Customer Conversion Prediction

Problem Statement You are working for a new-age insurance company and employ mutiple outreach plans to sell term insurance to your customers. Telephonic marketing campaigns still remain one of the most effective way to reach out to people however they incur a lot of cost. Hence, it is important to identify the customers that are most likely to convert beforehand so that they can be specifically targeted via call. We are given the historical marketing data of the insurance company and are required to build a ML model that will predict if a client will subscribe to the insurance.

Data The historical sales data is available as a compressed file here.

Features:

- age (numeric)
- job : type of job
- marital : marital status
- educational_qual : education status
- call_type : contact communication type
- day: last contact day of the month (numeric)
- mon: last contact month of year
- dur: last contact duration, in seconds (numeric)
- num_calls: number of contacts performed during this campaign and for this client
- prev_outcome: outcome of the previous marketing campaign (categorical: "unknown","other","failure","success")

Output variable (desired target):

- y - has the client subscribed to the insurance?

Minimum Requirements It is not sufficient to just fit a model - the model must be analysed to find the important factors that contribute towards the conversion rate. AUROC must be used as a metric to evaluate the performance of the models.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
data = pd.read_csv('C:/Users/Karthi/Desktop/train.csv') # reading file
data
```

Out[3]:

	age	job	marital	education_qual	call_type	day	mon	dur	num_calls	prev_outcome	y
0	58	management	married	tertiary	unknown	5	may	261	1	unknown	no
1	44	technician	single	secondary	unknown	5	may	151	1	unknown	no
2	33	entrepreneur	married	secondary	unknown	5	may	76	1	unknown	no
3	47	blue-collar	married	unknown	unknown	5	may	92	1	unknown	no
4	33	unknown	single	unknown	unknown	5	may	198	1	unknown	no
...	...	...	...	...	...	...	...	...	...	...	...
45206	51	technician	married	tertiary	cellular	17	nov	977	3	unknown	yes
45207	71	retired	divorced	primary	cellular	17	nov	456	2	unknown	yes
45208	72	retired	married	secondary	cellular	17	nov	1127	5	success	yes
45209	57	blue-collar	married	secondary	telephone	17	nov	508	4	unknown	no
45210	37	entrepreneur	married	secondary	cellular	17	nov	361	2	other	no

45211 rows × 11 columns

In [4]:
```
print(data.shape) # data shape
data.head()
```

```
(45211, 11)
```

Out[4]:

	age	job	marital	education_qual	call_type	day	mon	dur	num_calls	prev_outcome	y
0	58	management	married	tertiary	unknown	5	may	261	1	unknown	no
1	44	technician	single	secondary	unknown	5	may	151	1	unknown	no
2	33	entrepreneur	married	secondary	unknown	5	may	76	1	unknown	no
3	47	blue-collar	married	unknown	unknown	5	may	92	1	unknown	no
4	33	unknown	single	unknown	unknown	5	may	198	1	unknown	no

Data Cleaning

In [5]:
```
data = data.drop_duplicates()
print(data.shape)
```

```
(45205, 11)
```

In [6]:
```
data['age'].value_counts()
```

Out[6]:
```
32    2084
31    1996
33    1972
34    1929
35    1894
      ...
93       2
90       2
95       2
88       2
94       1
Name: age, Length: 77, dtype: int64
```

In [7]:
```
data.job.value_counts()
```

Out[7]:
```
blue-collar      9730
management       9457
technician       7596
admin.           5170
services         4153
retired          2264
self-employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student           938
unknown           288
Name: job, dtype: int64
```

In [8]:
```
data['job'] = data['job'].replace('unknown',data['job'].mode()[0])
```

In [9]:
```
data.job.value_counts()
```

Out[9]:
```
blue-collar      10018
management        9457
technician        7596
admin.            5170
services          4153
retired           2264
self-employed     1579
entrepreneur      1487
unemployed        1303
housemaid         1240
student            938
Name: job, dtype: int64
```

In [10]:
```
data['marital'].value_counts()
```

Out[10]:
```
married     27210
single      12788
divorced     5207
Name: marital, dtype: int64
```

In [11]:
```
data['education_qual'].value_counts()
```

Out[11]:
```
secondary    23199
tertiary     13299
primary       6850
unknown       1857
Name: education_qual, dtype: int64
```

In [12]:
```
data['education_qual'] = data['education_qual'].replace('unknown',data['education_qual'].mode()[(
```

In [13]:
```
data.education_qual.value_counts()
```

Out[13]:
```
secondary    25056
tertiary     13299
primary       6850
Name: education_qual, dtype: int64
```

In [14]:
```
data.call_type.value_counts()
```

Out[14]:
```
cellular     29282
unknown      13017
telephone     2906
Name: call_type, dtype: int64
```

In [15]:
```
data.day.value_counts()
```

Out[15]:
```
20    2752
18    2308
21    2026
17    1939
6     1932
5     1910
14    1848
8     1840
28    1829
7     1817
19    1756
29    1745
15    1703
12    1603
13    1585
30    1566
9     1560
11    1479
4     1445
16    1415
2     1292
27    1121
3     1079
26    1035
23     939
22     905
25     840
31     643
10     524
24     447
1      322
Name: day, dtype: int64
```

In [16]:
```
data.mon.value_counts()
```

Out[16]:
```
may    13765
jul     6894
aug     6245
jun     5339
nov     3970
apr     2932
feb     2649
jan     1403
oct      738
sep      579
mar      477
dec      214
Name: mon, dtype: int64
```

In [17]:
```
data.dur.value_counts()
```

Out[17]:
```
124     187
90      184
89      177
104     175
122     175
       ...
1833      1
1545      1
1352      1
1342      1
1556      1
Name: dur, Length: 1573, dtype: int64
```

In [18]:
```
data.num_calls.value_counts()
```

Out[18]:
```
1     17542
2     12503
3      5521
4      3520
5      1764
6      1291
7       735
8       540
9       327
10      266
11      201
12      155
13      133
14       93
15       84
16       79
17       69
18       51
19       44
20       43
21       35
22       23
25       22
23       22
24       20
29       16
28       16
26       13
31       12
27       10
32        9
30        8
33        6
34        5
36        4
35        4
43        3
38        3
37        2
50        2
41        2
46        1
58        1
55        1
63        1
51        1
39        1
44        1
Name: num_calls, dtype: int64
```

In [19]:
```
data.prev_outcome.value_counts()
```

Out[19]:
```
unknown    36953
failure     4901
other       1840
success     1511
Name: prev_outcome, dtype: int64
```

In [20]:
```
data.y.value_counts()
```

Out[20]:
```
no     39916
yes     5289
Name: y, dtype: int64
```

In [21]:
```
data.isnull().sum()
```

Out[21]:
```
age               0
job               0
marital           0
education_qual    0
call_type         0
day               0
mon               0
dur               0
num_calls         0
prev_outcome      0
y                 0
dtype: int64
```

In [22]:
```
data.shape
```

Out[22]:
```
(45205, 11)
```

In [23]:
```
data.describe() #describing the data
```

Out[23]:

	age	day	dur	num_calls
count	45205.000000	45205.00000	45205.000000	45205.000000
mean	40.937087	15.80688	258.183055	2.763898
std	10.619130	8.32234	257.538504	3.098189
min	18.000000	1.00000	0.000000	1.000000
25%	33.000000	8.00000	103.000000	1.000000
50%	39.000000	16.00000	180.000000	2.000000
75%	48.000000	21.00000	319.000000	3.000000
max	95.000000	31.00000	4918.000000	63.000000

In [24]:
```
iqr = data['age'].quantile(0.75) - data['age'].quantile(0.25)
upper_threshold = data['age'].quantile(0.75) + (1.5 * iqr)
lower_threshold = data['age'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[24]:
```
(70.5, 10.5)
```

In [51]:
```
data.age=data.age.clip(10.5,70.5)
```

In [26]:
```
iqr = data['day'].quantile(0.75) - data['day'].quantile(0.25)
upper_threshold = data['day'].quantile(0.75) + (1.5 * iqr)
lower_threshold = data['day'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[26]:
```
(40.5, -11.5)
```

In [27]:
```
iqr = data['dur'].quantile(0.75) - data['dur'].quantile(0.25)
upper_threshold = data['dur'].quantile(0.75) + (1.5 * iqr)
```

```
lower_threshold = data['dur'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[27]: (643.0, -221.0)

In [28]:
```
data.dur=data.dur.clip(-221.0,643.0)
```

In [29]:
```
iqr = data['num_calls'].quantile(0.75) - data['num_calls'].quantile(0.25)
upper_threshold = data['num_calls'].quantile(0.75) + (1.5 * iqr)
lower_threshold = data['num_calls'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[29]: (6.0, -2.0)

In [30]:
```
data.num_calls=data.num_calls.clip(-2.0,6.0)
```

In [52]:
```
data.describe()
```

Out[52]:

	age	day	dur	num_calls	target
count	45205.000000	45205.00000	45205.00000	45205.000000	45205.000000
mean	40.869052	15.80688	234.95620	2.392235	0.117000
std	10.395247	8.32234	176.75476	1.600152	0.321424
min	18.000000	1.00000	0.00000	1.000000	0.000000
25%	33.000000	8.00000	103.00000	1.000000	0.000000
50%	39.000000	16.00000	180.00000	2.000000	0.000000
75%	48.000000	21.00000	319.00000	3.000000	0.000000
max	70.500000	31.00000	643.00000	6.000000	1.000000

In [32]:
```
data['target'] = data['y'].map({'yes':1,'no':0})
```

In [33]:
```
data.groupby('prev_outcome')['target'].mean()
```

Out[33]:
```
prev_outcome
failure    0.126097
other      0.166848
success    0.647253
unknown    0.091630
Name: target, dtype: float64
```

In [34]:
```
data.groupby('job')['target'].mean()
```

Out[34]:
```
job
admin.           0.122050
blue-collar      0.074067
entrepreneur     0.082717
housemaid        0.087903
management       0.137570
retired          0.227915
self-employed    0.118429
services         0.088851
student          0.286780
technician       0.110585
unemployed       0.155027
Name: target, dtype: float64
```

In [35]:
```
data.groupby('marital')['target'].mean()
```

Out[35]:
```
marital
divorced    0.119455
married     0.101250
single      0.149515
Name: target, dtype: float64
```

In [36]:
```
data.groupby('education_qual')['target'].mean()
```

Out[36]:
```
education_qual
primary      0.086277
secondary    0.107838
tertiary     0.150086
Name: target, dtype: float64
```

In [37]:
```
data.groupby('call_type')['target'].mean()
```

Out[37]:
```
call_type
cellular     0.149204
telephone    0.134205
unknown      0.040716
Name: target, dtype: float64
```

In [38]:
```
data.groupby('mon')['target'].mean()
```

Out[38]:
```
mon
apr    0.196794
aug    0.110168
dec    0.467290
feb    0.166478
jan    0.101212
jul    0.090949
jun    0.102266
mar    0.519916
may    0.067199
nov    0.101511
oct    0.437669
sep    0.464594
Name: target, dtype: float64
```

In [53]:
```
data.isnull().sum()
```

Out[53]:
```
age              0
job              0
marital          0
education_qual   0
call_type        0
day              0
mon              0
dur              0
num_calls        0
prev_outcome     0
y                0
target           0
dtype: int64
```

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

Out[54]:
```
age               float64
job                object
marital            object
education_qual     object
call_type          object
day                 int64
mon                object
dur                 int64
num_calls           int64
prev_outcome       object
y                  object
target              int64
dtype: object
```

In [55]:
```
data.astype({'age':'int64'}).dtypes
```

Out[55]:
```
age                int64
job               object
marital           object
education_qual    object
call_type         object
day                int64
mon               object
dur                int64
num_calls          int64
prev_outcome      object
y                 object
target             int64
dtype: object
```

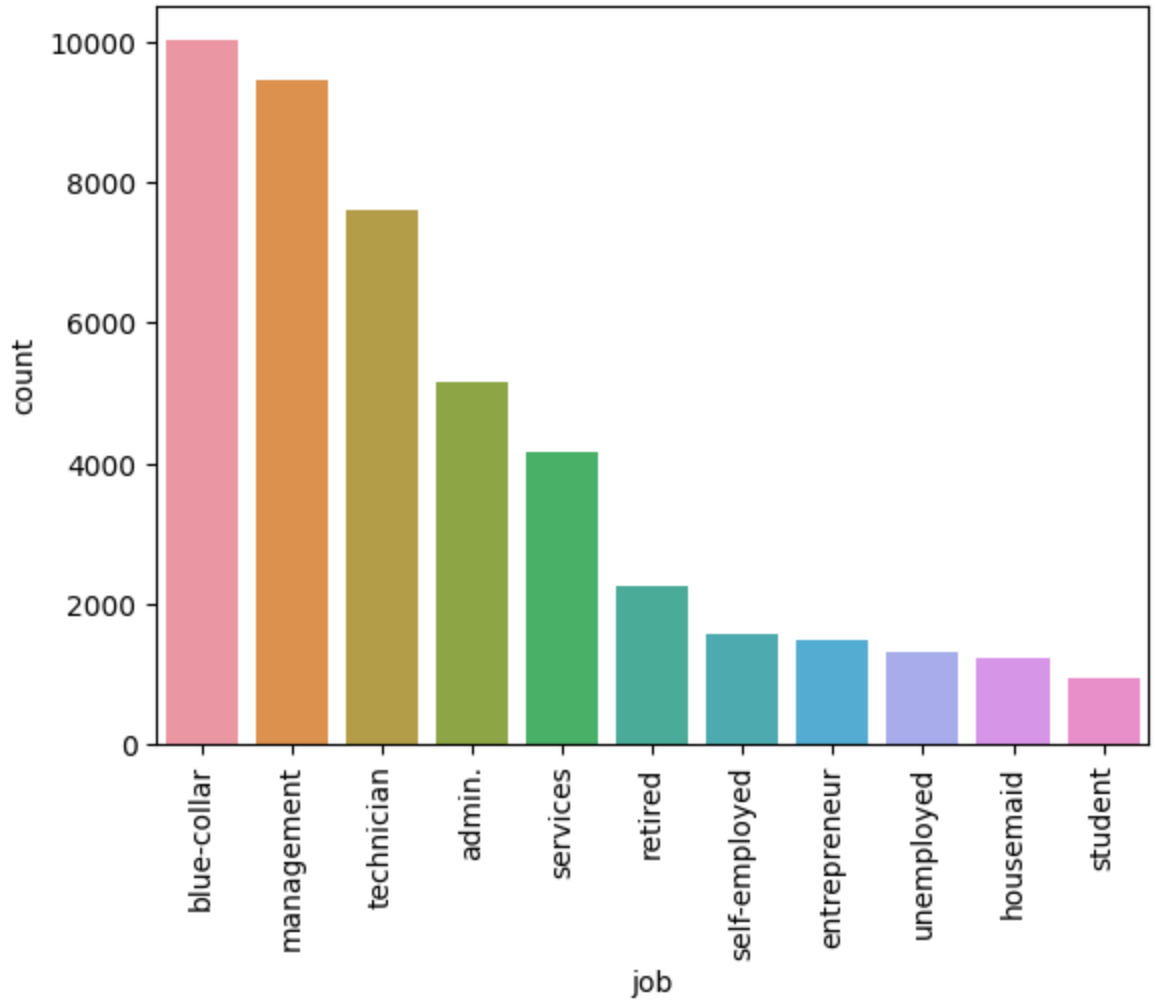
In [56]:
```
data.head()
```

Out[56]:

	age	job	marital	education_qual	call_type	day	mon	dur	num_calls	prev_outcome	y	target
0	58.0	management	married	tertiary	unknown	5	may	261	1	unknown	no	0
1	44.0	technician	single	secondary	unknown	5	may	151	1	unknown	no	0
2	33.0	entrepreneur	married	secondary	unknown	5	may	76	1	unknown	no	0
3	47.0	blue-collar	married	secondary	unknown	5	may	92	1	unknown	no	0
4	33.0	blue-collar	single	secondary	unknown	5	may	198	1	unknown	no	0

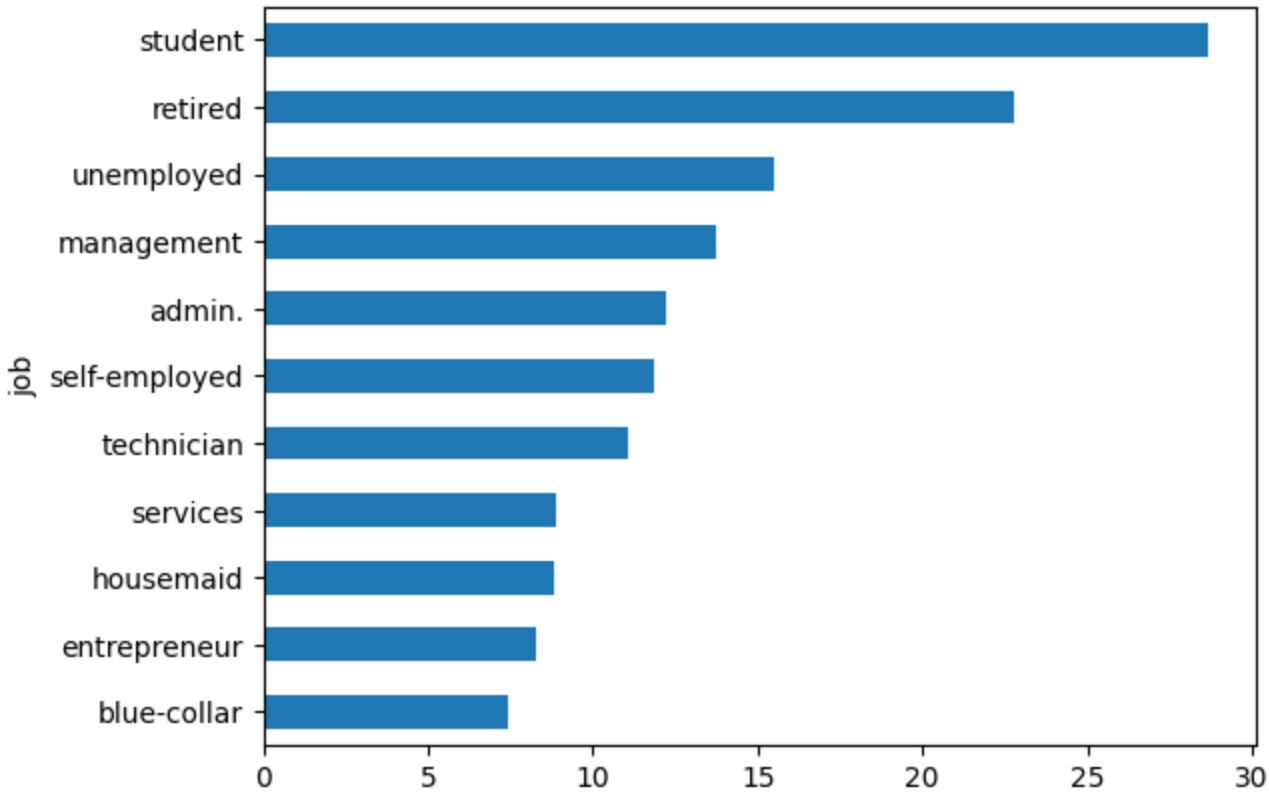
EDA

In [57]:
```
data_j = pd.DataFrame(data.job.value_counts()).sort_values('job', ascending = False).reset_index
data_j.rename(columns = {'index':'job','job':'count'},inplace=True)
bar = sns.barplot(x=data_j['job'], y=data_j['count'], data = data_j)
bar.tick_params(axis ='x', rotation=90)
```



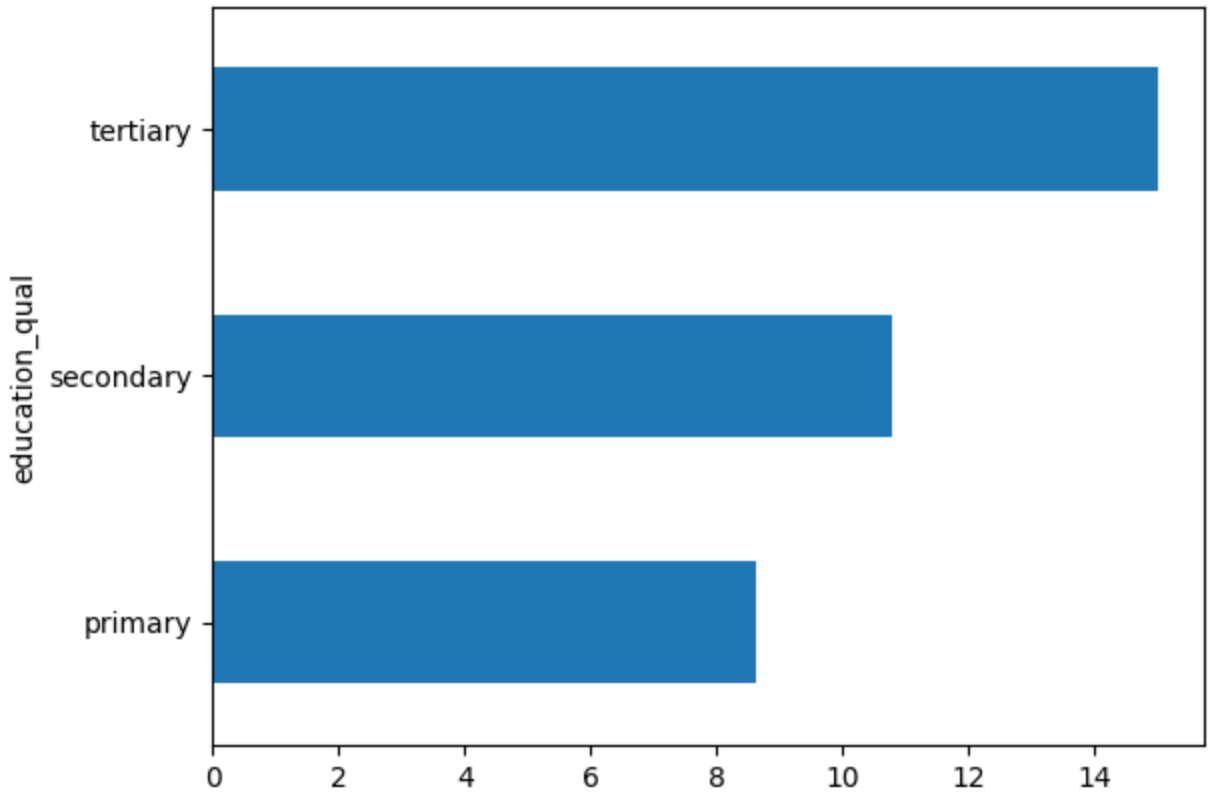
In [58]:
```
(data.groupby('job')['target'].mean()*100).sort_values().plot(kind='barh')
```

Out[58]:
```
<AxesSubplot: ylabel='job'>
```



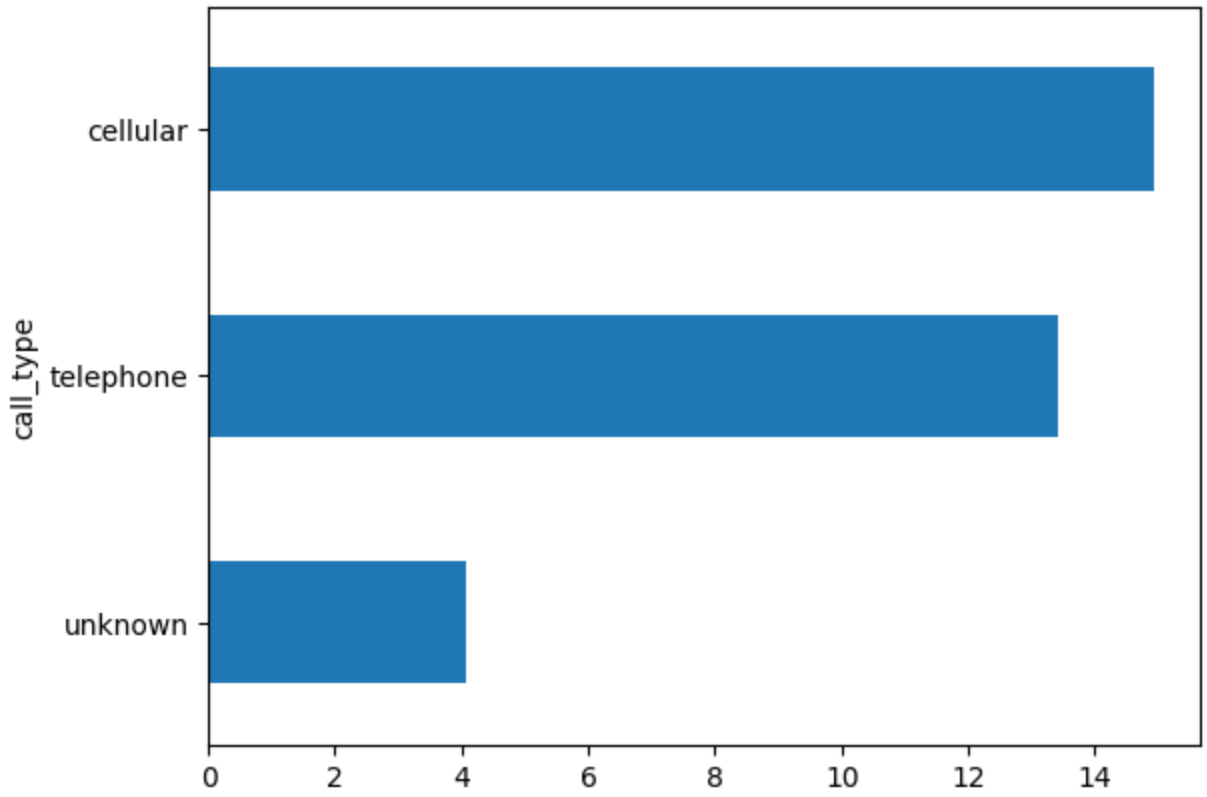
In [59]:
```
(data.groupby('education_qual')['target'].mean()*100).sort_values().plot(kind='barh')
```

Out[59]: <AxesSubplot: ylabel='education_qual'>



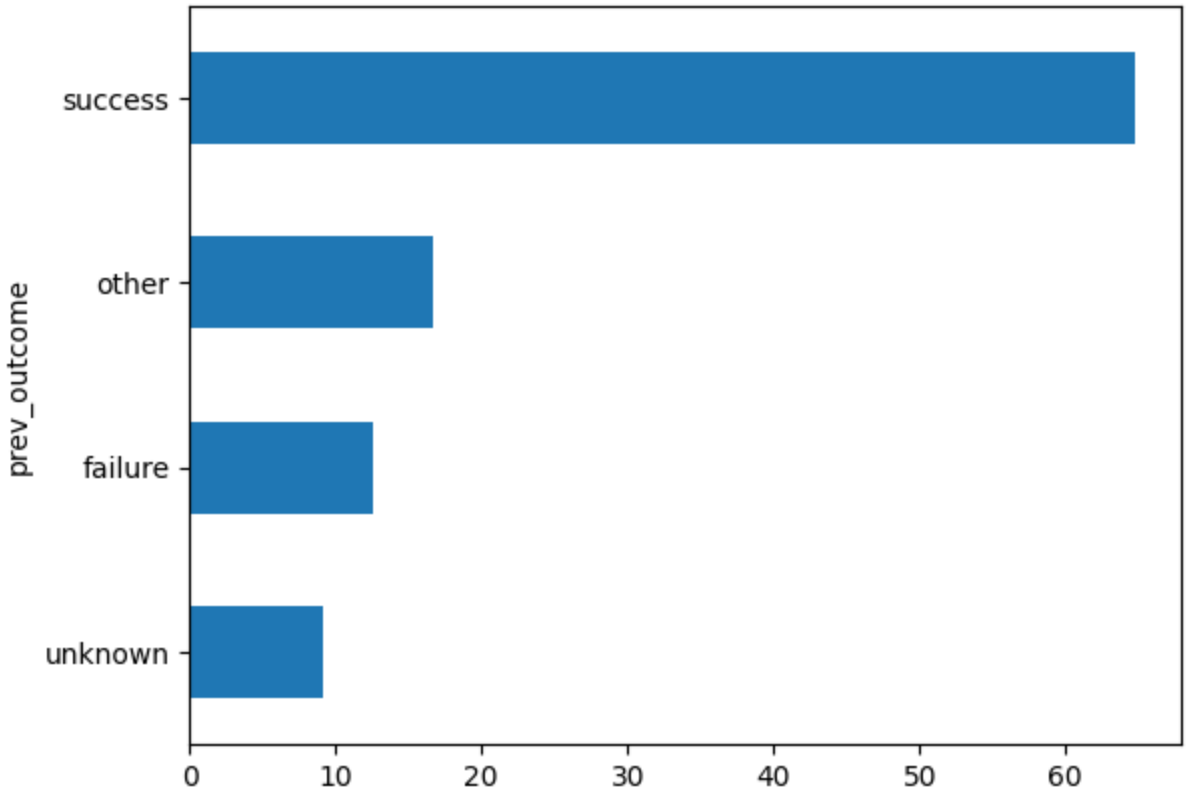
In [60]:
```
(data.groupby('call_type')['target'].mean()*100).sort_values().plot(kind='barh')
```

Out[60]: <AxesSubplot: ylabel='call_type'>



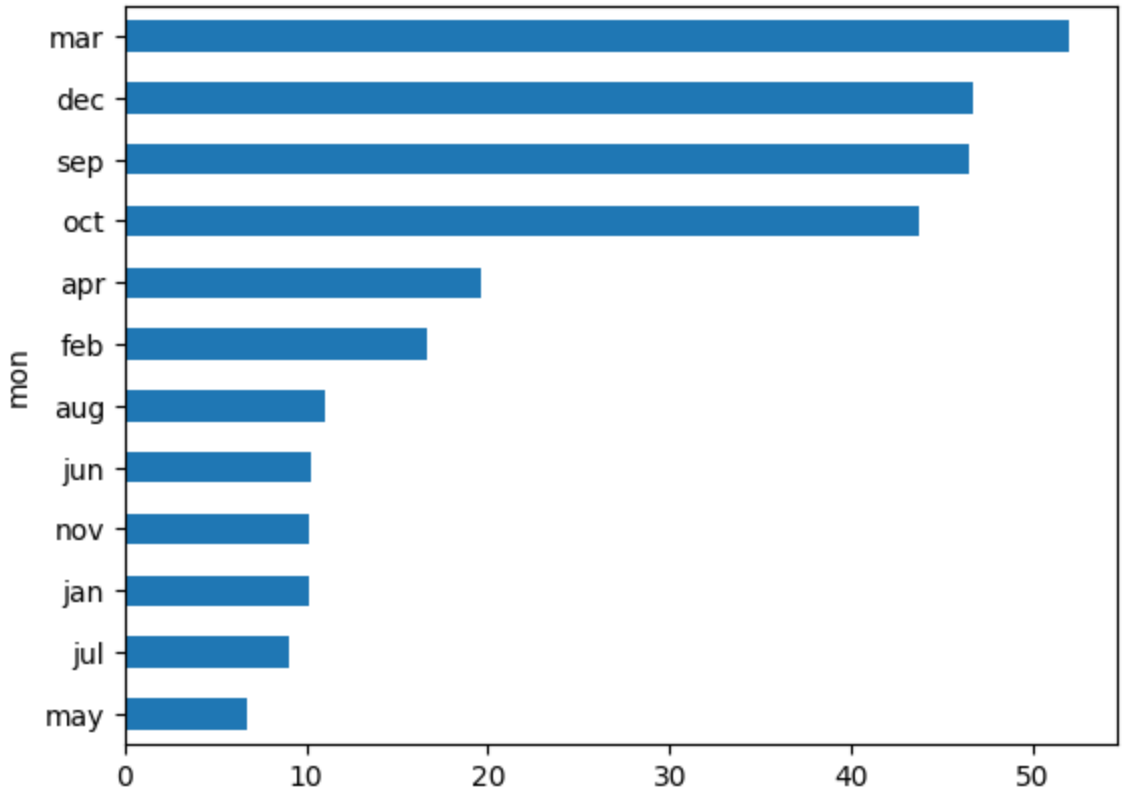
In [61]:
```
(data.groupby('prev_outcome')['target'].mean()*100).sort_values().plot(kind='barh')
```

Out[61]: <AxesSubplot: ylabel='prev_outcome'>



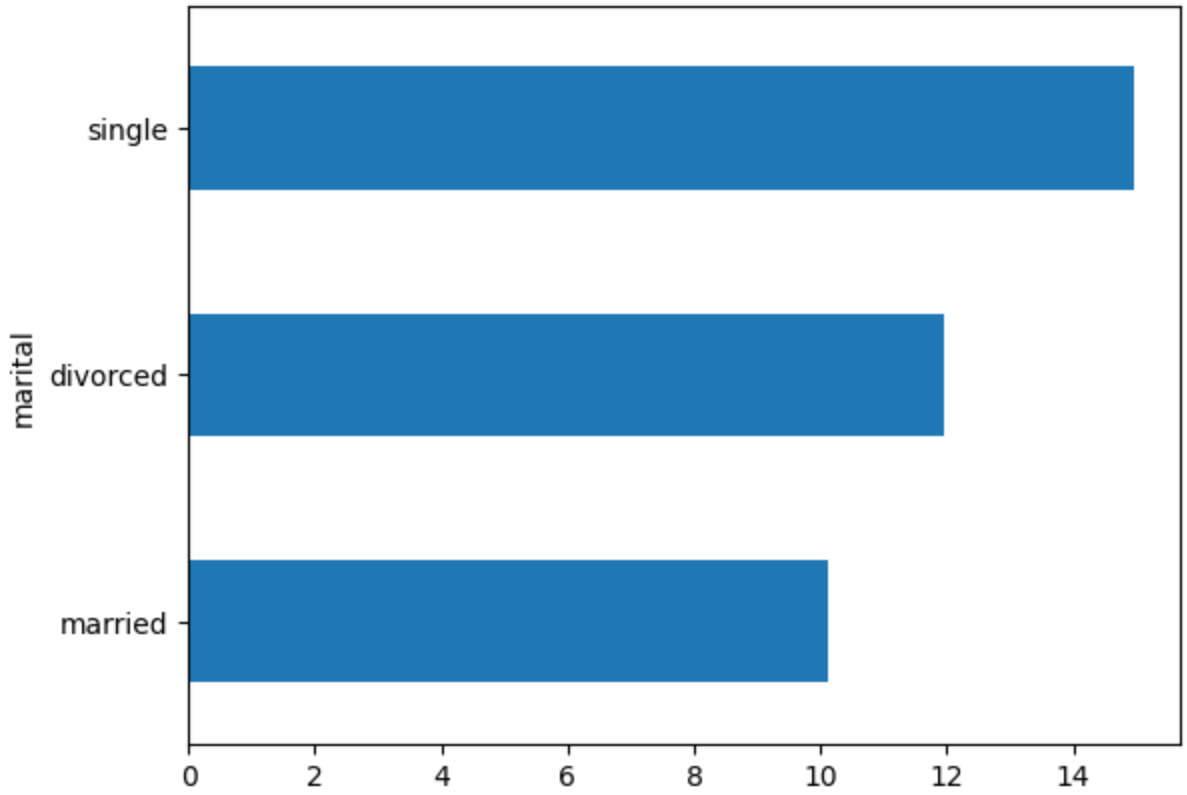
In [62]:
```
(data.groupby('mon')['target'].mean()*100).sort_values().plot(kind='barh')
```

Out[62]: <AxesSubplot: ylabel='mon'>



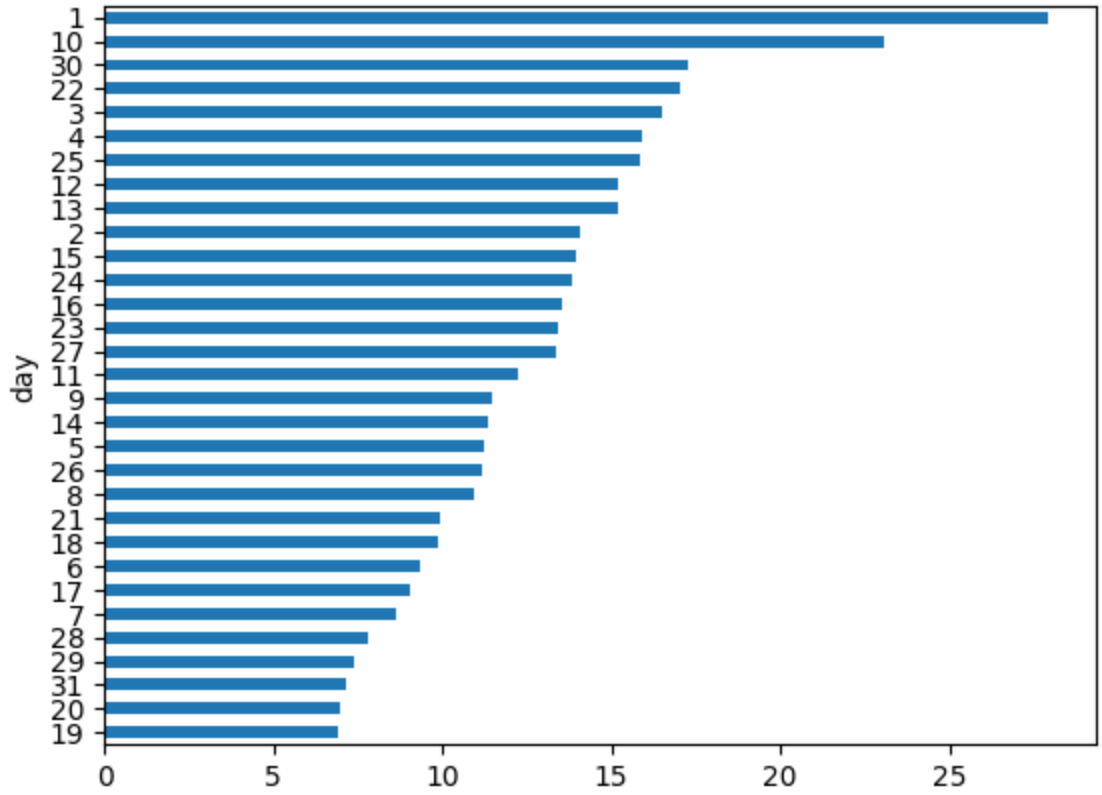
In [63]:
```
(data.groupby('marital')['target'].mean()*100).sort_values().plot(kind='barh')
```

Out[63]: <AxesSubplot: ylabel='marital'>



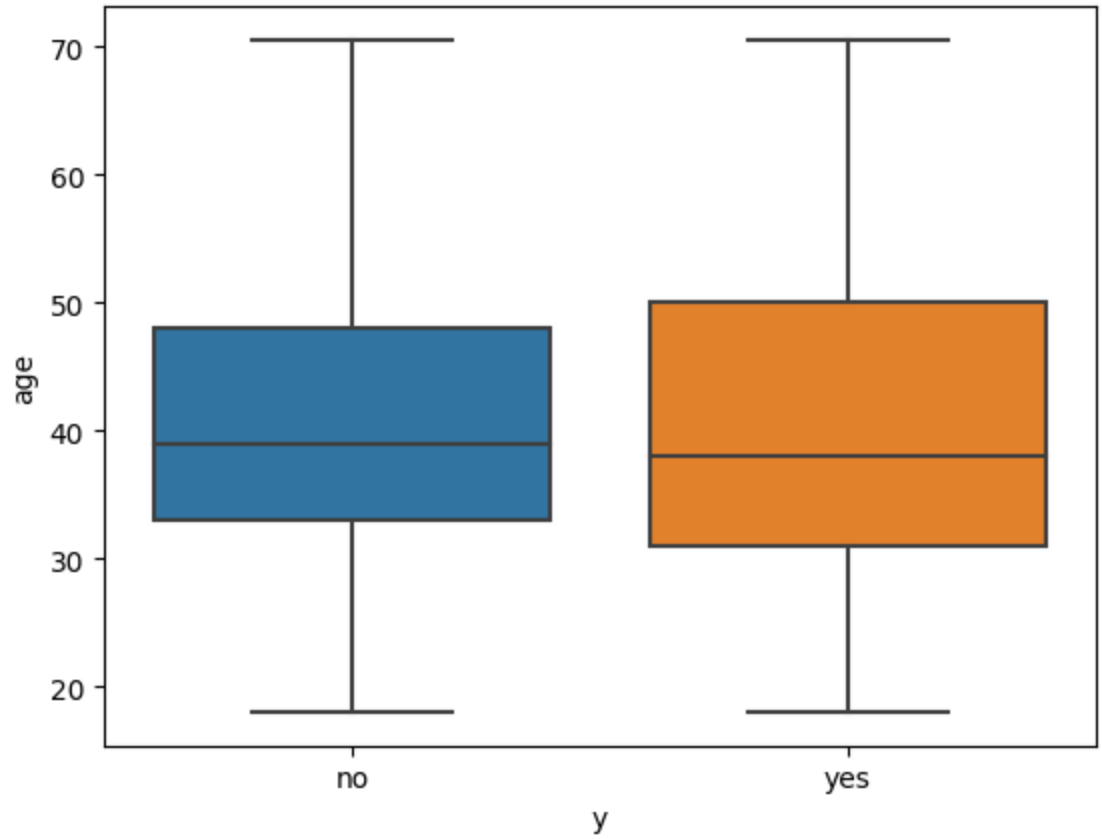
In [64]:
```
(data.groupby('day')['target'].mean()*100).sort_values().plot(kind='barh')
```

Out[64]: <AxesSubplot: ylabel='day'>



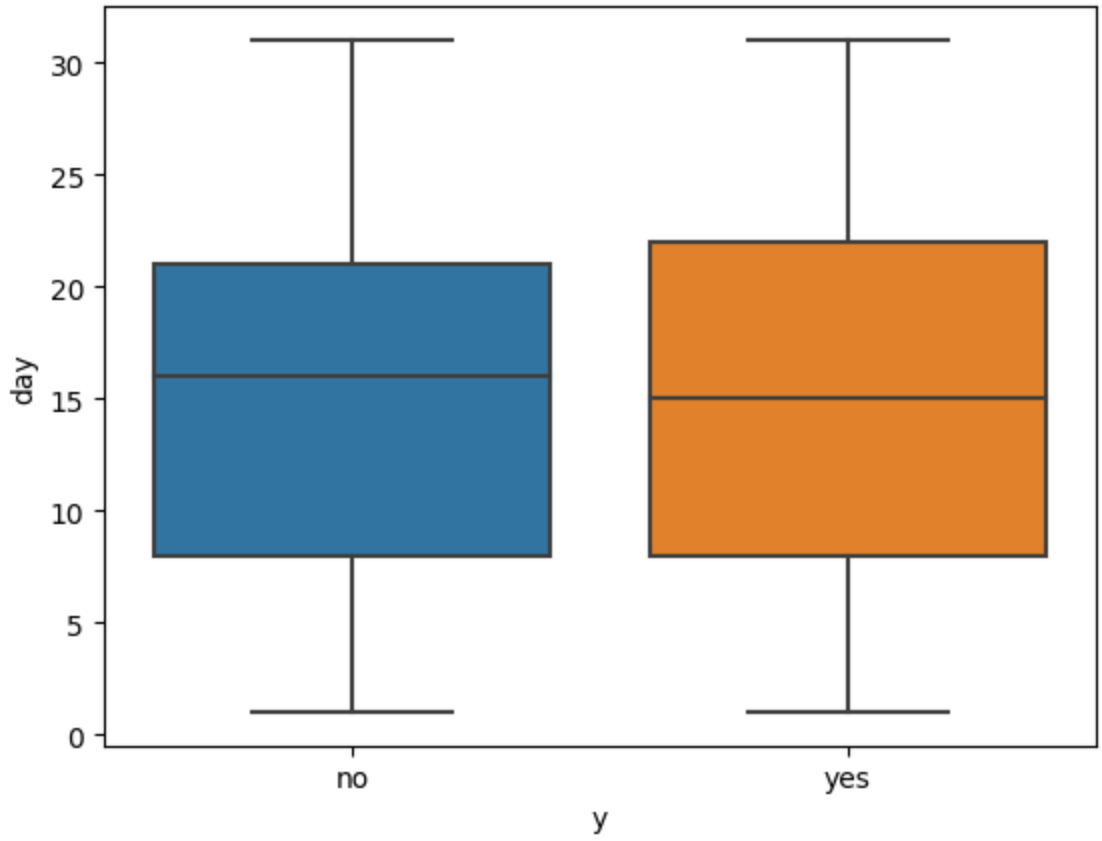
In [65]:
```
sns.boxplot(data, x ='y', y ='age')
```

Out[65]: <AxesSubplot: xlabel='y', ylabel='age'>



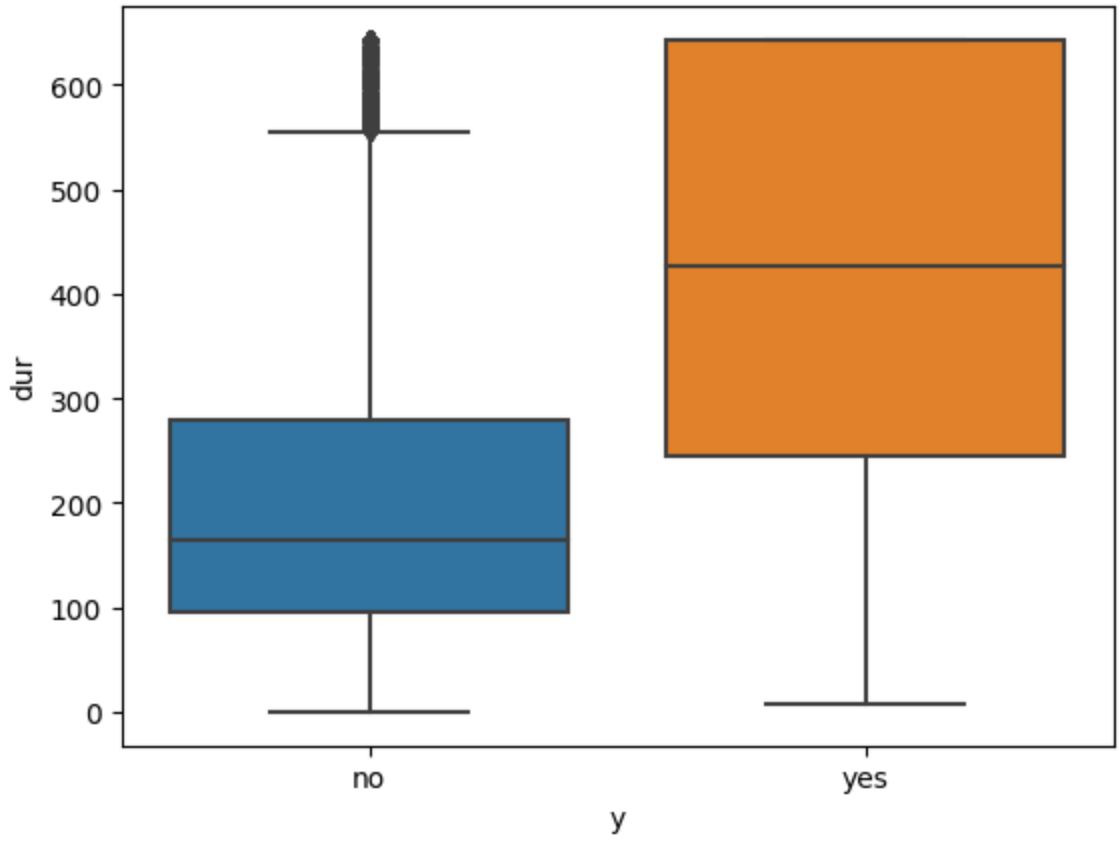
In [66]:
```
sns.boxplot(data, x ='y', y ='day')
```

Out[66]: <AxesSubplot: xlabel='y', ylabel='day'>



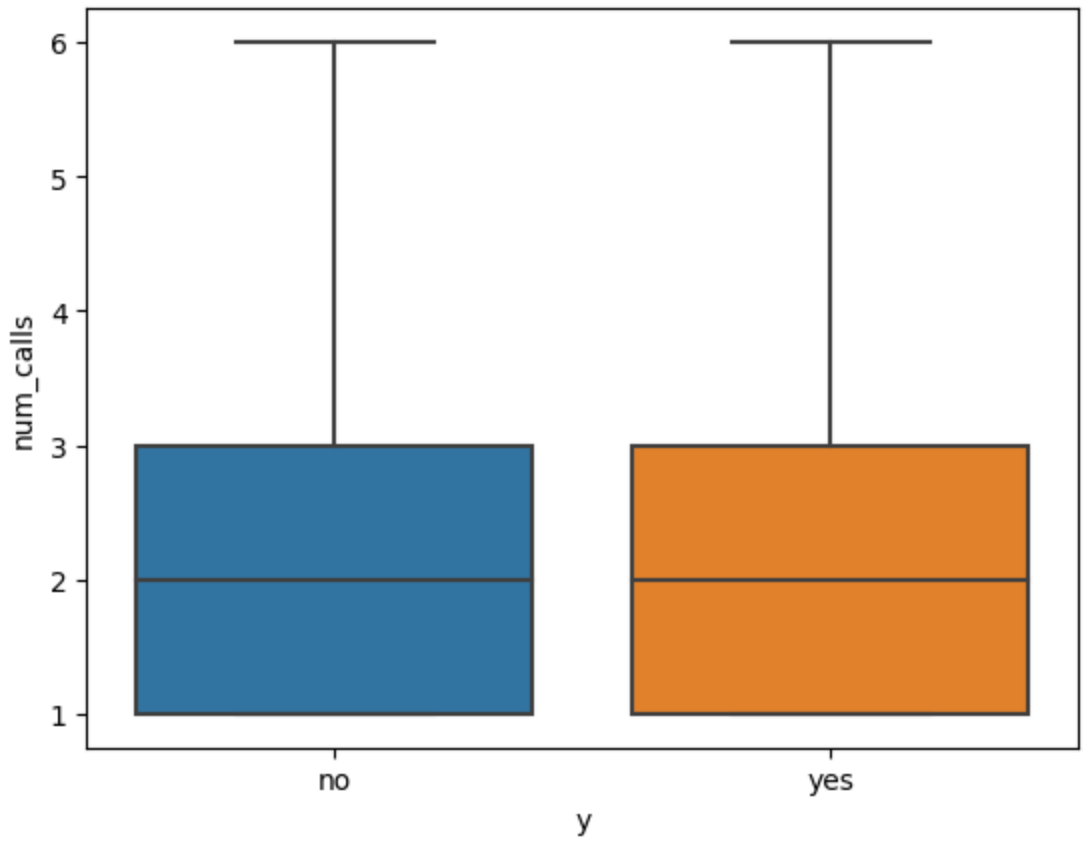
In [67]:
```
sns.boxplot(data, x ='y', y ='dur')
```

Out[67]: <AxesSubplot: xlabel='y', ylabel='dur'>



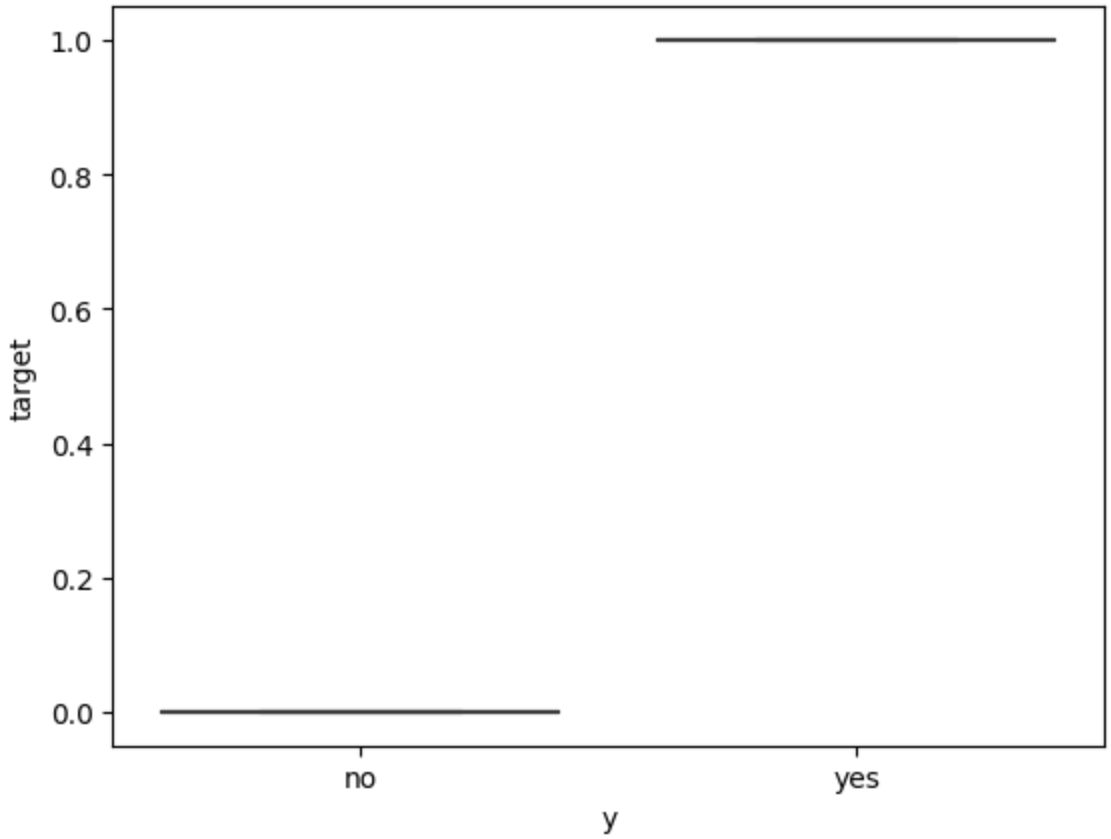
In [68]:
```
sns.boxplot(data, x ='y', y ='num_calls')
```

Out[68]: <AxesSubplot: xlabel='y', ylabel='num_calls'>



In [69]:
```
sns.boxplot(data, x ='y', y ='target')
```

Out[69]: <AxesSubplot: xlabel='y', ylabel='target'>



Data Encoding

In [70]:
```
col = data['job'].unique()
P=[]
for i in col:
 p = len(data[data['job']==i][data['y']=='yes'])/len(data[data['job']==i])
 P.append(p)
df = pd.DataFrame({'job':col, '%':P})
df = df.sort_values('%', ascending=True)
df = df.reset_index()
del df['index']
```

In [71]:
```
df
```

Out[71]:

	job	%
0	blue-collar	0.074067
1	entrepreneur	0.082717
2	housemaid	0.087903
3	services	0.088851
4	technician	0.110585
5	self-employed	0.118429
6	admin.	0.122050
7	management	0.137570
8	unemployed	0.155027
9	retired	0.227915
10	student	0.286780

In [72]:
```
data['job'] = data['job'].map({'blue-collar': 0,'entrepreneur': 1,'housemaid': 2,'services': 3,'
```

In [73]:
```
col = data['marital'].unique()
P =[]
for i in col:
 p = len(data[data['marital']==i][data['y']=='yes'])/len(data[data['marital']==i])
 P.append(p)
df = pd.DataFrame({'marital':col, '%':P})
df = df.sort_values('%', ascending=True)
df = df.reset_index()
del df['index']
```

In [74]:
```
df
```

Out[74]:

	marital	%
0	married	0.101250
1	divorced	0.119455
2	single	0.149515

In [75]:
```
data['marital'] = data['marital'].map({'married': 0, 'divorced': 1, 'single': 2})
```

In [76]:
```
col = data['education_qual'].unique()
P =[]
for i in col:
 p = len(data[data['education_qual']==i][data['y']=='yes'])/len(data[data['education_qual']==i])
 P.append(p)
df = pd.DataFrame({'education_qual':col, '%':P})
df = df.sort_values('%', ascending=True)
df = df.reset_index()
del df['index']
```

In [77]:
```
df
```

Out[77]:

	education_qual	%
0	primary	0.086277
1	secondary	0.107838
2	tertiary	0.150086

In [78]:
```
data['education_qual'] = data['education_qual'].map({'primary': 0, 'secondary': 1, 'tertiary': 2
```

In [79]:
```
col = data['call_type'].unique()
P =[]
for i in col:
 p = len(data[data['call_type']==i][data['y']=='yes'])/len(data[data['call_type']==i])
 P.append(p)
df = pd.DataFrame({'call_type':col, '%':P})
df = df.sort_values('%', ascending=True)
df = df.reset_index()
del df['index']
```

In [80]:
```
df
```

Out[80]:

	call_type	%
0	unknown	0.040716
1	telephone	0.134205
2	cellular	0.149204

In [81]:
```
data['call_type'] = data['call_type'].map({'unknown': 0, 'telephone': 1, 'cellular': 2})
```

In [82]:
```
col = data['mon'].unique()
P =[]
for i in col:
 p = len(data[data['mon']==i][data['y']=='yes'])/len(data[data['mon']==i])
 P.append(p)
df = pd.DataFrame({'mon':col, '%':P})
df = df.sort_values('%', ascending=True)
df = df.reset_index()
del df['index']
```

In [83]:
```
df
```

Out[83]:

	mon	%
0	may	0.067199
1	jul	0.090949
2	jan	0.101212
3	nov	0.101511
4	jun	0.102266
5	aug	0.110168
6	feb	0.166478
7	apr	0.196794
8	oct	0.437669
9	sep	0.464594
10	dec	0.467290
11	mar	0.519916

In [84]:
```
data['mon'] = data['mon'].map({'may': 0, 'jul': 1, 'jan': 2, 'nov': 3, 'jun': 4, 'aug': 5, 'feb'
```

In [85]:
```
col = data['prev_outcome'].unique()
P =[]
for i in col:
 p = len(data[data['prev_outcome']==i][data['y']=='yes'])/len(data[data['prev_outcome']==i])
 P.append(p)
df = pd.DataFrame({'prev_outcome':col, '%':P})
df = df.sort_values('%', ascending=True)
df = df.reset_index()
del df['index']
```

In [86]:
```
df
```

Out[86]:

	prev_outcome	%
0	unknown	0.091630
1	failure	0.126097
2	other	0.166848
3	success	0.647253

In [87]:
```
data['prev_outcome'] = data['prev_outcome'].map({'unknown': 0, 'failure': 1, 'other': 2, 'succes
```

In [88]:
```
data.head()
```

Out[88]:

	age	job	marital	education_qual	call_type	day	mon	dur	num_calls	prev_outcome	y	target
0	58.0	7.0	0	2	0	5	0	261	1	0	no	0
1	44.0	4.0	2	1	0	5	0	151	1	0	no	0
2	33.0	1.0	0	1	0	5	0	76	1	0	no	0
3	47.0	0.0	0	1	0	5	0	92	1	0	no	0
4	33.0	0.0	2	1	0	5	0	198	1	0	no	0

In [90]:
```
data.isnull().sum()
```

Out[90]:
```
age                0
job             5170
marital            0
education_qual     0
call_type          0
day                0
mon                0
dur                0
num_calls          0
prev_outcome       0
y                  0
target             0
dtype: int64
```

In [91]:
```
data.dropna(subset=['job'],inplace=True)
```

In [92]:
```
data.isnull().sum()
```

Out[92]:
```
age              0
job              0
marital          0
education_qual   0
call_type        0
day              0
mon              0
dur              0
num_calls        0
prev_outcome     0
y                0
target           0
dtype: int64
```

Splitting the Dataset

In [93]:
```
col = [*data.columns]
col[:-2]
```

Out[93]:
```
['age',
 'job',
 'marital',
 'education_qual',
 'call_type',
 'day',
 'mon',
 'dur',
 'num_calls',
 'prev_outcome']
```

In [94]:
```
x = data.loc[:, col[:-2]].values
y = data.loc[:, col[-1]].values
```

In [95]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
```

Balancing Dataset using SMOTEENN

In [96]:
```
data.shape
```

Out[96]:
```
(40035, 12)
```

In [97]:
```
len(x_train),len(y_train)
```

Out[97]:
```
(30026, 30026)
```

In [83]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\karthi\pycharmprojects\pythonproject\venv\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\karthi\pycharmprojects\pythonproject\venv\lib\site-packages (from imblearn) (0.10.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\karthi\pycharmprojects\pythonproject\venv\lib\site-packages (from imbalanced-learn->imblearn) (1.1.3)
Requirement already satisfied: numpy>=1.17.3 in c:\users\karthi\pycharmprojects\pythonproject\venv\lib\site-packages (from imbalanced-learn->imblearn) (1.23.4)
Requirement already satisfied: scipy>=1.3.2 in c:\users\karthi\pycharmprojects\pythonproject\venv\lib\site-packages (from imbalanced-learn->imblearn) (1.9.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\karthi\pycharmprojects\pythonproject\venv\lib\site-packages (from imbalanced-learn->imblearn) (3.1.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\karthi\pycharmprojects\pythonproject\venv\lib\site-packages (from imbalanced-learn->imblearn) (1.2.0)
```

In [98]:
```
from imblearn.combine import SMOTEENN
smt = SMOTEENN(sampling_strategy="all")
x_smt, y_smt = smt.fit_resample(x_train, y_train)
```

In [99]:
```
len(x_smt),len(y_smt)
```

Out[99]:
```
(45578, 45578)
```

In [100…
```
data_bal = pd.DataFrame(x_smt, columns=data.columns[:-2])
```

In [101…
```
data_bal['y'] = y_smt
```

In [102…
```
len(data_bal[data_bal['y']==1])/len(data_bal)
```

Out[102]:
```
0.559239106586511
```

Scaling of Dataset

In [103…
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_smt)
x_test_scaled = scaler.transform(x_test)
```

In [105…
```
x_train_scaled
```

Out[105]:
```
array([[ 1.52640222, -1.47143497, -0.85471737, ..., -1.1648402 ,
         1.90008879, -0.58321682],
       [ 0.66255879, -0.81432857, -0.85471737, ..., -1.46538529,
        -0.86084019, -0.58321682],
       [ 0.05786839,  0.82843741, -0.85471737, ..., -1.0312646 ,
        -0.17060795, -0.58321682],
       ...,
       [ 0.59912978,  0.26553201, -0.85471737, ..., -0.33126041,
        -0.33954578,  2.77839917],
       [-1.24924598,  0.41914351,  0.6041879 , ...,  1.57345955,
         1.47016168, -0.58321682],
       [-1.35937532, -1.17882063, -0.85471737, ...,  1.57345955,
         1.41476689, -0.58321682]])
```

In [106…
```
x_test_scaled
```

Out[106]:
```
array([[-0.28766898, -1.47143497, -0.85471737, ..., -0.41109218,
        -0.86084019, -0.58321682],
       [ 0.31702142,  0.17133102,  1.48750169, ..., -0.897689  ,
        -0.17060795, -0.58321682],
       [-0.46043766,  0.82843741, -0.85471737, ..., -1.16961075,
        -0.17060795, -0.58321682],
       ...,
       [-0.37405332, -0.15722218, -0.85471737, ..., -0.22504046,
        -0.86084019,  1.6578605 ],
       [ 0.05786839,  0.82843741,  0.31639216, ..., -1.15052852,
        -0.86084019, -0.58321682],
       [-1.49704977, -0.48577538,  1.48750169, ..., -0.31091048,
        -0.86084019, -0.58321682]])
```

Model : Logisgtic Regression

In [107…
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
```

In [108…
```
lr = LogisticRegression()
lr.fit(x_train_scaled,y_smt)
lr.score(x_test_scaled,y_test)
log = roc_auc_score(y_test, lr.predict_proba(x_test_scaled)[:, 1])
```

Model : KNN

In [111…
```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier()
KNN.fit(x_train_scaled, y_smt)
KNN.score(x_test_scaled, y_test)
```

Out[111]: 0.807473274053352

In [112…
```
'''for i in [1,2,3,4,5,6,7,8,9,10,20,30,40,50]:
    KNN = KNeighborsClassifier(n_neighbors=i)
    KNN.fit(x_train_scaled, y_smt)
    print('K-value:',i,'Accuracy Score:'KNN.score(x_train_scaled, y_smt), 'Cross-Val Score:', np
```

Out[112]:
```
"for i in [1,2,3,4,5,6,7,8,9,10,20,30,40,50]:\n    KNN = KNeighborsClassifier(n_neighbors=i)\n
  KNN.fit(x_train_scaled, y_smt)\n    print('K-value:',i,'Accuracy Score:'KNN.score(x_train_scal
ed, y_smt), 'Cross-Val Score:', np.mean(cross_val_score(KNN, x_train_scaled, y_smt, cv=10)))"
```

In [114…
```
KNN =KNeighborsClassifier(n_neighbors=3)
KNN.fit(x_train_scaled, y_smt)
KNN.score(x_test_scaled, y_test)
k = roc_auc_score(y_test, KNN.predict_proba(x_test_scaled)[:, 1])
```

Model : Decision Tree

In [115…
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train_scaled, y_smt)
dt.score(x_test_scaled, y_test)
```

Out[115]: 0.8502347886901789

In [116…
```
'''for d in [1,2,3,4,5,6,7,8,9,10,20,30,40,50]:
    dtt = DecisionTreeClassifier(max_depth=d)
    dtt.fit(x_train_scaled,y_smt)
    tt = DecisionTreeClassifier(max_depth=d)
    from sklearn.metrics import accuracy_score
    print('Depth:',d,'Accuracy Score:', accuracy_score(y_smt,dtt.predict(x_train_scaled)), 'cv:',
```

Out[116]:
```
"for d in [1,2,3,4,5,6,7,8,9,10,20,30,40,50]:\n    dtt = DecisionTreeClassifier(max_depth=d)\n
  dtt.fit(x_train_scaled,y_smt)\n    tt = DecisionTreeClassifier(max_depth=d)\n    from sklearn.
metrics import accuracy_score\n    print('Depth:',d,'Accuracy Score:', accuracy_score(y_smt,dtt.
predict(x_train_scaled)), 'cv:',np.mean(cross_val_score(tt,x_train_scaled, y_smt, cv=10)))"
```

In [117…
```
dtt = DecisionTreeClassifier(max_depth=10)
dtt.fit(x_train_scaled,y_smt)
dtt.score(x_test_scaled,y_test)
d = roc_auc_score(y_test, dtt.predict_proba(x_test_scaled)[:,1])
```

In [119…
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, dtt.predict(x_test_scaled))
```

Out[119]:
```
array([[7564, 1313],
       [ 223,  909]], dtype=int64)
```

Model :Random Forest

In [122…
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, max_depth=4, max_features='sqrt')
rf.fit(x_train_scaled, y_smt)
```

Out[122]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=4)
```

In [182…
```
from sklearn.metrics import roc_auc_score
r = roc_auc_score(y_test, rf.predict_proba(x_test_scaled)[:,1])
```

In [124…
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,rf.predict(x_test_scaled))
```

Out[124]:
```
array([[6819, 2058],
       [ 148,  984]], dtype=int64)
```

Model :XGBoost

In [127…
```
!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.7.4-py3-none-win_amd64.whl (89.1 MB)
     ---------------------------------------- 89.1/89.1 MB 975.3 kB/s eta 0:00:00
Requirement already satisfied: scipy in c:\users\karthi\pycharmprojects\pythonproject\venv\lib\s
ite-packages (from xgboost) (1.9.3)
Requirement already satisfied: numpy in c:\users\karthi\pycharmprojects\pythonproject\venv\lib\s
ite-packages (from xgboost) (1.23.4)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.4
```

In [128…
```
import xgboost as xgb
```

In [134…
```
for lr in [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10, 0.11, 0.12, 0.13, 0.14, 0
    xg = xgb.XGBClassifier(learning_rate=lr, n_estimators=100, verbosity=0)
    xg.fit(x_train_scaled, y_smt)
    xg.score(x_test_scaled, y_test)
    print('LR:',lr, 'Train_Score:',xg.score(x_train_scaled, y_smt),'CV:',np.mean(cross_val_score
```

```
LR: 0.01 Train_Score: 0.9445346439071481 CV: 0.9406076682821072
LR: 0.02 Train_Score: 0.9536179735837466 CV: 0.949208331154795
LR: 0.03 Train_Score: 0.9621527930141736 CV: 0.9564487049756278
LR: 0.04 Train_Score: 0.9669138619509412 CV: 0.9607271619599163
LR: 0.05 Train_Score: 0.9699416385098074 CV: 0.9618022911580801
LR: 0.06 Train_Score: 0.9723770240028083 CV: 0.9633381968903855
LR: 0.07 Train_Score: 0.9747465882662688 CV: 0.964501103134852
LR: 0.08 Train_Score: 0.9758874895783053 CV: 0.9651593443220259
LR: 0.09 Train_Score: 0.977006450480495 CV: 0.9661466916594377
LR: 0.1 Train_Score: 0.9783886963008469 CV: 0.9661905898114883
LR: 0.11 Train_Score: 0.9796173592522708 CV: 0.9661686888799597
LR: 0.12 Train_Score: 0.9801439290885954 CV: 0.9670463052805942
LR: 0.13 Train_Score: 0.9811531879415507 CV: 0.9661467879484309
LR: 0.14 Train_Score: 0.9817894598271095 CV: 0.9665636133715754
LR: 0.15 Train_Score: 0.9824476721225153 CV: 0.9664320103899676
LR: 0.16 Train_Score: 0.9830181227785335 CV: 0.9663223179687875
LR: 0.17 Train_Score: 0.9835666330247049 CV: 0.9670244139779649
LR: 0.18 Train_Score: 0.9841370836807232 CV: 0.9660371725584456
LR: 0.19 Train_Score: 0.9849708192549037 CV: 0.966826968582731
LR: 0.2 Train_Score: 0.9858923164684716 CV: 0.96630040259391
LR: 0.21 Train_Score: 0.9868796349115802 CV: 0.9662565525863561
LR: 0.22 Train_Score: 0.986747992452499 CV: 0.9663223516699352
LR: 0.23 Train_Score: 0.9867260520426522 CV: 0.9665856587365941
LR: 0.24 Train_Score: 0.9881521786826978 CV: 0.9663882181558096
LR: 0.25 Train_Score: 0.9887006889288692 CV: 0.9664759470576152
LR: 0.26 Train_Score: 0.9886348676993286 CV: 0.9667611694991518
LR: 0.27 Train_Score: 0.9895344245030497 CV: 0.9664101576029356
LR: 0.28 Train_Score: 0.989644126552284 CV: 0.9666295376308458
LR: 0.29 Train_Score: 0.9904559217166177 CV: 0.966366288337583
LR: 0.3 Train_Score: 0.9902803984378428 CV: 0.9660591216344709
LR: 0.5 Train_Score: 0.9958313221290974 CV: 0.9656861414044308
LR: 0.75 Train_Score: 0.9989688007371977 CV: 0.9659054829167438
```

In [183…
```
xg = xgb.XGBClassifier(learning_rate=0.24, n_estimators=100, verbosity=0)
xg.fit(x_train_scaled, y_smt)
g = roc_auc_score(y_test, xg.predict_proba(x_test_scaled)[:,1])
```

In [136…
```
xg.score(x_train_scaled, y_smt)
```

Out[136]:
```
0.9881521786826978
```

In [137…
```
confusion_matrix(y_test, xg.predict(x_test_scaled))
```

Out[137]:
```
array([[7878,  999],
       [ 247,  885]], dtype=int64)
```

Model: Ensemble Learning - Voting Classifier

In [140…
```
from sklearn.ensemble import VotingClassifier
from sklearn import tree
m1 = LogisticRegression(random_state=12)
m2 = tree.DecisionTreeClassifier(random_state=12)
m3 = KNeighborsClassifier(5)
m4 = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
m5 = RandomForestClassifier(n_estimators=100, max_depth=5, max_features='sqrt')
m = VotingClassifier(estimators=[('lr',m1),('dt',m2),('knn',m3),('xgb',m4),('rf',m5)], voting='s
```

In [184…
```
m.fit(x_train_scaled, y_smt)
y_pred = m.predict(x_test_scaled)
v = roc_auc_score(y_test, m.predict_proba(x_test_scaled)[:,1])
```

In [190…
```
pd.DataFrame({'Model':['Logistic Regression', 'KNN','Decision Tree','Random Forest','XGBoost','V
```

Out[190]:

	Model	AUROC
0	Logistic Regression	0.892721
1	KNN	0.846362
2	Decision Tree	0.873001
3	Random Forest	0.886998
4	XGBoost	0.908968
5	Voting Classifier	0.908913

In [148…
```
imp_ft = pd.DataFrame({'ft':col[:-2], 'imp':xg.feature_importances_})
imp_ft.sort_values('imp', ascending=False, inplace=True)
```

In [149…
```
imp_ft.iloc[0:5,0].values
```

Out[149]:
```
array(['call_type', 'prev_outcome', 'dur', 'mon', 'education_qual'],
      dtype=object)
```

In [151…
```
x_imp = data.loc[:, imp_ft.iloc[0:5,0]].values
y = data.loc[:, col[-1]].values
```

In [152…
```
from sklearn.model_selection import train_test_split
x_train_imp, x_test_imp, y_train, y_test = train_test_split(x_imp, y, test_size=0.25)
```

In [153…
```
from imblearn.combine import SMOTEENN
smt = SMOTEENN(sampling_strategy='all')
x_smt_imp, y_smt = smt.fit_resample(x_train_imp, y_train)
```

In [154…
```
data_bal_imp = pd.DataFrame(x_smt_imp, columns= imp_ft.iloc[0:5,0])
data_bal_imp['y'] = y_smt
len(data_bal_imp[data_bal_imp['y']==1])/len(data_bal_imp)
```

Out[154]:
```
0.40119587509748983
```

In [155…
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train_imp_scaled = scaler.fit_transform(x_smt_imp)
x_test_imp_scaled = scaler.transform(x_test_imp)
```

In [156…
```
'''for lr in [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10, 0.11, 0.12, 0.13, 0.14,
    xg = xgb.XGBClassifier(learning_rate=lr, n_estimators=100, verbosity=0)
    xg.fit(x_train_scaled, y_smt)
    xg.score(x_test_scaled, y_test)
    print('LR:',lr, 'Train_Score:',xg.score(x_train_scaled, y_smt),'CV:',np.mean(cross_val_score
```

Out[156]:
```
"for lr in [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10, 0.11, 0.12, 0.13, 0.14,
0.15, 0.16, 0.17, 0.18, 0.19, 0.20, 0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.30,
0.50, 0.75]:\n    xg = xgb.XGBClassifier(learning_rate=lr, n_estimators=100, verbosity=0)\n    x
g.fit(x_train_scaled, y_smt)\n    xg.score(x_test_scaled, y_test)\n    print('LR:',lr, 'Train_Sc
ore:',xg.score(x_train_scaled, y_smt),'CV:',np.mean(cross_val_score(xg, x_train_scaled, y_smt, c
v=10)))"
```

In [163…
```
xg = xgb.XGBClassifier(learning_rate=0.5, n_estimators=100, verbosity=0)
xg.fit(x_train_scaled, y_smt)
g = roc_auc_score(y_test, xg.predict_proba(x_test_imp_scaled)[:,1])
```

In [159…
```
g
```

Out[159]:
```
0.9219120381372278
```

In [164…
```
from itertools import combinations
comb_1 = list(combinations(col[:-2], 1))
[comb_1[0]]
```

Out[164]:
```
[('age',)]
```

In [165…
```
len(col[:-2])+1
```

Out[165]:
```
11
```

In [166…
```
auc = []
for i in comb_1:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc.append(g)
```

In [173…
```
comb_2 = list(combinations(col[:-2],2))
auc_2 = []
for i in comb_2:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc_2.append(g)
```

In [174…
```
comb_3 = list(combinations(col[:-2],3))
auc_3 = []
for i in comb_3:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
```

```
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc_3.append(g)
```

In [175…
```
comb_4 = list(combinations(col[:-2],4))
auc_4 = []
for i in comb_4:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc_4.append(g)
```

In [176…
```
comb_5 = list(combinations(col[:-2],5))
auc_5 = []
for i in comb_5:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc_5.append(g)
```

In [177…
```
comb_6 = list(combinations(col[:-2],6))
auc_6 = []
for i in comb_6:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc_6.append(g)
```

In [187…
```
comb_7 = list(combinations(col[:-2],7))
auc_7 = []
for i in comb_7:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc_7.append(g)
```

In [191…
```
comb_8 = list(combinations(col[:-2],8))
auc_8 = []
for i in comb_8:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc_8.append(g)
```

In [192…
```
comb_9 = list(combinations(col[:-2],9))
auc_9 = []
for i in comb_9:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc_9.append(g)
```

In [193…
```
comb_10 = list(combinations(col[:-2],10))
auc_10 = []
for i in comb_10:
    x = data.loc[:, i].values
    y = data.loc[:, col[-1]].values
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
```

```
    smt = SMOTEENN(sampling_strategy='all')
    x_smt, y_smt = smt.fit_resample(x_train, y_train)
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_smt)
    x_test_scaled = scaler.transform(x_test)
    import xgboost as xgb
    xgg = xgb.XGBClassifier(learning_rate=0.75, n_estimators=100, verbosity=0)
    xgg.fit(x_train_scaled, y_smt)
    g = roc_auc_score(y_test, xgg.predict_proba(x_test_scaled)[:,1])
    auc_10.append(g)
```

In [194…
```
m = max(auc_7)
i = auc_7.index(m)
be_ft = comb_7[i]
be_ft
```

Out[194]:
```
('age', 'call_type', 'day', 'mon', 'dur', 'num_calls', 'prev_outcome')
```

In [195…
```
m
```

Out[195]:
```
0.9171218204840541
```

In [ ]: