Prodigy Infotech Task 3

Name : Pavan yadav

Task : Build decision tree classifier

In [ ]:
```
# import all library to requreted
import numpy as np import pandas
as pd
from matplotlib import pyplot as plt import
seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier from
sklearn import tree from sklearn.tree import
plot_tree
from sklearn.ensemble import RandomForestClassifier from
sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import confusion_matrix,classification_report,accuracy_
import warnings warnings.filterwarnings('ignore')
```

In [4]:
```
data=pd.read_csv('dataset3.csv')
```

In [5]:
```
data.head()
```

Out[5]:

	age	job	marital	education	default	balance	housing	loan	contact	day	mon
0	58	management m	married	tertiary	no	2143	yes	no	unknown	5	
1	44	technician 5 m	single	secondary	no	29	yes	no	unknown		
2	33	entrepreneur 5 m	married	secondary	no	2	yes	yes	unknown		
3	47	blue-collar 5 m	married	unknown	no	1506	yes	no	unknown		
4	33	unknown 5 m	single	unknown	no	1	no	no	unknown		

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

Out[6]:
```
(45211, 17)
<class 'pandas.core.frame.DataFrame'>
```

In [7]:
```
data.info()
```

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

In [8]:
```
data.describe()
```

Out[8]:

	age	balance	day	duration	campaign	pdays
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000

Out[9]:

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
```

In [9]:
```
data.isnull().sum()
```

```
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

In [10]:
```
data.groupby('y').mean()
```

Out[10]:

y	age	balance	day	duration	campaign	pdays	previous
no	40.838986	1303.714969	15.892290	221.182806	2.846350	36.421372	0.502154
yes	41.670070	1804.267915	15.158253	537.294574	2.141047	68.702968	1.170354

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

Out[11]:
```
no     39922
yes     5289
Name: y, dtype: int64
```

In [ ]:

In [12]:
```
# checking no of parcentage yes and no
countyes=len(data[data.y=='yes'])
countno= len(data[data.y=='no'])
print(f'parcentage of yes=',countyes/len(data.y)*100)
print(f'parcentage of no=',countno/len(data.y)*100)
```

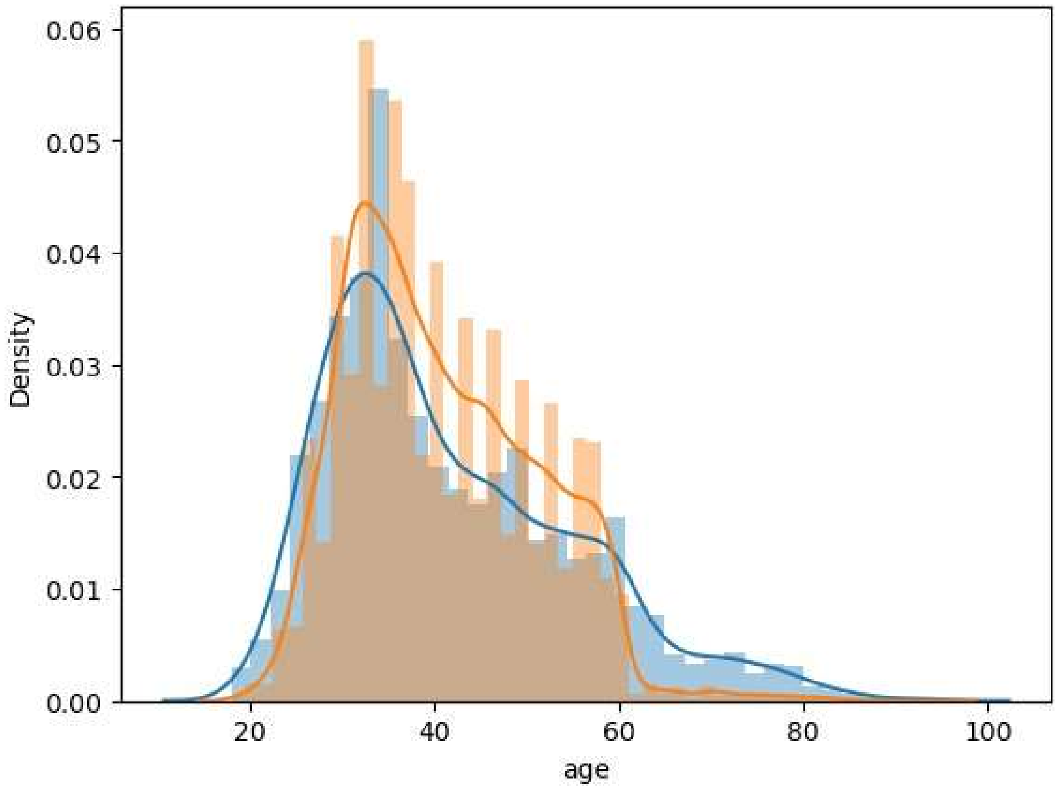
```
parcentage of yes= 11.698480458295547
parcentage of no= 88.30151954170445
```

In [13]:
```
# Replacing categorical columns
data.replace({'y':{'no':0,'yes':1}},inplace=True)
```

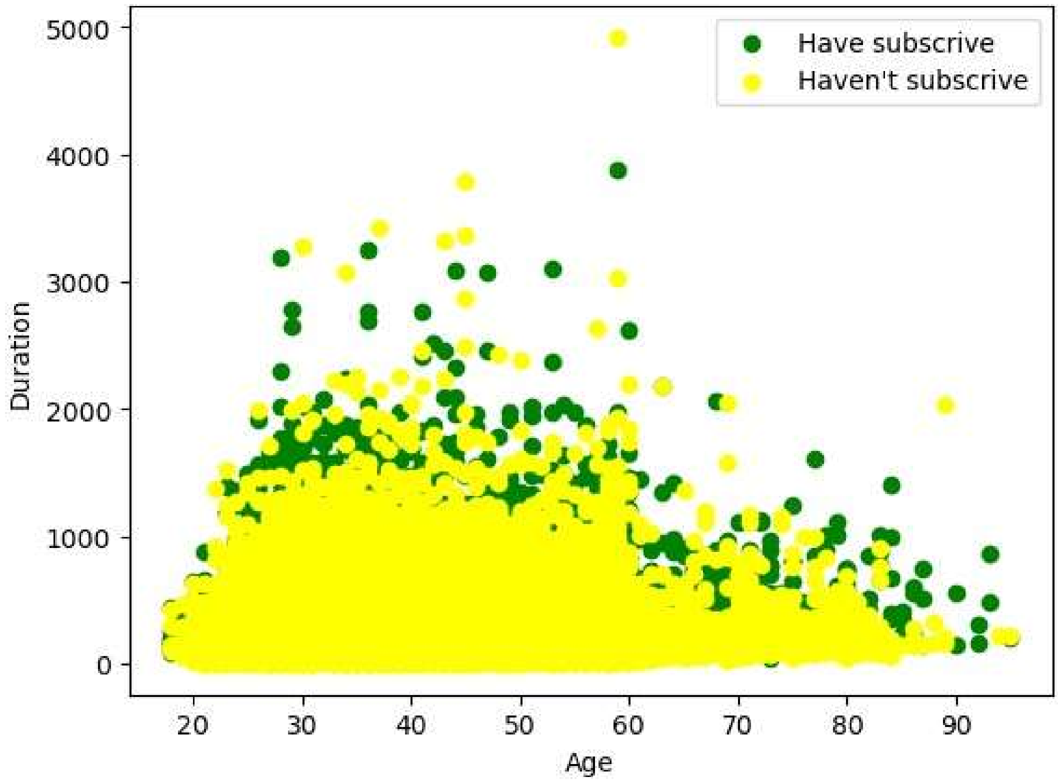
In [14]:
```
sns.distplot(data['age'][data['y']==1])
sns.distplot(data['age'][data['y']==0])
```

Out[14]: <Axes: xlabel='age', ylabel='Density'>

In [ ]:



In [15]:
```
plt.scatter(x=data.age[data.y==1],y=data.duration[data.y==1],c='green')
plt.scatter(x=data.age[data.y==0],y=data.duration[data.y==0],c='yellow')
plt.legend(['Have subscrive' , "Haven't subscrive"])
plt.xlabel('Age')
plt.ylabel('Duration')
plt.show()
```



In [16]:
```
# checking parcentage of people y in age
data.groupby(['age'])['y'].mean()
```

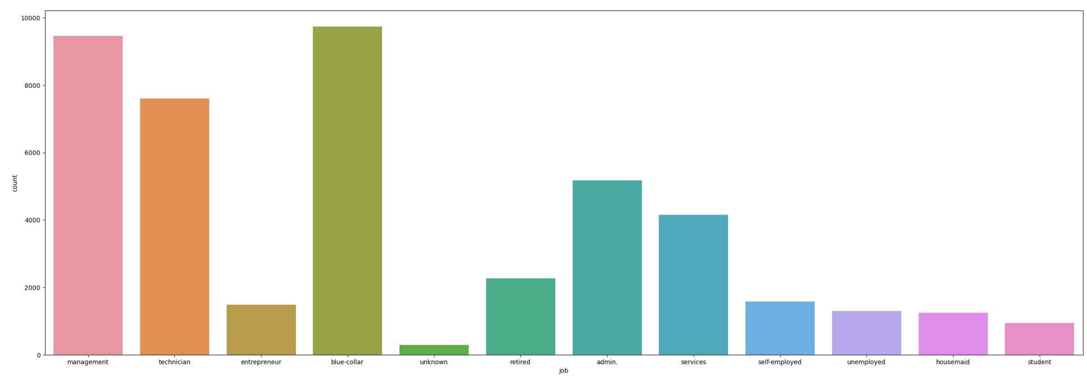
Out[16]:
```
age
18    0.583333
19    0.314286
20    0.300000
21    0.278481
22    0.310078         ...
90    1.000000
92    1.000000
93    1.000000
94    0.000000
95    0.500000
Name: y, Length: 77, dtype: float64
```

In [17]:
```
data['job'].value_counts()
```

Out[17]:
```
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264 self-
employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student           938
unknown           288
Name: job, dtype: int64
```

In [18]:
```
# making a count plot for job column
plt.figure(figsize=(30,10))
sns.countplot(x="job",data=data)
```

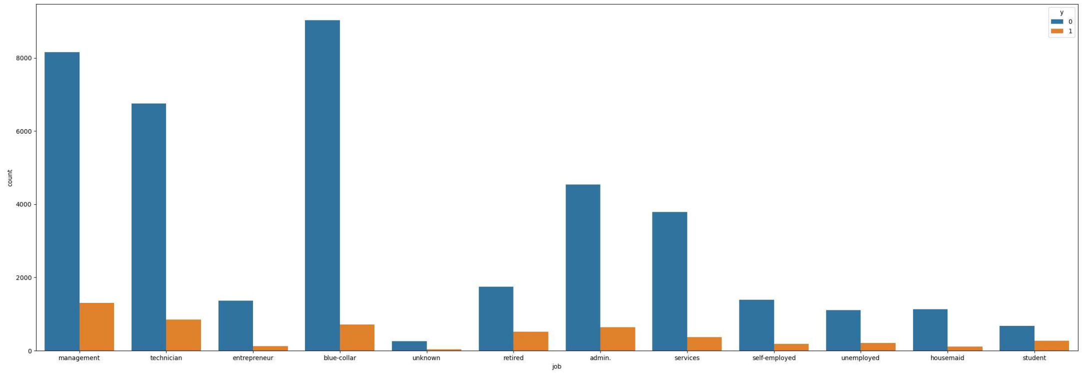
Out[18]: <Axes: xlabel='job', ylabel='count'>



In [19]:
```
# no of y job base
plt.figure(figsize=(30,10))
sns.countplot(x='job',hue='y',data=data
)
```

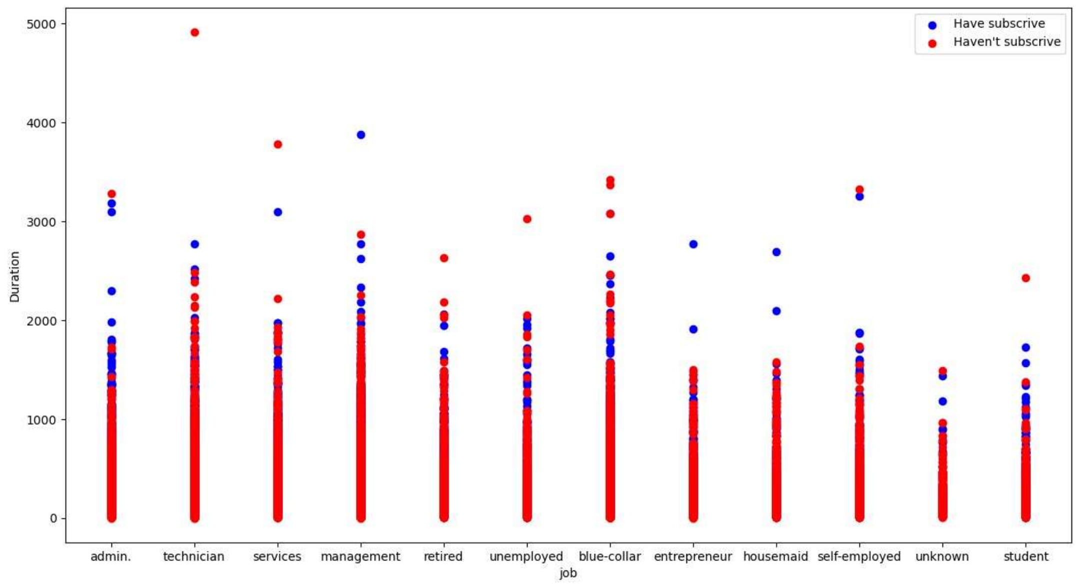
Out[19]: <Axes: xlabel='job', ylabel='count'>

In



[20]:

```
plt.figure(figsize=(15,8))
plt.scatter(x=data.job[data.y==1],y=data.duration[data.y==1],c='blue')
plt.scatter(x=data.job[data.y==0],y=data.duration[data.y==0],c='red')
plt.legend(['Have subscrive' , "Haven't subscrive"])
plt.xlabel('job')
plt.ylabel('Duration')
plt.show()
```



In [21]:

```
# checking parcentage of people y in job
data.groupby(['job'])['y'].mean()
```

Out[21]:

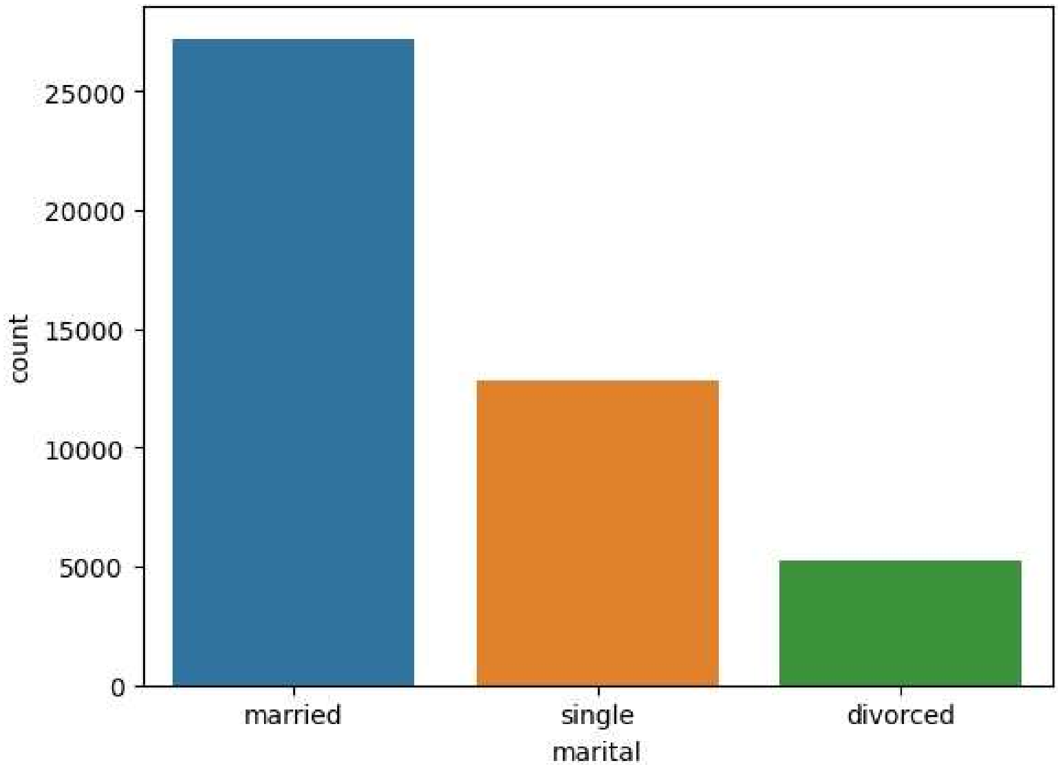
```
job
admin.           0.122027
blue-collar      0.072750
entrepreneur     0.082717
housemaid        0.087903
management       0.137556
retired          0.227915
```

In

```
self-employed    0.118429
services         0.088830
student          0.286780
technician       0.110570
unemployed       0.155027
unknown          0.118056
Name: y, dtype: float64
```

In [22]:
```
# making a count plot for marital column
sns.countplot(x="marital",data=data)
```

Out[22]: <Axes: xlabel='marital', ylabel='count'>

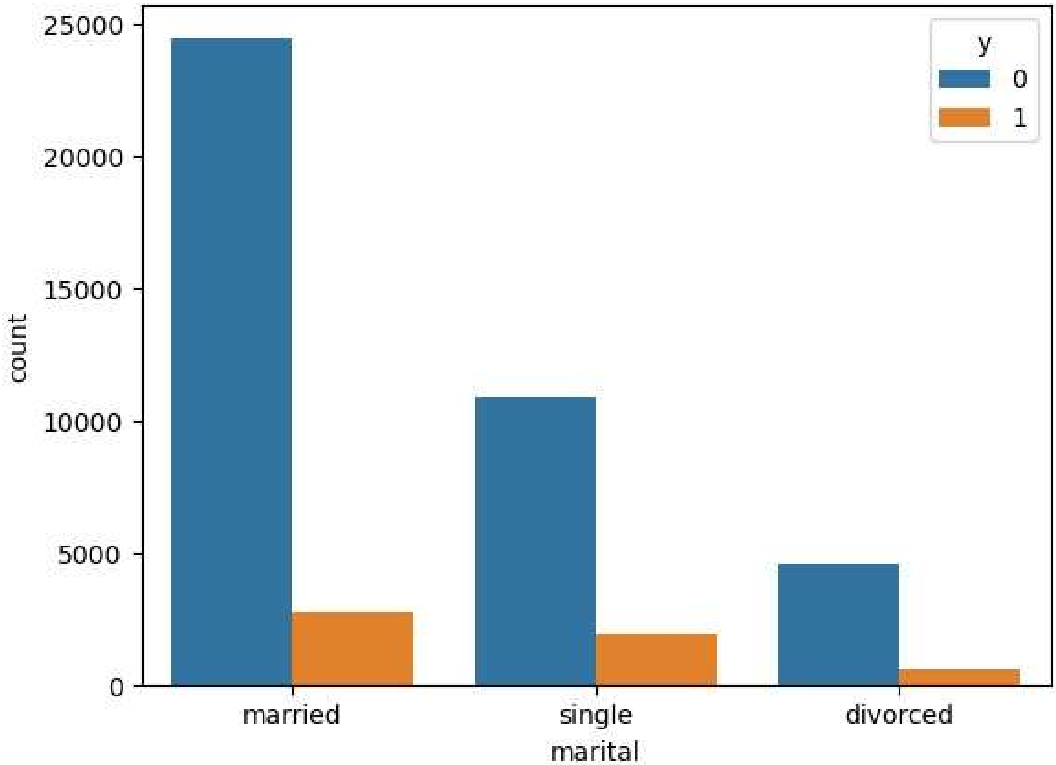


In [23]:
```
# no of y marital base
```

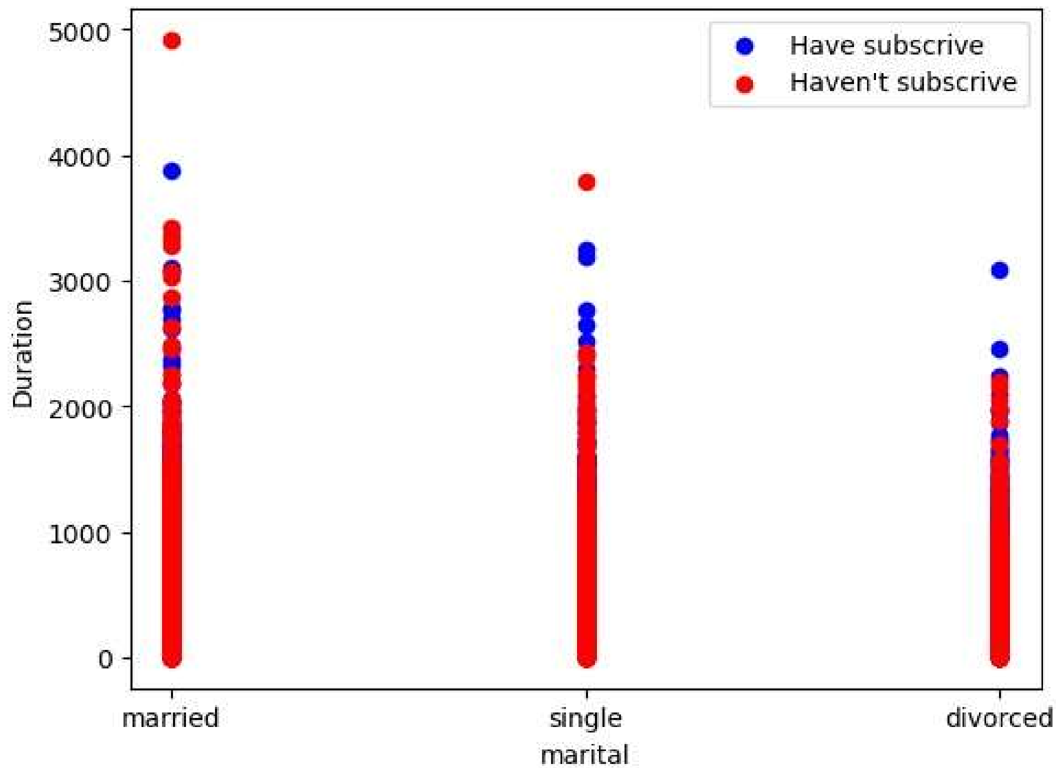
```
sns.countplot(x='marital',hue='y',data=data)
```

Out[23]: <Axes: xlabel='marital', ylabel='count'>

In


y
0
1
25000
20000
15000
10000
5000
0
count
married
single
divorced
marital

In [24]:
```
# plt.figure(figsize=(15,8))
plt.scatter(x=data.marital[data.y==1],y=data.duration[data.y==1],c='blue')
plt.scatter(x=data.marital[data.y==0],y=data.duration[data.y==0],c='red')
plt.legend(['Have subscrive' , "Haven't subscrive"])
plt.xlabel('marital')
plt.ylabel('Duration')
plt.show()
```



In [25]:
```
# checking percentage of people y in marital
data.groupby(['marital'])['y'].mean()
```

Out[25]:
```
marital
divorced    0.119455
married        0.101235
single         0.149492
Name: y, dtype: float64
```

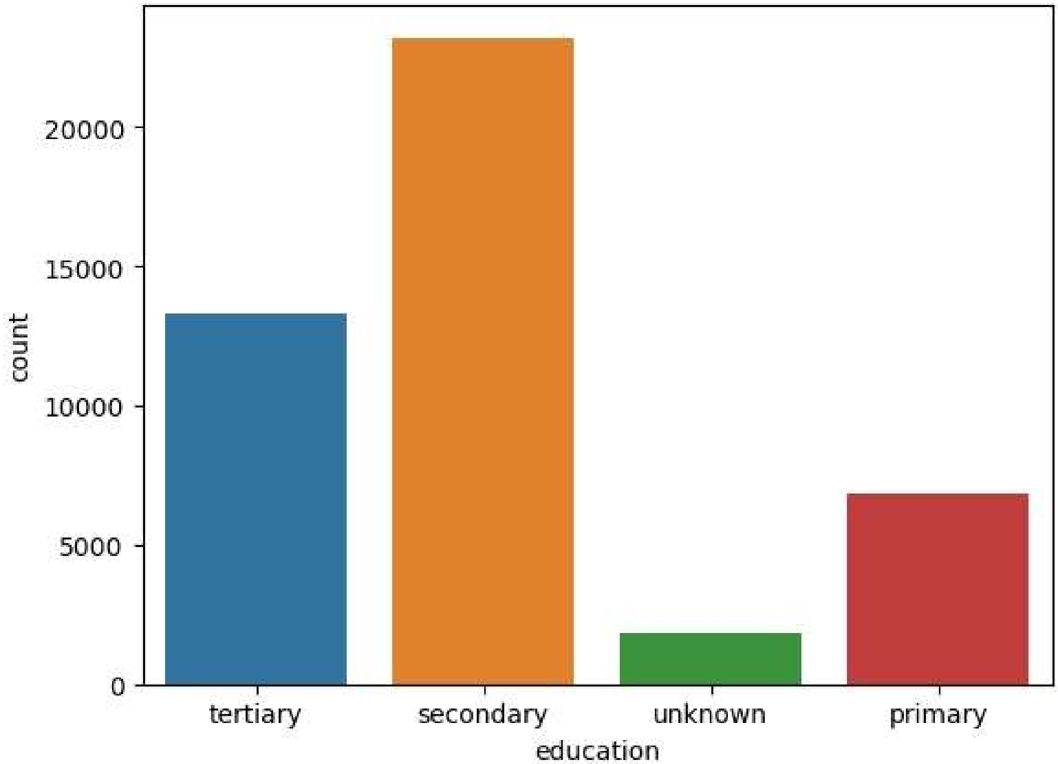
In [26]:
```
data['education'].value_counts()
```

Out[26]:
```
secondary    23202
tertiary     13301
primary       6851
unknown       1857
Name: education, dtype: int64
```

[27]:
```
# making a count plot for Y column
sns.countplot(x="education",data=data)
```

In

Out[27]: <Axes: xlabel='education', ylabel='count'>

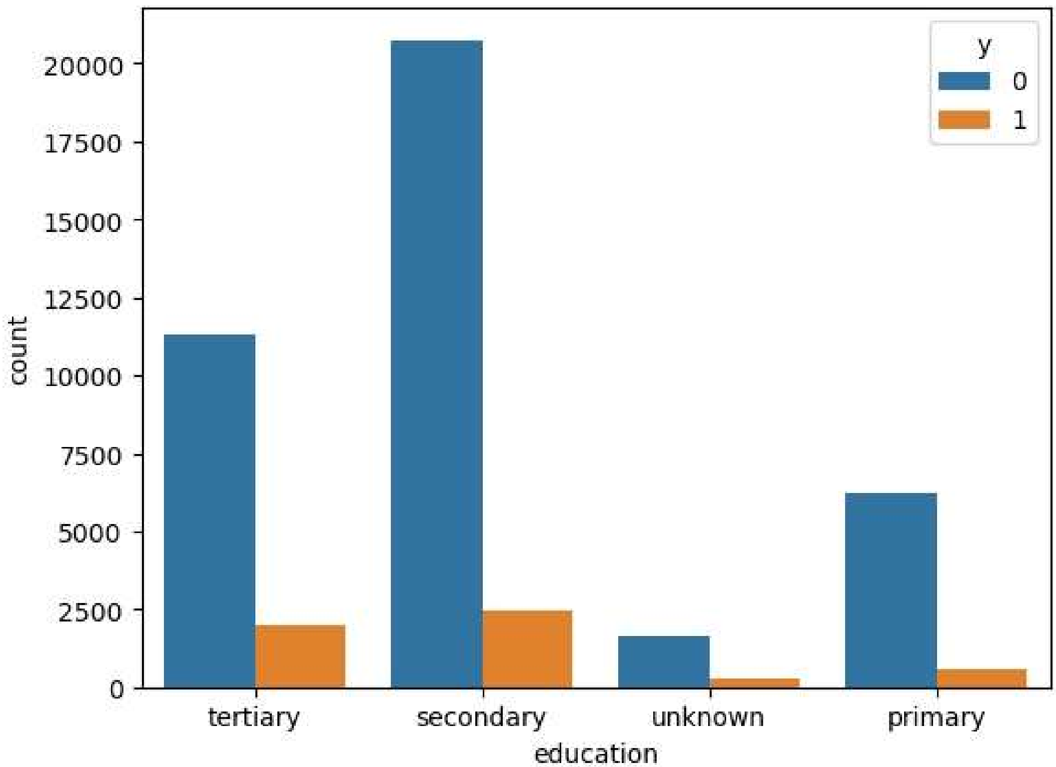


In [28]:
```
# no of y education base
```

```
sns.countplot(x='education',hue='y',data=data)
```

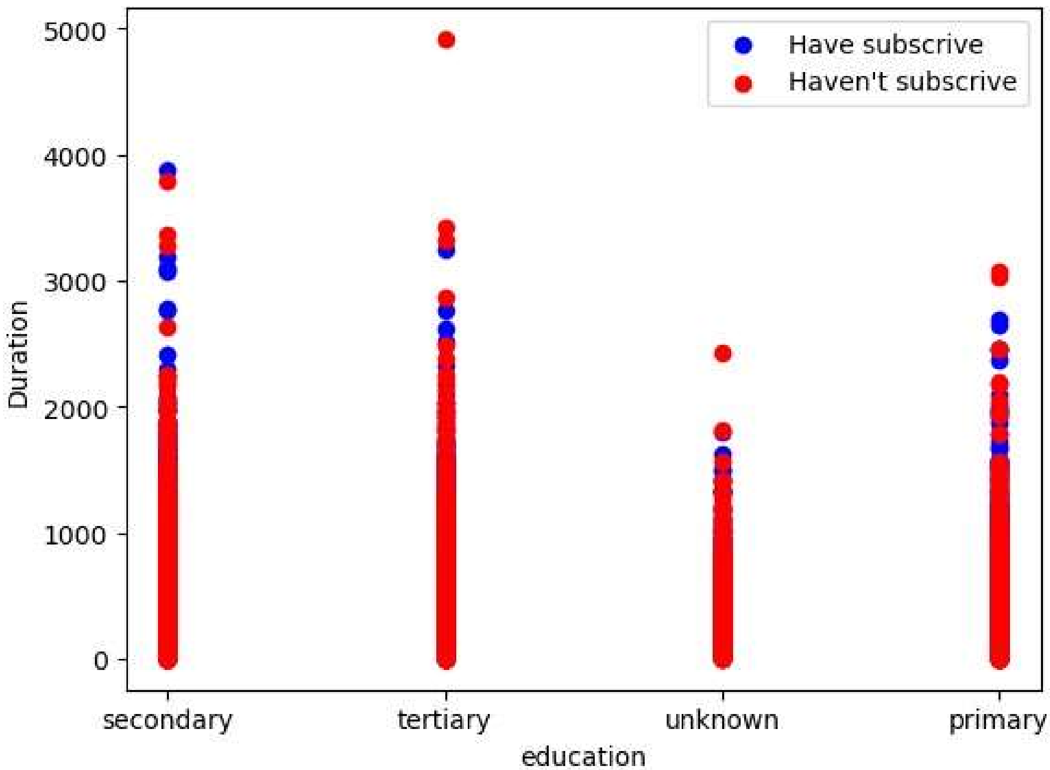
Out[28]: <Axes: xlabel='education', ylabel='count'>

In


y
0
1
20000
17500
15000
12500
10000
7500
5000
2500
0
count
tertiary
secondary
unknown
primary
education

In [29]:

```
plt.scatter(x=data.education[data.y==1],y=data.duration[data.y==1],c='blue')
plt.scatter(x=data.education[data.y==0],y=data.duration[data.y==0],c='red')
plt.legend(['Have subscrive' , "Haven't subscrive"]) plt.xlabel('education')
plt.ylabel('Duration')
plt.show()
```



In [30]:

```
# checking percentage of people y in education
data.groupby(['education'])['y'].mean()
```

Out[30]:

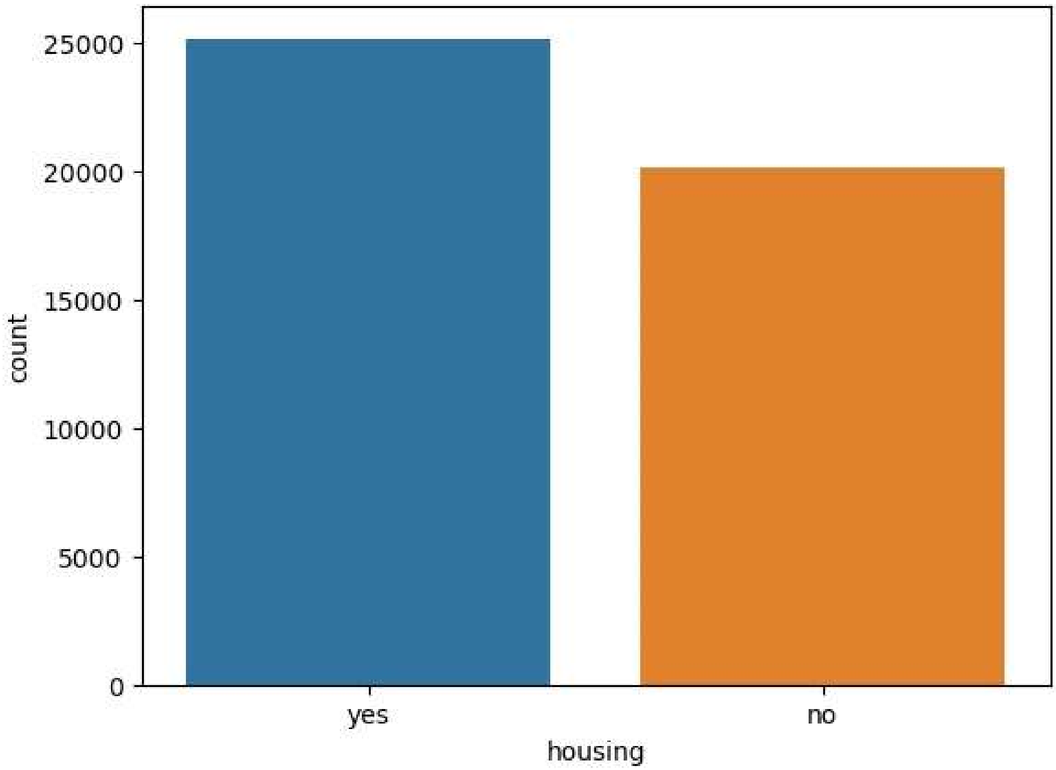
```
education
primary      0.086265
secondary    0.105594
tertiary     0.150064
unknown      0.135703
Name: y, dtype: float64
```

In [31]:

```
# making a count plot for housing column
sns.countplot(x="housing", data=data)
```

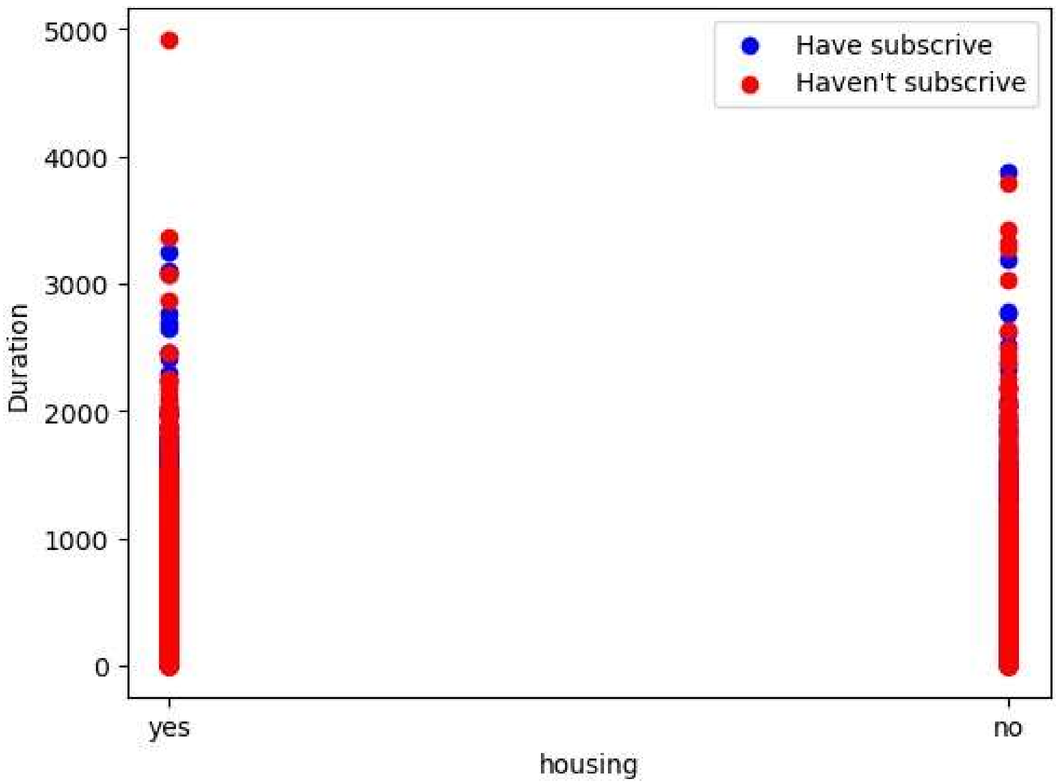
Out[31]: <Axes: xlabel='housing', ylabel='count'>

In



In [32]:

```
plt.scatter(x=data.housing[data.y==1],y=data.duration[data.y==1],c='blue')
plt.scatter(x=data.housing[data.y==0],y=data.duration[data.y==0],c='red')
plt.legend(['Have subscrive' , "Haven't subscrive"]) plt.xlabel('housing')
plt.ylabel('Duration')
plt.show()
```



In [33]:

```
# checking percentage of people y in housing
data.groupby(['housing'])['y'].mean()
```

Out[33]:

```
housing
no     0.167024
yes    0.077000 Name: y,
dtype: float64
```

In [34]:

```
data.columns
```

Out[34]:

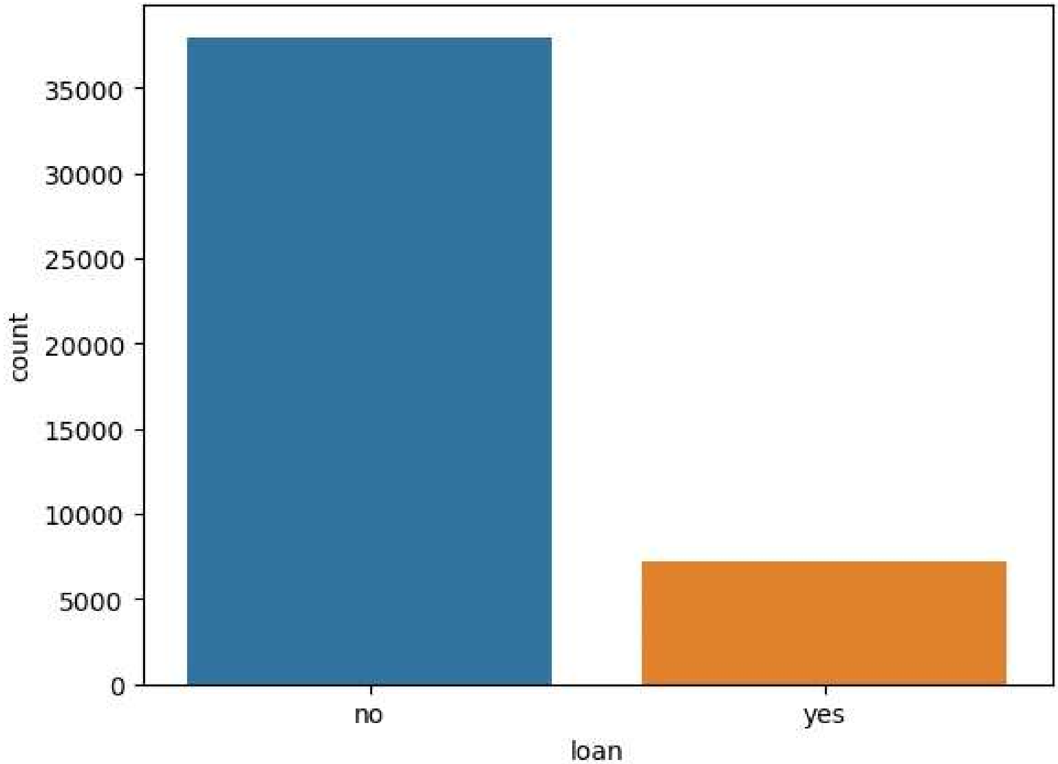
```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housin
g',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous',         'poutcome',         'y'],
 dtype='object')
```

[35]:

```
# making a count plot for loan column sns.countplot(x="loan",
data =data)
```

In

Out[35]: <Axes: xlabel='loan', ylabel='count'>

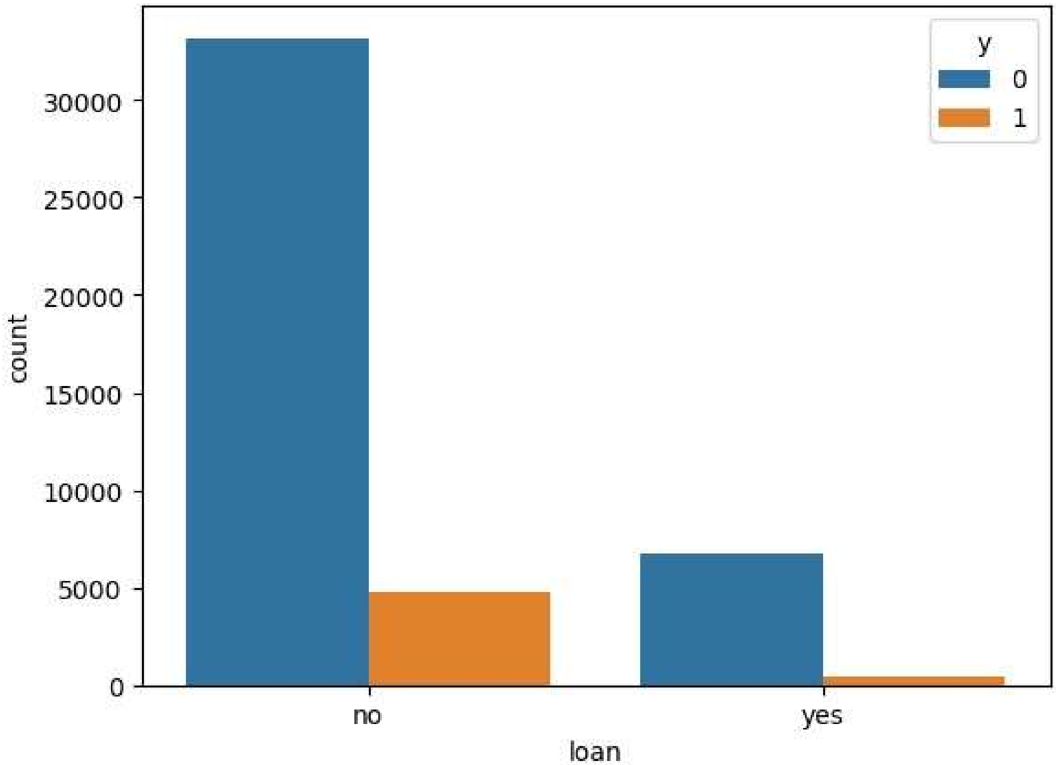


In [36]:
```
# no of y loan base
```

```
sns.countplot(x='loan',hue='y',data=data)
```

Out[36]: <Axes: xlabel='loan', ylabel='count'>

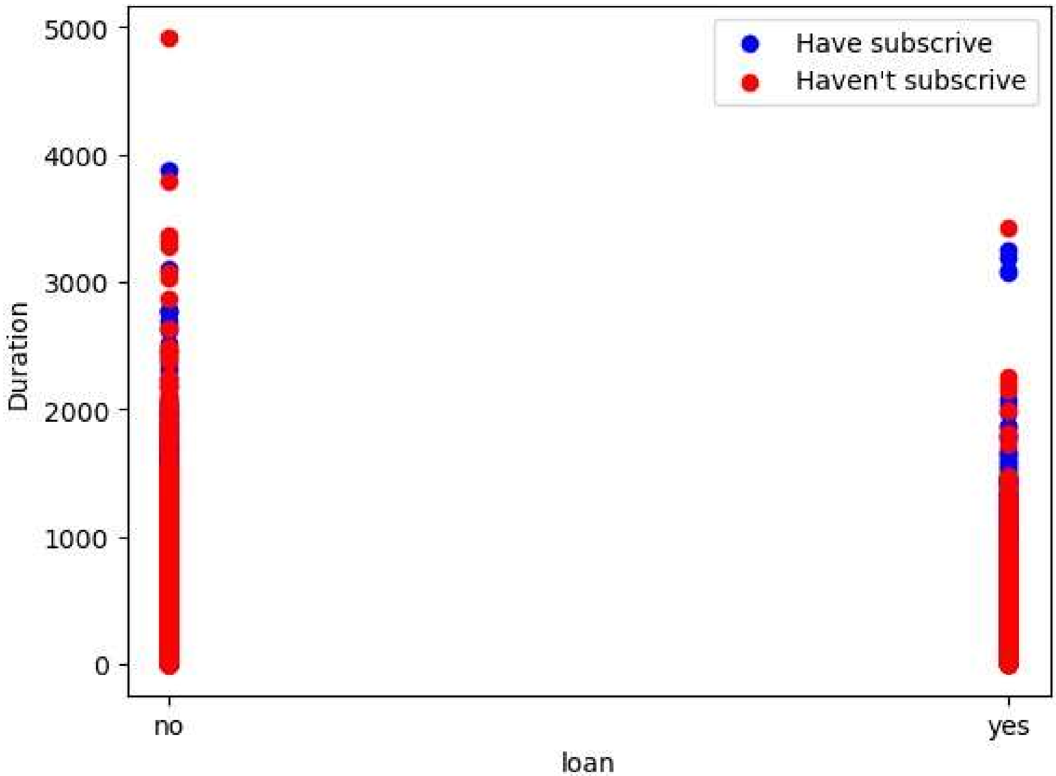
In



[37]:
```
plt.scatter(x=data.loan[data.y==1],y=data.duration[data.y==1],c='blue')
plt.scatter(x=data.loan[data.y==0],y=data.duration[data.y==0],c='red')
plt.legend(['Have subscrive' , "Haven't subscrive"]) plt.xlabel('loan')
```

In

```
plt.ylabel('Duration')
plt.show()
```

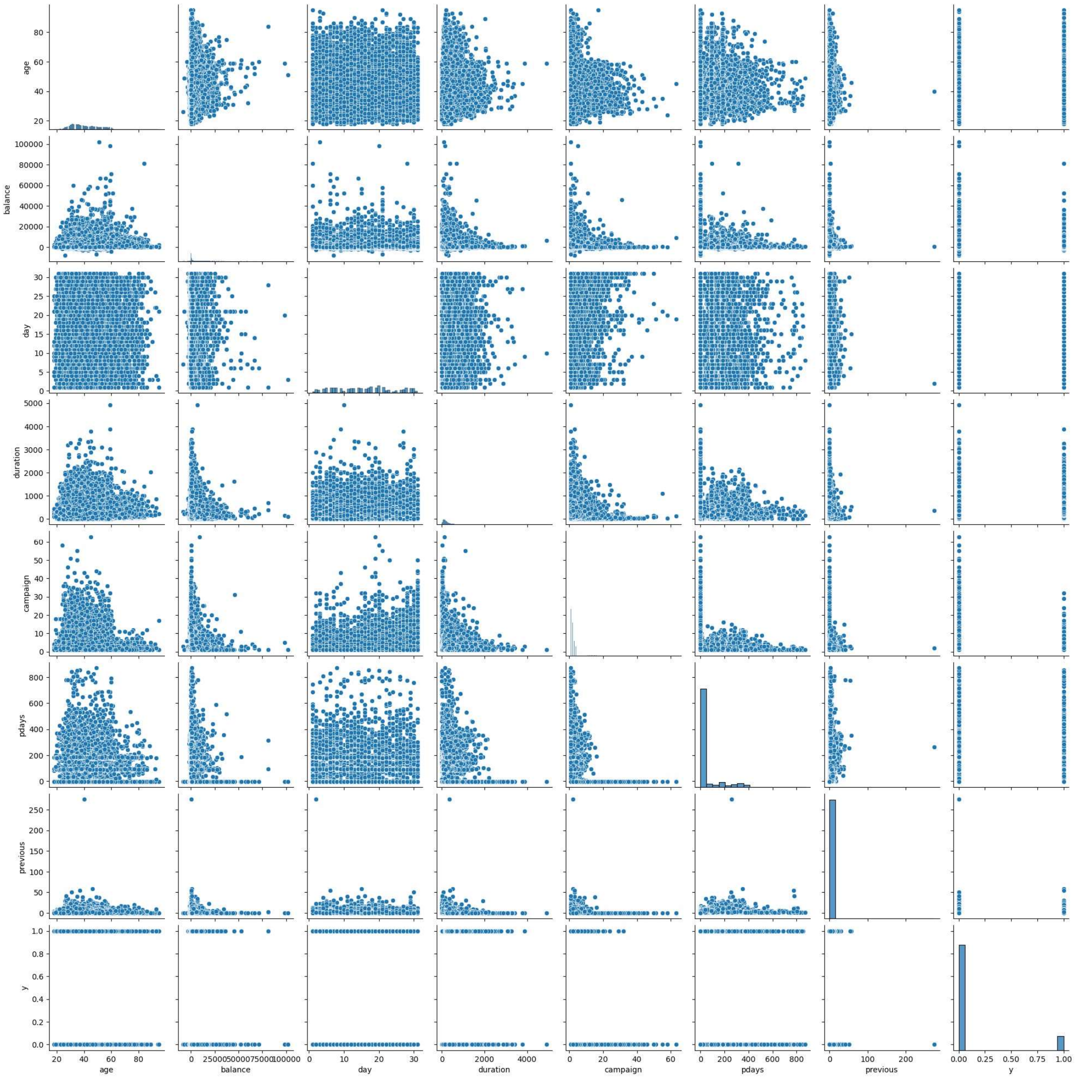


In [38]:
```
# checking percentage of people y in loan
data.groupby(['loan'])['y'].mean()
```

Out[38]:
```
loan
no     0.126557
yes    0.066814
Name: y, dtype: float64
```

In [39]:
```
sns.pairplot(data=data)
```

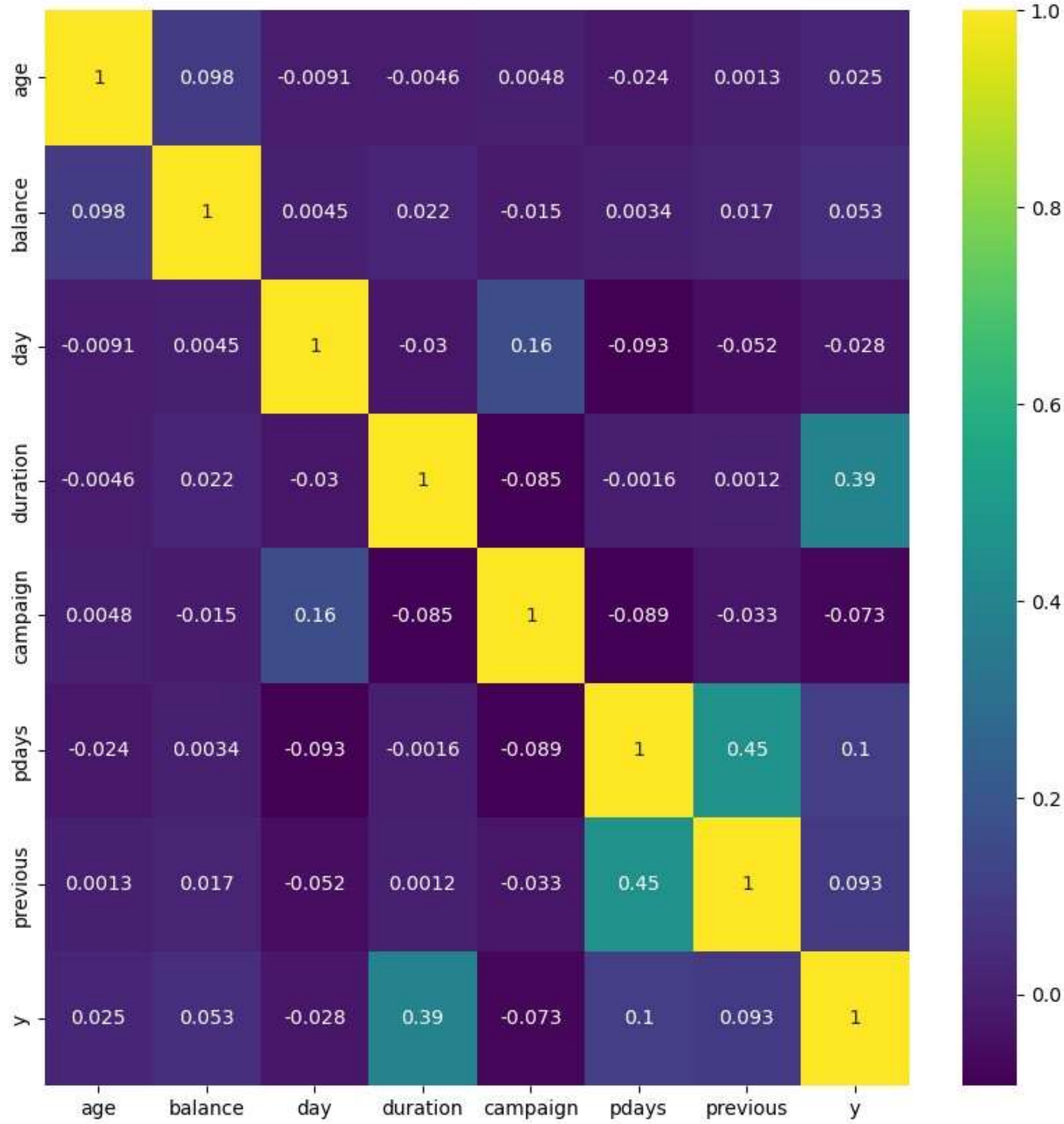
Out[39]: <seaborn.axisgrid.PairGrid at 0x23d1b151250>



[40]:
```
plt.figure(figsize=(10,10))
sns.heatmap(data=data.corr(),annot=True,cmap='viridis')
```

Out[40]: <Axes: >

In



In [41]:
```
data.corr()
```

Out[41]:

	age	balance	day	duration	campaign	pdays	previous	
age	1.000000	0.097783	-0.009120	-0.004648	0.004760	-0.023758	0.001288	0.02515
balance	0.097783	1.000000	0.004503	0.021560	-0.014578	0.003435	0.016674	0.05283
day	-0.009120	0.004503	1.000000	-0.030206	0.162490	-0.093044	-0.051710	-0.02834
duration	-0.004648	0.021560	-0.030206	1.000000	-0.084570	-0.001565	0.001203	0.39452
campaign	0.004760	-0.014578	0.162490	-0.084570	1.000000	-0.088628	-0.032855	-0.07317
pdays	-0.023758	0.003435	-0.093044	-0.001565	-0.088628	1.000000	0.454820	0.10362
previous	0.001288	0.016674	-0.051710	0.001203	-0.032855	0.454820	1.000000	0.09323
y	0.025155	0.052838	-0.028348	0.394521	-0.073172	0.103621	0.093236	1.00000

[42]:
```
# Encoding categorical columns
data['marital'].value_counts()
```

In
Out[42]:
```
married     27214 single
12790 divorced     5207
Name: marital, dtype: int64
```

In [43]:
```
# Encoding categorical columns
data['education'].value_counts()
```

Out[43]:
```
secondary    23202 tertiary
13301 primary       6851
unknown       1857 Name:
education, dtype: int64
```

In [44]:
```
# Encoding categorical columns
data['default'].value_counts()
```

Out[44]:
```
no     44396 yes      815
Name: default, dtype: int64
```

In [45]:
```
# Encoding categorical columns
data['housing'].value_counts()
```

Out[45]:
```
yes    25130 no     20081
Name: housing, dtype: int64
```

In [46]:
```
# Encoding categorical columns
data['loan'].value_counts()
```

Out[46]:
```
no     37967 yes
7244
   Name: loan, dtype:
         int64
```

[47]:
```
 #  Replacing  all  categorical  columns
data.replace({'job':{ 'blue-collar' : 0,
'management'  : 1,
                   'technician'   : 2,
                    'admin.'      : 3,
                    'services'    : 4,
                    'retired'     : 5,
                    'self-employed':6,
                    'entrepreneur' :7,
                    'unemployed'   :8,
                    'housemaid'    :9,
                    'student'      :10,
                    'unknown'      :11, },
            'marital':{'married':0,'single':1,'divorced':2},
            'education':{'secondary':0,'tertiary':1,'primary':2,'unknown':3}
            'default':{'no':0,'yes':1},
            'housing':{'no':0,'yes':1},
            'loan':{'no':0,'yes':1}
                       },inplace=True)
```

In [48]:
```
#                 Drop                 columns
data.drop(columns=['contact','day','month','poutcome'],inplace=True)
```

In
In [49]:
```
data.sample(5)
```

Out[49]:

	age	job	marital	education	default	balance	housing	loan	duration	campaign	pd
21340	54	0	0	2	0	0	0	0	144	2	
29839	36	4	0	0	0	372	0	0	94	1	
34013	53	4	0	0	0	341	0	0	423	2	
11820	56	6	0	0	0	549	0	1	181	2	
38336	33	4	1	0	0	450	1	0	148	2	

In [50]:
```
x=data.drop(columns=['y'])
y=data.y
```

In [51]:
```
x.shape
```

Out[51]: (45211, 12)

In [52]:
```
y.shape
```

Out[52]: (45211,)

In [53]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_st
```

Using LogisticRegression

[54]:
```
Lr=LogisticRegression()
Lr.fit(x_train,y_train)
```

▾ LogisticRegression

LogisticRegression()

In [55]:
```
Lr.score(x_test,y_test)
```

Out[55]: 0.8893066460245493
Out[54]:

In [56]:
```
train_score = Lr.score(x_train,y_train)
print(train_score)

test_score = Lr.score(x_test,y_test)
print(test_score)
```

```
0.8891561601415616
0.8893066460245493
```

In

In [57]:
```
print(classification_report(y_test,Lr.predict(x_test)))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      7994
           1       0.56      0.21      0.31      1049

    accuracy                           0.89      9043
   macro avg       0.73      0.59      0.62      9043
weighted avg       0.86      0.89      0.87      9043
```

In [58]:
```
print(confusion_matrix(y_test,Lr.predict(x_test)))
```

```
[[7822  172]
 [ 829  220]]
```

Using RandomForestClassifier

In [59]:
```
Rm = RandomForestClassifier()
Rm.fit(x_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [60]:
```
Rm.score(x_test,y_test)
```

Out[60]: 0.8989273471193188

Out[59]:

[61]:
```
train_score = Rm.score(x_train,y_train)
print(train_score)

test_score = Rm.score(x_test,y_test)
print(test_score)
```

```
0.9999723512497235
0.8989273471193188
```

In [62]:
```
print(classification_report(y_test,Rm.predict(x_test)))
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.94      7994
           1       0.61      0.36      0.45      1049

    accuracy                           0.90      9043
   macro avg       0.76      0.67      0.70      9043
weighted avg       0.88      0.90      0.89      9043
```

In [63]:
```
print(confusion_matrix(y_test,Rm.predict(x_test)))
```

In

```
[[7748  246]
 [ 668  381]]
```

Using DecisionTreeClassifier

Out[64]:

In [64]:
```python
Dtc = DecisionTreeClassifier()
Dtc.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [65]:
```python
y_pred = Dtc.predict(x_test)
y_pred
```

Out[65]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

[66]:
```python
y_pred = Dtc.predict(x_test)

cm = confusion_matrix(y_test,y_pred)
print(f"Confusion Matrix =\n",cm)
print("*"*50)
ac = accuracy_score(y_test,y_pred)
print(f"Accuracy Score = {ac}")
print("*"*50)
cr = classification_report(y_test,y_pred)
print(f"Classification report= \n{cr}")
```

```
Confusion Matrix =
 [[7290  704]
 [ 626  423]]
**************************************************
Accuracy Score = 0.8529249142983523
**************************************************
Classification report=               precision    recall
 f1-score   support

           0       0.92      0.91      0.92      7994
       1       0.38      0.40      0.39      1049

    accuracy                           0.85      9043
macro avg       0.65       0.66      0.65       9043
weighted avg       0.86      0.85      0.86      9043
```

In

In [67]:
```
# Training Data Evaluation

y_pred = Dtc.predict(x_train)

cm = confusion_matrix(y_train,y_pred)
print(f"Confusion Matrix =\n",cm)
print("*"*50)
ac = accuracy_score(y_train,y_pred)
print(f"Accuracy Score = {ac}")
print("*"*50)
cr = classification_report(y_train,y_pred)
print(f"Classification report= \n{cr}")
```

```
Confusion Matrix =
 [[31928     0]
 [    0  4240]] **************************************************
Accuracy Score = 1.0
**************************************************
Classification report=               precision    recall
 f1-score   support

           0       1.00      1.00      1.00     31928
      1       1.00      1.00      1.00      4240

    accuracy                           1.00     36168
macro avg       1.00      1.00      1.00     36168
weighted avg       1.00      1.00      1.00     36168
```

Randomised Search

Out[69]:

In [68]:
```
DC= DecisionTreeClassifier(min_samples_split =  15,min_samples_leaf =  16,r
```

In [69]:
```
DC.fit(x_train,y_train)
```

```
DecisionTreeClassifier

DecisionTreeClassifier(max_depth=3, min_samples_leaf=16, min_samples_spli
t=15)
```

In [70]:
```
# Testing Data Evaluation

y_pred = DC.predict(x_test)

cm = confusion_matrix(y_test,y_pred)
print(f"Confusion Matrix =\n",cm)
print("*"*50)
ac = accuracy_score(y_test,y_pred)
print(f"Accuracy Score = {ac}")
print("*"*50)
cr = classification_report(y_test,y_pred)
print(f"Classification report= \n{cr}")
```

```
Confusion Matrix =
 [[7774  220]
 [ 792  257]]
**************************************************
Accuracy Score = 0.8880902355413026
**************************************************
Classification report=               precision    recall
 f1-score   support

           0       0.91      0.97      0.94      7994
       1       0.54      0.24      0.34      1049

    accuracy                           0.89      9043
macro avg       0.72       0.61       0.64       9043
weighted avg       0.86      0.89      0.87      9043
```

In [71]:
```
# Training Data Evaluation

y_pred = DC.predict(x_train)

cm = confusion_matrix(y_train,y_pred)
print(f"Confusion Matrix =\n",cm)
print("*"*50)
ac = accuracy_score(y_train,y_pred)
print(f"Accuracy Score = {ac}")
print("*"*50)
cr = classification_report(y_train,y_pred)
print(f"Classification report= \n{cr}")
```

```
Confusion Matrix =
 [[31119   809]
 [ 3169  1071]]
**************************************************
Accuracy Score = 0.8900132714001328
**************************************************
Classification report=               precision    recall
 f1-score   support

           0       0.91      0.97      0.94     31928
      1       0.57      0.25      0.35      4240

    accuracy                           0.89     36168
macro avg       0.74       0.61       0.64      36168
weighted avg       0.87      0.89      0.87     36168
```

In [88]:
```
plt.figure(figsize=(100,50))
plot_tree(DC,filled=True)
```

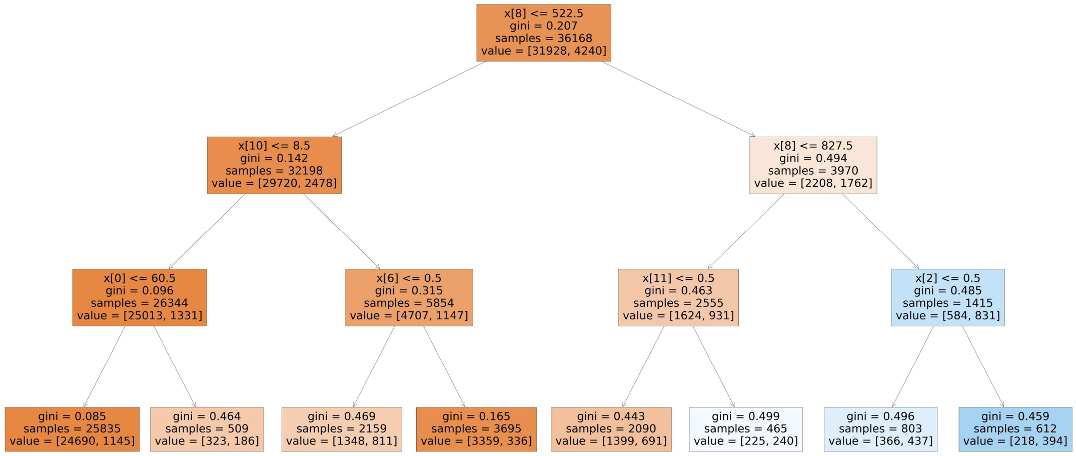
Out[88]:
```
[Text(0.5, 0.875, 'x[8] <= 522.5\ngini = 0.207\nsamples = 36168\nvalue =
[31928, 4240]'),
 Text(0.25, 0.625, 'x[10] <= 8.5\ngini = 0.142\nsamples = 32198\nvalue =
[29720, 2478]'),
 Text(0.125, 0.375, 'x[0] <= 60.5\ngini = 0.096\nsamples = 26344\nvalue =
[25013, 1331]'),
 Text(0.0625, 0.125, 'gini = 0.085\nsamples = 25835\nvalue = [24690, 114
5]'),
 Text(0.1875, 0.125, 'gini = 0.464\nsamples = 509\nvalue = [323, 186]'),
 Text(0.375, 0.375, 'x[6] <= 0.5\ngini = 0.315\nsamples = 5854\nvalue = [4
707, 1147]'),
 Text(0.3125, 0.125, 'gini = 0.469\nsamples = 2159\nvalue = [1348, 811]'),
 Text(0.4375, 0.125, 'gini = 0.165\nsamples = 3695\nvalue = [3359, 336]'),
 Text(0.75, 0.625, 'x[8] <= 827.5\ngini = 0.494\nsamples = 3970\nvalue =
[2208, 1762]'),
 Text(0.625, 0.375, 'x[11] <= 0.5\ngini = 0.463\nsamples = 2555\nvalue =
[1624, 931]'),
 Text(0.5625, 0.125, 'gini = 0.443\nsamples = 2090\nvalue = [1399, 691]'),
 Text(0.6875, 0.125, 'gini = 0.499\nsamples = 465\nvalue = [225, 240]'),
 Text(0.875, 0.375, 'x[2] <= 0.5\ngini = 0.485\nsamples = 1415\nvalue = [5
84, 831]'),
 Text(0.8125, 0.125, 'gini = 0.496\nsamples = 803\nvalue = [366, 437]'),
 Text(0.9375, 0.125, 'gini = 0.459\nsamples = 612\nvalue = [218, 394]')]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:


x[8] <= 522.5
gini = 0.207
samples = 36168
value = [31928, 4240]
x[10] <= 8.5
gini = 0.142
samples = 32198
value = [29720, 2478]
x[8] <= 827.5
gini = 0.494
samples = 3970
value = [2208, 1762]
x[0] <= 60.5
gini = 0.096
samples = 26344
value = [25013, 1331]
x[6] <= 0.5
gini = 0.315
samples = 5854
value = [4707, 1147]
x[11] <= 0.5
gini = 0.463
samples = 2555
value = [1624, 931]
x[2] <= 0.5
gini = 0.485
samples = 1415
value = [584, 831]
gini = 0.085
samples = 25835
value = [24690, 1145]
gini = 0.464
samples = 509
value = [323, 186]
gini = 0.469
samples = 2159
value = [1348, 811]
gini = 0.165
samples = 3695
value = [3359, 336]
gini = 0.443
samples = 2090
value = [1399, 691]
gini = 0.499
samples = 465
value = [225, 240]
gini = 0.496
samples = 803
value = [366, 437]
gini = 0.459
samples = 612
value = [218, 394]