Import libraries

In [2]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns


df = pd.read_csv('bank-additional-full.csv',sep=';')
term_deposits = df.copy()
# Have a grasp of how our data looks.
df.head()
```

Out[2]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaig
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	

5 rows × 21 columns

In [3]:
```
df.describe()
```

Out[3]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	4118
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-4
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-5
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-4
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-4
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-3
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-2

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

Fortunately, there are no missing values. If there were missing values we will have to fill them with the median, mean or mode. I tend to use the median but in this scenario there is no need to fill any missing values. This will definitely make our job easier!

In [6]:
```
df.nunique()
```

Out[6]:
```
age                 78
job                 12
marital              4
education            8
default              3
housing              3
loan                 3
contact              2
month               10
day_of_week          5
duration          1544
campaign            42
pdays               27
previous             8
poutcome             3
emp.var.rate        10
cons.price.idx      26
cons.conf.idx       26
euribor3m          316
nr.employed         11
y                    2
dtype: int64
```

Now,lets read the test data

In [13]:
```
test=pd.read_csv('bank-additional.csv',sep=';')
```

In [14]:
```
test.head()
```

Out[14]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	ca
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	...	
1	39	services	single	high.school	no	no	no	telephone	may	fri	...	
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	...	
3	38	services	married	basic.9y	no	unknown	unknown	telephone	jun	fri	...	
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	...	

5 rows × 21 columns

Concating train and test data

In [15]:
```
data = pd.concat([df,test])
data.replace(['basic.6y','basic.4y', 'basic.9y'], 'basic', inplace=True)
```

In [16]:
```
data.head()
```

Out[16]:

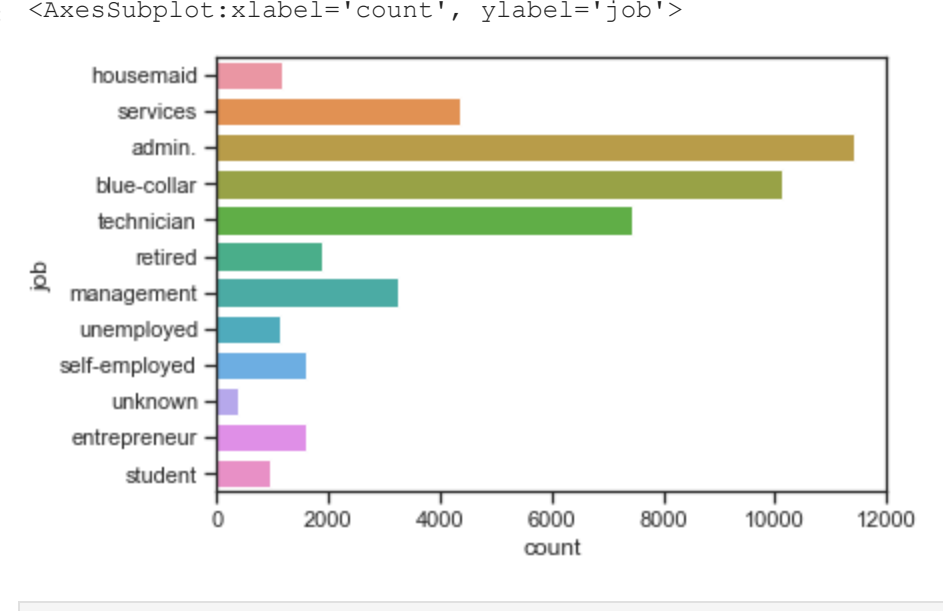
	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaig
0	56	housemaid	married	basic	no	no	no	telephone	may	mon	...	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	
3	40	admin.	married	basic	no	no	no	telephone	may	mon	...	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	

5 rows × 21 columns

Exploratory data analysis

In [17]:
```
sns.set(style="ticks", color_codes=True)
sns.countplot(y='job', data=data)
```

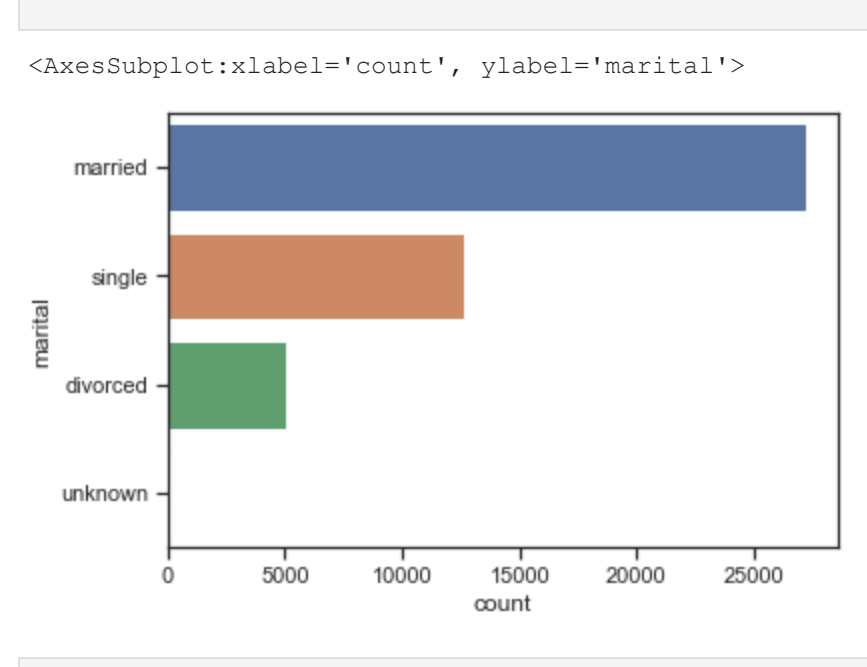
Out[17]: <AxesSubplot:xlabel='count', ylabel='job'>



In [18]:
```
data = data[data.job != 'unknown']
```

In [19]:
```
sns.countplot(y='marital', data=data)
```

Out[19]: <AxesSubplot:xlabel='count', ylabel='marital'>



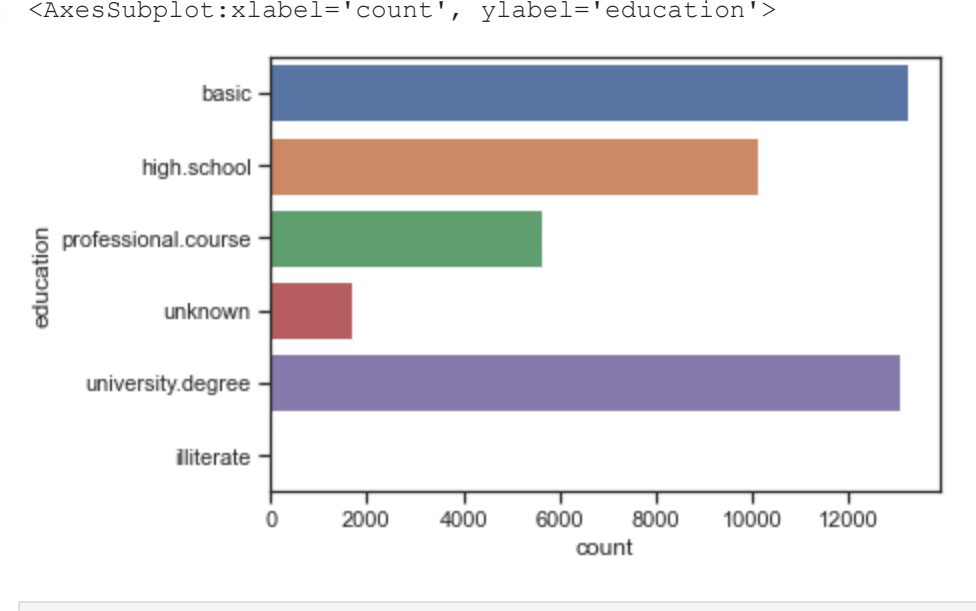
In [20]:
```
data.marital.value_counts()
```

Out[20]:
```
married     27175
single      12638
divorced     5043
unknown        82
Name: marital, dtype: int64
```

In [21]:
```
data = data[data.marital != 'unknown']
data = data[data.loan != 'unknown']
```

In [22]:
```
sns.countplot(y='education', data=data)
```

Out[22]: <AxesSubplot:xlabel='count', ylabel='education'>



In [23]:
```
data = data[data.education != 'illiterate']
```

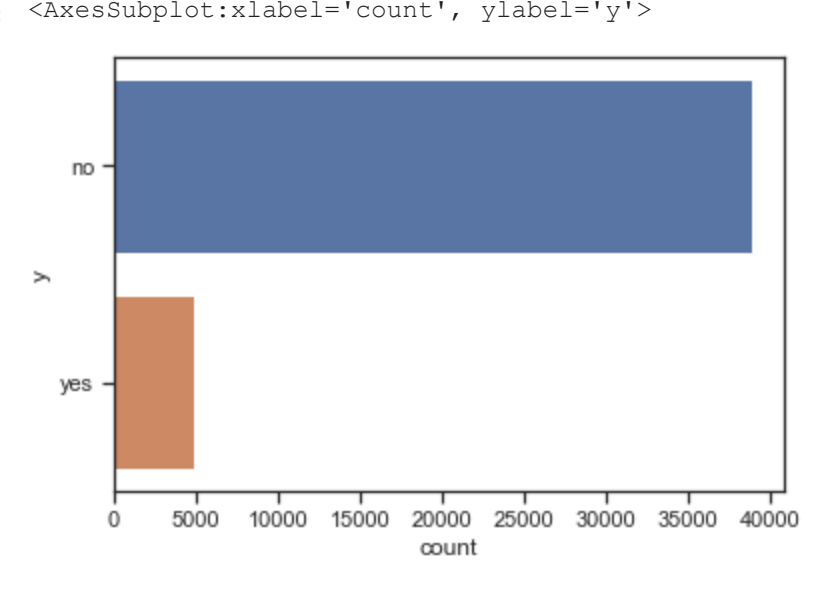
In [24]:
```
data.describe()
```

Out[24]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	con
count	43748.000000	43748.000000	43748.000000	43748.000000	43748.000000	43748.000000	43748.000000	437
mean	39.983634	258.406236	2.563660	962.479108	0.173997	0.079846	93.573810	-
std	10.392908	259.058736	2.747792	186.901953	0.498140	1.568906	0.578046	
min	17.000000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-
25%	32.000000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-
50%	38.000000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-
75%	47.000000	320.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-
max	98.000000	4918.000000	43.000000	999.000000	7.000000	1.400000	94.767000	-

In [25]:
```
sns.countplot(y='y', data=data)
```

Out[25]: <AxesSubplot:xlabel='count', ylabel='y'>



It is clear that the dataset is imblanced

Data preprocessing

Firstly,convert the categorical features into numeric using label encoding.

In label encoding in Python, we replace the categorical value with a numeric value between 0 and the number of classes minus 1. If the categorical variable value contains 5 distinct classes, we use (0, 1, 2, 3, and 4).

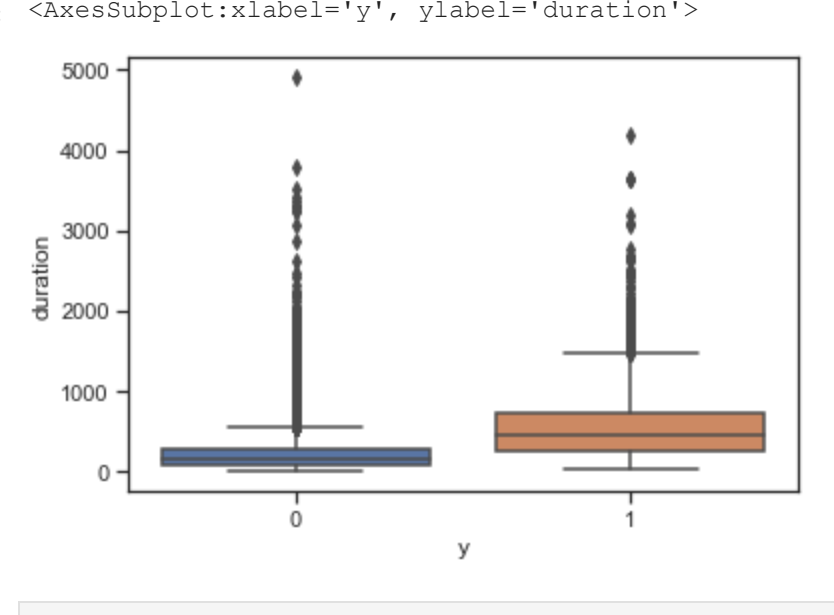
In [26]:
```
def categorize(df):
    new_df = df.copy()
    le = preprocessing.LabelEncoder()

    new_df['job'] = le.fit_transform(new_df['job'])
    new_df['marital'] = le.fit_transform(new_df['marital'])
    new_df['education'] = le.fit_transform(new_df['education'])
    new_df['default'] = le.fit_transform(new_df['default'])
    new_df['housing'] = le.fit_transform(new_df['housing'])
    new_df['month'] = le.fit_transform(new_df['month'])
    new_df['loan'] = le.fit_transform(new_df['loan'])
    new_df['contact'] = le.fit_transform(new_df['contact'])
    new_df['day_of_week'] = le.fit_transform(new_df['day_of_week'])
    new_df['poutcome'] = le.fit_transform(new_df['poutcome'])
    new_df['y'] = le.fit_transform(new_df['y'])
    return new_df
```

In [27]:
```
data = categorize(data)
#data = data.convert_objects(convert_numeric=True)
```

Checking for outliers using boxplots

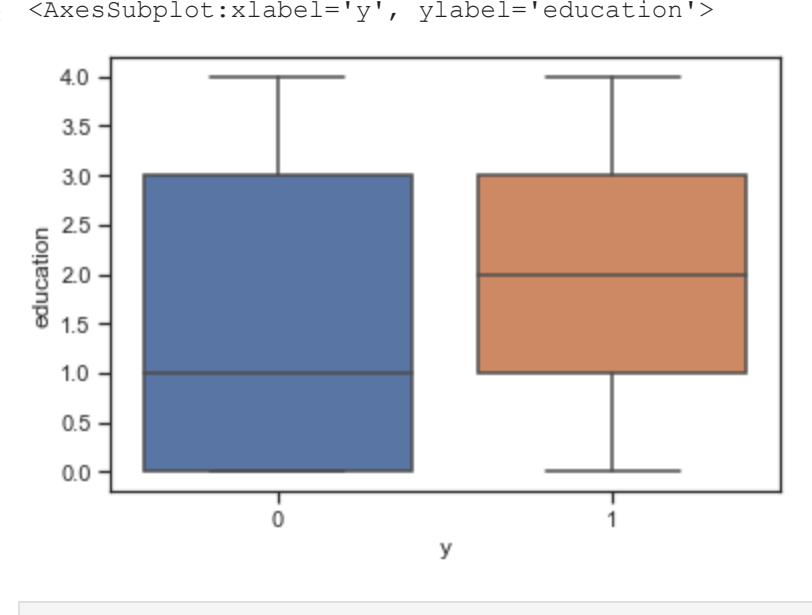
In [28]:
```
sns.boxplot(x='y', y='duration', data=data)
```

Out[28]: <AxesSubplot:xlabel='y', ylabel='duration'>



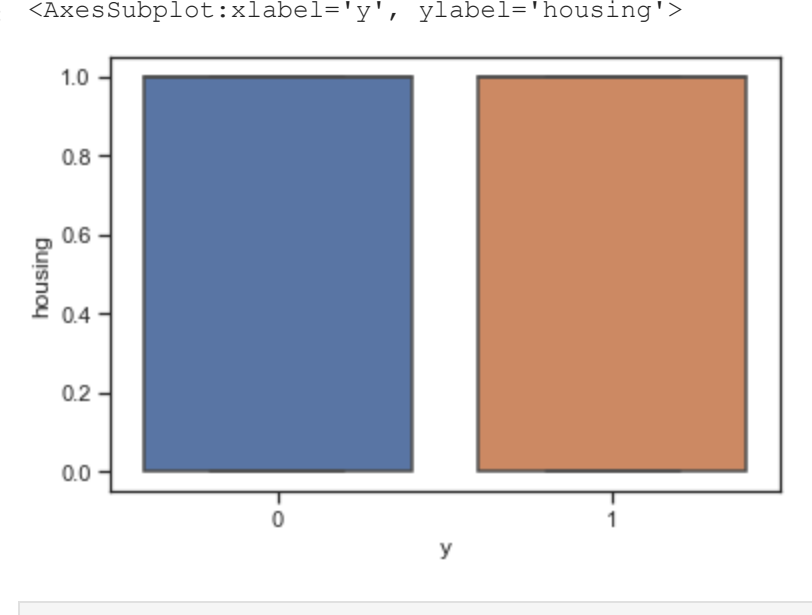
In [29]:
```
sns.boxplot(x='y', y='education', data=data)
```

Out[29]: <AxesSubplot:xlabel='y', ylabel='education'>



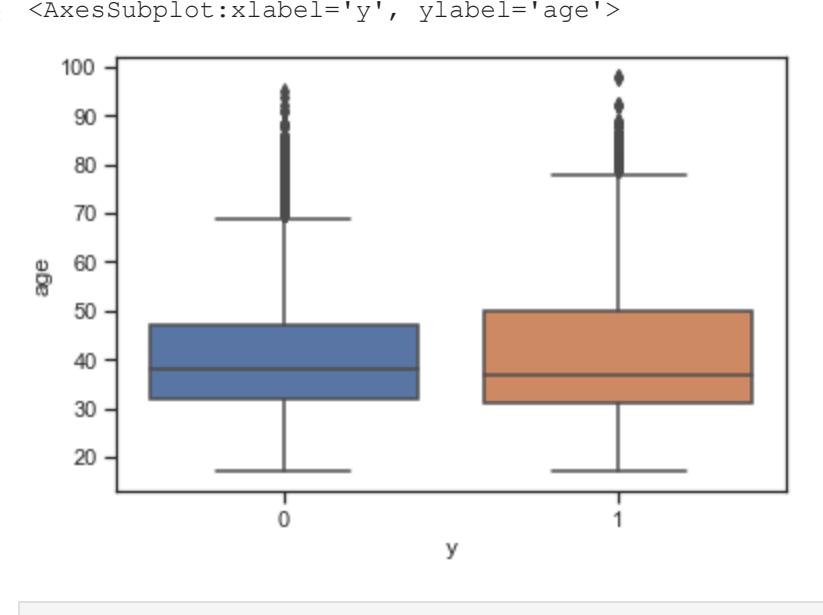
In [30]:
```
sns.boxplot(x='y', y='housing', data=data)
```

Out[30]: <AxesSubplot:xlabel='y', ylabel='housing'>



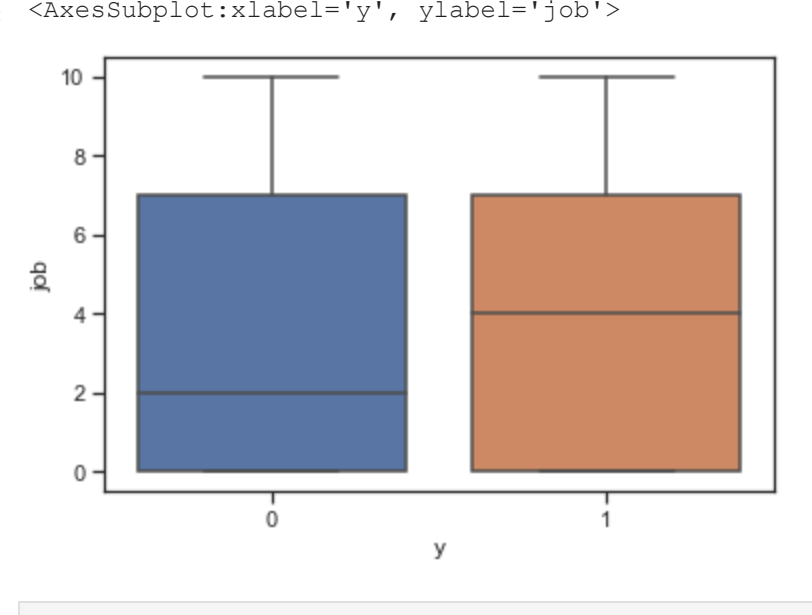
In [31]:
```
sns.boxplot(x='y', y='age', data=data)
```

Out[31]: <AxesSubplot:xlabel='y', ylabel='age'>



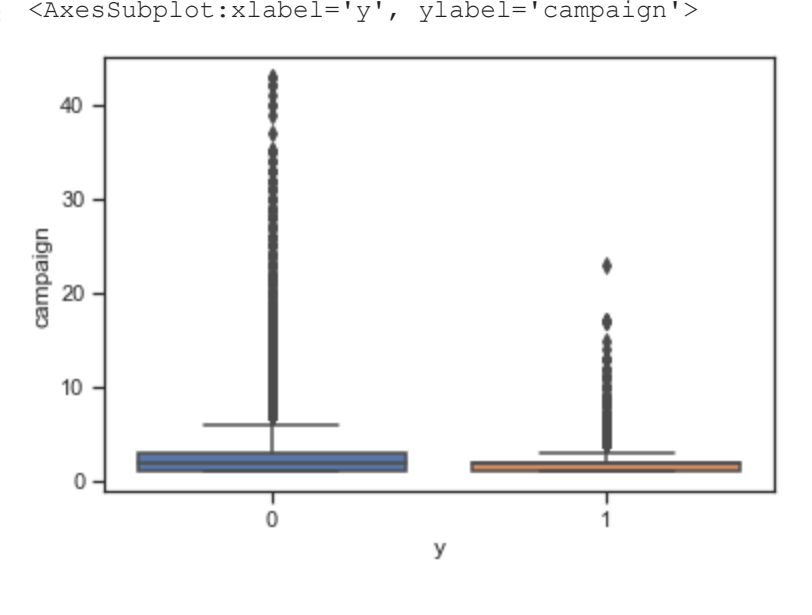
In [32]:
```
sns.boxplot(x='y', y='job', data=data)
```

Out[32]: <AxesSubplot:xlabel='y', ylabel='job'>



In [33]:
```
sns.boxplot(x='y', y='campaign', data=data )
```

Out[33]: <AxesSubplot:xlabel='y', ylabel='campaign'>



Removing outliers

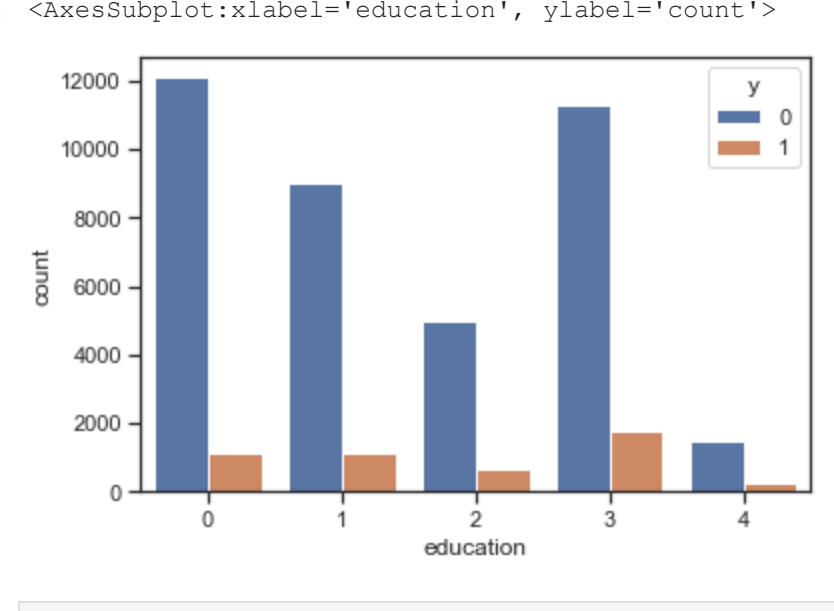
In [34]:
```
def remove_outliers(df, column , minimum, maximum):
    col_values = df[column].values
    df[column] = np.where(np.logical_or(col_values<minimum, col_values>maximum), col_v
    return df
```

In [35]:
```
min_val = data["duration"].min()
max_val = 1500
data = remove_outliers(df=data, column='duration' , minimum=min_val, maximum=max_val)

min_val = data["age"].min()
max_val = 80
data = remove_outliers(df=data, column='age' , minimum=min_val, maximum=max_val)

min_val = data["campaign"].min()
max_val = 6
data = remove_outliers(df=data, column='campaign' , minimum=min_val, maximum=max_val)
```

Dropping less meaningful columns

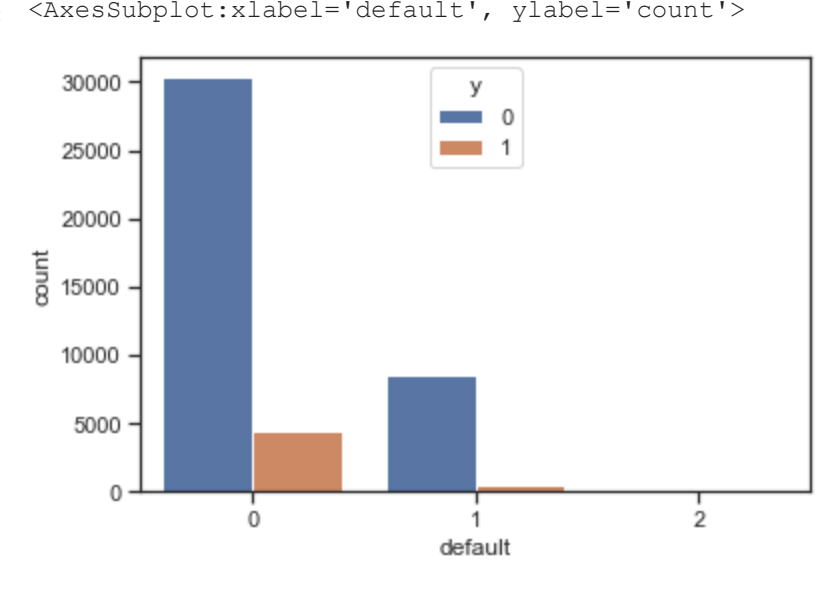
In [36]:
```
sns.countplot(x='education',hue='y',data=data)
```

Out[36]: <AxesSubplot:xlabel='education', ylabel='count'>



In [37]:
```
sns.countplot(x='default',hue='y',data=data)
```

Out[37]: <AxesSubplot:xlabel='default', ylabel='count'>

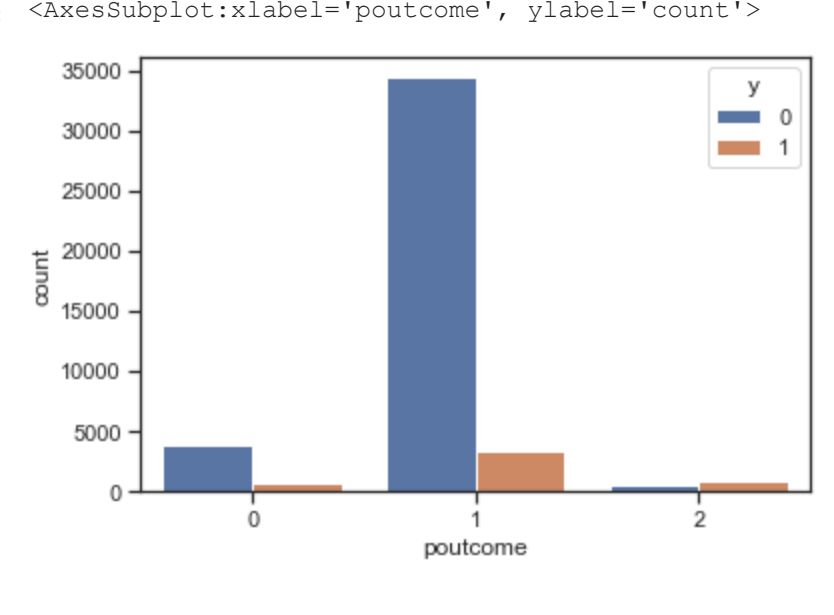


It is skewed to 0. So We can drop this

In [38]:
```
data = data.drop('default',axis=1)
```

In [39]:
```
sns.countplot(x='poutcome',hue='y',data=data)
```

Out[39]: <AxesSubplot:xlabel='poutcome', ylabel='count'>

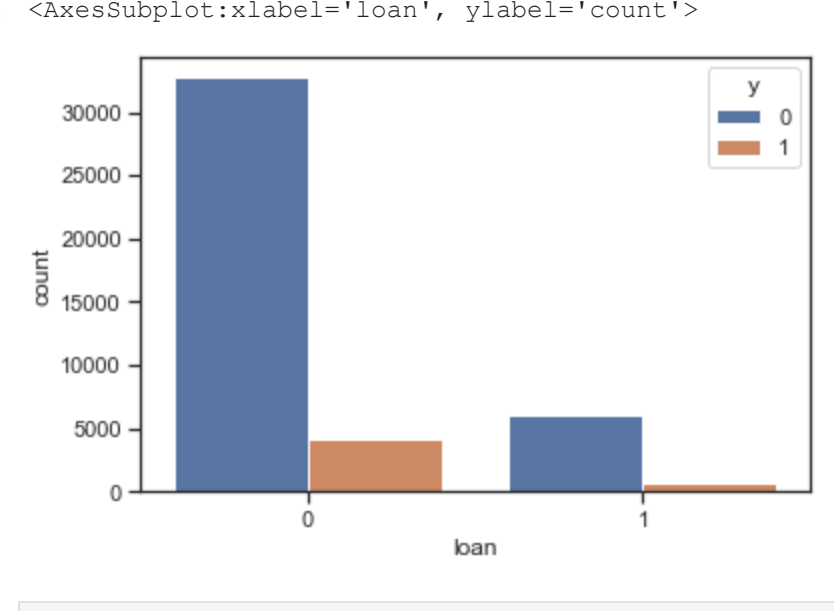


So many non existent values. We can drop this

In [40]:
```
data = data.drop('poutcome',axis=1)
```

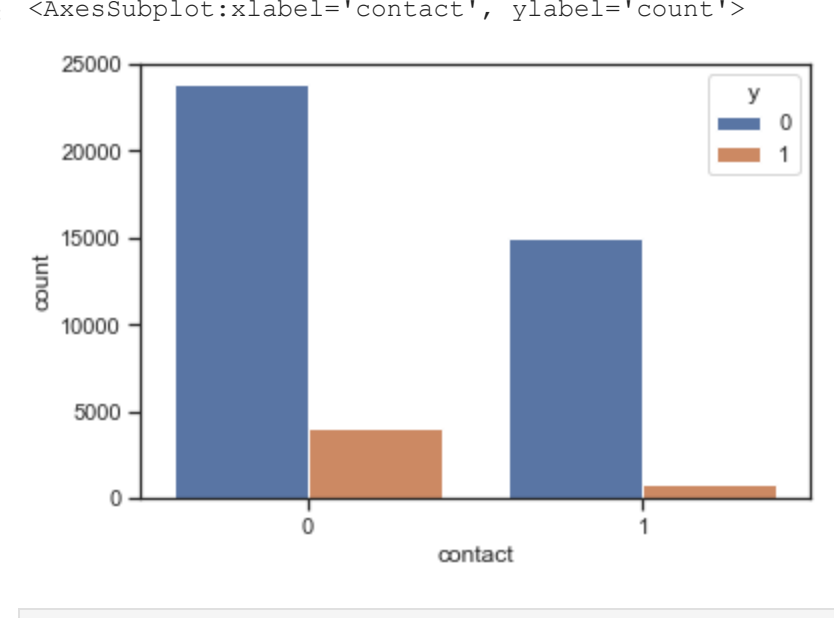
In [41]:
```
sns.countplot(x='loan',hue='y',data=data)
```

Out[41]: <AxesSubplot:xlabel='loan', ylabel='count'>



In [42]:
```
sns.countplot(x='contact',hue='y',data=data)
```

Out[42]: <AxesSubplot:xlabel='contact', ylabel='count'>



In [43]:
```
data = data.drop('contact',axis=1)
```

In [44]:
```
data = data.drop(['emp.var.rate','cons.price.idx','cons.conf.idx','euribor3m','nr.empl
```

In [45]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43748 entries, 0 to 4118
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   age          43748 non-null  float64
 1   job          43748 non-null  int32
 2   marital      43748 non-null  int32
 3   education    43748 non-null  int32
 4   housing      43748 non-null  int32
 5   loan         43748 non-null  int32
 6   month        43748 non-null  int32
 7   day_of_week  43748 non-null  int32
 8   duration     43748 non-null  float64
 9   campaign     43748 non-null  float64
 10  pdays        43748 non-null  int64
 11  previous     43748 non-null  int64
 12  y            43748 non-null  int32
dtypes: float64(3), int32(8), int64(2)
memory usage: 4.3 MB
```

In [46]:
```
data.head()
```

Out[46]:

	age	job	marital	education	housing	loan	month	day_of_week	duration	campaign	pdays	previous
0	56.0	3	1	0	0	0	6	1	261.0	1.0	999	0
1	57.0	7	1	1	0	0	6	1	149.0	1.0	999	0
2	37.0	7	1	1	1	0	6	1	226.0	1.0	999	0
3	40.0	0	1	0	0	0	6	1	151.0	1.0	999	0
4	56.0	7	1	1	0	1	6	1	307.0	1.0	999	0

Splitting into train and test data

import all necessary libraries

In [47]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.decomposition import PCA
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

In [48]:
```
X = data.drop('y',axis = 1).values
y = data['y'].values
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.25, random_state
```

In [49]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.fit_transform(X_train)
```

In [50]:
```
pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.fit_transform(X_train)
```

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

Out[51]: (32811, 10)

Building different Models and validating using 10 fold cross validation

In [52]:
```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('Decison-Tree', DecisionTreeClassifier()))
models.append(('Gaussian', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('RandForest',RandomForestClassifier(max_depth = 8, n_estimators = 120))
models.append(('ADA', AdaBoostClassifier(n_estimators = 120)))
```

In [53]:
```
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=42)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, sc
    results.append(cv_results)
    names.append(name)
    msg = "{}: {}".format(name, cv_results.mean())
    print(msg)
```

```
C:\Users\Thrinesh Duvvuru\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:293: FutureWarning: Setting a random_state has no effect since shuffle is False. Thi
s will raise an error in 0.24. You should leave random_state to its default (None), or
set shuffle=True.
  warnings.warn(
LR: 0.9035995662058859
C:\Users\Thrinesh Duvvuru\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:293: FutureWarning: Setting a random_state has no effect since shuffle is False. Thi
s will raise an error in 0.24. You should leave random_state to its default (None), or
set shuffle=True.
  warnings.warn(
LDA: 0.8985707323442396
C:\Users\Thrinesh Duvvuru\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:293: FutureWarning: Setting a random_state has no effect since shuffle is False. Thi
s will raise an error in 0.24. You should leave random_state to its default (None), or
set shuffle=True.
  warnings.warn(
KNN: 0.8959804395183542
C:\Users\Thrinesh Duvvuru\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:293: FutureWarning: Setting a random_state has no effect since shuffle is False. Thi
s will raise an error in 0.24. You should leave random_state to its default (None), or
set shuffle=True.
  warnings.warn(
Decison-Tree: 0.8760478265625903
C:\Users\Thrinesh Duvvuru\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:293: FutureWarning: Setting a random_state has no effect since shuffle is False. Thi
s will raise an error in 0.24. You should leave random_state to its default (None), or
set shuffle=True.
  warnings.warn(
Gaussian: 0.897686836904297
C:\Users\Thrinesh Duvvuru\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:293: FutureWarning: Setting a random_state has no effect since shuffle is False. Thi
s will raise an error in 0.24. You should leave random_state to its default (None), or
set shuffle=True.
  warnings.warn(
SVM: 0.9023804721327773
C:\Users\Thrinesh Duvvuru\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:293: FutureWarning: Setting a random_state has no effect since shuffle is False. Thi
s will raise an error in 0.24. You should leave random_state to its default (None), or
set shuffle=True.
  warnings.warn(
RandForest: 0.9044224396145628
C:\Users\Thrinesh Duvvuru\anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:293: FutureWarning: Setting a random_state has no effect since shuffle is False. Thi
s will raise an error in 0.24. You should leave random_state to its default (None), or
set shuffle=True.
  warnings.warn(
ADA: 0.8990278171682992
```

Logistic Regression obtained the highest accuracy with lesser runtime and is more stable in the results. Even SVM and Random Forest obtained the similar accuracy but have higher runtime compared to Logistic Regession

In [54]:
```
scaler.fit(X_test)
X_test = scaler.fit_transform(X_test)
```

In [55]:
```
pca.fit(X_test)
X_test = pca.fit_transform(X_test)
```

In [56]:
```
lr = LogisticRegression()
lr.fit(X_train, Y_train)
predictions = lr.predict(X_test)
print("Accuracy : ", accuracy_score(Y_test, predictions))
print("Confusion Matrix : \n",confusion_matrix(Y_test, predictions))
print("Classification Report: \n",classification_report(Y_test, predictions))
```

```
Accuracy :  0.9012526286915973
Confusion Matrix :
 [[9611  126]
 [ 954  246]]
Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.99      0.95      9737
           1       0.66      0.20      0.31      1200

    accuracy                           0.90     10937
   macro avg       0.79      0.60      0.63     10937
weighted avg       0.88      0.90      0.88     10937
```

In [57]:
```
rf=RandomForestClassifier(max_depth = 8, n_estimators = 120)
rf.fit(X_train, Y_train)
predictions = rf.predict(X_test)
print("Accuracy : ", accuracy_score(Y_test, predictions))
print("Confusion Matrix : \n",confusion_matrix(Y_test, predictions))
print("Classification Report: \n",classification_report(Y_test, predictions))
```

```
Accuracy :  0.9022583889549236
Confusion Matrix :
 [[9622  115]
 [ 954  246]]
Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.99      0.95      9737
           1       0.68      0.20      0.32      1200

    accuracy                           0.90     10937
   macro avg       0.80      0.60      0.63     10937
weighted avg       0.88      0.90      0.88     10937
```