In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
import io
%cd "G:\PGW23\python\Bank data mock test"
```

```
G:\PGW23\python\Bank data mock test
```

In [3]:

```
bank=pd.read_csv("bank-full.csv")
```

In [4]:

```
# Run - head, tail, info
bank.head()
```

Out[4]:

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

In [5]:

```
bank.tail()
```

Out[5]:

	age	job	marital	education	default	balance	housing	loan	contact	day
45206	51	technician	married	tertiary	no	825	no	no	cellular	17
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17
45208	72	retired	married	secondary	no	5715	no	no	cellular	17
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17

In [6]:

```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        45211 non-null  int64 
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64 
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        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  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

In [7]:

```
bank.describe()
```

Out[7]:

	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

In [8]:

```
bank.shape
```

Out[8]:

```
(45211, 17)
```

In [9]:

```
bank.describe(include=['object'])
```

Out[9]:

	job	marital	education	default	housing	loan	contact	month	poutcome	y
count	45211	45211	45211	45211	45211	45211	45211	45211	45211	4521
unique	12	3	4	2	2	2	3	12	4	2
top	blue-collar	married	secondary	no	yes	no	cellular	may	unknown	n
freq	9732	27214	23202	44396	25130	37967	29285	13766	36959	3992

In [10]:

```
#finding any missing values in data set
bank.isnull().sum().sort_values(ascending=False)
```

Out[10]:

```
age          0
day          0
poutcome     0
previous     0
pdays        0
campaign     0
duration     0
month        0
contact      0
job          0
loan         0
housing      0
balance      0
default      0
education    0
marital      0
y            0
dtype: int64
```

In [11]:

```
bank.count()
```

Out[11]:

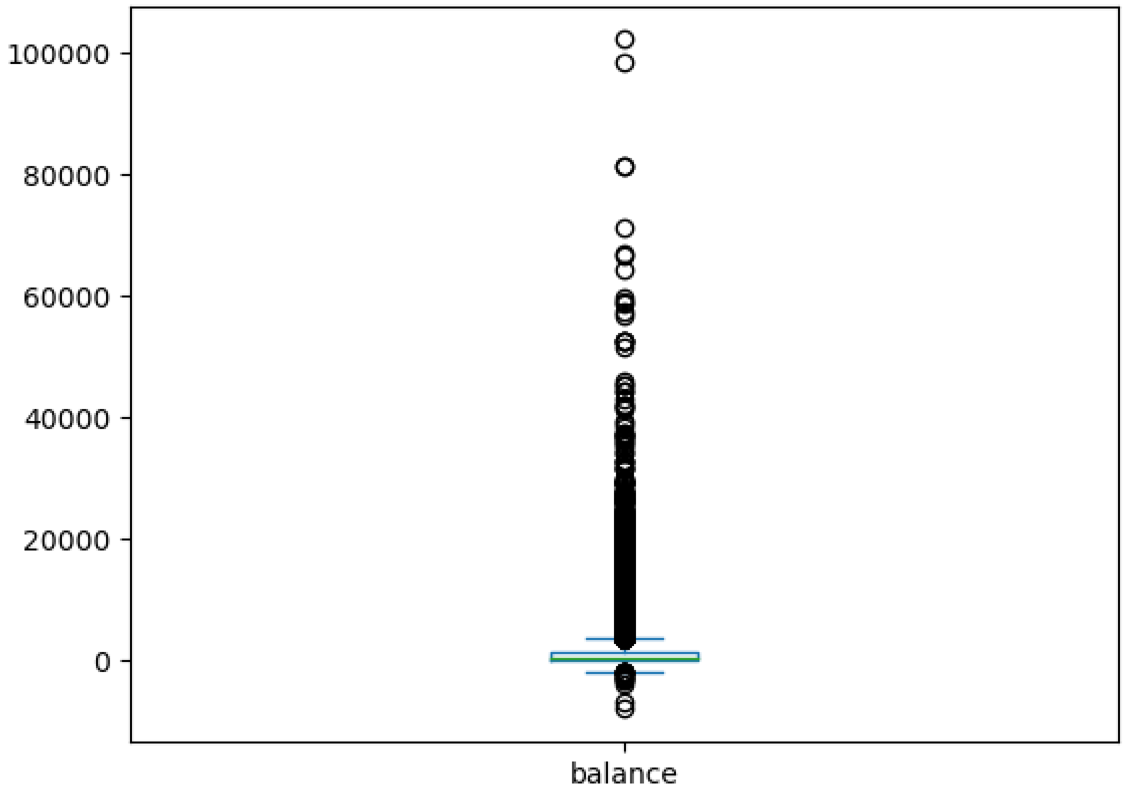
```
age          45211
job          45211
marital      45211
education    45211
default      45211
balance      45211
housing      45211
loan         45211
contact      45211
day          45211
month        45211
duration     45211
campaign     45211
pdays        45211
previous     45211
poutcome     45211
y            45211
dtype: int64
```

In [12]:

```
# histogram, boxplot and density curve - balance and duration
bank.balance.plot(kind="box")
```

Out[12]:

```
<AxesSubplot:>
```

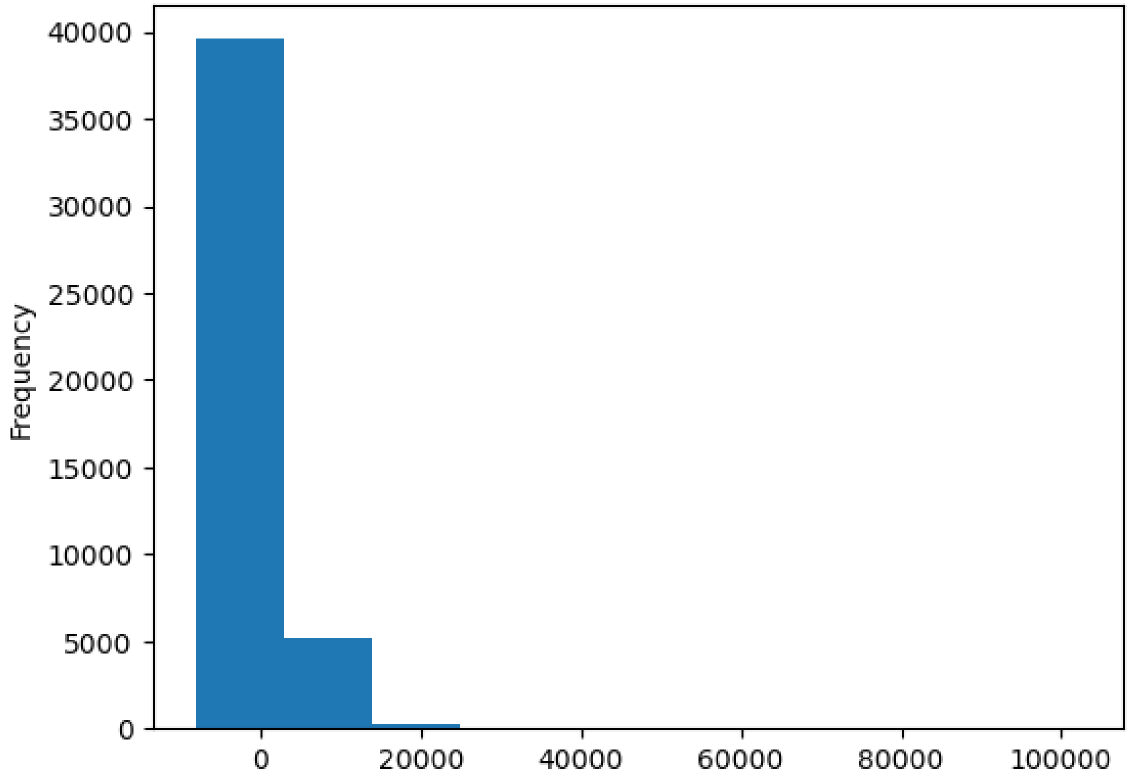


In [13]:

```
bank.balance.plot(kind="hist")
```

Out[13]:

```
<AxesSubplot:ylabel='Frequency'>
```

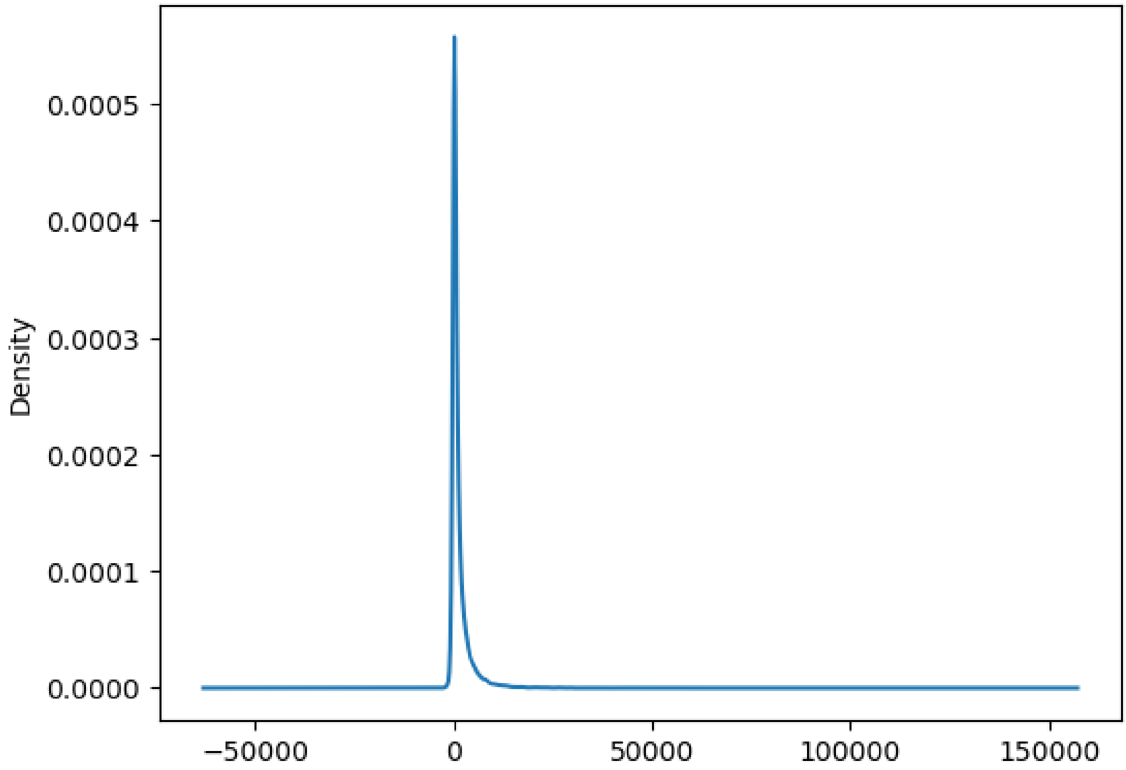


In [14]:

```
bank.balance.plot(kind="density")
```

Out[14]:

```
<AxesSubplot:ylabel='Density'>
```

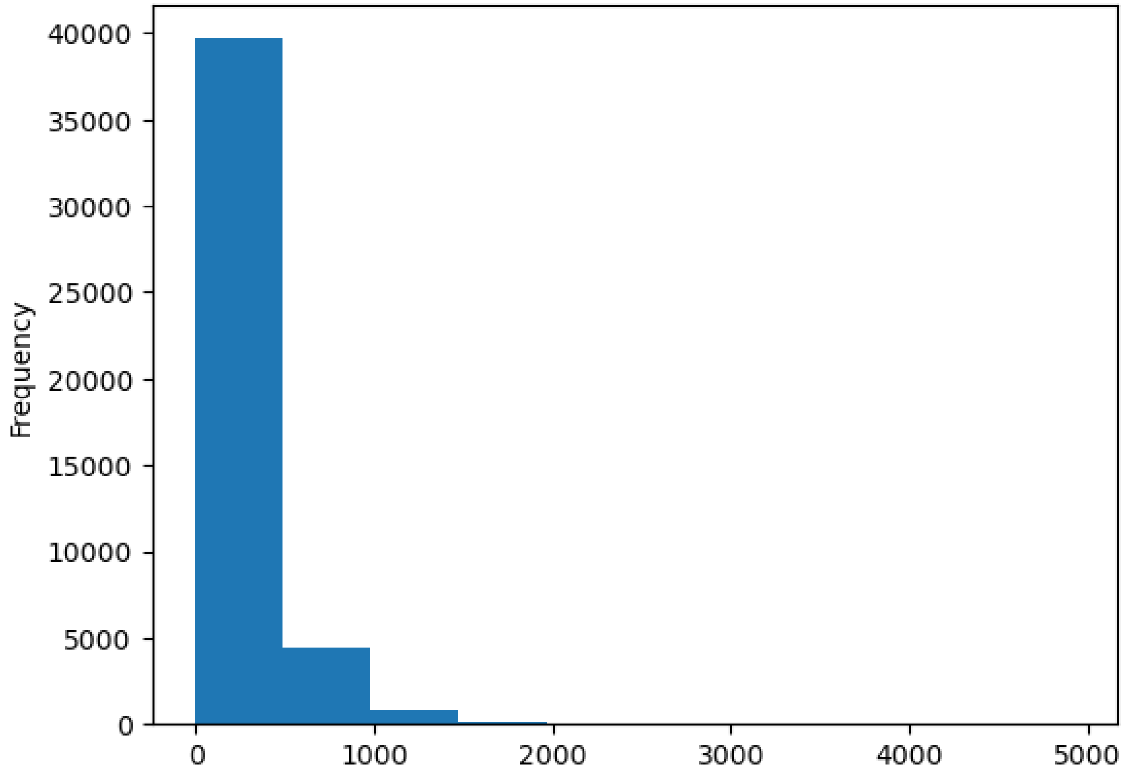


In [15]:

```
bank.duration.plot(kind="hist")
```

Out[15]:

```
<AxesSubplot:ylabel='Frequency'>
```

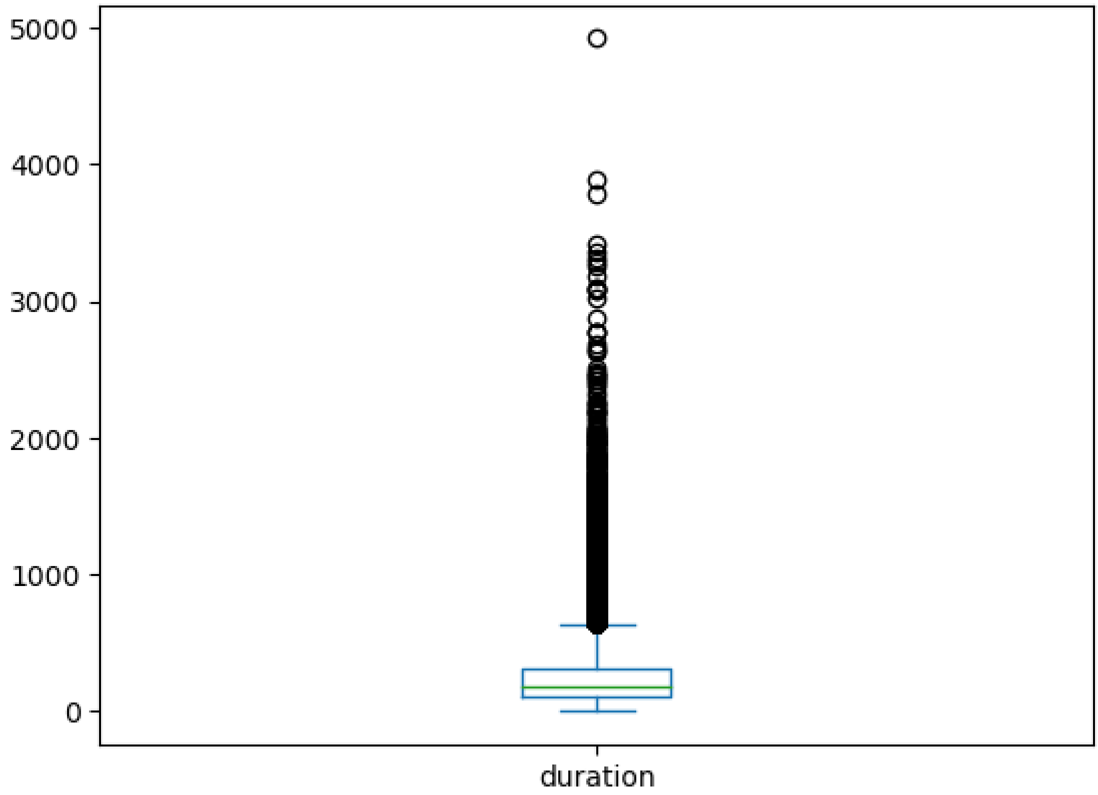


In [16]:

```
bank.duration.plot(kind="box")
```

Out[16]:

<AxesSubplot:>

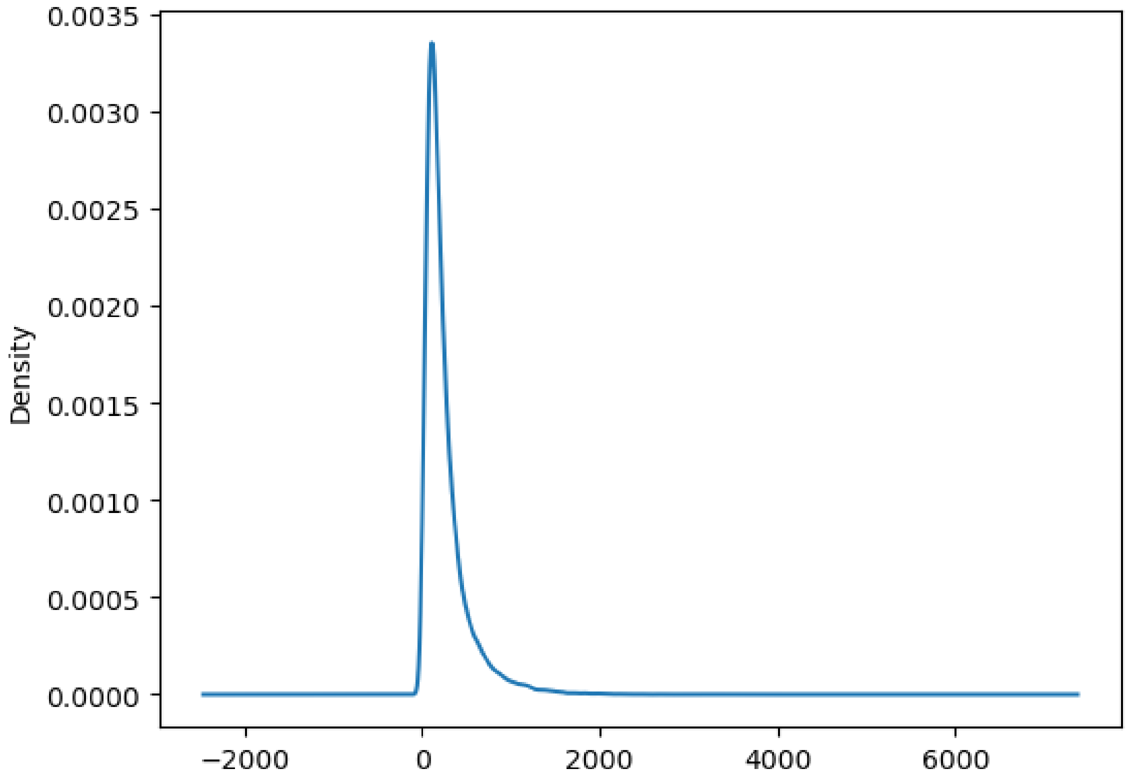


In [17]:

```
bank.duration.plot(kind="density")
```

Out[17]:

```
<AxesSubplot:ylabel='Density'>
```



In [18]:

```
#using count function we find the number of cells in data set
bank.y.count()
```

Out[18]:

```
45211
```

In [25]:

```
# Frequency Counts of the following variable - y,
bank.y.value_counts()
```

Out[25]:

```
no     39922
yes     5289
Name: y, dtype: int64
```

In [20]:

```
# Frequency Counts of the following variable -marital
bank.marital.value_counts()
```

Out[20]:

```
married     27214
single      12790
divorced     5207
Name: marital, dtype: int64
```

In [21]:

```
# Frequency Counts of the following variable -education
bank.education.value_counts()
```

Out[21]:

```
secondary    23202
tertiary     13301
primary       6851
unknown       1857
Name: education, dtype: int64
```

In [22]:

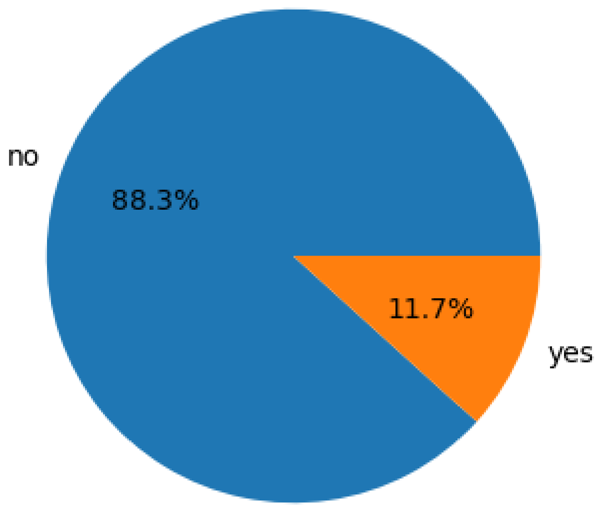
```
# Frequency Counts of the following variable - job
bank.job.value_counts()
```

Out[22]:

```
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
```

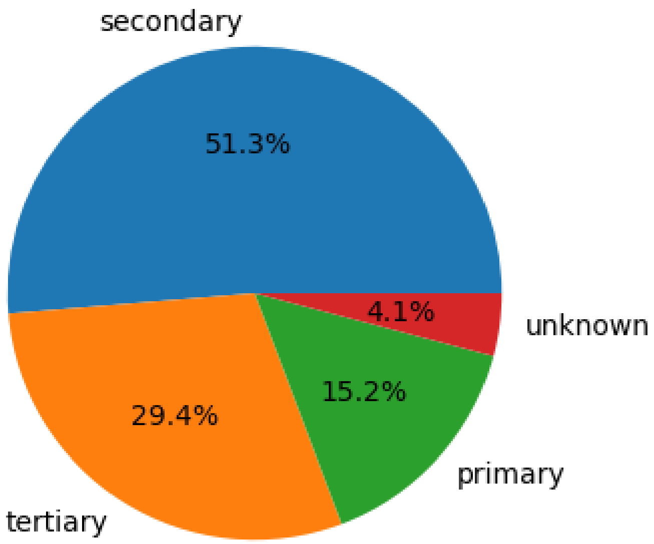
In [23]:

```
# Frequency Counts of the following variable - y and pie diagrams
plt.figure(figsize=(4,4))
bank1=bank['y'].value_counts()
keys=bank1.keys().to_list()
count=bank1.to_list()
plt.pie(x=count,labels=keys,autopct='%1.1f%%')
plt.show()
```



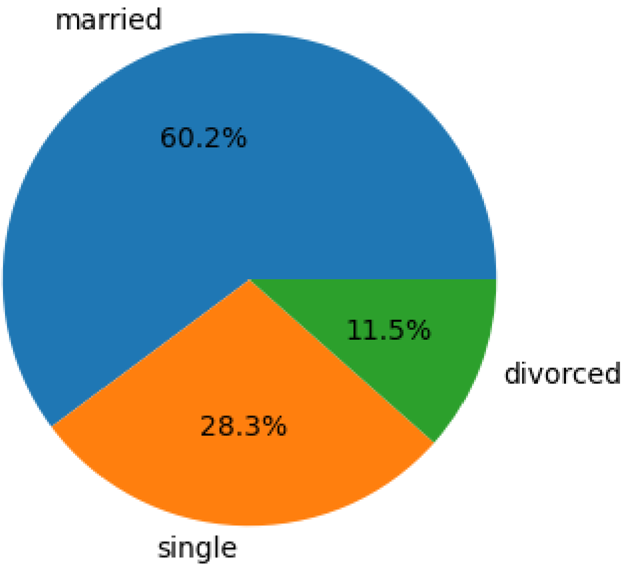
In [43]:

```
# Frequency Counts of the following variable -  education and  pie diagrams
plt.figure(figsize=(4,4))
bank1=bank['education'].value_counts()
keys=bank1.keys().to_list()
count=bank1.to_list()
plt.pie(x=count,labels=keys,autopct='%1.1f%%')
plt.show()
```



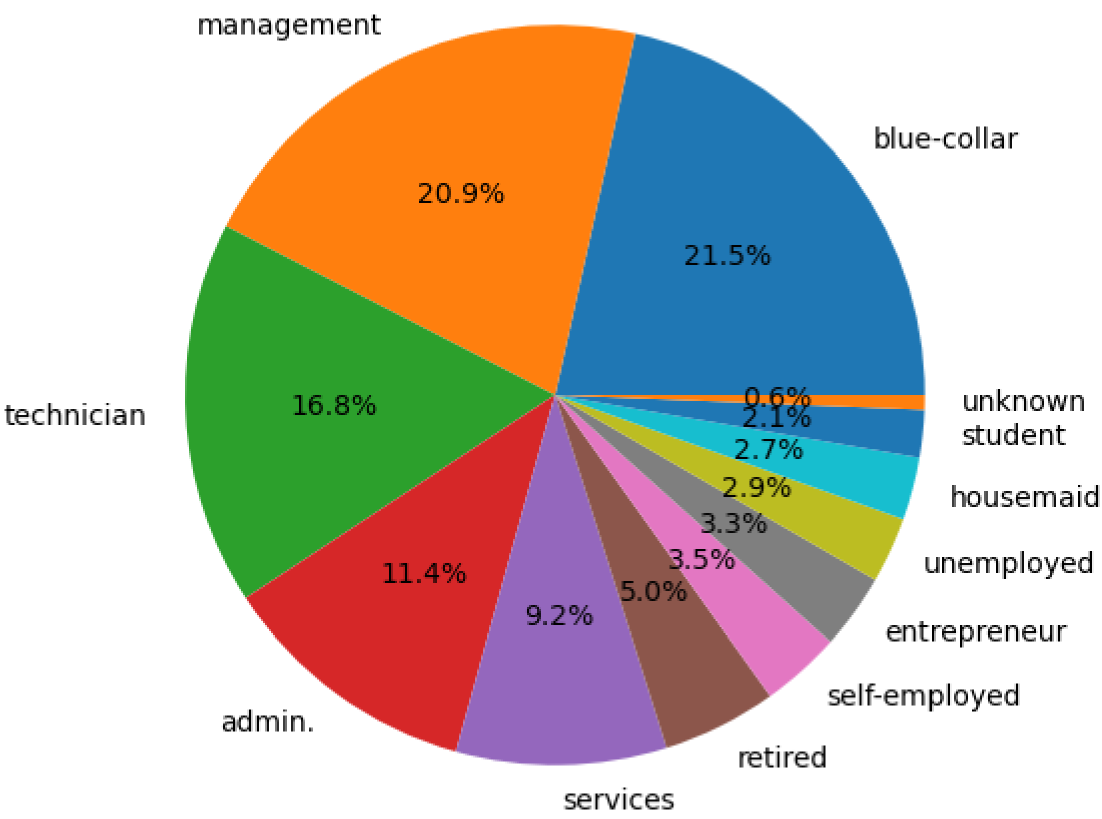
In [44]:

```
# Frequency Counts of the following variable -marital and pie diagrams
plt.figure(figsize=(4,4))
bank1=bank['marital'].value_counts()
keys=bank1.keys().to_list()
count=bank1.to_list()
plt.pie(x=count,labels=keys,autopct='%1.1f%%')
plt.show()
```



In [48]:

```
# Frequency Counts of the following variable - job and pie diagrams
plt.figure(figsize=(6,6))
bank1=bank['job'].value_counts()
keys=bank1.keys().to_list()
count=bank1.to_list()
plt.pie(x=count,labels=keys,autopct='%1.1f%%')
plt.show()
```



In [62]:

```
# Cross Tabulations of y and default:

pd.crosstab(bank.y,bank.default)
```

Out[62]:

default	no	yes
y		
no	39159	763
yes	5237	52

In [51]:

```
# Cross Tabulations y and housing:

pd.crosstab(bank.y,bank.housing)
```

Out[51]:

housing	no	yes
y		
no	16727	23195
yes	3354	1935

In [52]:

```
# Cross Tabulations of  y and Loan:

pd.crosstab(bank.y,bank.loan)
```

Out[52]:

loan	no	yes
y		
no	33162	6760
yes	4805	484

In [53]:

```
# Cross Tabulations of  y and marital:

pd.crosstab(bank.y,bank.marital)
```

Out[53]:

marital	divorced	married	single
y			
no	4585	24459	10878
yes	622	2755	1912

In [57]:

```
# What is the Average balance of y (yes/no)?

bank.balance.groupby(bank.y).mean()
```

Out[57]:

```
y
no      1303.714969
yes     1804.267915
Name: balance, dtype: float64
```

In [58]:

```
# What is the Median age of y(yes/no)?

bank.age.groupby(bank.y).median()
```

Out[58]:

```
y
no     39.0
yes    38.0
Name: age, dtype: float64
```

In [59]:

```
# What is the Average balance of different marital?

bank.balance.groupby(bank.marital).mean()
```

Out[59]:

```
marital
divorced    1178.872287
married     1425.925590
single      1301.497654
Name: balance, dtype: float64
```

In [60]:

```
# What is the Average balance of different job:

bank.balance.groupby(bank.job).mean()
```

Out[60]:

```
job
admin.           1135.838909
blue-collar      1078.826654
entrepreneur     1521.470074
housemaid        1392.395161
management       1763.616832
retired          1984.215106
self-employed    1647.970868
services          997.088108
student          1388.060768
technician       1252.632092
unemployed       1521.745971
unknown          1772.357639
Name: balance, dtype: float64
```

In [61]:

```
# What is the Average balance of different education?

bank.balance.groupby(bank.education).mean()
```

Out[61]:

```
education
primary      1250.949934
secondary    1154.880786
tertiary     1758.416435
unknown      1526.754443
Name: balance, dtype: float64
```

In [63]:

```
# Hypothesis Testing - groupby()-mean & variance(ttest), Null & Alt,
# Split Data, Conduct test and interpret based on p-value
```

In [64]:

```
# Test Null Average balance of y(yes/no) equal
bank.balance.groupby(bank.y).mean()
```

Out[64]:

```
y
no     1303.714969
yes    1804.267915
Name: balance, dtype: float64
```

In [65]:

```
#Splititng data ,# Test Null Average balance of y(yes/no) equal
N=bank[bank.y=="no"]
Y=bank[bank.y=="yes"]
```

In [66]:

```
from scipy.stats import ttest_ind
```

In [68]:

```
#Null-No association b/w two variables
#alt- association b/w two variables
ttest_ind(N.balance,Y.balance,equal_var=False)
# pvalue=4.3837327771001536e-23 is less than 0.05,Fail to reject null
```

Out[68]:

```
Ttest_indResult(statistic=-9.933545392962255, pvalue=4.3837327771001536e-2
3)
```

In [69]:

```
# Test Null Average duration of y(yesy/no) equal
bank.duration.groupby(bank.y).mean()
```

Out[69]:

```
y
no     221.182806
yes    537.294574
Name: duration, dtype: float64
```

In [70]:

```
#Splititng data ,# Test Null Average duration of y(yesy/no) equal
N=bank[bank.y=="no"]
Y=bank[bank.y=="yes"]
```

In [71]:

```
#Null-No association b/w two variables
#alt- association b/w two variables

ttest_ind(N.duration,Y.duration,equal_var=False)

#pvalue=0.0 is then 0.05 reject null
```

Out[71]:

```
Ttest_indResult(statistic=-57.51412654456789, pvalue=0.0)
```

In [72]:

```
# Test Null Average age of y(yesy/no) equal
bank.age.groupby(bank.y).mean()
```

Out[72]:

```
y
no     40.838986
yes    41.670070
Name: age, dtype: float64
```

In [73]:

```
#Splititng data ,# Test Null Average age of y(yesy/no) equal
N=bank[bank.y=="no"]
Y=bank[bank.y=="yes"]
```

In [74]:

```
#Null-No association b/w two variables
#alt- association b/w two variables

ttest_ind(N.age,Y.age,equal_var=False)

#pvalue=1.5971046743760372e-05 is more the 0.05 fail to reject null
```

Out[74]:

```
Ttest_indResult(statistic=-4.318317591167348, pvalue=1.5971046743760372e-0
5)
```

In [26]:

```
# Test Null Average balance of different marital equal

bank.balance.groupby(bank.marital).mean()
```

Out[26]:

```
marital
divorced    1178.872287
married     1425.925590
single      1301.497654
Name: balance, dtype: float64
```

In [27]:

```
#split data

divorcedbank=bank[bank.marital=="divorced"]
marriedbank=bank[bank.marital=="married"]
singlebank=bank[bank.marital=="single"]
```

In [28]:

```
from scipy.stats import f_oneway
```

In [29]:

```
f_oneway(divorcedbank.balance,marriedbank.balance,singlebank.balance)

#pvalue=1.6055869132631893e-08 is less then 0.05,reject null
```

Out[29]:

```
F_onewayResult(statistic=17.954318144453257, pvalue=1.6055869132631893e-0
8)
```

In [30]:

```
# Test Null Average duration of different marital equal

bank.duration.groupby(bank.marital).mean()
```

Out[30]:

```
marital
divorced    262.517188
married     253.412765
single      266.497967
Name: duration, dtype: float64
```

In [31]:

```
f_oneway(divorcedbank.duration,marriedbank.duration,singlebank.duration)

#pvalue=5.697950277614421e-06 is less then 0.05 ,reject null
```

Out[31]:

```
F_onewayResult(statistic=12.078630055775221, pvalue=5.697950277614421e-06)
```

In [32]:

```
# Test Null Average age of different marital equal

bank.age.groupby(bank.marital).mean()
```

Out[32]:

```
marital
divorced    45.782984
married     43.408099
single      33.703440
Name: age, dtype: float64
```

In [33]:

```
f_oneway(divorcedbank.age,marriedbank.age,singlebank.age)

#pvalue=0.0 is less then 0.05, reject null
```

Out[33]:

```
F_onewayResult(statistic=5228.732920484922, pvalue=0.0)
```

In [34]:

```
# Test Null No Association between job and education

pd.crosstab(bank.job,bank.education)
```

Out[34]:

education	primary	secondary	tertiary	unknown
job				
admin.	209	4219	572	171
blue-collar	3758	5371	149	454
entrepreneur	183	542	686	76
housemaid	627	395	173	45
management	294	1121	7801	242
retired	795	984	366	119
self-employed	130	577	833	39
services	345	3457	202	150
student	44	508	223	163
technician	158	5229	1968	242
unemployed	257	728	289	29
unknown	51	71	39	127

In [35]:

```
from scipy.stats import chi2_contingency
```

In [36]:

```
#Null-No association b/w two variables
#alt- association b/w two variables

chi2_contingency(pd.crosstab(bank.job,bank.education))

#p-value=0.0 ,less then 0.05,reject null
```

Out[36]:

```
(28483.136453176405,
 0.0,
 33,
 array([[ 783.58189379, 2653.7245803 , 1521.29948464,  212.39404127],
        [1474.72809714, 4994.4010086 , 2863.13799739,  399.73289686],
        [ 225.33093716,  763.11901971,  437.47289376,   61.07714937],
        [ 187.90205923,  636.36017783,  364.80591007,   50.93185287],
        [1433.20780341, 4853.78593705, 2782.5276592 ,  388.47860034],
        [ 343.07279202, 1161.87051824,  666.06498419,   92.99170556],
        [ 239.27205769,  810.33283935,  464.53913871,   64.85596426],
        [ 629.47189843, 2131.80659574, 1222.09979872,  170.62170711],
        [ 142.13881577,  481.37568291,  275.95801907,   38.52748225],
        [1151.20318064, 3898.73247661, 2235.02459578,  312.03974696],
        [ 197.44869611,  668.69138042,  383.34040388,   53.51951959],
        [  43.6417686 ,  147.79978324,   84.7291146 ,   11.82933357]]))
```

In [37]:

```
# Test Null No Association between y and default

pd.crosstab(bank.y,bank.default)
```

Out[37]:

default	no	yes
y		
no	39159	763
yes	5237	52

In [38]:

```
#Null-No association b/w two variables
#alt- association b/w two variables

chi2_contingency(pd.crosstab(bank.y,bank.default))

#p-value=2.4538606753508344e-06 is less then 0.05 ,Reject null
```

Out[38]:

```
(22.20224995571685,
 2.4538606753508344e-06,
 1,
 array([[39202.34261574,   719.65738426],
        [ 5193.65738426,    95.34261574]]))
```

In [39]:

```
# Test Null No Association between y and marital

pd.crosstab(bank.y,bank.marital)
```

Out[39]:

marital	divorced	married	single
y			
no	4585	24459	10878
yes	622	2755	1912

In [41]:

```
#Null-No association b/w two variables
#alt- association b/w two variables

chi2_contingency(pd.crosstab(bank.y,bank.marital))

#p_value=2.1450999986791486e-43 less then 0.05, reject null
```

Out[41]:

```
(196.4959456560396,
 2.1450999986791486e-43,
 2,
 array([[ 4597.86012254, 24030.37552808, 11293.76434938],
        [  609.13987746,  3183.62447192,  1496.23565062]]))
```

In [42]:

```
# Test Null No Association between marital and job

pd.crosstab(bank.marital,bank.job)
```

Out[42]:

job	admin.	blue-collar	entrepreneur	housemaid	management	retired	self-employed	services
marital								
divorced	750	750	179	184	1111	425	140	549
married	2693	6968	1070	912	5400	1731	993	2407
single	1728	2014	238	144	2947	108	446	1198

In [44]:

```
#Null-No association b/w two variables
#alt- association b/w two variables

chi2_contingency(pd.crosstab(bank.marital,bank.job))

#p-value=0.0, less then 0.05, reject null
```

Out[44]:

```
(3837.6026593315473,
 0.0,
 22,
 array([[ 595.54968923, 1120.84501559,  171.2594059 ,  142.81214749,
         1089.28813784,  260.74734025,  181.85514587,  478.42069408,
          108.03047931,  874.95474553,  150.06792595,   33.16927296],
        [3112.59635929, 5858.01349229,  895.07460574,  746.39711575,
         5693.08380704, 1362.77666939,  950.45245626, 2500.43033775,
          564.61330207, 4572.88620026,  784.31890469,  173.35674946],
        [1462.85395147, 2753.14149211,  420.66598837,  350.79073677,
         2675.62805512,  640.47599036,  446.69239787, 1175.14896817,
          265.35621862, 2149.15905421,  368.61316936,   81.47397757]]))
```

In [45]:

```
# Split data into numeric and object variables and dummy encode object variables. Scale n


numcol=bank.select_dtypes(include=np.number)
objcol=bank.select_dtypes(include=['object'])
```

In [46]:

```
numcol.head()
```

Out[46]:

	age	balance	day	duration	campaign	pdays	previous
0	58	2143	5	261	1	-1	0
1	44	29	5	151	1	-1	0
2	33	2	5	76	1	-1	0
3	47	1506	5	92	1	-1	0
4	33	1	5	198	1	-1	0

In [47]:

```
objcol.head()
```

Out[47]:

	job	marital	education	default	housing	loan	contact	month	poutcome	y
0	management	married	tertiary	no	yes	no	unknown	may	unknown	no
1	technician	single	secondary	no	yes	no	unknown	may	unknown	no
2	entrepreneur	married	secondary	no	yes	yes	unknown	may	unknown	no
3	blue-collar	married	unknown	no	yes	no	unknown	may	unknown	no
4	unknown	single	unknown	no	no	no	unknown	may	unknown	no

In [49]:

```
objcol.columns
```

Out[49]:

```
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'conta
ct',
       'month', 'poutcome', 'y'],
      dtype='object')
```

In [50]:

```
#now we convert object colums using labelencoder

from sklearn.preprocessing import LabelEncoder
```

In [51]:

```
le=LabelEncoder()
```

In [52]:

```
objcoldummy=objcol.apply(le.fit_transform)
```

In [53]:

```
objcoldummy.head()
```

Out[53]:

	job	marital	education	default	housing	loan	contact	month	poutcome	y
0	4	1	2	0	1	0	2	8	3	0
1	9	2	1	0	1	0	2	8	3	0
2	2	1	1	0	1	1	2	8	3	0
3	1	1	3	0	1	0	2	8	3	0
4	11	2	3	0	0	0	2	8	3	0

In [54]:

```
#Now we will combining the both numcol and objdummuy cols into single data set

bankclear=pd.concat([numcol,objcoldummy],axis=1)
```

In [55]:

```
bankclear.head()
```

Out[55]:

	age	balance	day	duration	campaign	pdays	previous	job	marital	education	default
0	58	2143	5	261	1	-1	0	4	1	2	0
1	44	29	5	151	1	-1	0	9	2	1	0
2	33	2	5	76	1	-1	0	2	1	1	0
3	47	1506	5	92	1	-1	0	1	1	3	0
4	33	1	5	198	1	-1	0	11	2	3	0

In [56]:

```
bankclear.shape
```

Out[56]:

```
(45211, 17)
```

In [57]:

```
#spliting data into X and y to perform ML.

X=bankclear.drop("y",axis=1)
y=bankclear.y
```

In [61]:

```
X.head()
```

Out[61]:

	age	balance	day	duration	campaign	pdays	previous	job	marital	education	default
0	58	2143	5	261	1	-1	0	4	1	2	0
1	44	29	5	151	1	-1	0	9	2	1	0
2	33	2	5	76	1	-1	0	2	1	1	0
3	47	1506	5	92	1	-1	0	1	1	3	0
4	33	1	5	198	1	-1	0	11	2	3	0

In [60]:

```
y.head()
```

Out[60]:

```
0    0
1    0
2    0
3    0
4    0
Name: y, dtype: int32
```

In [62]:

```
#Build the following Models

#Logistic Regression

from sklearn.linear_model import LogisticRegression
```

In [63]:

```
lg=LogisticRegression(max_iter=3000)
```

In [64]:

```
lgmodel=lg.fit(X,y)
```

```
C:\Users\RELIANCE\anaconda3\lib\site-packages\sklearn\linear_model\_logist
ic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [65]:

```
lgmodel.score(X,y)
```

Out[65]:

```
0.8903364225520338
```

In [66]:

```
lgpredict=lgmodel.predict(X)
```

In [67]:

```
lgresidual=y-lgpredict
```

In [68]:

```
pd.crosstab(y,lgpredict)
```

Out[68]:

col_0	0	1
y		
0	39146	776
1	4182	1107

In [69]:

```
from sklearn.metrics import classification_report,plot_roc_curve
```

In [70]:

```
print(classification_report(y,lgpredict))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94     39922
           1       0.59      0.21      0.31      5289

    accuracy                           0.89     45211
   macro avg       0.75      0.59      0.62     45211
weighted avg       0.87      0.89      0.87     45211
```

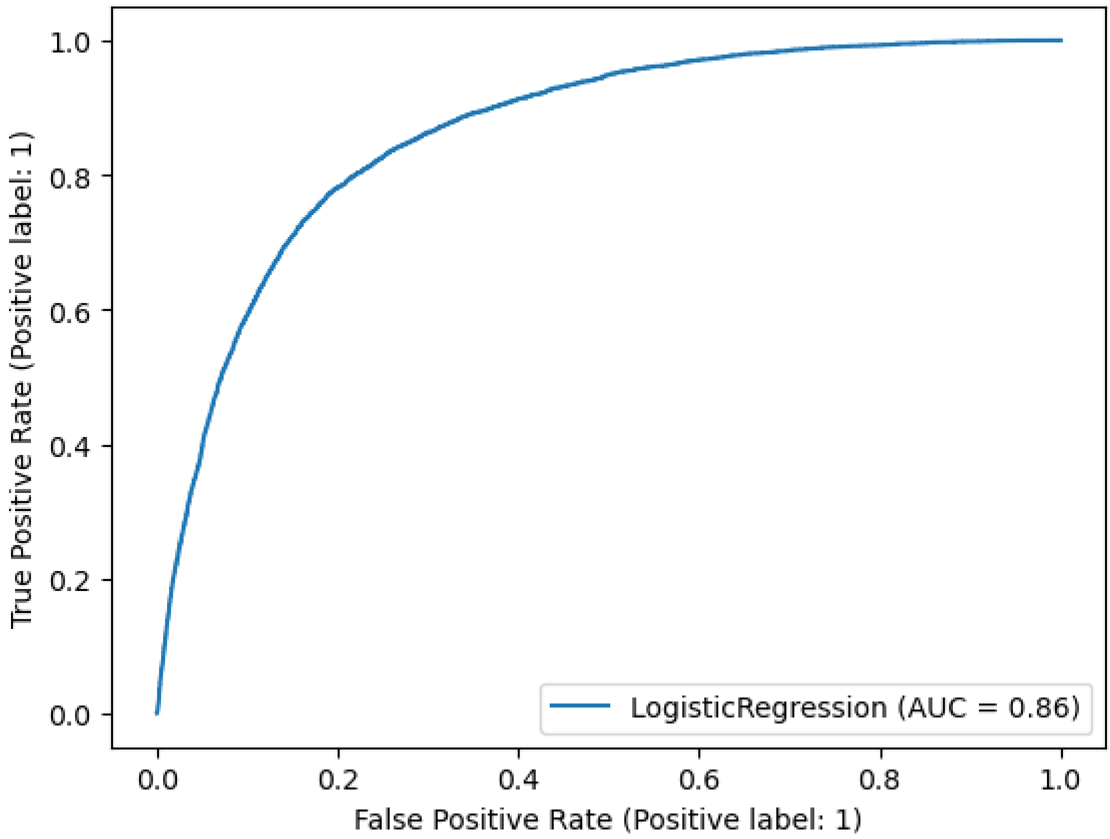
In [71]:

```
plot_roc_curve(lg,X,y)
```

```
C:\Users\RELIANCE\anaconda3\lib\site-packages\sklearn\utils\deprecation.p
y:87: FutureWarning: Function plot_roc_curve is deprecated; Function :fun
c:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use on
e of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predict
ions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

Out[71]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x25b07a49610>
```



In [72]:

```
# Decision Tree

from sklearn.tree import DecisionTreeClassifier
```

In [74]:

```
dc=DecisionTreeClassifier(max_depth=10)
```

In [75]:

```
dcmodel=dc.fit(X,y)
```

In [76]:

```
dcmodel.score(X,y)
```

Out[76]:

```
0.9267656101391254
```

In [77]:

```
dcpredict=dcmodel.predict(X)
```

In [78]:

```
dcresudical=y-dcpredict
```

In [79]:

```
pd.crosstab(y,dcpredict)
```

Out[79]:

col_0	0	1
y		
0	39122	800
1	2511	2778

In [80]:

```
print(classification_report(y,dcpredict))
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96     39922
           1       0.78      0.53      0.63      5289

    accuracy                           0.93     45211
   macro avg       0.86      0.75      0.79     45211
weighted avg       0.92      0.93      0.92     45211
```

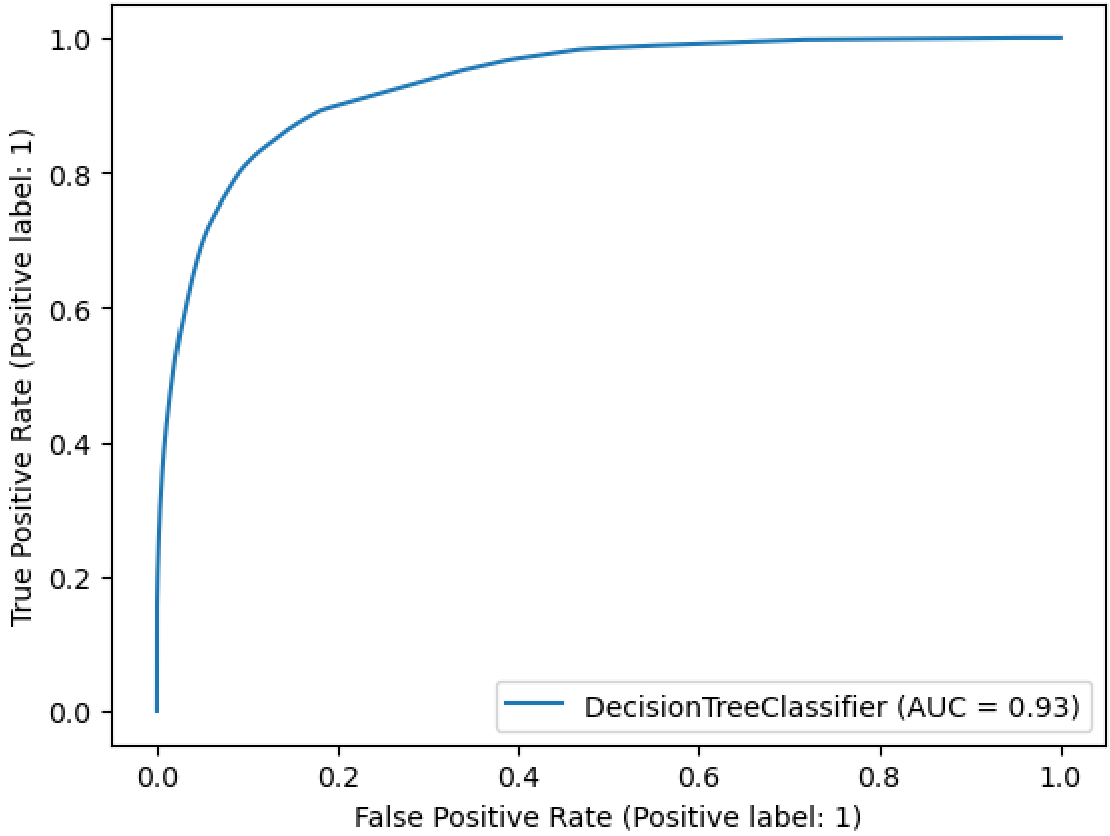
In [81]:

```
plot_roc_curve(dc,X,y)
```

```
C:\Users\RELIANCE\anaconda3\lib\site-packages\sklearn\utils\deprecation.p
y:87: FutureWarning: Function plot_roc_curve is deprecated; Function :fun
c:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use on
e of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predict
ions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

Out[81]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x25b06b64d30>
```



In [82]:

```
# Random tree
from sklearn.ensemble import RandomForestClassifier
```

In [83]:

```
rf=RandomForestClassifier(max_depth=8)
```

In [84]:

```
rfmodel=rf.fit(X,y)
```

In [85]:

```
rfmodel.score(X,y)
```

Out[85]:

```
0.9138041627037667
```

In [86]:

```
rfpredict=rfmodel.predict(X)
```

In [87]:

```
rfresudical=y-rfpredict
```

In [88]:

```
pd.crosstab(y,rfpredict)
```

Out[88]:

col_0	0	1
y		
0	39515	407
1	3490	1799

In [89]:

```
print(classification_report(y,rfpredict))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     39922
           1       0.82      0.34      0.48      5289

    accuracy                           0.91     45211
   macro avg       0.87      0.66      0.72     45211
weighted avg       0.91      0.91      0.90     45211
```

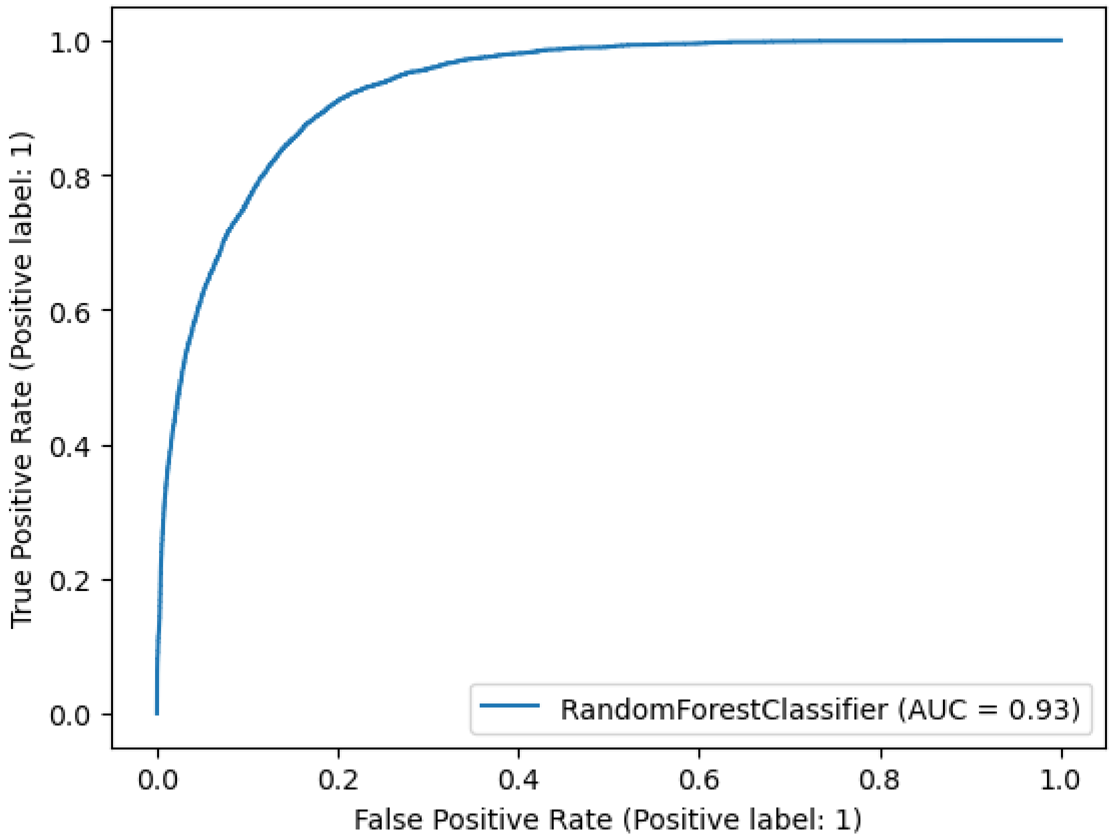
In [90]:

```
plot_roc_curve(rf,X,y)
```

```
C:\Users\RELIANCE\anaconda3\lib\site-packages\sklearn\utils\deprecation.p
y:87: FutureWarning: Function plot_roc_curve is deprecated; Function :fun
c:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use on
e of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predict
ions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

Out[90]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x25b0122e310>
```



In [91]:

```
#Gradient Boosting

from sklearn.ensemble import GradientBoostingClassifier
```

In [92]:

```
gb=GradientBoostingClassifier(max_depth=8)
```

In [93]:

```
gbmodel=gb.fit(X,y)
```

In [94]:

```
gbmodel.score(X,y)
```

Out[94]:

```
0.9632832717701444
```

In [95]:

```
gbpredict=gbmodel.predict(X)
```

In [96]:

```
gbresiduiacal=y-gbpredict
```

In [97]:

```
pd.crosstab(y,gbpredict)
```

Out[97]:

col_0	0	1
y		
0	39584	338
1	1322	3967

In [98]:

```
print(classification_report(y,gbpredict))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     39922
           1       0.92      0.75      0.83      5289

    accuracy                           0.96     45211
   macro avg       0.94      0.87      0.90     45211
weighted avg       0.96      0.96      0.96     45211
```

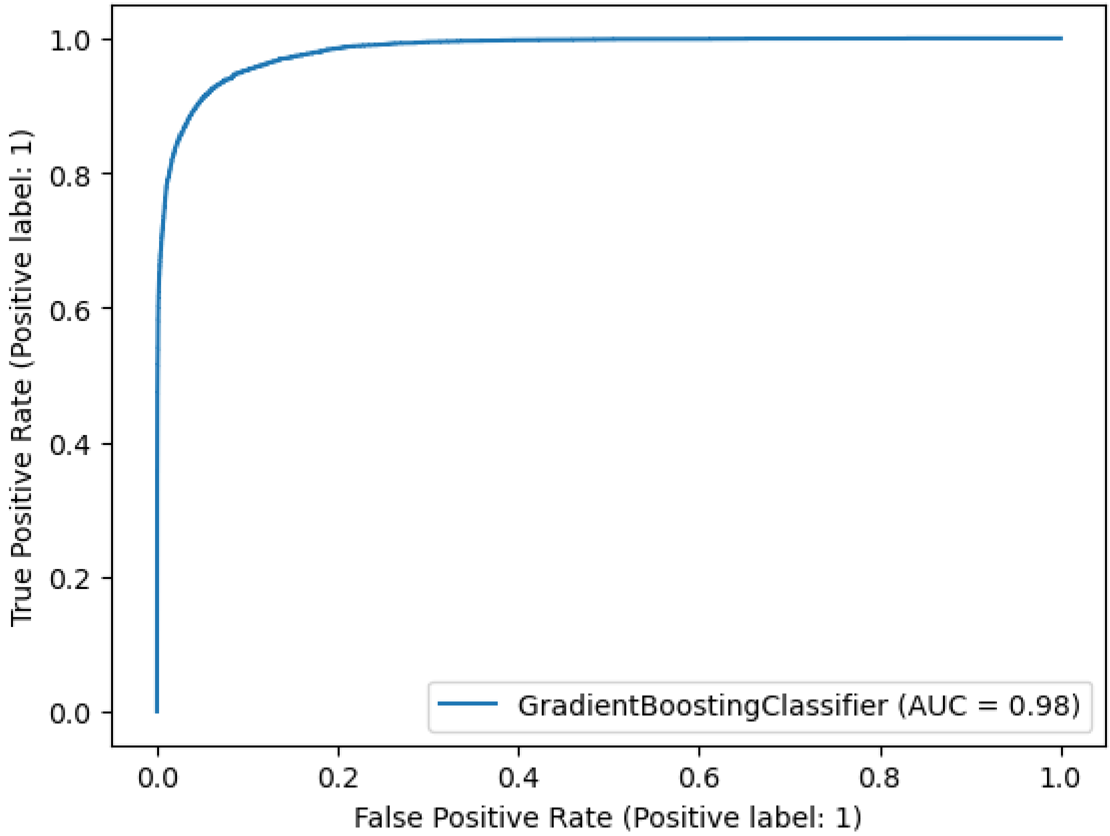
In [99]:

```
plot_roc_curve(gb,X,y)
```

```
C:\Users\RELIANCE\anaconda3\lib\site-packages\sklearn\utils\deprecation.p
y:87: FutureWarning: Function plot_roc_curve is deprecated; Function :fun
c:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use on
e of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predict
ions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

Out[99]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x25b05ea3250>
```



In [100]:

```
#Naïve Bayes

from sklearn.naive_bayes import BernoulliNB
```

In [101]:

```
nbber=BernoulliNB()
```

In [102]:

```
nbbermodel=nbber.fit(X,y)
```

In [103]:

```
nbbermodel.score(X,y)
```

Out[103]:

```
0.8674216451748469
```

In [104]:

```
nbberpredict=nbbermodel.predict(X)
```

In [105]:

```
pd.crosstab(y,nbberpredict)
```

Out[105]:

col_0	0	1
y		
0	38095	1827
1	4167	1122

In [106]:

```
print(classification_report(y,nbberpredict))
```

```
              precision    recall  f1-score   support

           0       0.90      0.95      0.93     39922
           1       0.38      0.21      0.27      5289

    accuracy                           0.87     45211
   macro avg       0.64      0.58      0.60     45211
weighted avg       0.84      0.87      0.85     45211
```

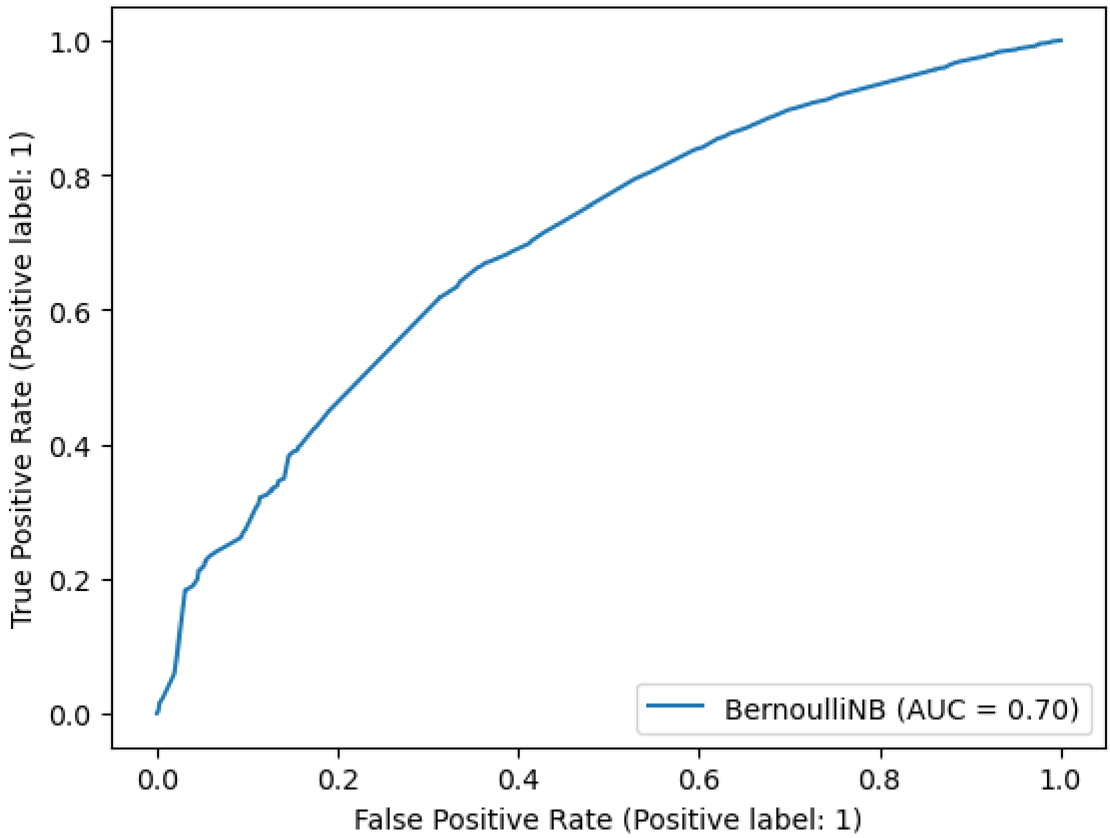
In [107]:

```
plot_roc_curve(nbber,X,y)
```

```
C:\Users\RELIANCE\anaconda3\lib\site-packages\sklearn\utils\deprecation.p
y:87: FutureWarning: Function plot_roc_curve is deprecated; Function :fun
c:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use on
e of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predict
ions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

Out[107]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x25b02c5f490>
```



In [108]:

```
#Support Vector

from sklearn.svm import SVC
```

In [109]:

```
svc=SVC()
```

In [110]:

```
svmodel=svc.fit(X,y)
```

In [111]:

```
svmodel.score(X,y)
```

Out[111]:

```
0.8832363805268629
```

In [112]:

```
#KNN

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
```

In [113]:

```
knn=KNeighborsClassifier()
```

In [114]:

```
k_range=list(range(1,20))
param_grid=dict(n_neighbors=k_range)
grid=GridSearchCV(knn,param_grid,cv=5)
```

In [115]:

```
grid_search=grid.fit(X,y)
```

```
C:\Users\RELIANCE\anaconda3\lib\site-packages\sklearn\neighbors\_classif
ication.py:228: FutureWarning: Unlike other reduction functions (e.g. `s
kew`, `kurtosis`), the default behavior of `mode` typically preserves th
e axis it acts along. In SciPy 1.11.0, this behavior will change: the de
fault value of `keepdims` will become False, the `axis` over which the s
tatistic is taken will be eliminated, and the value None will no longer
be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\RELIANCE\anaconda3\lib\site-packages\sklearn\neighbors\_classif
ication.py:228: FutureWarning: Unlike other reduction functions (e.g. `s
kew`, `kurtosis`), the default behavior of `mode` typically preserves th
e axis it acts along. In SciPy 1.11.0, this behavior will change: the de
fault value of `keepdims` will become False, the `axis` over which the s
tatistic is taken will be eliminated, and the value None will no longer
be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\RELIANCE\anaconda3\lib\site-packages\sklearn\neighbors\_classif
ication.py:228: FutureWarning: Unlike other reduction functions (e.g. `s
kew`, `kurtosis`), the default behavior of `mode` typically preserves th
```

In [116]:

```
grid_search.best_params_
```

Out[116]:

```
{'n_neighbors': 16}
```

In [117]:

```
grid_search.best_score_
```

Out[117]:

```
0.8792768327039703
```

In [ ]: