In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
import seaborn as sns
from sklearn.model_selection import train_test_split
```

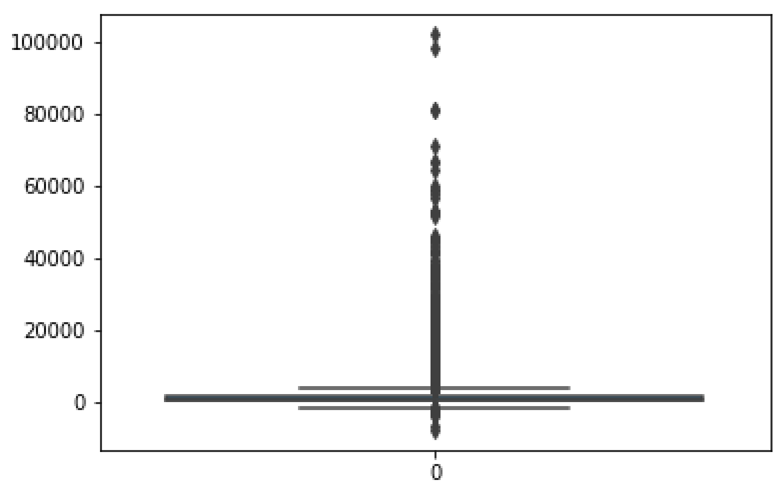
In [2]:
```
raw_data= pd.read_csv('bank-full.csv')
raw_data.head()
```

Out[2]:

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

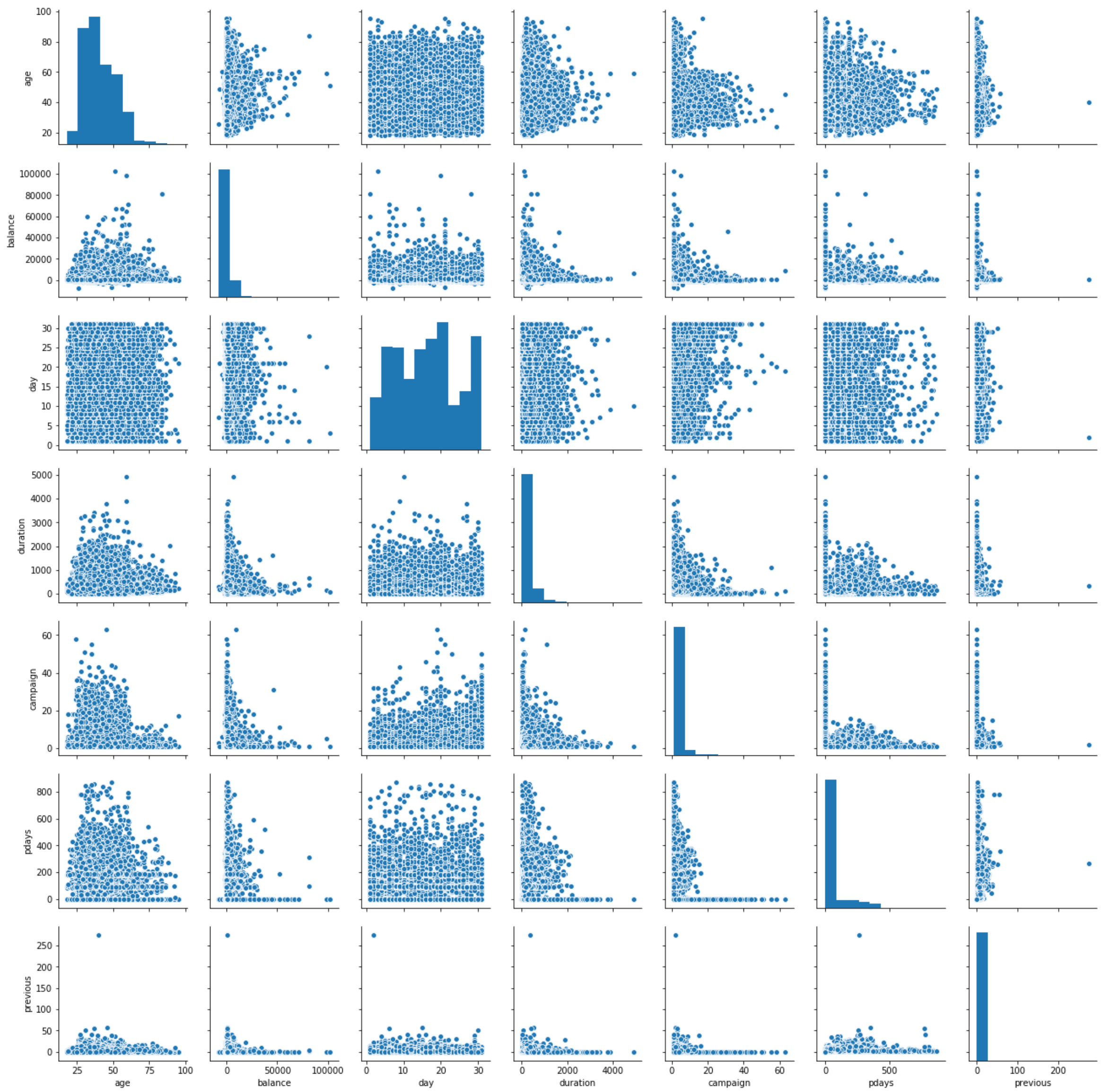
In [3]:
```
sns.boxplot(data= raw_data['balance'])
```

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x192d2bdd550>



In [4]:
```
sns.pairplot(raw_data)
```

Out[4]: <seaborn.axisgrid.PairGrid at 0x192d2b94c88>



In [5]:
```
raw_data.dtypes
```

Out[5]:
```
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
Target       object
dtype: object
```

In [6]:
```
raw_data= raw_data.drop(['contact'], axis= 1)
```

In [7]:
```
raw_data.describe()
```

Out[7]:

	age	balance	day	duration	campaign	pdays	
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	4
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	

In [8]:
```
raw_data['job'].unique()
```

Out[8]:
```
array(['management', 'technician', 'entrepreneur', 'blue-collar',
       'unknown', 'retired', 'admin.', 'services', 'self-employed',
       'unemployed', 'housemaid', 'student'], dtype=object)
```

In [9]:
```
raw_data['marital'].unique()
```

Out[9]:
```
array(['married', 'single', 'divorced'], dtype=object)
```

In [10]:
```
raw_data['education'].unique()
```

Out[10]: `array(['tertiary', 'secondary', 'unknown', 'primary'], dtype=object)`

In [11]:
```
raw_data['default'].unique()
```

Out[11]: `array(['no', 'yes'], dtype=object)`

In [12]:
```
raw_data['housing'].unique()
```

Out[12]: `array(['yes', 'no'], dtype=object)`

In [13]:
```
raw_data['loan'].unique()
```

Out[13]: `array(['no', 'yes'], dtype=object)`

In [14]:
```
raw_data['month'].unique()
```

Out[14]:
```
array(['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'jan', 'feb',
       'mar', 'apr', 'sep'], dtype=object)
```

In [15]:
```
raw_data['poutcome'].unique()
```

Out[15]: `array(['unknown', 'failure', 'other', 'success'], dtype=object)`

In [16]:
```
raw_data['Target'].unique()
```

Out[16]: `array(['no', 'yes'], dtype=object)`

In [17]:
```
raw_data.corr()
```

Out[17]:

	age	balance	day	duration	campaign	pdays	previous
age	1.000000	0.097783	-0.009120	-0.004648	0.004760	-0.023758	0.001288
balance	0.097783	1.000000	0.004503	0.021560	-0.014578	0.003435	0.016674
day	-0.009120	0.004503	1.000000	-0.030206	0.162490	-0.093044	-0.051710
duration	-0.004648	0.021560	-0.030206	1.000000	-0.084570	-0.001565	0.001203
campaign	0.004760	-0.014578	0.162490	-0.084570	1.000000	-0.088628	-0.032855
pdays	-0.023758	0.003435	-0.093044	-0.001565	-0.088628	1.000000	0.454820
previous	0.001288	0.016674	-0.051710	0.001203	-0.032855	0.454820	1.000000

In [18]:

```
jobe= raw_data['job']
educatione= raw_data['education']
balancee= raw_data['balance']
previouse= raw_data['previous']
for i in range (45211):
    if jobe[i] == "unknown":
        raw_data= raw_data.drop(i, axis= 0)

    elif educatione[i] == "unknown":
        raw_data= raw_data.drop(i, axis= 0)

    elif balancee[i]<0:
        raw_data= raw_data.drop(i, axis= 0)

    elif previouse[i]>200:
        raw_data= raw_data.drop(i, axis= 0)
raw_data.head()
```

Out[18]:

	age	job	marital	education	default	balance	housing	loan	day	month	duratio
0	58	management	married	tertiary	no	2143	yes	no	5	may	26
1	44	technician	single	secondary	no	29	yes	no	5	may	15
2	33	entrepreneur	married	secondary	no	2	yes	yes	5	may	7
5	35	management	married	tertiary	no	231	yes	no	5	may	13
6	28	management	single	tertiary	no	447	yes	yes	5	may	21

In [19]:

```
raw_data.describe()
```

Out[19]:

	age	balance	day	duration	campaign	pdays
count	39558.000000	39558.000000	39558.000000	39558.000000	39558.000000	39558.000000
mean	40.887077	1507.659361	15.756813	258.783179	2.744224	40.873300
std	10.629567	3132.353076	8.279917	258.738881	3.027217	100.463052
min	18.000000	0.000000	1.000000	0.000000	1.000000	-1.000000
25%	33.000000	145.000000	8.000000	103.000000	1.000000	-1.000000
50%	39.000000	536.000000	16.000000	180.000000	2.000000	-1.000000
75%	48.000000	1583.000000	21.000000	320.000000	3.000000	-1.000000
max	95.000000	102127.000000	31.000000	4918.000000	58.000000	871.000000

In [20]:

```
#raw_data['job']= raw_data['job'].map({'management': 0, 'technician': 1, 'ent
#raw_data['marital']= raw_data['marital'].map({'single': 0, 'married': 1, 'di
#raw_data['education']= raw_data['education'].map({'primary': 0, 'secondary':
#raw_data['default']= raw_data['default'].map({'yes': 0, 'no': 1})
#raw_data['housing']= raw_data['housing'].map({'yes': 0, 'no': 1})
#raw_data['loan']= raw_data['loan'].map({'yes': 0, 'no': 1})
#raw_data['month']= raw_data['month'].map({'jan': 0, 'feb': 1, 'mar': 2, 'apr
#raw_data['poutcome']= raw_data['poutcome'].map({'success': 0, 'failure': 1,
raw_data['Target']= raw_data['Target'].map({'yes': 0, 'no': 1})
```

In [21]:

```
raw_data.head()
```

Out[21]:

	age	job	marital	education	default	balance	housing	loan	day	month	duratio
0	58	management	married	tertiary	no	2143	yes	no	5	may	26
1	44	technician	single	secondary	no	29	yes	no	5	may	15
2	33	entrepreneur	married	secondary	no	2	yes	yes	5	may	7
5	35	management	married	tertiary	no	231	yes	no	5	may	13
6	28	management	single	tertiary	no	447	yes	yes	5	may	21

In [22]:

```
job= pd.get_dummies(raw_data['job'], prefix= 'job')
marital= pd.get_dummies(raw_data['marital'], prefix= 'marital')
education= pd.get_dummies(raw_data['education'], prefix= 'education')
default= pd.get_dummies(raw_data['default'], prefix= 'default')
housing= pd.get_dummies(raw_data['housing'], prefix= 'housing')
loan= pd.get_dummies(raw_data['loan'], prefix= 'loan')
month= pd.get_dummies(raw_data['month'], prefix= 'month')
poutcome= pd.get_dummies(raw_data['poutcome'], prefix= 'potcome')
#Target= pd.get_dummies(raw_data['Target'], prefix= 'Target')
data= pd.concat([raw_data['age'], raw_data['balance'], raw_data['day'], raw_d
```

In [23]:
```
data
```

24	40	0	5	181	1	-1	0	0	0
26	39	255	5	296	1	-1	0	0	0
27	52	113	5	127	1	-1	0	0	0
29	36	265	5	348	1	-1	0	0	0
30	57	839	5	225	1	-1	0	0	0
31	49	378	5	230	1	-1	0	0	0
32	60	39	5	208	1	-1	0	1	0
33	59	0	5	226	1	-1	0	0	1
34	51	10635	5	336	1	-1	0	0	0
35	57	63	5	242	1	-1	0	0	0
...	...	...	...	...	...	...	...	...	...
45180	66	3409	15	414	2	27	6	0	0
45181	46	6879	15	74	2	118	3	0	1

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

Out[24]:

	age	balance	day	duration	campaign	pdays
count	39558.000000	39558.000000	39558.000000	39558.000000	39558.000000	39558.000000
mean	40.887077	1507.659361	15.756813	258.783179	2.744224	40.873300
std	10.629567	3132.353076	8.279917	258.738881	3.027217	100.463052
min	18.000000	0.000000	1.000000	0.000000	1.000000	-1.000000
25%	33.000000	145.000000	8.000000	103.000000	1.000000	-1.000000
50%	39.000000	536.000000	16.000000	180.000000	2.000000	-1.000000
75%	48.000000	1583.000000	21.000000	320.000000	3.000000	-1.000000
max	95.000000	102127.000000	31.000000	4918.000000	58.000000	871.000000

8 rows × 47 columns

Data Split

In [25]:
```
x_train, x_test, y_train, y_test= train_test_split(data.drop(['Target'], axis
```

In [26]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
C:\Users\ppragallapati\AppData\Local\Continuum\anaconda3\lib\site-packages
\sklearn\preprocessing\data.py:625: DataConversionWarning: Data with input
dtype uint8, int64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
C:\Users\ppragallapati\AppData\Local\Continuum\anaconda3\lib\site-packages
\sklearn\base.py:462: DataConversionWarning: Data with input dtype uint8, i
nt64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
C:\Users\ppragallapati\AppData\Local\Continuum\anaconda3\lib\site-packages
\ipykernel_launcher.py:5: DataConversionWarning: Data with input dtype uint
8, int64 were all converted to float64 by StandardScaler.
  """
```

Logistic regression

In [27]:
```
from sklearn.linear_model import LogisticRegression
logistic= LogisticRegression()
logistic.fit(x_train, y_train)
logistic_prediction= logistic.predict(x_test)
```

```
C:\Users\ppragallapati\AppData\Local\Continuum\anaconda3\lib\site-packages
\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will b
e changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [28]:
```
from sklearn.metrics import classification_report
from sklearn import metrics
print(classification_report(y_test,logistic_prediction))
print("Accuracy:",metrics.accuracy_score(y_test, logistic_prediction))
metrics.confusion_matrix(y_test, logistic_prediction)
```

```
              precision    recall  f1-score   support

           0       0.64      0.34      0.45      1411
           1       0.92      0.97      0.94     10457

   micro avg       0.90      0.90      0.90     11868
   macro avg       0.78      0.66      0.70     11868
weighted avg       0.88      0.90      0.89     11868

Accuracy: 0.8987192450286484
```

Out[28]:
```
array([[  484,   927],
       [  275, 10182]], dtype=int64)
```

Random Forest

In [29]:
```
from sklearn.ensemble import RandomForestRegressor
#tree = DecisionTreeClassifier(random_state=RSEED)
regressor = RandomForestRegressor(n_estimators=20, random_state=0)
regressor.fit(x_train, y_train)
y_pred = regressor.predict(x_test)
```

In [30]:
```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,
```

```
Mean Absolute Error: 0.13370285642062688
Mean Squared Error: 0.06693979438405796
Root Mean Squared Error: 0.2587272586799813
```

In [31]:
```
regressor.score(x_test, y_test)
```

Out[31]:
```
0.360993423936926
```

XGBOOST

In [33]:
```
from numpy import loadtxt
from xgboost import XGBClassifier
# fit model no training data
model = XGBClassifier()
model.fit(x_train, y_train)
```

Out[33]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
```

In [35]:
```
# make predictions for test data
y_pred = model.predict(x_test)
predictions = [round(value) for value in y_pred]
```

In [37]:
```
# evaluate predictions
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 90.19%
```

In [ ]:
```
#XGBOOST gave the best accuracy
```