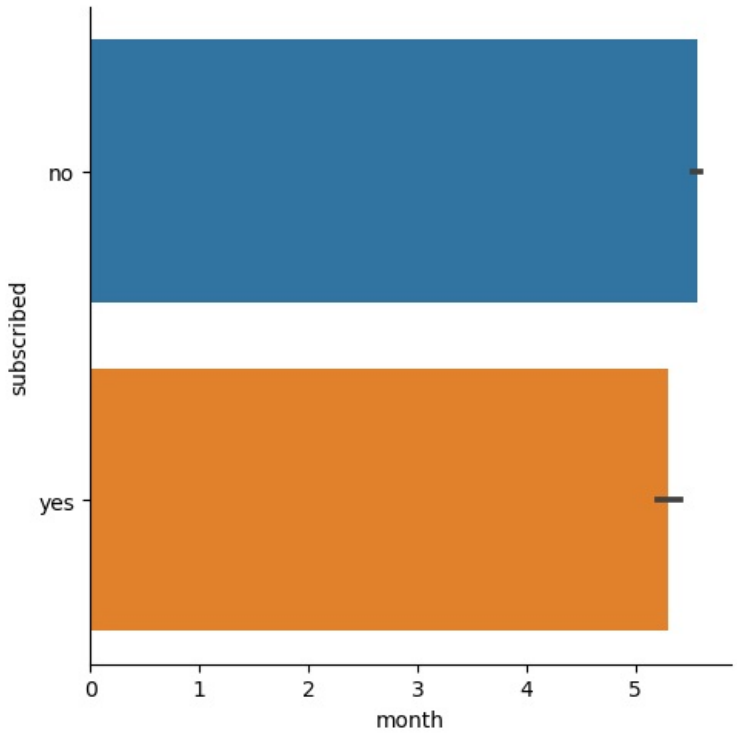
```
<Figure size 600x400 with 0 Axes>
```



As we go through the day, we notice that the majority of subscribers are on days 1, 10, and 32.

In [64]:
```
plt.figure(figsize=(6,4))
sns.catplot(x='month',y='subscribed',data=df_tr,kind='bar')
plt.show()
```

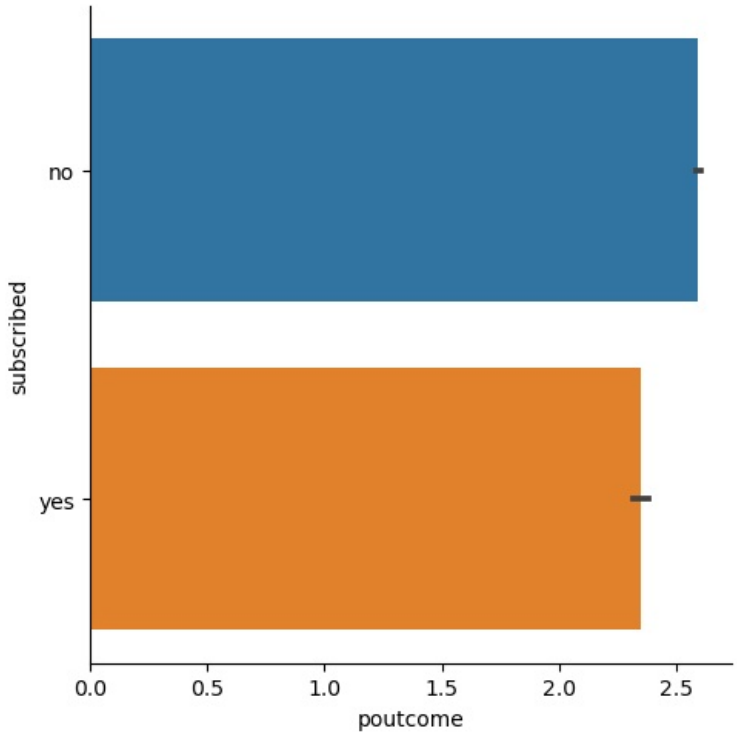
```
<Figure size 600x400 with 0 Axes>
```



As we track the months, we see that the months 2, 7, 10, and 11 have the most subscribers.

In [65]:
```
sns.catplot(x='poutcome',y='subscribed',data=df_tr,kind='bar')
```

Out[65]:
```
<seaborn.axisgrid.FacetGrid at 0x21fec77d670>
```



According to the results, the majority of respondents are type 2 subscribers.

Having finished our bivariate analysis, we can now go on to the multivariate analysis.

Multivariate Research

In [66]:
```
df_tr.corr()
```

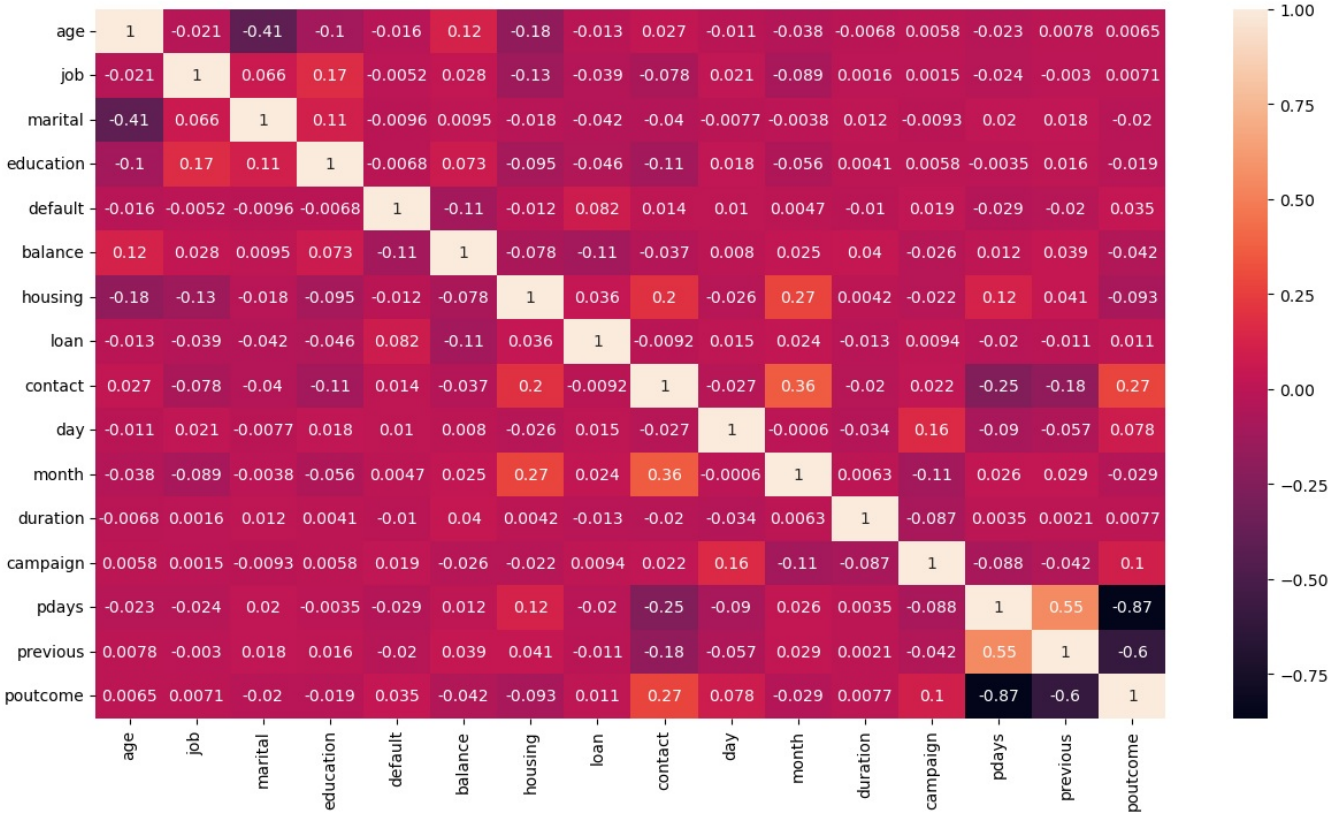
Out[66]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	c
age	1.000000	-0.020785	-0.406096	-0.102359	-0.016113	0.123262	-0.183755	-0.012631	0.026747	-0.011047	-0.038380	-0.006820	
job	-0.020785	1.000000	0.066124	0.170347	-0.005217	0.028363	-0.125347	-0.039049	-0.078253	0.020951	-0.089384	0.001568	
marital	-0.406096	0.066124	1.000000	0.107547	-0.009591	0.009503	-0.017846	-0.041654	-0.040180	-0.007686	-0.003834	0.012132	-
education	-0.102359	0.170347	0.107547	1.000000	-0.006774	0.073391	-0.095376	-0.045950	-0.112000	0.017902	-0.055967	0.004107	
default	-0.016113	-0.005217	-0.009591	-0.006774	1.000000	-0.112012	-0.011823	0.081567	0.014053	0.009996	0.004694	-0.010035	
balance	0.123262	0.028363	0.009503	0.073391	-0.112012	1.000000	-0.078310	-0.114014	-0.036729	0.007979	0.025117	0.039534	-
housing	-0.183755	-0.125347	-0.017846	-0.095376	-0.011823	-0.078310	1.000000	0.036244	0.195103	-0.025666	0.271442	0.004227	-
loan	-0.012631	-0.039049	-0.041654	-0.045950	0.081567	-0.114014	0.036244	1.000000	-0.009213	0.014769	0.024186	-0.013499	
contact	0.026747	-0.078253	-0.040180	-0.112000	0.014053	-0.036729	0.195103	-0.009213	1.000000	-0.027046	0.362934	-0.020445	
day	-0.011047	0.020951	-0.007686	0.017902	0.009996	0.007979	-0.025666	0.014769	-0.027046	1.000000	-0.000604	-0.034033	
month	-0.038380	-0.089384	-0.003834	-0.055967	0.004694	0.025117	0.271442	0.024186	0.362934	-0.000604	1.000000	0.006256	-
duration	-0.006820	0.001568	0.012132	0.004107	-0.010035	0.039534	0.004227	-0.013499	-0.020445	-0.034033	0.006256	1.000000	-
campaign	0.005781	0.001504	-0.009349	0.005805	0.018730	-0.026384	-0.021613	0.009357	0.022402	0.159929	-0.106394	-0.087361	
pdays	-0.023029	-0.023661	0.020328	-0.003459	-0.029432	0.012430	0.123482	-0.019742	-0.249535	-0.089611	0.026139	0.003479	-
previous	0.007787	-0.002984	0.017754	0.016342	-0.020128	0.039181	0.040659	-0.011372	-0.181549	-0.057131	0.029266	0.002061	-
poutcome	0.006468	0.007101	-0.019770	-0.019007	0.035067	-0.042371	-0.092677	0.010919	0.272083	0.077586	-0.029369	0.007738	

We use a heat map to better visualise what we can't see clearly.

In [67]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df_tr.corr(),annot=True)
```

Out[67]:
```
<AxesSubplot:>
```



lets list out the close variables to our target variable 'Subscribed' Subscribed vs housing Subscribed vs contact Subscribed vs duration Subscribed vs pdays Now plot correlation for features.

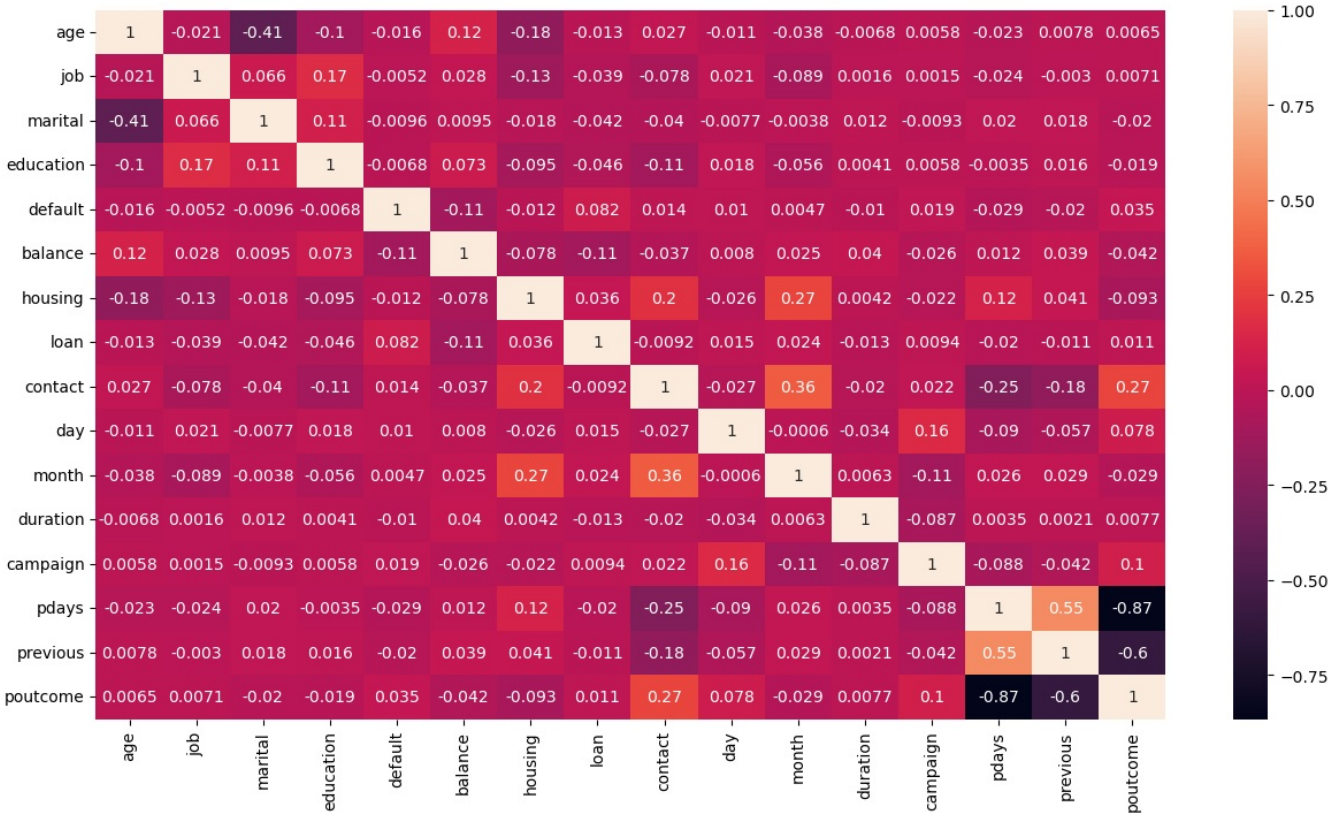
In [68]:
```
df_tr.drop(['subscribed'],axis=1).corr()
```

Out[68]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	c
age	1.000000	-0.020785	-0.406096	-0.102359	-0.016113	0.123262	-0.183755	-0.012631	0.026747	-0.011047	-0.038380	-0.006820	
job	-0.020785	1.000000	0.066124	0.170347	-0.005217	0.028363	-0.125347	-0.039049	-0.078253	0.020951	-0.089384	0.001568	
marital	-0.406096	0.066124	1.000000	0.107547	-0.009591	0.009503	-0.017846	-0.041654	-0.040180	-0.007686	-0.003834	0.012132	-
education	-0.102359	0.170347	0.107547	1.000000	-0.006774	0.073391	-0.095376	-0.045950	-0.112000	0.017902	-0.055967	0.004107	
default	-0.016113	-0.005217	-0.009591	-0.006774	1.000000	-0.112012	-0.011823	0.081567	0.014053	0.009996	0.004694	-0.010035	
balance	0.123262	0.028363	0.009503	0.073391	-0.112012	1.000000	-0.078310	-0.114014	-0.036729	0.007979	0.025117	0.039534	-
housing	-0.183755	-0.125347	-0.017846	-0.095376	-0.011823	-0.078310	1.000000	0.036244	0.195103	-0.025666	0.271442	0.004227	-
loan	-0.012631	-0.039049	-0.041654	-0.045950	0.081567	-0.114014	0.036244	1.000000	-0.009213	0.014769	0.024186	-0.013499	
contact	0.026747	-0.078253	-0.040180	-0.112000	0.014053	-0.036729	0.195103	-0.009213	1.000000	-0.027046	0.362934	-0.020445	
day	-0.011047	0.020951	-0.007686	0.017902	0.009996	0.007979	-0.025666	0.014769	-0.027046	1.000000	-0.000604	-0.034033	
month	-0.038380	-0.089384	-0.003834	-0.055967	0.004694	0.025117	0.271442	0.024186	0.362934	-0.000604	1.000000	0.006256	-
duration	-0.006820	0.001568	0.012132	0.004107	-0.010035	0.039534	0.004227	-0.013499	-0.020445	-0.034033	0.006256	1.000000	-
campaign	0.005781	0.001504	-0.009349	0.005805	0.018730	-0.026384	-0.021613	0.009357	0.022402	0.159929	-0.106394	-0.087361	
pdays	-0.023029	-0.023661	0.020328	-0.003459	-0.029432	0.012430	0.123482	-0.019742	-0.249535	-0.089611	0.026139	0.003479	-
previous	0.007787	-0.002984	0.017754	0.016342	-0.020128	0.039181	0.040659	-0.011372	-0.181549	-0.057131	0.029266	0.002061	-
poutcome	0.006468	0.007101	-0.019770	-0.019007	0.035067	-0.042371	-0.092677	0.010919	0.272083	0.077586	-0.029369	0.007738	

In [69]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df_tr.drop(['subscribed'],axis=1).corr(),annot=True)
```

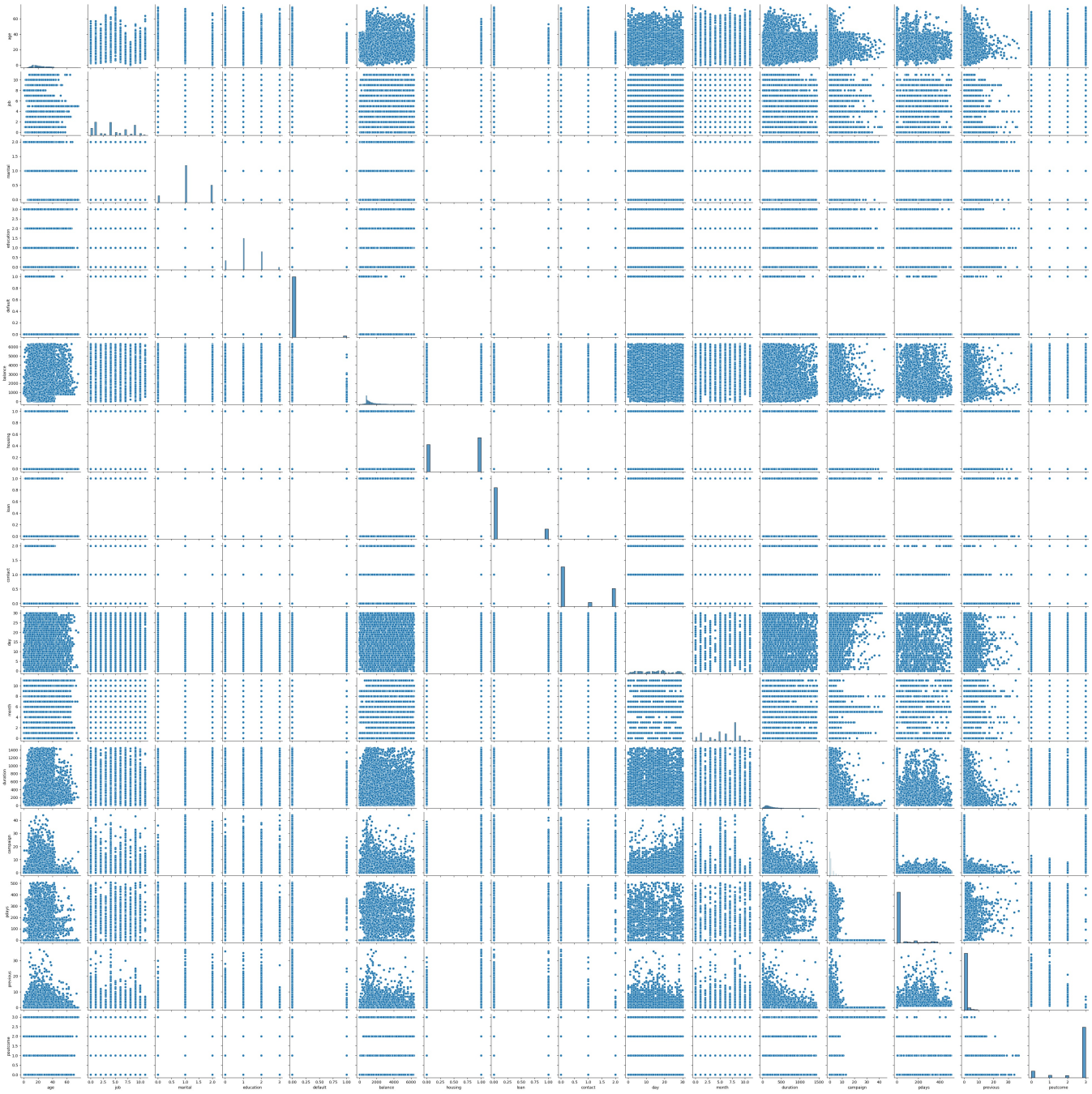
Out[69]:
```
<AxesSubplot:>
```



Relationship. age vs marital job vs education job vs housing education vs marital housing vs contact housing vs month contact vs poutcome contact vs months day vs campaign pdays vs previous pdays vs poutcome previous vs poutcome

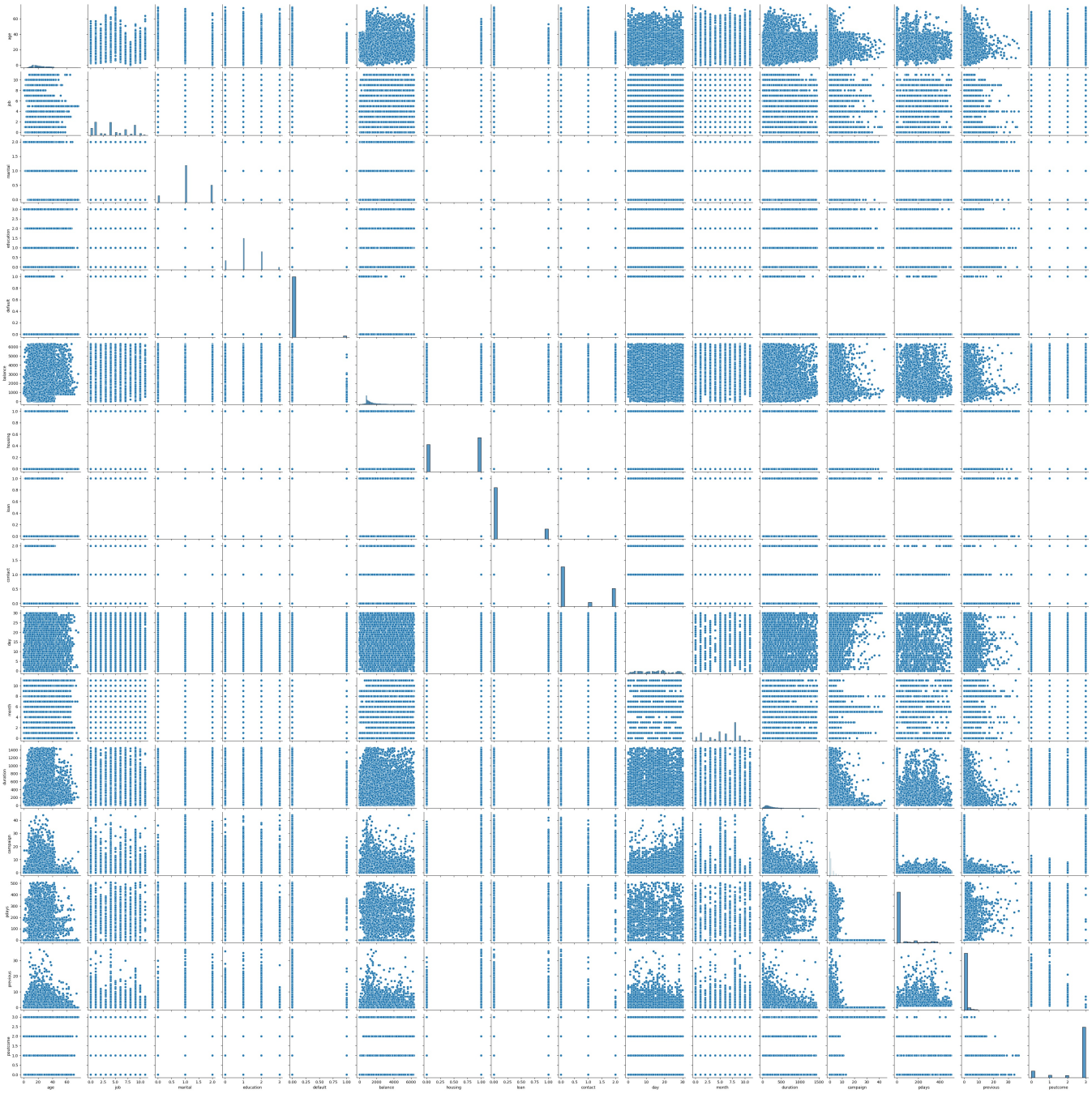
In [72]:
```
# With the target variable, pair plot
sns.pairplot(df_tr)
```

Out[72]:
```
<seaborn.axisgrid.PairGrid at 0x21fec564940>
```



In [73]:
```
sns.pairplot(df_tr.drop(['subscribed'],axis=1))
```

Out[73]:
```
<seaborn.axisgrid.PairGrid at 0x21fa2d61ca0>
```



Looking at the Statistics

In [74]:
```
df_tr.describe()
```

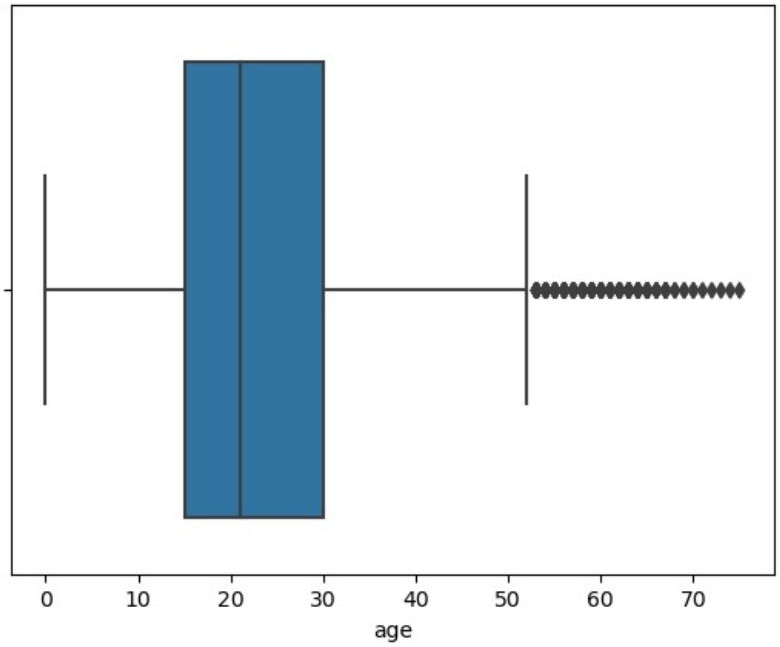
Out[74]:

	age	job	marital	education	default	balance	housing	loan	contact	
count	31647.000000	31647.000000	31647.000000	31647.000000	31647.000000	31647.000000	31647.000000	31647.000000	31647.000000	3164
mean	22.956520	4.332923	1.167220	1.225013	0.018485	1822.242266	0.555629	0.162132	0.644642	1
std	10.621873	3.271868	0.607186	0.749195	0.134700	1352.093883	0.496904	0.368578	0.899480	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	15.000000	1.000000	1.000000	1.000000	0.000000	892.000000	0.000000	0.000000	0.000000	
50%	21.000000	4.000000	1.000000	1.000000	0.000000	1269.000000	1.000000	0.000000	0.000000	1
75%	30.000000	7.000000	2.000000	2.000000	0.000000	2245.000000	1.000000	0.000000	2.000000	2
max	75.000000	11.000000	2.000000	3.000000	1.000000	6325.000000	1.000000	1.000000	2.000000	3

Checking to see if there are any outliers at this point.

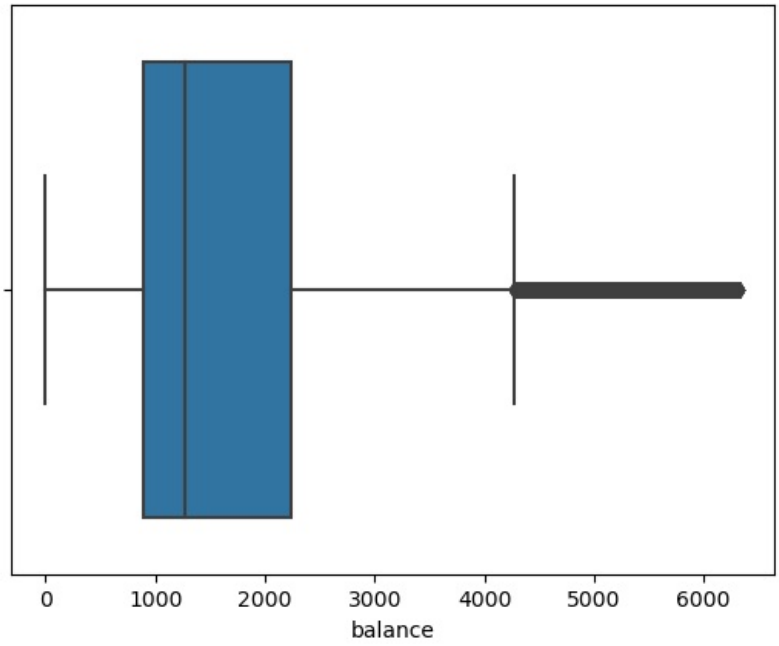
Looking for anomalies in the features.

In [75]:
```
plt.plot(figsize=(6,4))
sns.boxplot(df_tr['age'])
plt.show()
```



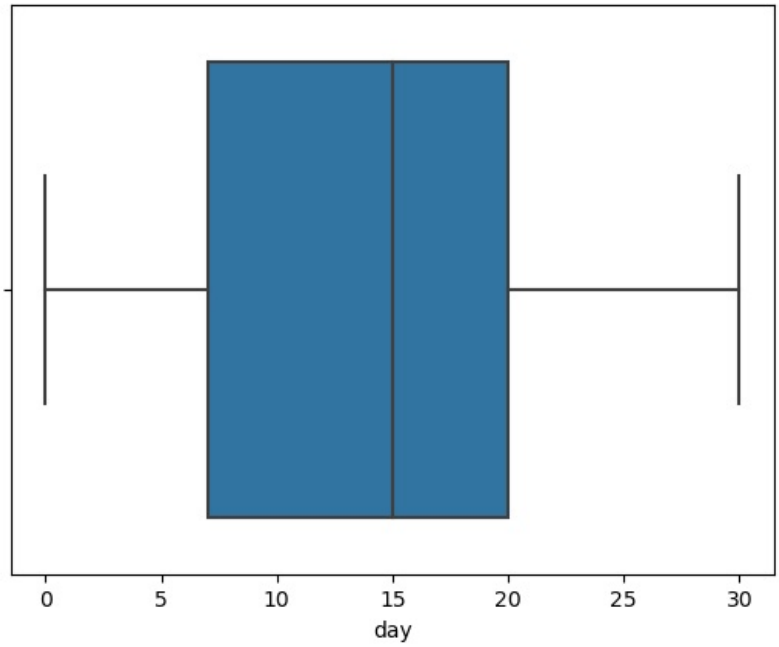
Since there are more outliers, we may see this.

In [76]:
```
plt.plot(figsize=(6,4))
sns.boxplot(df_tr['balance'])
plt.show()
```



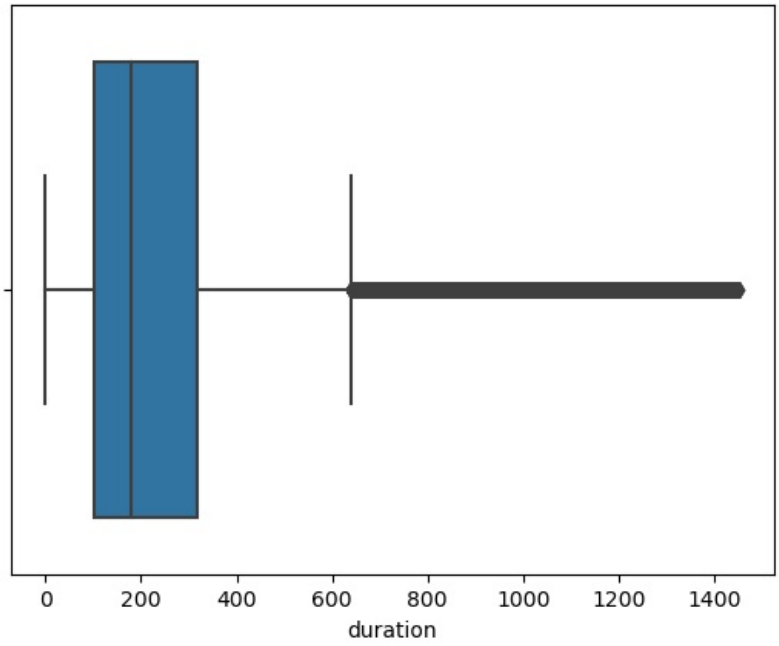
As can be seen, the majority of outliers are present in these columns.

In [77]:
```
plt.plot(figsize=(6,4))
sns.boxplot(df_tr['day'])
plt.show()
```



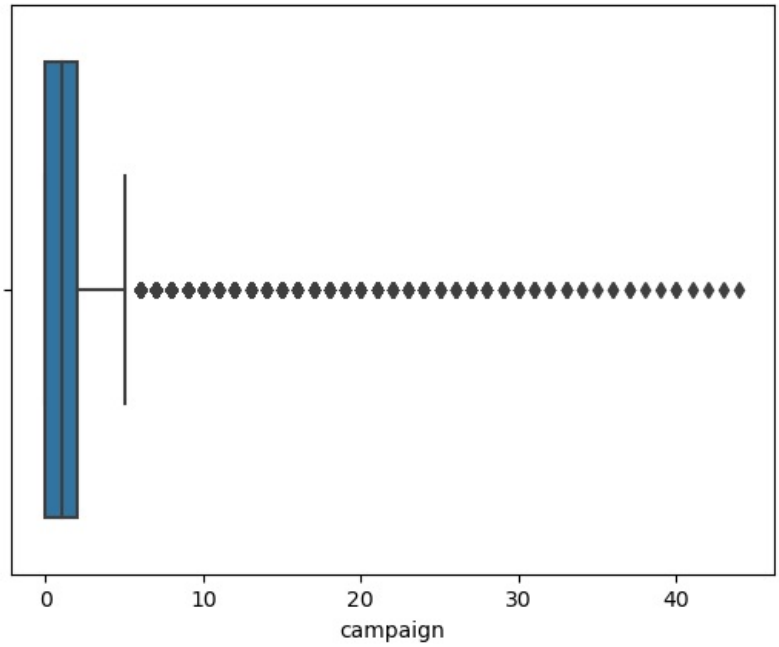
As we can see, there are no outliers in these columns.

In [78]:
```
plt.plot(figsize=(6,4))
sns.boxplot(df_tr['duration'])
plt.show()
```



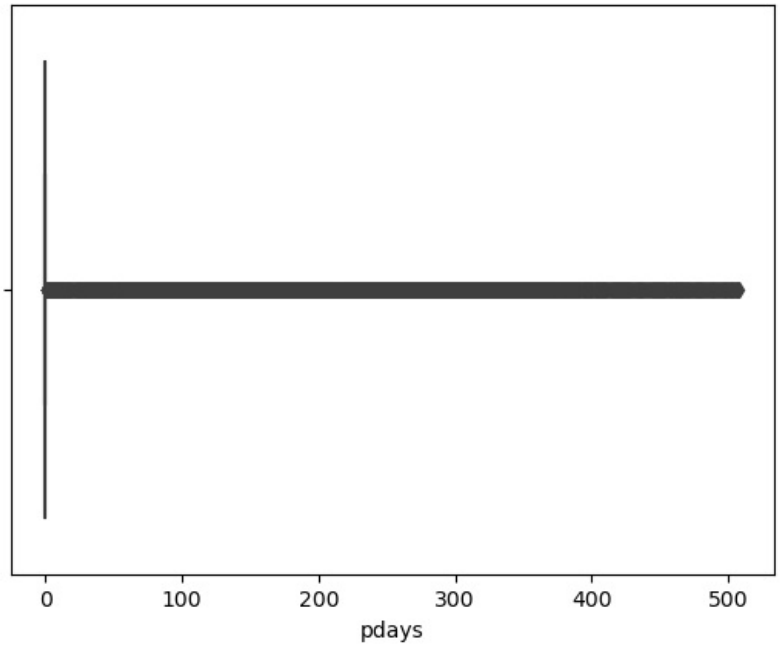
Since there are more outliers, we may see this.

In [79]:
```
plt.plot(figsize=(6,4))
sns.boxplot(df_tr['campaign'])
plt.show()
```



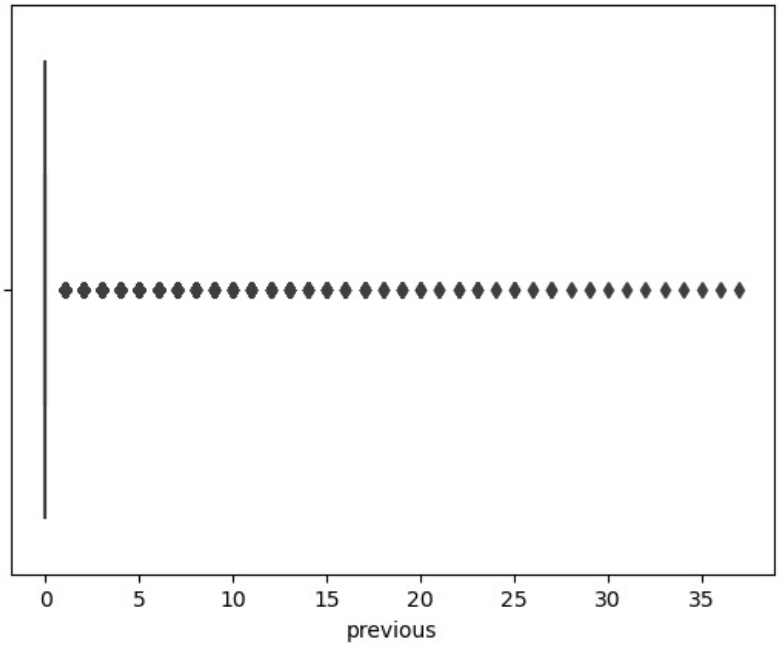
when we notice that this column contains the majority of outliers.

In [80]:
```
plt.plot(figsize=(6,4))
sns.boxplot(df_tr['pdays'])
plt.show()
```



seeing as how the bulk are anomalies.

In [81]:
```
plt.plot(figsize=(6,4))
sns.boxplot(df_tr['previous'])
plt.show()
```



Most of the columns in this list are outliers.

We should eliminate Previous, Pdays, and balance because they currently have too many outliers. To remove these columns, we first examine their skewness and if it is high, we remove them.

In [82]:
```
df_tr['previous'].skew()
```

Out[82]:
```
6.739236991675542
```

In [83]:
```
df_tr['pdays'].skew()
```

Out[83]:
```
2.456606849432811
```

In [84]:
```
df_tr['balance'].skew()
```

Out[84]:
```
1.5408527490653934
```

We should eliminate it since there is too much skewness in these columns, as we have noticed.

In [85]:
```
df_tr.drop(['previous','pdays','balance'],axis=1,inplace=True)
```

In [86]:
```
df_tr.head()
```

Out[86]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	poutcome	subscribed
0	38	0	1	3	0	0	0	1	18	9	43	1	3	no
1	13	11	1	1	0	0	0	0	19	5	90	1	3	no
2	9	7	1	1	0	1	0	0	17	5	239	0	3	no
3	39	4	0	2	0	0	0	0	21	6	864	0	2	yes
4	13	9	1	1	0	1	0	0	3	3	379	0	3	no

As we watch, columns are taken out.

We must divide our data into features and a target DataFrame before removing outliers from the feature columns in order to remove outliers.

In [87]:
```
x=df_tr.drop(['subscribed'],axis=1)
y=df_tr['subscribed']
```

In [88]:
```
x.head(2)
```

Out[88]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	poutcome
0	38	0	1	3	0	0	0	1	18	9	43	1	3
1	13	11	1	1	0	0	0	0	19	5	90	1	3

In [89]:
```
y.head(5)
```

Out[89]:
```
0     no
1     no
2     no
3    yes
4     no
Name: subscribed, dtype: object
```

We can now see that our data has been divided.

We now eliminate outliers from only features columns.

In [90]:
```
from scipy.stats import zscore
z=np.abs(zscore(x))
threshold=3

x=x[(z<3).all(axis=1)]
print(x.shape)

y=y[(z<3).all(axis=1)]
print(y.shape)
```

```
(29428, 13)
(29428,)
```

% Data loss check

In [91]:
```
loss=round((31647-29558)/31647*100,2)
loss
```

Out[91]:
```
6.6
```

Observably, we detest just 6.6% of the data.

Checking for skewness now, and if it exists, eliminate it using the best technique.

In [92]:
```
x.skew()
```

Out[92]:
```
age          0.449298
job          0.264750
marital     -0.101252
education    0.196828
default      0.000000
housing     -0.248198
loan         1.865499
contact      0.767699
day          0.100236
month       -0.505261
duration     1.504138
campaign     2.067293
poutcome    -1.945938
dtype: float64
```

The remainder of all acceptable and and which are not acceptable is it converted object to must be removed from the duration and campaign.

In [93]:
```
from scipy.stats import boxcox,yeojohnson
from scipy.stats import skew
```

In [94]:
```
#Determining the most effective strategy for skewness for both positive and negative numbers.
def skee(a):
    model=[np.sqrt(a),np.log(a),yeojohnson(a)[0]]
    print('original skewness is:',a.skew())
    print('\n')
    for m in model:
        x=m
        print(skew(m))
        print('\n')
```

In [95]:
```
#Finding the optimal skewness calculation method for just positive values.
def skeep(a):
    model=[np.sqrt(a),np.log(a),boxcox(a)[0],yeojohnson(a)[0]]
    print('original skewness is:',a.skew())
    print('\n')
    for m in model:
        x=m
        print(skew(m))
        print('\n')
```

now looking for independent variables one by one. When using a function, we start by using the skeep function because it has all of the options.

In [96]:
```
skee(x['duration'])
```

```
original skewness is: 1.504137758397215


0.6005787705715926


nan


0.0023467131068794812
```

Yeojohnson, as we can see, works well to eliminate skewness in this column.

In [97]:
```
x['duration']=yeojohnson(x['duration'])[0]
```

In [98]:
```
skee(x['campaign'])
```

```
original skewness is: 2.0672925110814595


0.4917682307550108


nan


0.1500425951749947
```

We can see that boxcox works best to eliminate skewness in this column.

In [100…
```
x.skew()
```

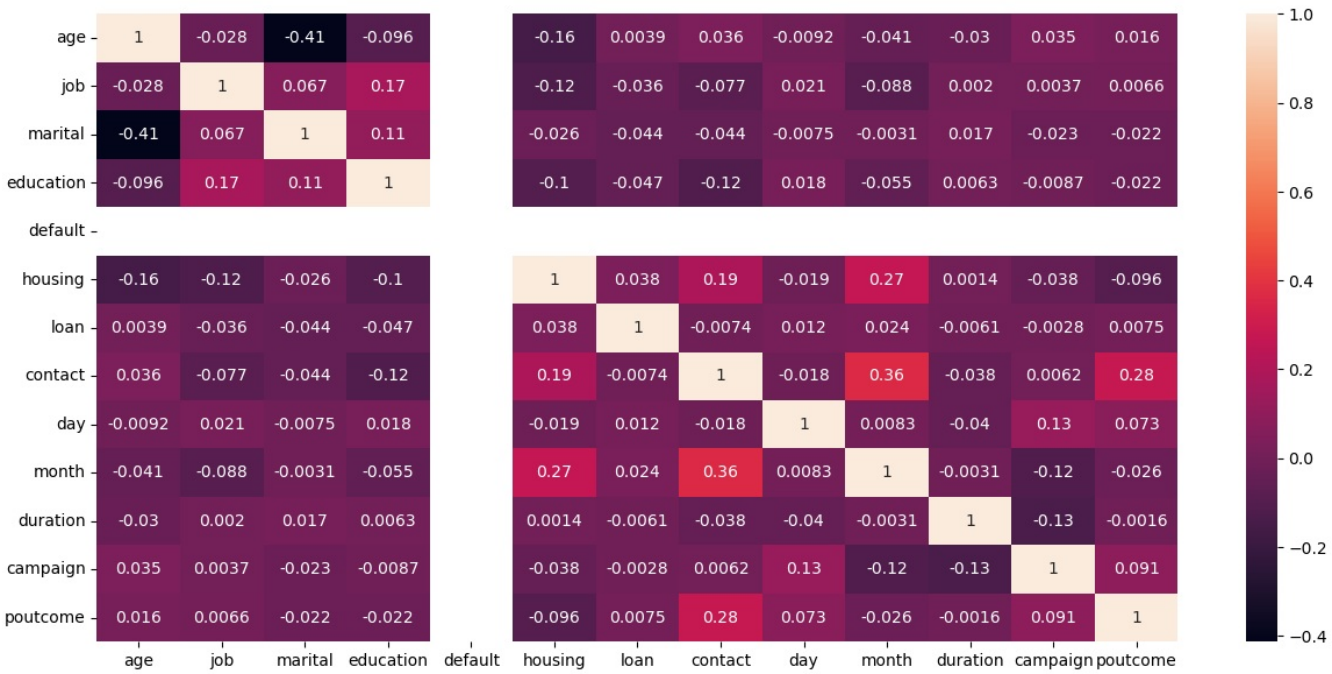
Out[100]:
```
age          0.449298
job          0.264750
marital     -0.101252
education    0.196828
default      0.000000
housing     -0.248198
loan         1.865499
contact      0.767699
day          0.100236
month       -0.505261
duration     0.002347
campaign     2.067293
poutcome    -1.945938
dtype: float64
```

We can now see that the skewness has been eliminated, or accepted.

Next, verifying the independent variables for multicollinearity.

In [101…
```
plt.figure(figsize=(15,7))
sns.heatmap(x.corr(),annot=True)
```

Out[101]:
```
<AxesSubplot:>
```



In [102…
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [103…
```
def vif_calc():
    vif=pd.DataFrame()
    vif["VIF Factor"]=[variance_inflation_factor(x.values,i) for i in range(x.shape[1])]
    vif['features']=x.columns
    print(vif)
```

In [104…
```
vif_calc()
```

```
    VIF Factor   features
0     5.422443        age
1     2.793359        job
2     4.700817    marital
3     3.737618  education
4          NaN    default
5     2.472995    housing
6     1.190587       loan
7     1.972084    contact
8     4.052702        day
9     5.124384      month
10   10.726093   duration
11    1.618088   campaign
12    7.472763   poutcome
```

We now see that the age and duration both have values above 10, therefore we first delete the duration before checking the VIF.

In [105…
```
x.drop(['duration'],axis=1,inplace=True)
```

In [106…
```
def vif_calc():
    vif=pd.DataFrame()
    vif["VIF Factor"]=[variance_inflation_factor(x.values,i) for i in range(x.shape[1])]
    vif['features']=x.columns
    print(vif)
```

```
vif_calc()
```

```
    VIF Factor   features
0     4.684862        age
1     2.763295        job
2     4.252491    marital
3     3.656259  education
4          NaN    default
5     2.392052    housing
6     1.188793       loan
7     1.950684    contact
8     4.006833        day
9     4.985226      month
10    1.611584   campaign
11    6.902713   poutcome
```

We can now see that every VIF number is valid.

Scaling the input data

In [107…
```
#We employ a conventional scaler for scaling.
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)
x
```

Out[107]:
```
array([[ 1.51982216, -1.31718769, -0.28278261, ...,  1.14547748,
        -0.23087928,  0.44931293],
       [-0.95521605,  2.0300211 , -0.28278261, ..., -0.18482151,
        -0.23087928,  0.44931293],
       [-1.35122216,  0.81285427, -0.28278261, ..., -0.18482151,
        -0.75008419,  0.44931293],
       ...,
       [ 1.22281757, -0.10002085, -1.93260949, ...,  0.14775324,
        -0.23087928,  0.44931293],
       [-0.85621452, -0.10002085,  1.36704426, ..., -1.5151205 ,
         0.28832563,  0.44931293],
       [ 1.61882368,  1.42143769, -0.28278261, ...,  0.81290273,
         2.36514527, -2.56353398]])
```

Verifying that our data is balanced, and if not, taking appropriate action.

In [108…
```
y.value_counts()
```

Out[108]:
```
no     26357
yes     3071
Name: subscribed, dtype: int64
```

We must balance our data since, as we have seen, it is not balanced. We can employ the SMOTE approach for it.

To balance the data, we employ an oversampling method here.

In [109…
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\lenovo\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\lenovo\anaconda3\lib\site-packages (from imblearn)
(0.11.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\lenovo\anaconda3\lib\site-packages (from imbalanced-le
arn->imblearn) (1.21.5)
Requirement already satisfied: joblib>=1.1.1 in c:\users\lenovo\anaconda3\lib\site-packages (from imbalanced-le
arn->imblearn) (1.3.2)
Requirement already satisfied: scipy>=1.5.0 in c:\users\lenovo\anaconda3\lib\site-packages (from imbalanced-lea
rn->imblearn) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\lenovo\anaconda3\lib\site-packages (from imbala
nced-learn->imblearn) (2.2.0)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\lenovo\anaconda3\lib\site-packages (from imbalan
ced-learn->imblearn) (1.0.2)
```

In [110…
```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
x,y=sm.fit_resample(x,y)
```

In [111…
```
y.value_counts()
```

Out[111]:
```
no     26357
yes    26357
Name: subscribed, dtype: int64
```

Now it is quite evident that we have used oversampling to balance our data.

We should employ a classifier model because our target variable is seen to be of a binary kind.

In [47]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
```

```
from sklearn.metrics import classification_report,confusion_matrix,roc_auc_score,roc_curve,accuracy_score
```

This for loop can be used to discover the optimal random state.

Since our model only has two values, stratifying is used in this instance since it is crucial to divide the data appropriately.0 and 1 have the same info.

In [113…

```
for i in range(0,1000):
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=i,test_size=0.25,stratify=y)
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    if round(accuracy_score(y_train,pred_train)*100,1)==round(accuracy_score(y_test,pred_test)*100,1):
        print('At random state',i,'The model perform very well')
        print('Random State = ',i)
        print("Training accuracy score is = ",accuracy_score(y_train,pred_train))
        print("Test accuracy score is = ",accuracy_score(y_test,pred_test))
        print('\n')
```

```
At random state 6 The model perform very well
Random State =  6
Training accuracy score is =  0.6831668142152523
Test accuracy score is =  0.683132255861598


At random state 13 The model perform very well
Random State =  13
Training accuracy score is =  0.6838750474263311
Test accuracy score is =  0.6843463085211321


At random state 26 The model perform very well
Random State =  26
Training accuracy score is =  0.6835209308207917
Test accuracy score is =  0.6836634039001441


At random state 57 The model perform very well
Random State =  57
Training accuracy score is =  0.6843050461616289
Test accuracy score is =  0.6844980651035739


At random state 74 The model perform very well
Random State =  74
Training accuracy score is =  0.6834956367775389
Test accuracy score is =  0.6832081341528189


At random state 76 The model perform very well
Random State =  76
Training accuracy score is =  0.6829897559124826
Test accuracy score is =  0.6827528644054935


At random state 78 The model perform very well
Random State =  78
Training accuracy score is =  0.683217402301758
Test accuracy score is =  0.6826769861142727


At random state 80 The model perform very well
Random State =  80
Training accuracy score is =  0.6830150499557355
Test accuracy score is =  0.6827528644054935


At random state 81 The model perform very well
Random State =  81
Training accuracy score is =  0.6837232831668142
Test accuracy score is =  0.6844980651035739


At random state 82 The model perform very well
Random State =  82
Training accuracy score is =  0.6834450486910333
Test accuracy score is =  0.6833598907352606


At random state 89 The model perform very well
Random State =  89
Training accuracy score is =  0.6843050461616289
Test accuracy score is =  0.6836634039001441


At random state 103 The model perform very well
Random State =  103
```

```
Training accuracy score is =  0.6835968129505502
Test accuracy score is =  0.6839669170650277


At random state 115 The model perform very well
Random State =  115
Training accuracy score is =  0.6830656380422411
Test accuracy score is =  0.6834357690264815


At random state 130 The model perform very well
Random State =  130
Training accuracy score is =  0.6835209308207917
Test accuracy score is =  0.6840427953562486


At random state 136 The model perform very well
Random State =  136
Training accuracy score is =  0.6831415201719995
Test accuracy score is =  0.683132255861598


At random state 137 The model perform very well
Random State =  137
Training accuracy score is =  0.684153281902112
Test accuracy score is =  0.6843463085211321


At random state 155 The model perform very well
Random State =  155
Training accuracy score is =  0.683622106993803
Test accuracy score is =  0.6839669170650277


At random state 165 The model perform very well
Random State =  165
Training accuracy score is =  0.6836979891235614
Test accuracy score is =  0.6841186736474695


At random state 179 The model perform very well
Random State =  179
Training accuracy score is =  0.6833185784747692
Test accuracy score is =  0.683132255861598


At random state 184 The model perform very well
Random State =  184
Training accuracy score is =  0.6830909320854939
Test accuracy score is =  0.6834357690264815


At random state 216 The model perform very well
Random State =  216
Training accuracy score is =  0.684431516377893
Test accuracy score is =  0.6840427953562486


At random state 236 The model perform very well
Random State =  236
Training accuracy score is =  0.6837232831668142
Test accuracy score is =  0.6843463085211321


At random state 244 The model perform very well
Random State =  244
Training accuracy score is =  0.6840015176425952
Test accuracy score is =  0.6835875256089233


At random state 245 The model perform very well
Random State =  245
Training accuracy score is =  0.6835968129505502
Test accuracy score is =  0.6835875256089233


At random state 267 The model perform very well
Random State =  267
Training accuracy score is =  0.6834956367775389
Test accuracy score is =  0.6829804992791563


At random state 288 The model perform very well
Random State =  288
Training accuracy score is =  0.6822056405716453
Test accuracy score is =  0.6822217163669474


At random state 300 The model perform very well
```

```
Random State =  300
Training accuracy score is =  0.6831668142152523
Test accuracy score is =  0.6833598907352606


At random state 319 The model perform very well
Random State =  319
Training accuracy score is =  0.6830656380422411
Test accuracy score is =  0.6834357690264815


At random state 322 The model perform very well
Random State =  322
Training accuracy score is =  0.6833944606045277
Test accuracy score is =  0.6833598907352606


At random state 340 The model perform very well
Random State =  340
Training accuracy score is =  0.6837738712533199
Test accuracy score is =  0.6843463085211321


At random state 344 The model perform very well
Random State =  344
Training accuracy score is =  0.6830909320854939
Test accuracy score is =  0.6832840124440398


At random state 348 The model perform very well
Random State =  348
Training accuracy score is =  0.6825850512204376
Test accuracy score is =  0.6829804992791563


At random state 349 The model perform very well
Random State =  349
Training accuracy score is =  0.6830150499557355
Test accuracy score is =  0.6827528644054935


At random state 352 The model perform very well
Random State =  352
Training accuracy score is =  0.6839003414695839
Test accuracy score is =  0.6840427953562486


At random state 355 The model perform very well
Random State =  355
Training accuracy score is =  0.6826356393069433
Test accuracy score is =  0.6832081341528189


At random state 376 The model perform very well
Random State =  376
Training accuracy score is =  0.683343872518022
Test accuracy score is =  0.6832840124440398


At random state 386 The model perform very well
Random State =  386
Training accuracy score is =  0.6835462248640445
Test accuracy score is =  0.6840427953562486


At random state 390 The model perform very well
Random State =  390
Training accuracy score is =  0.6831668142152523
Test accuracy score is =  0.6834357690264815


At random state 415 The model perform very well
Random State =  415
Training accuracy score is =  0.6836726950803086
Test accuracy score is =  0.6841186736474695


At random state 425 The model perform very well
Random State =  425
Training accuracy score is =  0.6841279878588592
Test accuracy score is =  0.6841186736474695


At random state 436 The model perform very well
Random State =  436
Training accuracy score is =  0.6829644618692298
Test accuracy score is =  0.6830563775703771
```

```
At random state 443 The model perform very well
Random State =  443
Training accuracy score is =  0.6840015176425952
Test accuracy score is =  0.6835116473177024


At random state 475 The model perform very well
Random State =  475
Training accuracy score is =  0.6833944606045277
Test accuracy score is =  0.6832081341528189


At random state 477 The model perform very well
Random State =  477
Training accuracy score is =  0.6834450486910333
Test accuracy score is =  0.6834357690264815


At random state 495 The model perform very well
Random State =  495
Training accuracy score is =  0.683217402301758
Test accuracy score is =  0.6832840124440398


At random state 502 The model perform very well
Random State =  502
Training accuracy score is =  0.6841279878588592
Test accuracy score is =  0.6836634039001441


At random state 504 The model perform very well
Random State =  504
Training accuracy score is =  0.6832679903882636
Test accuracy score is =  0.6827528644054935


At random state 518 The model perform very well
Random State =  518
Training accuracy score is =  0.6833185784747692
Test accuracy score is =  0.6832840124440398


At random state 520 The model perform very well
Random State =  520
Training accuracy score is =  0.6832932844315164
Test accuracy score is =  0.6832840124440398


At random state 531 The model perform very well
Random State =  531
Training accuracy score is =  0.6835715189072973
Test accuracy score is =  0.6836634039001441


At random state 537 The model perform very well
Random State =  537
Training accuracy score is =  0.6839256355128367
Test accuracy score is =  0.6839669170650277


At random state 556 The model perform very well
Random State =  556
Training accuracy score is =  0.6837991652965727
Test accuracy score is =  0.6836634039001441


At random state 562 The model perform very well
Random State =  562
Training accuracy score is =  0.683217402301758
Test accuracy score is =  0.6827528644054935


At random state 589 The model perform very well
Random State =  589
Training accuracy score is =  0.6830150499557355
Test accuracy score is =  0.6828287426967145


At random state 592 The model perform very well
Random State =  592
Training accuracy score is =  0.6839256355128367
Test accuracy score is =  0.683815160482586


At random state 616 The model perform very well
Random State =  616
Training accuracy score is =  0.6836474010370558
Test accuracy score is =  0.6836634039001441
```

```
At random state 626 The model perform very well
Random State =  626
Training accuracy score is =  0.6830403439989883
Test accuracy score is =  0.6832840124440398


At random state 633 The model perform very well
Random State =  633
Training accuracy score is =  0.6837738712533199
Test accuracy score is =  0.6839669170650277


At random state 644 The model perform very well
Random State =  644
Training accuracy score is =  0.6831415201719995
Test accuracy score is =  0.6834357690264815


At random state 648 The model perform very well
Random State =  648
Training accuracy score is =  0.6838244593398255
Test accuracy score is =  0.6835875256089233


At random state 658 The model perform very well
Random State =  658
Training accuracy score is =  0.6837738712533199
Test accuracy score is =  0.6838910387738069


At random state 660 The model perform very well
Random State =  660
Training accuracy score is =  0.6834197546477805
Test accuracy score is =  0.6826769861142727


At random state 700 The model perform very well
Random State =  700
Training accuracy score is =  0.6834703427342861
Test accuracy score is =  0.6829804992791563


At random state 707 The model perform very well
Random State =  707
Training accuracy score is =  0.6839256355128367
Test accuracy score is =  0.683739282191365


At random state 734 The model perform very well
Random State =  734
Training accuracy score is =  0.6843303402048817
Test accuracy score is =  0.683815160482586


At random state 741 The model perform very well
Random State =  741
Training accuracy score is =  0.6831668142152523
Test accuracy score is =  0.6828287426967145


At random state 748 The model perform very well
Random State =  748
Training accuracy score is =  0.6839256355128367
Test accuracy score is =  0.6839669170650277


At random state 752 The model perform very well
Random State =  752
Training accuracy score is =  0.6843050461616289
Test accuracy score is =  0.6839669170650277


At random state 801 The model perform very well
Random State =  801
Training accuracy score is =  0.6835462248640445
Test accuracy score is =  0.684422186812353


At random state 835 The model perform very well
Random State =  835
Training accuracy score is =  0.6839509295560895
Test accuracy score is =  0.6842704302299112


At random state 840 The model perform very well
Random State =  840
Training accuracy score is =  0.6837738712533199
Test accuracy score is =  0.6836634039001441
```

```
At random state 853 The model perform very well
Random State =  853
Training accuracy score is =  0.6833691665612748
Test accuracy score is =  0.6825252295318309


At random state 854 The model perform very well
Random State =  854
Training accuracy score is =  0.684153281902112
Test accuracy score is =  0.683815160482586


At random state 859 The model perform very well
Random State =  859
Training accuracy score is =  0.6833185784747692
Test accuracy score is =  0.6825252295318309


At random state 865 The model perform very well
Random State =  865
Training accuracy score is =  0.683217402301758
Test accuracy score is =  0.6826011078230518


At random state 897 The model perform very well
Random State =  897
Training accuracy score is =  0.6832679903882636
Test accuracy score is =  0.6826769861142727


At random state 908 The model perform very well
Random State =  908
Training accuracy score is =  0.6836726950803086
Test accuracy score is =  0.683739282191365


At random state 918 The model perform very well
Random State =  918
Training accuracy score is =  0.683217402301758
Test accuracy score is =  0.6832840124440398


At random state 933 The model perform very well
Random State =  933
Training accuracy score is =  0.6820285822688756
Test accuracy score is =  0.6815388117459594


At random state 941 The model perform very well
Random State =  941
Training accuracy score is =  0.6835209308207917
Test accuracy score is =  0.683739282191365


At random state 959 The model perform very well
Random State =  959
Training accuracy score is =  0.6832679903882636
Test accuracy score is =  0.6829046209879354


At random state 960 The model perform very well
Random State =  960
Training accuracy score is =  0.6837991652965727
Test accuracy score is =  0.6842704302299112


At random state 975 The model perform very well
Random State =  975
Training accuracy score is =  0.6841026938156064
Test accuracy score is =  0.6841186736474695


At random state 978 The model perform very well
Random State =  978
Training accuracy score is =  0.6830403439989883
Test accuracy score is =  0.6832081341528189


At random state 980 The model perform very well
Random State =  980
Training accuracy score is =  0.6840773997723536
Test accuracy score is =  0.684422186812353


At random state 988 The model perform very well
Random State =  988
Training accuracy score is =  0.6835715189072973
```

```
Test accuracy score is =  0.683815160482586


At random state 996 The model perform very well
Random State =  996
Training accuracy score is =  0.6843809282913874
Test accuracy score is =  0.6835116473177024
```

As we can see, random state 5 has the highest accuracy for train and test, coming in at 67.45.

In [114…
```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=5,test_size=0.25,stratify=y)
```

In [115…
```
y_train.value_counts()
```

Out[115]:
```
no     19768
yes    19767
Name: subscribed, dtype: int64
```

In [116…
```
y_test.value_counts()
```

Out[116]:
```
yes    6590
no     6589
Name: subscribed, dtype: int64
```

Now we can easily see that our data is roughly divided into no and yes groups.

In [117…
```
#Logistic Regression
lr.fit(x_train,y_train)
pred_lr=lr.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_lr)*100,'%',)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_lr))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_lr))
```

```
Accuracy of model = 69.42104863798467 %


Confusion matrix
 [[4212 2377]
 [1653 4937]]


Classification Report
               precision    recall  f1-score   support

          no       0.72      0.64      0.68      6589
         yes       0.68      0.75      0.71      6590

    accuracy                           0.69     13179
   macro avg       0.70      0.69      0.69     13179
weighted avg       0.70      0.69      0.69     13179
```

The accuracy score for Logistic Regression is now 66.93%, and confusion matrix predicts that 8907 out of 13206 predictions are accurate.

Check the cross val score now.

In [118…
```
from sklearn.model_selection import cross_val_score
pred_lr_ac=accuracy_score(y_test,pred_lr)
for i in range(2,15):
    lsscore=cross_val_score(lr,x,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_lr_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 68.32340554691353
Accuracy score is = 69.42104863798467


At cv = 3
Cross validation score is = 68.34048278190035
Accuracy score is = 69.42104863798467


At cv = 4
Cross validation score is = 68.35376847765657
Accuracy score is = 69.42104863798467


At cv = 5
Cross validation score is = 68.36324822613945
Accuracy score is = 69.42104863798467


At cv = 6
Cross validation score is = 68.35376821656605
Accuracy score is = 69.42104863798467


At cv = 7
Cross validation score is = 68.37084488899748
Accuracy score is = 69.42104863798467


At cv = 8
Cross validation score is = 68.34997360755072
Accuracy score is = 69.42104863798467


At cv = 9
Cross validation score is = 68.3613609016227
Accuracy score is = 69.42104863798467


At cv = 10
Cross validation score is = 68.32531389004284
Accuracy score is = 69.42104863798467


At cv = 11
Cross validation score is = 68.36896585255784
Accuracy score is = 69.42104863798467


At cv = 12
Cross validation score is = 68.36516127039333
Accuracy score is = 69.42104863798467


At cv = 13
Cross validation score is = 68.35374675589118
Accuracy score is = 69.42104863798467


At cv = 14
Cross validation score is = 68.3594665668812
Accuracy score is = 69.42104863798467
```

We pick cv=2 because it produces the least difference between the Cross Validation score and the Accuracy score.

In [119…
```
lsscore_selected=cross_val_score(lr,x,y,cv=2)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','Accuracy score =',pred_lr_ac*100)
```

```
Cross validation score = 68.32340554691353
 Accuracy score = 69.42104863798467
```

In [120…
```
from sklearn.ensemble import GradientBoostingClassifier,AdaBoostClassifier,BaggingClassifier,RandomForestClassi
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

Now, we apply hyperparameter adjustment to the accurate model we obtain using all of these models for the same random state.

In [121…
```
gb=GradientBoostingClassifier()
ada=AdaBoostClassifier()
```

```
bg=BaggingClassifier()
rfc=RandomForestClassifier()
ext=ExtraTreesClassifier()
gnb=GaussianNB()
svc=SVC()
dtc=DecisionTreeClassifier()
knn=KNeighborsClassifier()
```

Check each accuracy score individually now, along with the matrices and the cross-validation score.

In [122…
```
#GradientBoostingClassifier
gb.fit(x_train,y_train)
pred_gb=gb.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_gb)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_gb))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_gb))
```

```
Accuracy of model = 89.11146520980347


Confusion matrix
 [[6173  416]
 [1019 5571]]


Classification Report
              precision    recall  f1-score   support

          no       0.86      0.94      0.90      6589
         yes       0.93      0.85      0.89      6590

    accuracy                           0.89     13179
   macro avg       0.89      0.89      0.89     13179
weighted avg       0.89      0.89      0.89     13179
```

The accuracy score for GradientBoostingClassifier is now 89.20%, and confusion matrix predicts that 11780 out of 13206 predictions are valid.

In [123…
```
pred_gb_ac=accuracy_score(y_test,pred_gb)
for i in range(2,15):
    gsscore=cross_val_score(gb,x,y,cv=i)
    gsc=gsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',gsc*100)
    print('Accuracy score is =',pred_gb_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 88.46985620518268
Accuracy score is = 89.11146520980347


At cv = 3
Cross validation score is = 88.72606375342308
Accuracy score is = 89.11146520980347


At cv = 4
Cross validation score is = 88.6293072662095
Accuracy score is = 89.11146520980347


At cv = 5
Cross validation score is = 88.60648767738043
Accuracy score is = 89.11146520980347


At cv = 6
Cross validation score is = 88.65016816952756
Accuracy score is = 89.11146520980347


At cv = 7
Cross validation score is = 88.63885114989579
Accuracy score is = 89.11146520980347


At cv = 8
Cross validation score is = 88.62765050086927
Accuracy score is = 89.11146520980347


At cv = 9
Cross validation score is = 88.7017183699848
Accuracy score is = 89.11146520980347


At cv = 10
Cross validation score is = 88.70920034005174
Accuracy score is = 89.11146520980347


At cv = 11
Cross validation score is = 88.84973903834882
Accuracy score is = 89.11146520980347


At cv = 12
Cross validation score is = 88.83608333398985
Accuracy score is = 89.11146520980347


At cv = 13
Cross validation score is = 88.92329348165693
Accuracy score is = 89.11146520980347


At cv = 14
Cross validation score is = 88.90858234613326
Accuracy score is = 89.11146520980347
```

As can be seen, the cross val score is 88.87% at cv=13.

In [124…
```
gsscore_selected=cross_val_score(gb,x,y,cv=12)
print('Cross validation score =',gsscore_selected.mean()*100,'\n','Accuracy score =',pred_gb_ac*100)
```

```
Cross validation score = 88.83608333398985
 Accuracy score = 89.11146520980347
```

In [125…
```
# AdaBoostClassifier
ada.fit(x_train,y_train)
pred_ada=ada.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_ada)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_ada))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_ada))
```

```
Accuracy of model = 83.95174140678353


Confusion matrix
 [[5803  786]
 [1329 5261]]


Classification Report
              precision    recall  f1-score   support

          no       0.81      0.88      0.85      6589
         yes       0.87      0.80      0.83      6590

    accuracy                           0.84     13179
   macro avg       0.84      0.84      0.84     13179
weighted avg       0.84      0.84      0.84     13179
```

The accuracy score for AdaBoostClassifier is currently 82.90%, and confusion matrix predicts that out of 13206 predictions, 10948 are right.

In [126…

```
pred_ada_ac=accuracy_score(y_test,pred_ada)
for i in range(2,15):
    adasscore=cross_val_score(ada,x,y,cv=i)
    adasc=adasscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',adasc*100)
    print('Accuracy score is =',pred_ada_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 84.29259779185796
Accuracy score is = 83.95174140678353


At cv = 3
Cross validation score is = 83.5129753281462
Accuracy score is = 83.95174140678353


At cv = 4
Cross validation score is = 83.77475103162635
Accuracy score is = 83.95174140678353


At cv = 5
Cross validation score is = 83.79561036622064
Accuracy score is = 83.95174140678353


At cv = 6
Cross validation score is = 83.60026102650416
Accuracy score is = 83.95174140678353


At cv = 7
Cross validation score is = 83.62495764990949
Accuracy score is = 83.95174140678353


At cv = 8
Cross validation score is = 83.41827731232746
Accuracy score is = 83.95174140678353


At cv = 9
Cross validation score is = 83.8015583746502
Accuracy score is = 83.95174140678353


At cv = 10
Cross validation score is = 83.51122750849338
Accuracy score is = 83.95174140678353


At cv = 11
Cross validation score is = 83.97801355856231
Accuracy score is = 83.95174140678353


At cv = 12
Cross validation score is = 83.70649402765979
Accuracy score is = 83.95174140678353


At cv = 13
Cross validation score is = 83.59641613331473
Accuracy score is = 83.95174140678353


At cv = 14
Cross validation score is = 83.58726135339279
Accuracy score is = 83.95174140678353
```

With cv=8, we see the least amount of variation between the Cross val score and the Accuracy score.

In [127…
```
lsscore_selected=cross_val_score(ada,x,y,cv=8)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','Accuracy score =',pred_ada_ac*100)
```

```
Cross validation score = 83.41827731232746
 Accuracy score = 83.95174140678353
```

In [128…
```
# BaggingClassifier()
bg.fit(x_train,y_train)
pred_bg=bg.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_bg)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_bg))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_bg))
```

```
Accuracy of model = 93.10266332802185


Confusion matrix
 [[6260  329]
 [ 580 6010]]


Classification Report
              precision    recall  f1-score   support

          no       0.92      0.95      0.93      6589
         yes       0.95      0.91      0.93      6590

    accuracy                           0.93     13179
   macro avg       0.93      0.93      0.93     13179
weighted avg       0.93      0.93      0.93     13179
```

The accuracy score for BaggingClassifier is currently 92.97%, and confusion matrix predicts that 12278 out of 13206 predictions are right.

In [129…
```
pred_bg_ac=accuracy_score(y_test,pred_bg)
for i in range(2,10):
    bgsscore=cross_val_score(bg,x,y,cv=i)
    bgsc=bgsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',bgsc*100)
    print('Accuracy score is =',pred_bg_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 91.53545547672344
Accuracy score is = 93.10266332802185


At cv = 3
Cross validation score is = 92.31904367321854
Accuracy score is = 93.10266332802185


At cv = 4
Cross validation score is = 92.55237542524058
Accuracy score is = 93.10266332802185


At cv = 5
Cross validation score is = 92.69270186454716
Accuracy score is = 93.10266332802185


At cv = 6
Cross validation score is = 92.81037989112134
Accuracy score is = 93.10266332802185


At cv = 7
Cross validation score is = 93.01341390386881
Accuracy score is = 93.10266332802185


At cv = 8
Cross validation score is = 92.89788459682771
Accuracy score is = 93.10266332802185


At cv = 9
Cross validation score is = 92.92643052109959
Accuracy score is = 93.10266332802185
```

Cross validation score and accuracy score differ the least for cv=8.

In [130…
```
lsscore_selected=cross_val_score(bg,x,y,cv=8)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','Accuracy score =',pred_bg_ac*100)
```

```
Cross validation score = 92.93012956021106
 Accuracy score = 93.10266332802185
```

In [131…
```
# RandomForestClassifier
rfc.fit(x_train,y_train)
pred_rfc=rfc.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_rfc)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_rfc))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_rfc))
```

```
Accuracy of model = 94.2256620380909


Confusion matrix
 [[6282  307]
 [ 454 6136]]


Classification Report
              precision    recall  f1-score   support

          no       0.93      0.95      0.94      6589
         yes       0.95      0.93      0.94      6590

    accuracy                           0.94     13179
   macro avg       0.94      0.94      0.94     13179
weighted avg       0.94      0.94      0.94     13179
```

The accuracy score for RandomForestClassifier is now 94.23%, and confusion matrix predicts that 12453 out of 13206 predictions are right.

In [132…
```
pred_rfc_ac=accuracy_score(y_test,pred_rfc)
for i in range(2,10):
    gsscore=cross_val_score(rfc,x,y,cv=i)
    gsc=gsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',gsc*100)
    print('Accuracy score is =',pred_rfc_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 93.22570854042569
Accuracy score is = 94.2256620380909


At cv = 3
Cross validation score is = 93.81387602302524
Accuracy score is = 94.2256620380909


At cv = 4
Cross validation score is = 94.12688609922257
Accuracy score is = 94.2256620380909


At cv = 5
Cross validation score is = 94.20651598652296
Accuracy score is = 94.2256620380909


At cv = 6
Cross validation score is = 94.30141681655543
Accuracy score is = 94.2256620380909


At cv = 7
Cross validation score is = 94.40769740936224
Accuracy score is = 94.2256620380909


At cv = 8
Cross validation score is = 94.4363073163508
Accuracy score is = 94.2256620380909


At cv = 9
Cross validation score is = 94.42879187508395
Accuracy score is = 94.2256620380909
```

As can be seen, with cv=8, there is the least discrepancy between the accuracy and cross-val scores.

In [133…
```
gsscore_selected=cross_val_score(rfc,x,y,cv=8)
print('Cross validation score =',gsscore_selected.mean()*100,'\n','Accuracy score =',pred_rfc_ac*100)
```

```
Cross validation score = 94.37560238001856
 Accuracy score = 94.2256620380909
```

In [134…
```
# ExtraTreesClassifier
ext.fit(x_train,y_train)
pred_ext=ext.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_ext)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_ext))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_ext))
```

```
Accuracy of model = 95.09067455800896


Confusion matrix
 [[6233  356]
 [ 291 6299]]


Classification Report
              precision    recall  f1-score   support

          no       0.96      0.95      0.95      6589
         yes       0.95      0.96      0.95      6590

    accuracy                           0.95     13179
   macro avg       0.95      0.95      0.95     13179
weighted avg       0.95      0.95      0.95     13179
```

The accuracy score for ExtraTreesClassifier is now 94.75%, and confusion matrix predicts that 12513 out of 13206 predictions are accurate.

In [135…
```
pred_ext_ac=accuracy_score(y_test,pred_ext)
for i in range(2,10):
    gsscore=cross_val_score(ext,x,y,cv=i)
    gsc=gsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',gsc*100)
    print('Accuracy score is =',pred_ext_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 94.00349053382403
Accuracy score is = 95.09067455800896


At cv = 3
Cross validation score is = 94.80786641572205
Accuracy score is = 95.09067455800896


At cv = 4
Cross validation score is = 95.10380268852828
Accuracy score is = 95.09067455800896


At cv = 5
Cross validation score is = 95.32763396804151
Accuracy score is = 95.09067455800896


At cv = 6
Cross validation score is = 95.36559840656452
Accuracy score is = 95.09067455800896


At cv = 7
Cross validation score is = 95.49081654551686
Accuracy score is = 95.09067455800896


At cv = 8
Cross validation score is = 95.5401971281054
Accuracy score is = 95.09067455800896


At cv = 9
Cross validation score is = 95.51367322563533
Accuracy score is = 95.09067455800896
```

As we can see, the cross-validation score and accuracy score are least different at cv=4.

In [136…
```
gsscore_selected=cross_val_score(ext,x,y,cv=4)
print('Cross validation score =',gsscore_selected.mean()*100,'\n','Accuracy score =',pred_ext_ac*100)
```

```
Cross validation score = 95.09052470730862
 Accuracy score = 95.09067455800896
```

In [137…
```
# GaussianNB
gnb.fit(x_train,y_train)
pred_gnb=gnb.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_gnb)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_gnb))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_gnb))
```

```
Accuracy of model = 68.47256999772365


Confusion matrix
 [[3642 2947]
 [1208 5382]]


Classification Report
              precision    recall  f1-score   support

          no       0.75      0.55      0.64      6589
         yes       0.65      0.82      0.72      6590

    accuracy                           0.68     13179
   macro avg       0.70      0.68      0.68     13179
weighted avg       0.70      0.68      0.68     13179
```

The accuracy score for GaussianNB is now 67.21%, and confusion matrix predicts that 8877 out of 13206 predictions are accurate.

In [138…
```
pred_gnb_ac=accuracy_score(y_test,pred_gnb)
for i in range(2,10):
    gsscore=cross_val_score(gnb,x,y,cv=i)
    gsc=gsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',gsc*100)
    print('Accuracy score is =',pred_gnb_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 67.57787305080244
Accuracy score is = 68.47256999772365


At cv = 3
Cross validation score is = 67.5645956220182
Accuracy score is = 68.47256999772365


At cv = 4
Cross validation score is = 67.56270591084058
Accuracy score is = 68.47256999772365


At cv = 5
Cross validation score is = 67.5551060618436
Accuracy score is = 68.47256999772365


At cv = 6
Cross validation score is = 67.56459922291475
Accuracy score is = 68.47256999772365


At cv = 7
Cross validation score is = 67.56839982042479
Accuracy score is = 68.47256999772365


At cv = 8
Cross validation score is = 67.57788651292873
Accuracy score is = 68.47256999772365


At cv = 9
Cross validation score is = 67.58358423138388
Accuracy score is = 68.47256999772365
```

As can be seen, with cv=4, there is the least discrepancy between the accuracy and cross-val scores.

In [139…
```
gsscore_selected=cross_val_score(gnb,x,y,cv=5)
print('Cross validation score =',gsscore_selected.mean()*100,'\n','Accuracy score =',pred_gnb_ac*100)
```

```
Cross validation score = 67.5551060618436
 Accuracy score = 68.47256999772365
```

In [140…
```
# SVC
svc.fit(x_train,y_train)
pred_svc=svc.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_svc)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_svc))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_svc))
```

```
Accuracy of model = 76.49290537977085


Confusion matrix
 [[5071 1518]
 [1580 5010]]


Classification Report
              precision    recall  f1-score   support

          no       0.76      0.77      0.77      6589
         yes       0.77      0.76      0.76      6590

    accuracy                           0.76     13179
   macro avg       0.76      0.76      0.76     13179
weighted avg       0.76      0.76      0.76     13179
```

The accuracy score for SVC is now 75.45%, and confusion matrix predicts that 9974 out of 13206 predictions are accurate.

In [141…
```
pred_svc_ac=accuracy_score(y_test,pred_svc)
for i in range(2,5):
    gsscore=cross_val_score(svc,x,y,cv=i)
    gsc=gsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',gsc*100)
    print('Accuracy score is =',pred_svc_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 74.67655651250142
Accuracy score is = 76.49290537977085


At cv = 3
Cross validation score is = 75.21152439366148
Accuracy score is = 76.49290537977085


At cv = 4
Cross validation score is = 75.36708469743535
Accuracy score is = 76.49290537977085
```

As we can see, with cv=4, there is the least gap between the cross-val score and accuracy score.

In [142…
```
gsscore_selected=cross_val_score(svc,x,y,cv=4)
print('Cross validation score =',gsscore_selected.mean()*100,'\n','Accuracy score =',pred_svc_ac*100)
```

```
Cross validation score = 75.36708469743535
 Accuracy score = 76.49290537977085
```

In [143…
```
# DecisionTreeClassifier
dtc.fit(x_train,y_train)
pred_dtc=dtc.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_dtc)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_dtc))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_dtc))
```

```
Accuracy of model = 90.31034221109341


Confusion matrix
 [[5930  659]
 [ 618 5972]]


Classification Report
              precision    recall  f1-score   support

          no       0.91      0.90      0.90      6589
         yes       0.90      0.91      0.90      6590

    accuracy                           0.90     13179
   macro avg       0.90      0.90      0.90     13179
weighted avg       0.90      0.90      0.90     13179
```

Currently, we can see that the DecisionTreeClassifier's accuracy score is 90.21% and that, according to the confusion matrix, 11914 out of 13206 predictions are accurate.

In [144…
```
pred_dtc_ac=accuracy_score(y_test,pred_dtc)
for i in range(2,10):
    gsscore=cross_val_score(dtc,x,y,cv=i)
```

```
    gsc=gsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',gsc*100)
    print('Accuracy score is =',pred_dtc_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 89.2115946427894
Accuracy score is = 90.31034221109341


At cv = 3
Cross validation score is = 89.91357779599186
Accuracy score is = 90.31034221109341


At cv = 4
Cross validation score is = 90.19815081990184
Accuracy score is = 90.31034221109341


At cv = 5
Cross validation score is = 90.28536864497583
Accuracy score is = 90.31034221109341


At cv = 6
Cross validation score is = 90.41061902218233
Accuracy score is = 90.31034221109341


At cv = 7
Cross validation score is = 90.5776200923113
Accuracy score is = 90.31034221109341


At cv = 8
Cross validation score is = 90.42032767860906
Accuracy score is = 90.31034221109341


At cv = 9
Cross validation score is = 90.55890540747598
Accuracy score is = 90.31034221109341
```

As we can see, at cv=9, there is the least discrepancy between the accuracy and cross-val scores.

In [145…
```
gsscore_selected=cross_val_score(dtc,x,y,cv=9)
print('Cross validation score =',gsscore_selected.mean()*100,'\n','Accuracy score =',pred_dtc_ac*100)
```

```
Cross validation score = 90.44697499605188
 Accuracy score = 90.31034221109341
```

In [146…
```
# KNeighborsClassifier
knn.fit(x_train,y_train)
pred_knn=knn.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_knn)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_knn))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_knn))
```

```
Accuracy of model = 86.79717732756659


Confusion matrix
 [[5106 1483]
 [ 257 6333]]


Classification Report
              precision    recall  f1-score   support

          no       0.95      0.77      0.85      6589
         yes       0.81      0.96      0.88      6590

    accuracy                           0.87     13179
   macro avg       0.88      0.87      0.87     13179
weighted avg       0.88      0.87      0.87     13179
```

The accuracy score for KNeighborsClassifier is now 86.83%, and confusion matrix predicts that 11467 out of 13206 predictions are right.

In [147…
```
pred_knn_ac=accuracy_score(y_test,pred_knn)
for i in range(2,10):
    gsscore=cross_val_score(knn,x,y,cv=i)
    gsc=gsscore.mean()
```

```
    print('At cv =',i)
    print('Cross validation score is =',gsc*100)
    print('Accuracy score is =',pred_knn_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 84.44056607352886
Accuracy score is = 86.79717732756659


At cv = 3
Cross validation score is = 86.35277079419826
Accuracy score is = 86.79717732756659


At cv = 4
Cross validation score is = 86.95603130557036
Accuracy score is = 86.79717732756659


At cv = 5
Cross validation score is = 87.27283123257796
Accuracy score is = 86.79717732756659


At cv = 6
Cross validation score is = 87.51376206338512
Accuracy score is = 86.79717732756659


At cv = 7
Cross validation score is = 87.82108208099369
Accuracy score is = 86.79717732756659


At cv = 8
Cross validation score is = 87.89695734901896
Accuracy score is = 86.79717732756659


At cv = 9
Cross validation score is = 87.9519588727137
Accuracy score is = 86.79717732756659
```

As we can see, with cv=4, there is the least gap between the cross-val score and accuracy score.

In [148…
```
gsscore_selected=cross_val_score(knn,x,y,cv=4)
print('Cross validation score =',gsscore_selected.mean()*100,'\n','Accuracy score =',pred_knn_ac*100)
```

```
Cross validation score = 86.95603130557036
 Accuracy score = 86.79717732756659
```

Reviewing all the models.

We see that ExtraTreesClassifier and RandomForestClassifier have differences of 0.01 and 0.1, respectively, hence we should apply the hyper parameter to these two classifiers.

Hyper Parameter

RandomForestClassifier

In [149…
```
grid_paramrfc={

'n_estimators':[100,200,300],
     'criterion' : ['entropy','gini'],
     'max_depth' :[None,1,2,3],
     'min_samples_split' : [2,3,4,5],
    'max_features' :['sqrt','log2',None],
     'bootstrap' :[True,False],
     'random_state':[None,10,20,30]
 }
```

In [150…
```
#gridsearch cv use
from sklearn.model_selection import GridSearchCV
```

In [151…
```
gd_rfc=GridSearchCV(estimator=rfc,n_jobs=-1,
                    param_grid=grid_paramrfc,
                    scoring='accuracy',
                    cv=8)
```

In [152…
```
#right now The parameter in our model is this.
#gd_rfc.fit(x,y)
```

Checking the best parameter and accuracy now that all of our hyperparameters have been applied to the model.

ExtraTreesClassifier

In [153…
```
# grid_parametc={

#     'n_estimators':[100,200,300,50],
#     'criterion' : ['entropy','gini'],
#     'max_depth' :[None,1,2,10,20],
#     'min_samples_split' : [2,3,4,5],
#     'max_features' :['sqrt','log2',None],
#     'bootstrap' :[True,False],
#     'random_state':[None,10,50,1000,500,30],
#     'n_jobs':[None,-1,1,0,2]
# }
```

In [154…
```
# gd_etc=GridSearchCV(estimator=ext,
#                     param_grid=grid_paramrfc,
#                     scoring='accuracy',
#                     cv=4)
```

In [155…
```
#right now The parameter in our model is this.

#gd_etc.fit(x,y)
```

Checking the best parameter and accuracy now that all of our hyperparameters have been applied to the model.

In [156…
```
#best_parameters=gd_rfc.best_params_
#print(best_parameters)
#best_result=gd_rfc.best_score_
#print(best_result*100)
```

Due to system configurations, it is not feasible for me to plot a hyper parameter in my system. Therefore, I utilised Google Colab and Kaggle to find a hyper parameter, which I then plotted here. I attempted this for almost 13 hours on my computer, but it did not work, so I turned to internet services. Now that we have all of the rfc hyperparameters, we can assess accuracy and Cross val score before selecting the best model.

RandomForestClassifier

In [157…
```
rfch=RandomForestClassifier(n_estimators=200,criterion='gini',max_depth=None,min_samples_split=3,max_features='
```

In [158…
```
# RandomForestClassifier
rfch.fit(x_train,y_train)
pred_rfch=rfch.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_rfch)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_rfch))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_rfch))
```

```
Accuracy of model = 94.38500644965475


Confusion matrix
 [[6293  296]
 [ 444 6146]]


Classification Report
               precision    recall  f1-score   support

           no       0.93      0.96      0.94      6589
          yes       0.95      0.93      0.94      6590

    accuracy                           0.94     13179
   macro avg       0.94      0.94      0.94     13179
weighted avg       0.94      0.94      0.94     13179
```

The accuracy score for RandomForestClassifier is now 94.26%, and confusion matrix predicts that 12448 out of 13206 predictions are right.

In [159…
```
pred_rfch_ac=accuracy_score(y_test,pred_rfch)
for i in range(2,10):
    gsscore=cross_val_score(rfch,x,y,cv=i)
    gsc=gsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',gsc*100)
    print('Accuracy score is =',pred_rfch_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 93.248472891452
Accuracy score is = 94.38500644965475


At cv = 3
Cross validation score is = 93.84612818333798
Accuracy score is = 94.38500644965475


At cv = 4
Cross validation score is = 94.1819016030265
Accuracy score is = 94.38500644965475


At cv = 5
Cross validation score is = 94.28239643693487
Accuracy score is = 94.38500644965475


At cv = 6
Cross validation score is = 94.36212247257164
Accuracy score is = 94.38500644965475


At cv = 7
Cross validation score is = 94.41149679408964
Accuracy score is = 94.38500644965475


At cv = 8
Cross validation score is = 94.40975711116448
Accuracy score is = 94.38500644965475


At cv = 9
Cross validation score is = 94.44776868047882
Accuracy score is = 94.38500644965475
```

As we can see, the cross-validation score and accuracy score diverge the least when cv=7.

In [160…
```
gsscore_selected=cross_val_score(rfch,x,y,cv=7)
print('Cross validation score =',gsscore_selected.mean()*100,'\n','Accuracy score =',pred_rfch_ac*100)
```

```
Cross validation score = 94.43425833815941
 Accuracy score = 94.38500644965475
```

ExtraTreesClassifier

In [161…
```
exth=ExtraTreesClassifier(n_estimators=200,criterion='entropy',max_depth=None,min_samples_split=3,max_features=
```

In [162…
```
# ExtraTreesClassifier
exth.fit(x_train,y_train)
pred_exth=ext.predict(x_test)
print('Accuracy of model =',accuracy_score(y_test,pred_exth)*100)
print('\n')
print('Confusion matrix','\n',confusion_matrix(y_test,pred_exth))
print('\n')
print('Classification Report','\n',classification_report(y_test,pred_exth))
```

```
Accuracy of model = 95.09067455800896


Confusion matrix
 [[6233  356]
 [ 291 6299]]


Classification Report
              precision    recall  f1-score   support

          no       0.96      0.95      0.95      6589
         yes       0.95      0.96      0.95      6590

    accuracy                           0.95     13179
   macro avg       0.95      0.95      0.95     13179
weighted avg       0.95      0.95      0.95     13179
```

The accuracy score for ExtraTreesClassifier is now 94.752%, and confusion matrix predicts that 12513 out of 13206 predictions are right.

In [163…
```
pred_exth_ac=accuracy_score(y_test,pred_exth)
for i in range(2,5):
    gsscore=cross_val_score(exth,x,y,cv=i)
    gsc=gsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',gsc*100)
    print('Accuracy score is =',pred_exth_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 94.03953408961566
Accuracy score is = 95.09067455800896


At cv = 3
Cross validation score is = 94.83253333794191
Accuracy score is = 95.09067455800896


At cv = 4
Cross validation score is = 95.17779150780622
Accuracy score is = 95.09067455800896
```

As can be seen, with cv=3, there is the least discrepancy between the accuracy and cross-val scores.

In [164…
```
gsscore_selected=cross_val_score(exth,x,y,cv=3)
print('Cross validation score =',gsscore_selected.mean()*100,'\n','Accuracy score =',pred_exth_ac*100)
```

```
Cross validation score = 94.86857563423138
 Accuracy score = 95.09067455800896
```

Now, in order to choose the optimal model, we are comparing the differences between the Cros val score and the Accuracy score.

0.05 for RandomForestClassifier and extra Tree Classifier = 0.18

Observe that there is essentially no difference between cross-val-core and accuracy, therefore that is why we chose rfc.

The model is saved

In [166…
```
import joblib
joblib.dump(rfch,'RandomForestClassifierbankmarketing.obj')
```

Out[166]:
```
['RandomForestClassifierbankmarketing.obj']
```

Our model is currently saved on the drive and ready to use.

Currently, we import our test data and evaluate its accuracy score, but first, we must apply the entire method.

In [167…
```
df_te=pd.read_csv('termdeposit_test.csv')
df_te.head(5)
```

Out[167]:

	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous
0	38441	32	services	married	secondary	no	118	yes	no	cellular	15	may	20	6	-1	0
1	40403	78	retired	divorced	primary	no	2787	no	no	telephone	1	jul	372	1	-1	0
2	3709	31	self-employed	single	tertiary	no	144	yes	no	unknown	16	may	676	1	-1	0
3	37422	57	services	single	primary	no	3777	yes	no	telephone	13	may	65	2	-1	0
4	12527	45	blue-collar	divorced	secondary	no	-705	no	yes	unknown	3	jul	111	1	-1	0

In [168…
```
df_te.drop(['ID','previous','pdays','balance','duration'],axis=1,inplace=True)
df_te.head(2)
```

Out[168]:

	age	job	marital	education	default	housing	loan	contact	day	month	campaign	poutcome
0	32	services	married	secondary	no	yes	no	cellular	15	may	6	unknown
1	78	retired	divorced	primary	no	no	no	telephone	1	jul	1	unknown

data encoding now.

In [169…
```
le=LabelEncoder()
df_te['job']=le.fit_transform(df_te['job'])
df_te['marital']=le.fit_transform(df_te['marital'])
df_te['education']=le.fit_transform(df_te['education'])
df_te['default']=le.fit_transform(df_te['default'])
df_te['housing']=le.fit_transform(df_te['housing'])
df_te['loan']=le.fit_transform(df_te['loan'])
df_te['contact']=le.fit_transform(df_te['contact'])
df_te['month']=le.fit_transform(df_te['month'])
df_te['poutcome']=le.fit_transform(df_te['poutcome'])
```

In [170…
```
df_te.head()
```

Out[170]:

	age	job	marital	education	default	housing	loan	contact	day	month	campaign	poutcome
0	32	7	1	1	0	1	0	0	15	8	6	3
1	78	5	0	0	0	0	0	1	1	5	1	3
2	31	6	2	2	0	1	0	2	16	8	1	3
3	57	7	2	0	0	1	0	1	13	8	2	3
4	45	1	0	1	0	0	1	2	3	5	1	3

Eliminating outliers

In [171…
```
from scipy.stats import zscore
z=np.abs(zscore(df_te))
threshold=3

Xte=df_te[(z<3).all(axis=1)]
print(Xte.shape)
```

```
(12940, 12)
```

In [172…
```
Xte.head()
```

Out[172]:

	age	job	marital	education	default	housing	loan	contact	day	month	campaign	poutcome
0	32	7	1	1	0	1	0	0	15	8	6	3
2	31	6	2	2	0	1	0	2	16	8	1	3
3	57	7	2	0	0	1	0	1	13	8	2	3
4	45	1	0	1	0	0	1	2	3	5	1	3
5	32	4	2	2	0	1	0	0	22	5	2	3

In [173…
```
Xte['campaign']=boxcox(Xte['campaign'])[0]
```

Data Scalability with a basic scaller.

In [174…
```
#We employ a conventional scaler for scaling.
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
Xtes=sc.fit_transform(Xte)
Xtes
```

Out[174]:
```
array([[-0.8527007 ,  0.80620379, -0.28174092, ...,  0.8214164 ,
         1.55491544,  0.45857576],
       [-0.95162171,  0.50198481,  1.35826247, ...,  0.8214164 ,
        -1.11431094,  0.45857576],
       [ 1.62032473,  0.80620379,  1.35826247, ...,  0.8214164 ,
         0.22122354,  0.45857576],
       ...,
       [-0.55593764,  0.19776584, -0.28174092, ..., -0.50415923,
        -1.11431094,  0.45857576],
       [-1.14946375, -1.32332902,  1.35826247, ...,  1.1528103 ,
         0.22122354, -0.52647907],
       [-1.14946375, -1.32332902, -0.28174092, ...,  0.8214164 ,
        -1.11431094, -2.49658873]])
```

Our data is now ready to be predicted.

In [175…
```
import joblib
load_bank=joblib.load("RandomForestClassifierbankmarketing.obj")
```

In [176…
```
load_bank
```

Out[176]:
```
RandomForestClassifier(max_features='log2', min_samples_split=3,
                       n_estimators=200)
```

In [177…
```
pre_test=load_bank.predict(Xtes)
```

In [178…
```
dff=pd.DataFrame({
    'ans':pre_test
})
dff
```

Out[178]:

	ans
0	yes
1	yes
2	yes
3	no
4	yes
...	...
12935	yes
12936	no
12937	no
12938	yes
12939	yes

12940 rows × 1 columns

In [179…
```
XX=Xte
result = pd.concat([XX, dff], axis=1)
result.head()
```

Out[179]:

	age	job	marital	education	default	housing	loan	contact	day	month	campaign	poutcome	ans
0	32.0	7.0	1.0	1.0	0.0	1.0	0.0	0.0	15.0	8.0	1.161296	3.0	yes
2	31.0	6.0	2.0	2.0	0.0	1.0	0.0	2.0	16.0	8.0	0.000000	3.0	yes
3	57.0	7.0	2.0	0.0	0.0	1.0	0.0	1.0	13.0	8.0	0.581049	3.0	no
4	45.0	1.0	0.0	1.0	0.0	0.0	1.0	2.0	3.0	5.0	0.000000	3.0	yes
5	32.0	4.0	2.0	2.0	0.0	1.0	0.0	0.0	22.0	5.0	0.581049	3.0	yes

Our data is now prepared for use.

In [4]:
```
#the data being loaded
train = pd.read_csv("termdeposit_train.csv")
test = pd.read_csv("termdeposit_test.csv")
```

checking each dataset's columns

In [6]:
```
train.columns
```

Out[6]:
```
Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome', 'subscribed'],
      dtype='object')
```

In [7]:
```
test.columns
```

Out[7]:
```
Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome'],
      dtype='object')
```

'Subscribed' is the desired variable as a result.

Examining the variable data types

In [8]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31647 entries, 0 to 31646
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   ID          31647 non-null  int64 
 1   age         31647 non-null  int64 
 2   job         31647 non-null  object
 3   marital     31647 non-null  object
 4   education   31647 non-null  object
 5   default     31647 non-null  object
 6   balance     31647 non-null  int64 
 7   housing     31647 non-null  object
 8   loan        31647 non-null  object
 9   contact     31647 non-null  object
 10  day         31647 non-null  int64 
 11  month       31647 non-null  object
 12  duration    31647 non-null  int64 
 13  campaign    31647 non-null  int64 
 14  pdays       31647 non-null  int64 
 15  previous    31647 non-null  int64 
 16  poutcome    31647 non-null  object
 17  subscribed  31647 non-null  object
dtypes: int64(8), object(10)
memory usage: 4.3+ MB
```

Examining each dataset's shapes

In [9]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13564 entries, 0 to 13563
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   ID         13564 non-null  int64 
 1   age        13564 non-null  int64 
 2   job        13564 non-null  object
 3   marital    13564 non-null  object
 4   education  13564 non-null  object
 5   default    13564 non-null  object
 6   balance    13564 non-null  int64 
 7   housing    13564 non-null  object
 8   loan       13564 non-null  object
 9   contact    13564 non-null  object
 10  day        13564 non-null  int64 
 11  month      13564 non-null  object
 12  duration   13564 non-null  int64 
 13  campaign   13564 non-null  int64 
 14  pdays      13564 non-null  int64 
 15  previous   13564 non-null  int64 
 16  poutcome   13564 non-null  object
dtypes: int64(8), object(9)
memory usage: 1.8+ MB
```

In [10]:
```
train.shape
```

Out[10]:
```
(31647, 18)
```

In [11]:
```
test.shape
```

Out[11]:
```
(13564, 17)
```

As a result, we can observe that the dataset and the variable to be predicted, "Subscribed," have a total of 17 attributes in common.

Exploration of Data

In [12]:
```
#printing the train dataset's initial five rows
train.head()
```

Out[12]:

	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previou
0	26110	56	admin.	married	unknown	no	1933	no	no	telephone	19	nov	44	2	-1	
1	40576	31	unknown	married	secondary	no	3	no	no	cellular	20	jul	91	2	-1	
2	15320	27	services	married	secondary	no	891	yes	no	cellular	18	jul	240	1	-1	
3	43962	57	management	divorced	tertiary	no	3287	no	no	cellular	22	jun	867	1	84	
4	29842	31	technician	married	secondary	no	119	yes	no	cellular	4	feb	380	1	-1	

In [13]:
```
#printing the test dataset's initial five rows
test.head()
```

Out[13]:

	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous
0	38441	32	services	married	secondary	no	118	yes	no	cellular	15	may	20	6	-1	0
1	40403	78	retired	divorced	primary	no	2787	no	no	telephone	1	jul	372	1	-1	0
2	3709	31	self-employed	single	tertiary	no	144	yes	no	unknown	16	may	676	1	-1	0
3	37422	57	services	single	primary	no	3777	yes	no	telephone	13	may	65	2	-1	0
4	12527	45	blue-collar	divorced	secondary	no	-705	no	yes	unknown	3	jul	111	1	-1	0

In [14]:
```
#examining the train dataset for missing values
train.isnull().sum()
```

Out[14]:
```
ID            0
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
subscribed    0
dtype: int64
```

In [15]:
```
#Testing the test dataset for missing values
test.isnull().sum()
```

Out[15]:
```
ID           0
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
dtype: int64
```

Using a Single Variable For Analysis

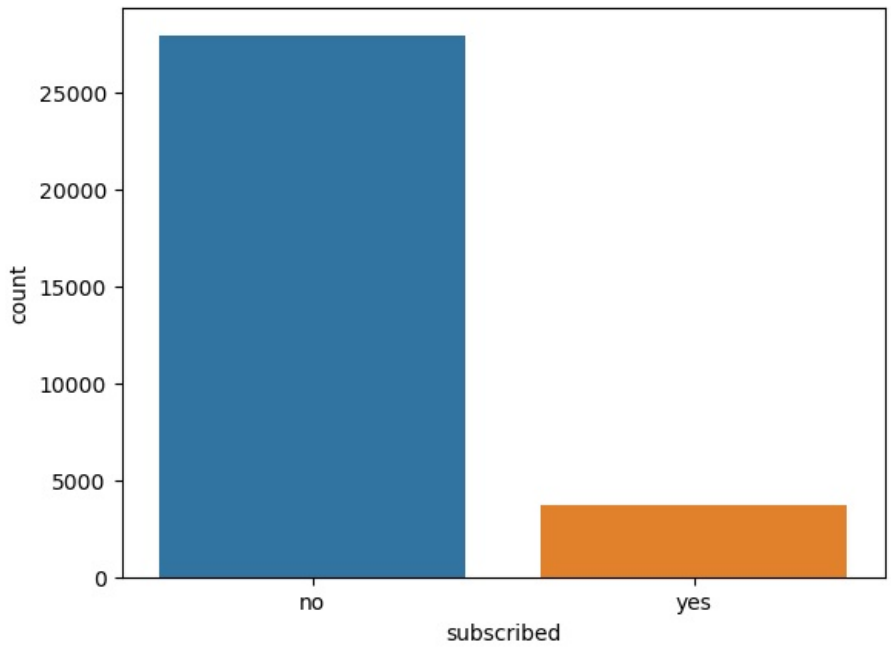
'Subscribed' variable analysis

In [16]:
```
#Frequency of 'subscribed'
train['subscribed'].value_counts()
```

Out[16]:
```
no     27932
yes     3715
Name: subscribed, dtype: int64
```

In [17]:
```
# Plotting the'subscribed' frequency
sns.countplot(data=train, x='subscribed')
```

Out[17]:
```
<AxesSubplot:xlabel='subscribed', ylabel='count'>
```



In [18]:
```
#the 'Subscribed' variable's frequency table has been normalised
train['subscribed'].value_counts(normalize=True)
```

Out[18]:
```
no     0.882611
yes    0.117389
Name: subscribed, dtype: float64
```

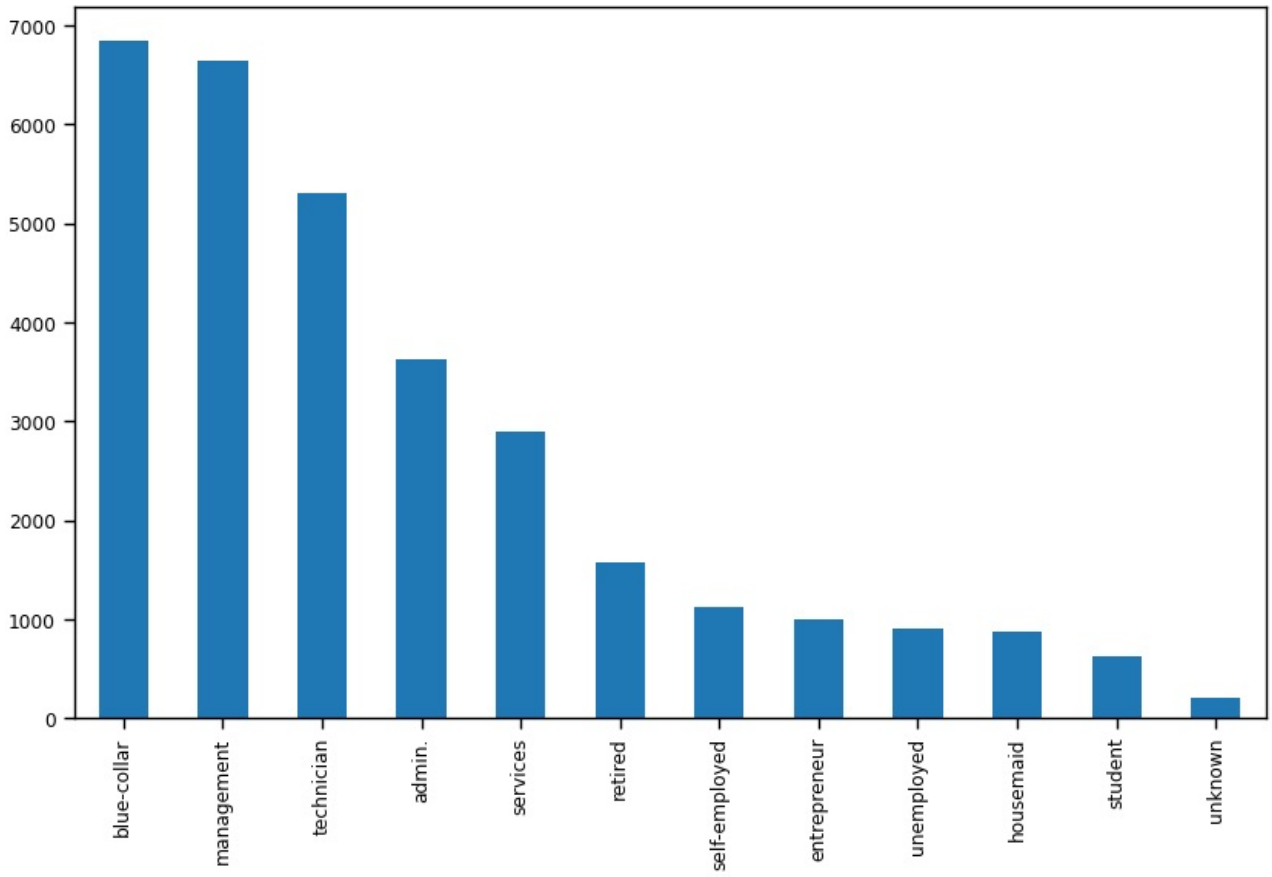
We can observe from the analysis above that just 3,715 out of 31,647 persons have subscribed, or about 12%.

The 'Job' variable is being examined.

In [19]:
```
# table for Frequency
train['job'].value_counts()
```

Out[19]:
```
blue-collar      6842
management       6639
technician       5307
admin.           3631
services         2903
retired          1574
self-employed    1123
entrepreneur     1008
unemployed        905
housemaid         874
student           635
unknown           206
Name: job, dtype: int64
```

In [20]:
```
#the job frequency table being plotted
sns.set_context('paper')
train['job'].value_counts().plot(kind='bar', figsize=(10,6));
```



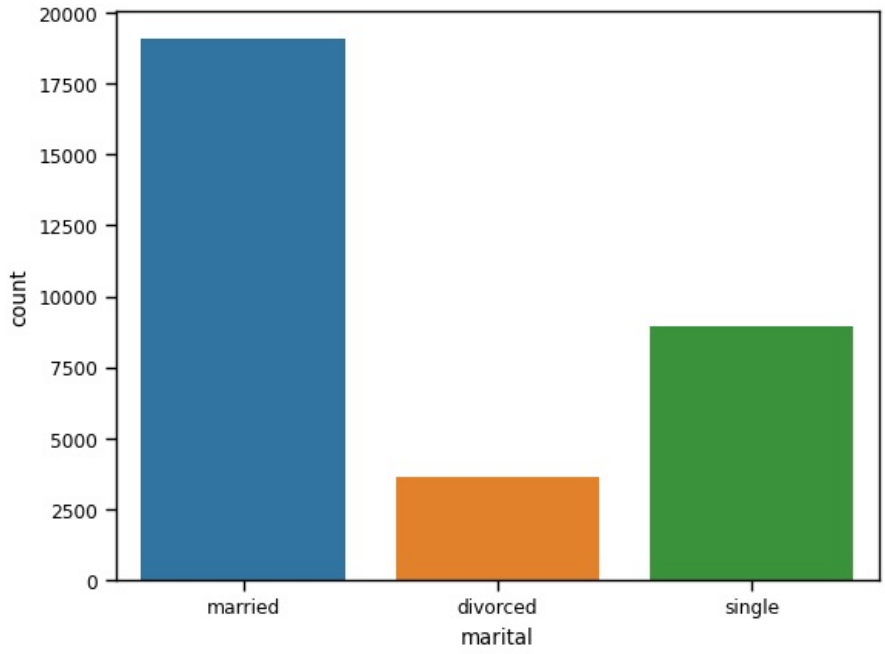
We can see that the majority of the clientele are those with blue-collar jobs, and students are generally the least active, as they rarely make term deposits.

Examining'marital' status

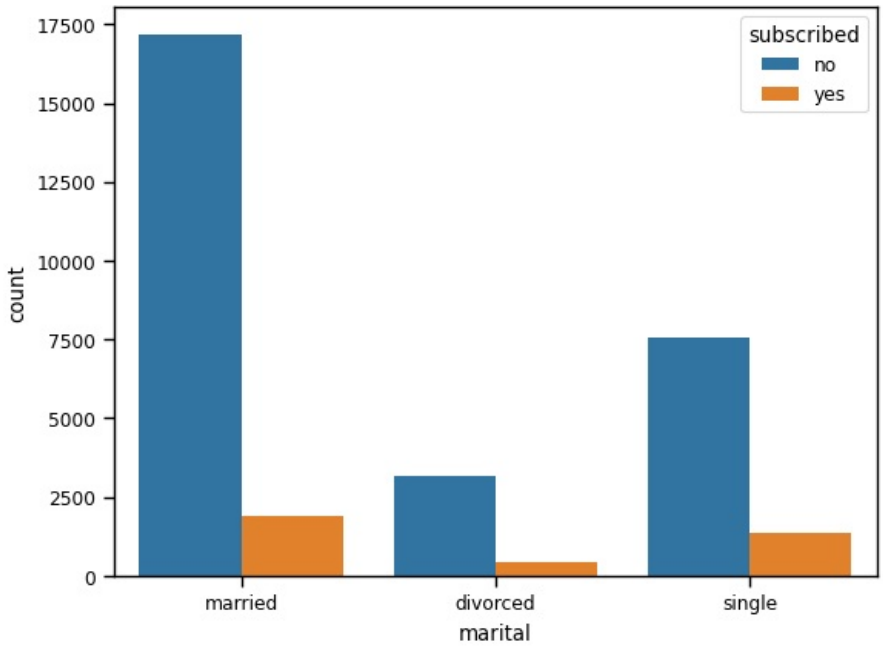
In [21]:
```
train['marital'].value_counts()
```

Out[21]:
```
married     19095
single       8922
divorced     3630
Name: marital, dtype: int64
```

In [22]:
```
sns.countplot(data=train, x='marital');
```

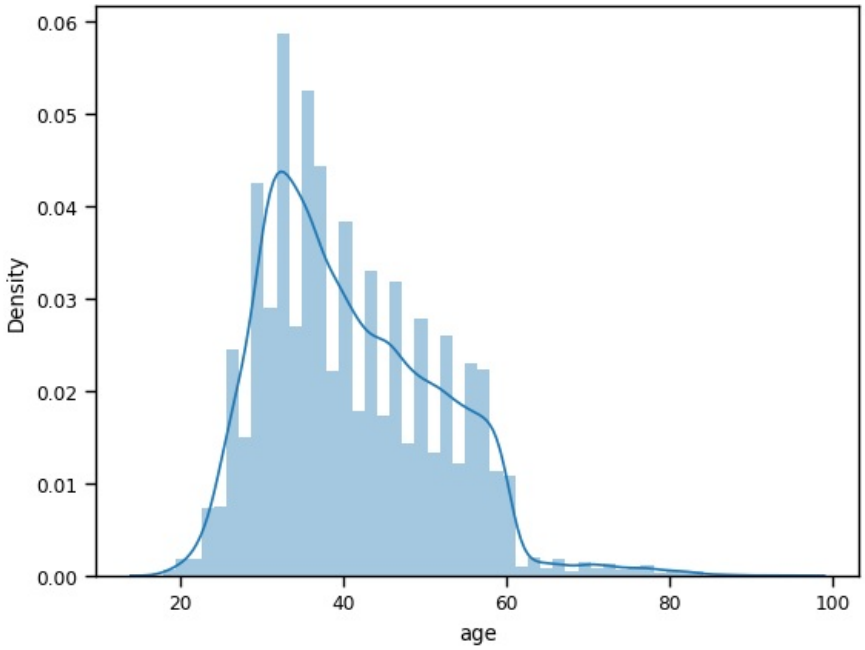


In [23]:
```
sns.countplot(data=train, x='marital', hue='subscribed');
```



Examination of the 'age' variable

In [24]:
```
sns.distplot(train['age']);
```



We can assume that the majority of the clients are between the ages of 20 and 60.

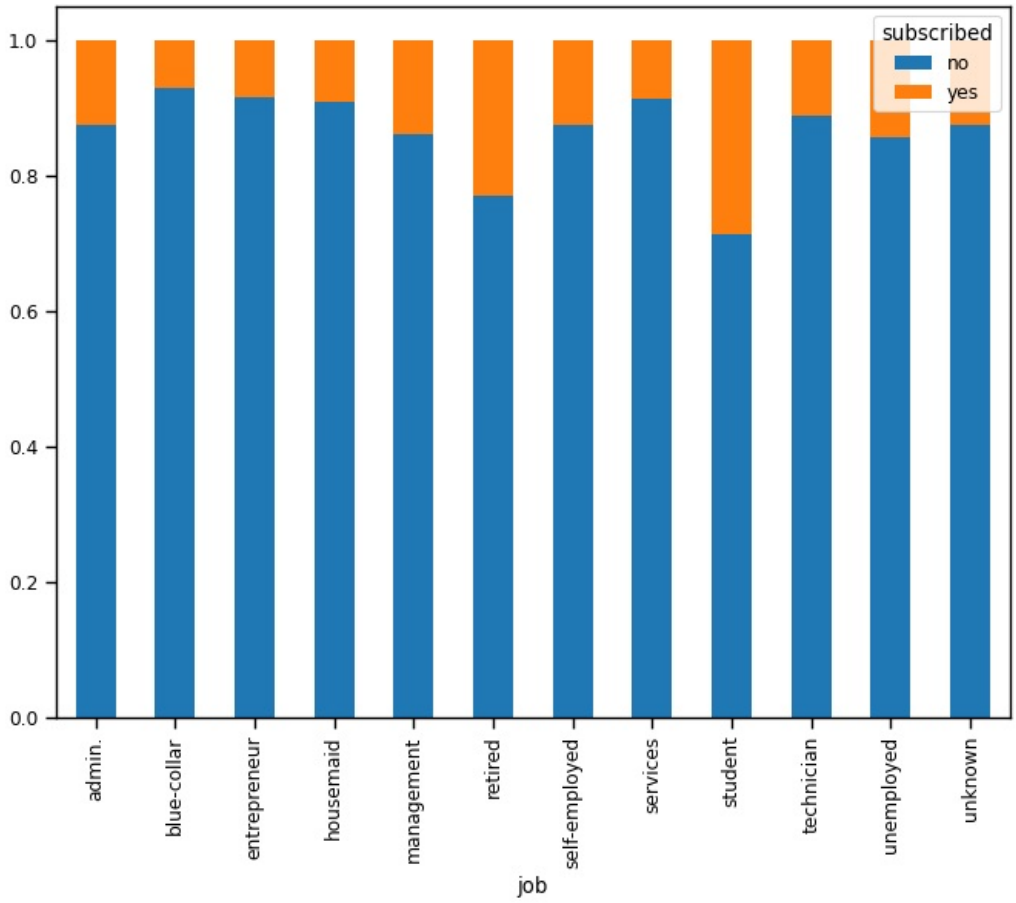
Analysis of BiVariance

In [25]:
```
#job versus subscribed
print(pd.crosstab(train['job'],train['subscribed']))
```

```
subscribed       no  yes
job
admin.         3179  452
blue-collar    6353  489
entrepreneur    923   85
housemaid       795   79
management     5716  923
retired        1212  362
self-employed   983  140
services       2649  254
student         453  182
technician     4713  594
unemployed      776  129
unknown         180   26
```

In [26]:
```
job = pd.crosstab(train['job'],train['subscribed'])
job_norm = job.div(job.sum(1).astype(float), axis=0)
```

In [27]:
```
job_norm.plot.bar(stacked=True,figsize=(8,6));
```



It is unexpected that students are more likely to subscribe to a term deposit than retirees based on the aforementioned graph because students don't often do so. The dataset contains fewer students than other employment categories, which may be the case because more students have term deposit subscriptions than other job kinds.

In [28]:
```
#Getting married versus subscribing
pd.crosstab(train['marital'], train['subscribed'])
```

Out[28]:

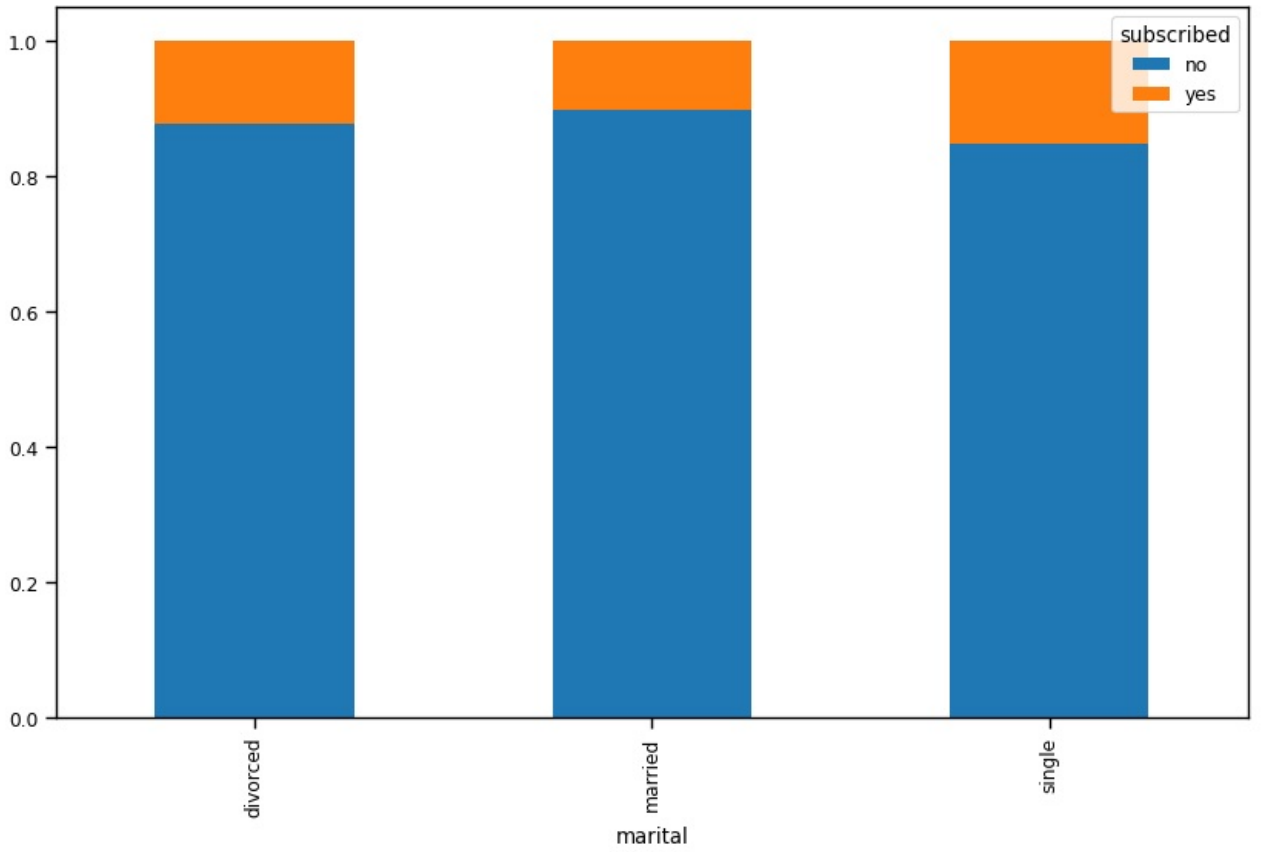
subscribed	no	yes
marital		
divorced	3185	445
married	17176	1919
single	7571	1351

In [29]:
```
marital = pd.crosstab(train['marital'], train['subscribed'])
marital_norm = marital.div(marital.sum(1).astype(float), axis=0)
marital_norm
```

Out[29]:

subscribed	no	yes
marital		
divorced	0.877410	0.122590
married	0.899502	0.100498
single	0.848577	0.151423

In [30]:
```
marital_norm.plot.bar(stacked=True, figsize=(10,6));
```



The results of the analysis above suggest that the subscription to term deposits is not significantly influenced by marital status.

In [31]:
```
#default versus subscription
pd.crosstab(train['default'], train['subscribed'])
```

Out[31]:

subscribed	no	yes
default		
no	27388	3674
yes	544	41

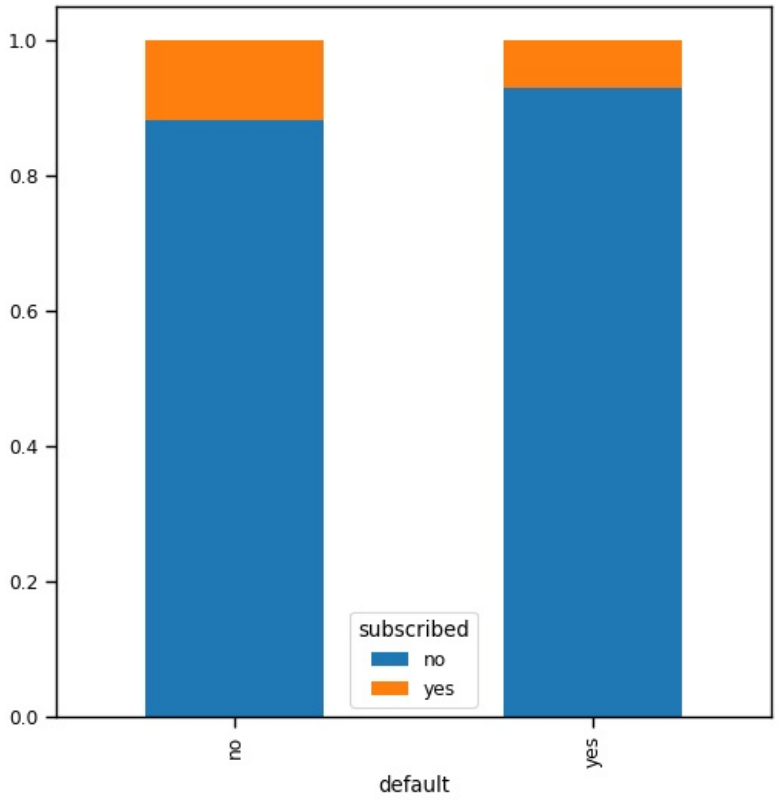
In [32]:
```
dflt = pd.crosstab(train['default'], train['subscribed'])
dflt_norm = dflt.div(dflt.sum(1).astype(float), axis=0)
dflt_norm
```

Out[32]:

subscribed	no	yes
default		
no	0.881720	0.118280
yes	0.929915	0.070085

In [33]:
```
dflt_norm.plot.bar(stacked=True, figsize=(6,6))
```

Out[33]:
```
<AxesSubplot:xlabel='default'>
```



We may conclude that clients without a history of default have a little bit better chances of signing up for a term loan than clients with a history of default.

In [34]:
```
# 0s and 1s are created by converting the target variables.
train['subscribed'].replace('no', 0,inplace=True)
train['subscribed'].replace('yes', 1,inplace=True)
```

In [35]:
```
train['subscribed']
```

Out[35]:
```
0        0
1        0
2        0
3        1
4        0
       ..
31642    0
31643    1
31644    0
31645    0
31646    1
Name: subscribed, Length: 31647, dtype: int64
```

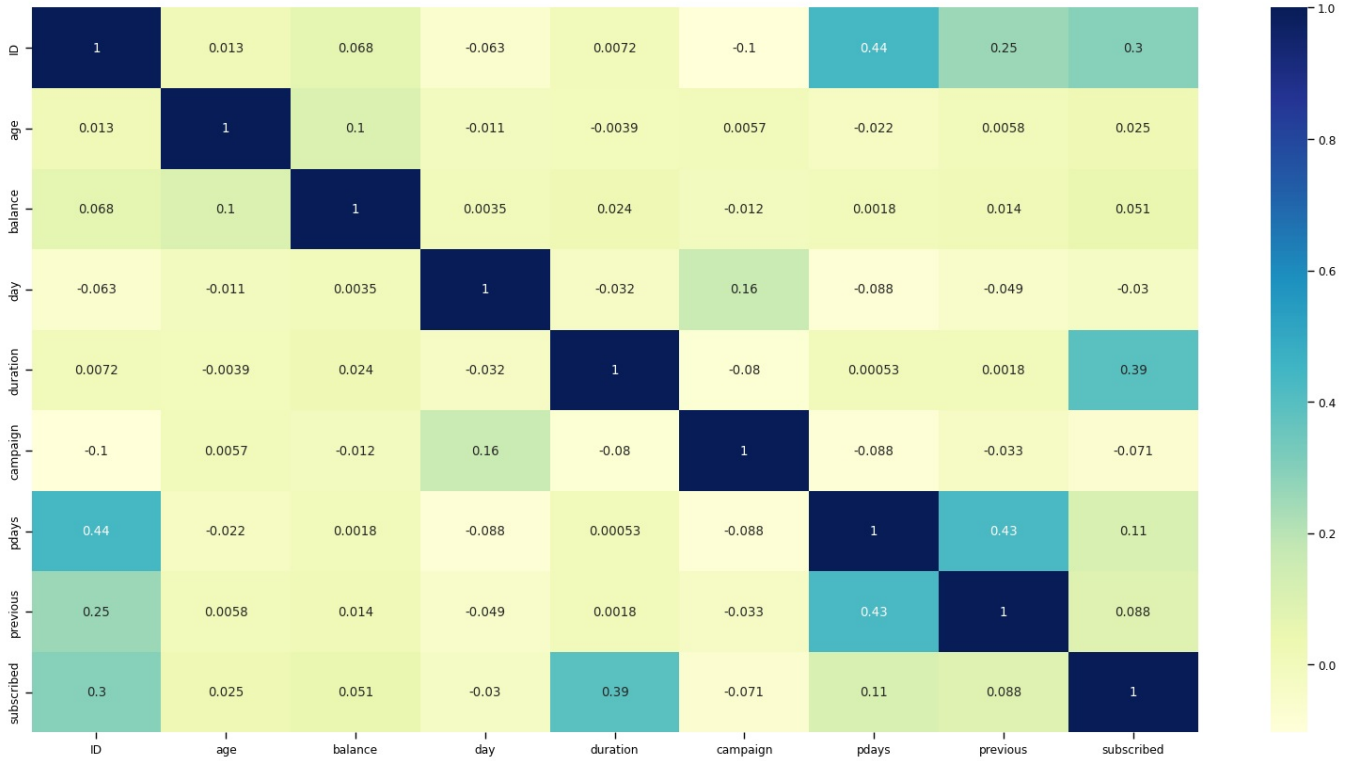
In [36]:
```
#Relationship matrix
tc = train.corr()
tc
```

Out[36]:

	ID	age	balance	day	duration	campaign	pdays	previous	subscribed
ID	1.000000	0.013337	0.067897	-0.063399	0.007183	-0.103508	0.436148	0.254476	0.296663
age	0.013337	1.000000	0.103245	-0.011056	-0.003870	0.005733	-0.021947	0.005761	0.024538
balance	0.067897	0.103245	1.000000	0.003461	0.024274	-0.012032	0.001789	0.013843	0.050807
day	-0.063399	-0.011056	0.003461	1.000000	-0.032288	0.159168	-0.087626	-0.048752	-0.029600
duration	0.007183	-0.003870	0.024274	-0.032288	1.000000	-0.080305	0.000529	0.001783	0.389838
campaign	-0.103508	0.005733	-0.012032	0.159168	-0.080305	1.000000	-0.087570	-0.033151	-0.070607
pdays	0.436148	-0.021947	0.001789	-0.087626	0.000529	-0.087570	1.000000	0.428938	0.108290
previous	0.254476	0.005761	0.013843	-0.048752	0.001783	-0.033151	0.428938	1.000000	0.088081
subscribed	0.296663	0.024538	0.050807	-0.029600	0.389838	-0.070607	0.108290	0.088081	1.000000

In [37]:
```
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sns.heatmap(tc, annot=True, cmap='YlGnBu')
```

Out[37]:
```
<AxesSubplot:>
```



We can conclude that the target variable and the call's duration have a strong correlation. The likelihood that the client will subscribe to a term deposit increases as the call length increases since there is a higher likelihood that the client will express interest in a term deposit.

Building a model

In [38]:
```
target = train['subscribed']
train = train.drop('subscribed', axis=1)
```

In [39]:
```
#on the train dataset, creating dummy values
train = pd.get_dummies(train)
train.head()
```

Out[39]:

	ID	age	balance	day	duration	campaign	pdays	previous	job_admin.	job_blue-collar	...	month_jun	month_mar	month_may	month_nc
0	26110	56	1933	19	44	2	-1	0	1	0	...	0	0	0	
1	40576	31	3	20	91	2	-1	0	0	0	...	0	0	0	
2	15320	27	891	18	240	1	-1	0	0	0	...	0	0	0	
3	43962	57	3287	22	867	1	84	3	0	0	...	1	0	0	
4	29842	31	119	4	380	1	-1	0	0	0	...	0	0	0	

5 rows × 52 columns

Dividing the data into a train and validation set will allow us to test our model's predictions using the validation set. 20% of the dataset will

serve as our validation set, while the remaining 80% will serve as our training set.

In [40]:
```
from sklearn.model_selection import train_test_split
```

In [41]:
```
X_train, X_val, y_train, y_val = train_test_split(train, target, test_size=0.2, random_state=12)
```

The time has come to develop our model and evaluate its performance because our data is now ready. I'll use a logistic regression model for this issue because it is a classification challenge.

Logistic Regression

In [42]:
```
from sklearn.linear_model import LogisticRegression
```

In [43]:
```
#making a logistic regression model object
lreg = LogisticRegression()
```

In [44]:
```
#data fitting into the model
lreg.fit(X_train,y_train)
```

Out[44]:
```
LogisticRegression()
```

In [45]:
```
#Using the validation set to make predictions
pred = lreg.predict(X_val)
```

Evaluating the model's accuracy

In [49]:
```
from sklearn.metrics import accuracy_score
```

In [50]:
```
#Accuracy score calculation
accuracy_score(y_val,pred)
```

Out[50]:
```
0.8911532385466034
```

Our accuracy rating on the validation dataset was about 89%. The decision boundary for logistic regression is linear. A model that can account for this nonlinearity is necessary.

Utilising the Decision Tree technique to handle non-linearity

Decision Tree

In [51]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [52]:
```
#making a decision tree object
clf = DecisionTreeClassifier(max_depth=4, random_state=0)
```

In [53]:
```
#making the model fit
clf.fit(X_train, y_train)
```

Out[53]:
```
DecisionTreeClassifier(max_depth=4, random_state=0)
```

In [54]:
```
#using the validation set to make predictions
predict = clf.predict(X_val)
predict
```

Out[54]:
```
array([0, 0, 0, ..., 1, 0, 0], dtype=int64)
```

In [55]:
```
#determining the precision
accuracy_score(y_val,predict)
```

Out[55]:
```
0.9042654028436019
```

On the validation set, we achieved an accuracy of greater than 90%.

Now let's predict using the test dataset.

In [56]:
```
test = pd.get_dummies(test)
test.head()
```

Out[56]:

	ID	age	balance	day	duration	campaign	pdays	previous	job_admin.	job_blue-collar	...	month_jun	month_mar	month_may	month_nc
0	38441	32	118	15	20	6	-1	0	0	0	...	0	0	1	
1	40403	78	2787	1	372	1	-1	0	0	0	...	0	0	0	
2	3709	31	144	16	676	1	-1	0	0	0	...	0	0	1	
3	37422	57	3777	13	65	2	-1	0	0	0	...	0	0	1	
4	12527	45	-705	3	111	1	-1	0	0	1	...	0	0	0	

5 rows × 52 columns

In [57]:
```
test_pred = clf.predict(test)
test_pred
```

Out[57]:
```
array([0, 1, 0, ..., 0, 1, 0], dtype=int64)
```

Finally, we will create a csv file and save these forecasts.

In [58]:
```
submissions = pd.DataFrame()
```

In [59]:
```
submissions['ID'] = test['ID']
submissions['subscribed'] = test_pred
```

In [60]:
```
submissions['subscribed']
```

Out[60]:
```
0        0
1        1
2        0
3        0
4        0
        ..
13559    0
13560    0
13561    0
13562    1
13563    0
Name: subscribed, Length: 13564, dtype: int64
```

We will translate 1 and 0 in the predictions to yes and no, respectively, because the target variable might be either yes or no.

In [61]:
```
submissions['subscribed'].replace(0,'no',inplace=True)
submissions['subscribed'].replace(1,'yes',inplace=True)
```

In [62]:
```
submissions['subscribed']
```

Out[62]:
```
0         no
1        yes
2         no
3         no
4         no
        ...
13559     no
13560     no
13561     no
13562    yes
13563     no
Name: subscribed, Length: 13564, dtype: object
```

In [63]:
```
submissions.to_csv('submission file.csv', header=True, index=False)
```

In [ ]:
```

```