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

In [2]:
```
os.chdir("C:/Users/RAM1/Desktop/DSP") #change directory or to set the path
```

In [3]:
```
bank=pd.read_csv("bank.csv")
```

In [4]:
```
bank.head()
```

Out[4]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	dura
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	

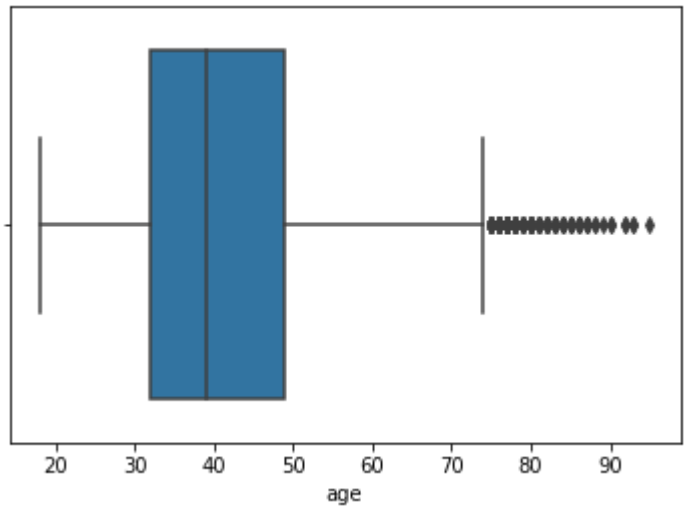
In [5]:
```
bank.isnull().sum()
```

Out[5]:
```
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
deposit      0
dtype: int64
```

In [6]:
```
bank.dtypes
```

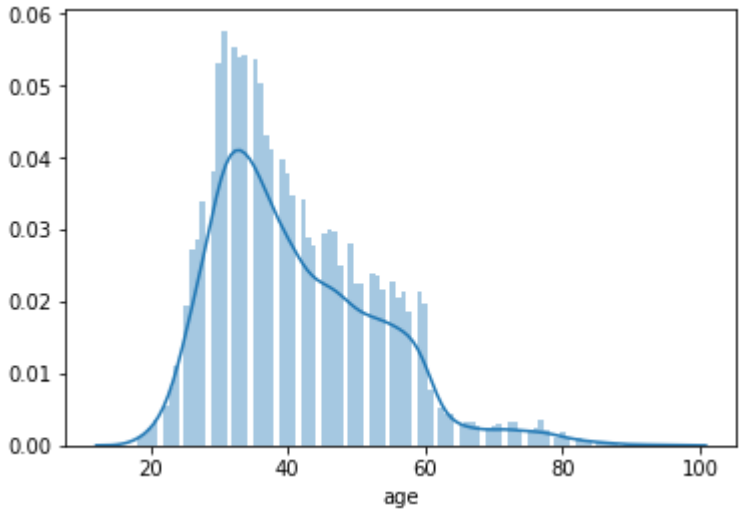
Out[6]:
```
age           int64
job          object
marital      object
education    object
default      object
balance       int64
housing      object
loan         object
contact      object
day           int64
month        object
duration      int64
campaign      int64
pdays         int64
previous      int64
poutcome     object
deposit      object
dtype: object
```

In [7]:
```
g=sns.boxplot(x=bank["age"])
```

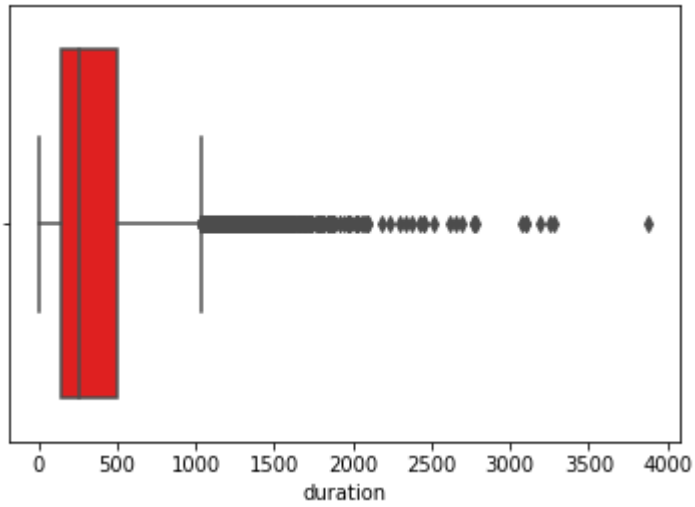


In [8]:
```
sns.distplot(bank.age,bins=100)
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xcd1f859be0>
```

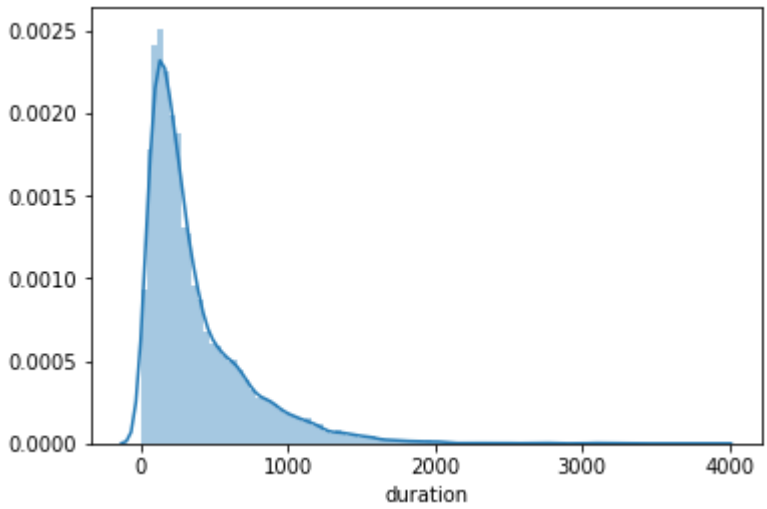


In [9]:
```
g=sns.boxplot(x=bank["duration"],color="red")
```



In [10]:
```
sns.distplot(bank.duration,bins=100)
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xcd1fab3eb8>
```



In [11]:
```
#remove th outliers from the graph
```

In [12]:
```
bank=bank.drop(["day","month","pdays","contact"],axis=1)
```

In [13]:
```
bank.head()
```

Out[13]:

	age	job	marital	education	default	balance	housing	loan	duration	campaign	previou
0	59	admin.	married	secondary	no	2343	yes	no	1042	1	
1	56	admin.	married	secondary	no	45	no	no	1467	1	
2	41	technician	married	secondary	no	1270	yes	no	1389	1	
3	55	services	married	secondary	no	2476	yes	no	579	1	
4	54	admin.	married	tertiary	no	184	no	no	673	2	

In [14]:
```
bank_data=bank #Creating duplicate copy of bank data
```

In [15]:
```
one_hot_data = pd.get_dummies(bank_data[['job','marital','education','default','h
one_hot_data.iloc[0:5,10:15]
one_hot_data.loc[0:5,["job_unknown","marital_single"]]
```

Out[15]:

	job_unknown	marital_single
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	1

In [16]:
```
x= bank_data[["age","duration","campaign","previous","balance"]]
x=pd.concat([x,one_hot_data],axis=1)


y=bank_data["deposit"]
print(x.shape)
print(y.shape)
```

```
(11162, 27)
(11162,)
```

In [17]:
```
from sklearn import model_selection
validation_size=0.40

seed=6
x_train,x_test,y_train,y_test=model_selection.train_test_split(x,y,test_size= val

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(6697, 27)
(6697,)
(4465, 27)
(4465,)
```

In [18]:
```
#create tree object
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
model=tree.DecisionTreeClassifier(criterion="gini")
model.fit(x_train,y_train)
```

Out[18]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [19]:
```
#predicted_dt=model.predict(x_test)
```

In [20]:
```
#It will compare with actual and predicted
#predicted_dt=pd.DataFrame(predicted_dt)
#predicted_dt=pd.concat([y_test,predicted_dt],axis=1)
#predicted_dt.head(10)
```

In [21]:
```
predicted_dt=[]
predicted_dt=model.predict(x_test)
```

In [22]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(predicted_dt,y_test))

from sklearn.metrics import accuracy_score

accuracy=round(accuracy_score(predicted_dt,y_test)*100,2)
print("Accuracy of this model is ",accuracy,"%")
```

```
[[1782  671]
 [ 576 1436]]
Accuracy of this model is  72.07 %
```

In [23]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

model_rand=RandomForestClassifier(n_estimators=100)
model_rand.fit(x_train,y_train)

predicted_rand=model_rand.predict(x_test)

from sklearn.metrics import confusion_matrix
print(confusion_matrix(predicted_rand,y_test))

from sklearn.metrics import accuracy_score

accuracy=round(accuracy_score(predicted_rand,y_test)*100,2)
print("Accuracy of this model is ",accuracy,"%")
```

```
[[1934  417]
 [ 424 1690]]
Accuracy of this model is  81.16 %
```

In [24]:
```
from sklearn.neighbors import KNeighborsClassifier
#models_knn=[]
#models_knn.append(("KNN",KNeighborsClassifier()))


from sklearn import metrics
k_range=(1,3,5,7,9,11,13,15,29,31,33,35,37,39,41)

#scores={}
score_list=[]
for k in k_range:
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    y_pred=knn.predict(x_test)
    scores[k]=metrics.accuracy_score(y_test,y_pred)
    score_list.append(metrics.accuracy_score(y_test,y_pred))
    accuracy_knn=round(accuracy_score(y_pred, y_test)*100,2)
    print("for k= ",k,"The accuracy is: " ,accuracy_knn,"%")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-4c23d0529c30> in <module>()
     13     knn.fit(x_train,y_train)
     14     y_pred=knn.predict(x_test)
---> 15     scores[k]=metrics.accuracy_score(y_test,y_pred)
     16     score_list.append(metrics.accuracy_score(y_test,y_pred))
     17     accuracy_knn=round(accuracy_score(y_pred, y_test)*100,2)

NameError: name 'scores' is not defined
```

In [ ]:
```
%matplotlib inline  #to draw line
plt.plot(k_range,score_list)
plt.xlabel("Number of neighbors")
plt.ylabel("Accuracy")
plt.title("Accuracy Calculation for KNN")
plt.show()
```

In [ ]:
```
```

In [ ]:
```
x=bank_data.drop(["deposit"],axis=1)
y=bank_data["deposit"]
```

In [ ]:
```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=1/3,random_state=0)
```

In [ ]:
```
xtrain.shape
```

In [ ]:
```
xtest.shape
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: