Step-1: Import Libraries

In [80]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Step-2: Read the Data

In [81]:
```
bank_df=pd.read_csv(r"F:\FSDS\Data Files\bank.csv", sep=';')
```

In [82]:
```
bank_df
```

Out[82]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may
...	...	...	...	...	...	...	...	...	...	...	...
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	may
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr

4521 rows × 17 columns

Step-3: Data quick check

In [83]:
```
bank_df.head()
```

Out[83]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	du
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	

In [84]:
```
bank_df.tail()
```

Out[84]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	may
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr

In [85]:
```
bank_df.shape
```

Out[85]:
```
(4521, 17)
```

In [86]:
```
bank_df.columns
```

Out[86]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

In [87]:
```
bank_df.dtypes
```

Out[87]:
```
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
y            object
dtype: object
```

Step-4: Null value Analysis

- check if Null values are present
  - Fill the null values with Median or KNN- Imputer for Numerical column
  - Fill the null values with mode for categorical columns

In [88]:
```
bank_df.isnull().sum()
# there is no missing values
```

Out[88]:
```
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
y            0
dtype: int64
```

Step-5: Do some data preprocessing

- if any column corrupted
- ex. Numerical values in categorical columns

- ex. Categorical values in numerical columns

Step-6: Drop the id type columns

- Which means a data has more unique labels
- Drop the single value columns

In [89]:
```
bank_df.head()
```

Out[89]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	du
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	

Step-7: Categorical column Analysis

In [90]:
```
categorical=bank_df.select_dtypes(include='object').columns
categorical
```

Out[90]:
```
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
       'month', 'poutcome', 'y'],
      dtype='object')
```

Frequency Table

In [91]:
```
unique=bank_df['job'].unique()
unique
```

Out[91]:
```
array(['unemployed', 'services', 'management', 'blue-collar',
       'self-employed', 'technician', 'entrepreneur', 'admin.', 'student',
       'housemaid', 'retired', 'unknown'], dtype=object)
```

In [92]:
```
bank_df[['job']].value_counts()
```

Out[92]:
```
job
management       969
blue-collar      946
technician       768
admin.           478
services         417
retired          230
self-employed    183
entrepreneur     168
unemployed       128
housemaid        112
student           84
unknown           38
dtype: int64
```

In [93]:
```
count=[]
for i in unique:
    con=bank_df['job']==i
    count.append(len(bank_df[con]))

count
```

Out[93]:
```
[128, 417, 969, 946, 183, 768, 168, 478, 84, 112, 230, 38]
```

In [94]:
```
df=pd.DataFrame(zip(unique, count), columns=['labels','count'])
df
```

Out[94]:

	labels	count
0	unemployed	128
1	services	417
2	management	969
3	blue-collar	946
4	self-employed	183
5	technician	768
6	entrepreneur	168
7	admin.	478
8	student	84
9	housemaid	112
10	retired	230
11	unknown	38

In [95]:
```
# creating a new folder
import os
new_folder='Project3'
cwd=os.getcwd()
new_dir=os.path.join(cwd, new_folder)
try:
    os.makedirs(new_dir)
```

```
except Exception as e:
    print(e)
```

```
[WinError 183] Cannot create a file when that file already exists: 'C:\\Users\\NEHA
\\Project3'
```

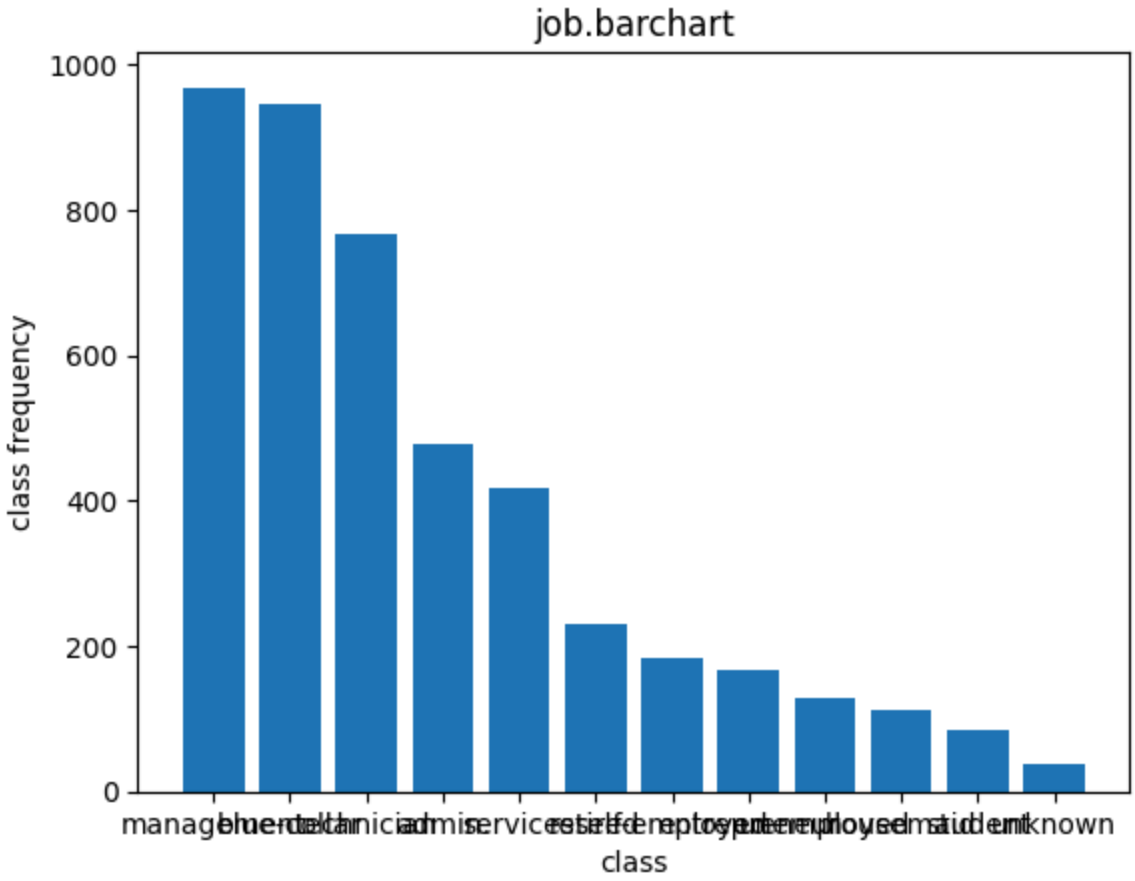
In [96]:
```
for i in categorical:
    keys=bank_df[i].value_counts().keys()
    values=bank_df[i].value_counts().values
    cols=['labels', 'count']
    df1= pd.DataFrame(zip(keys, values),columns=cols)
    df1
    file_name=f'{i}_table.csv'
    new_path1=os.path.join(new_dir, file_name)
    df1.to_csv(new_path1)
```

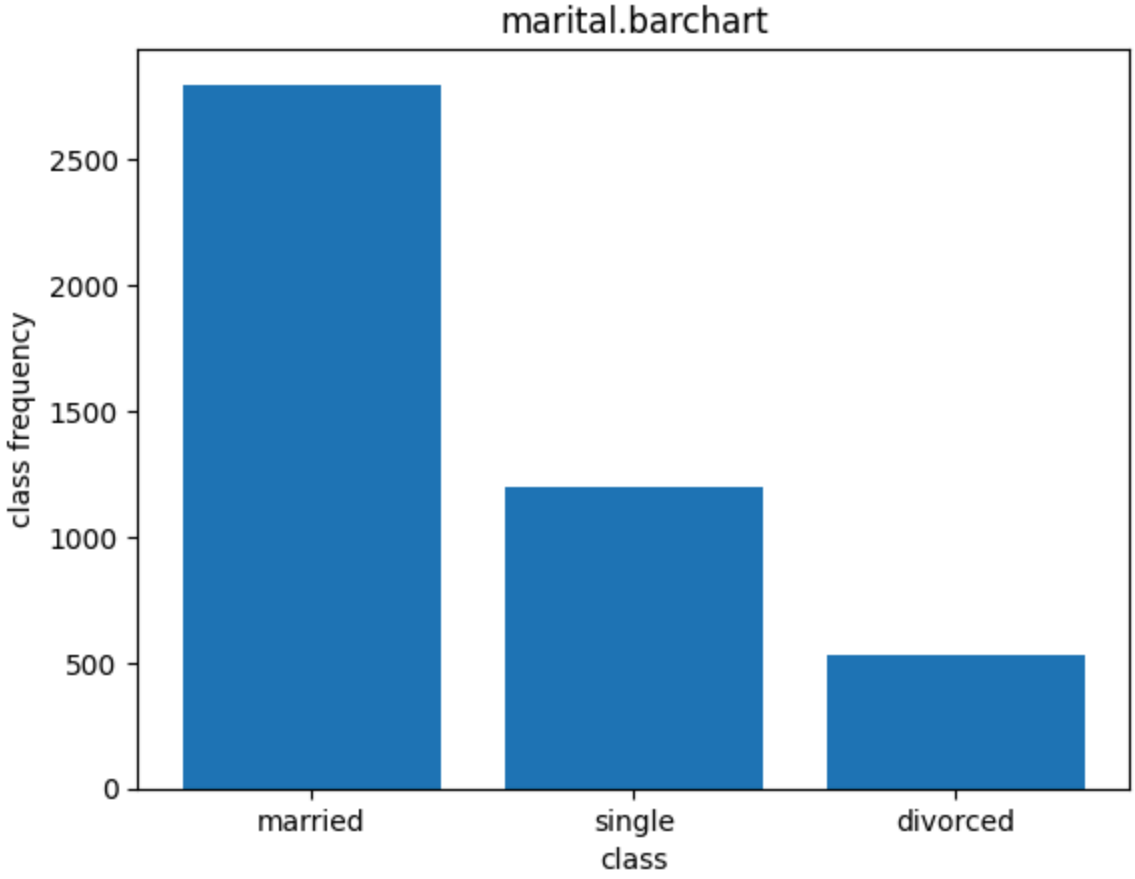
Barchart

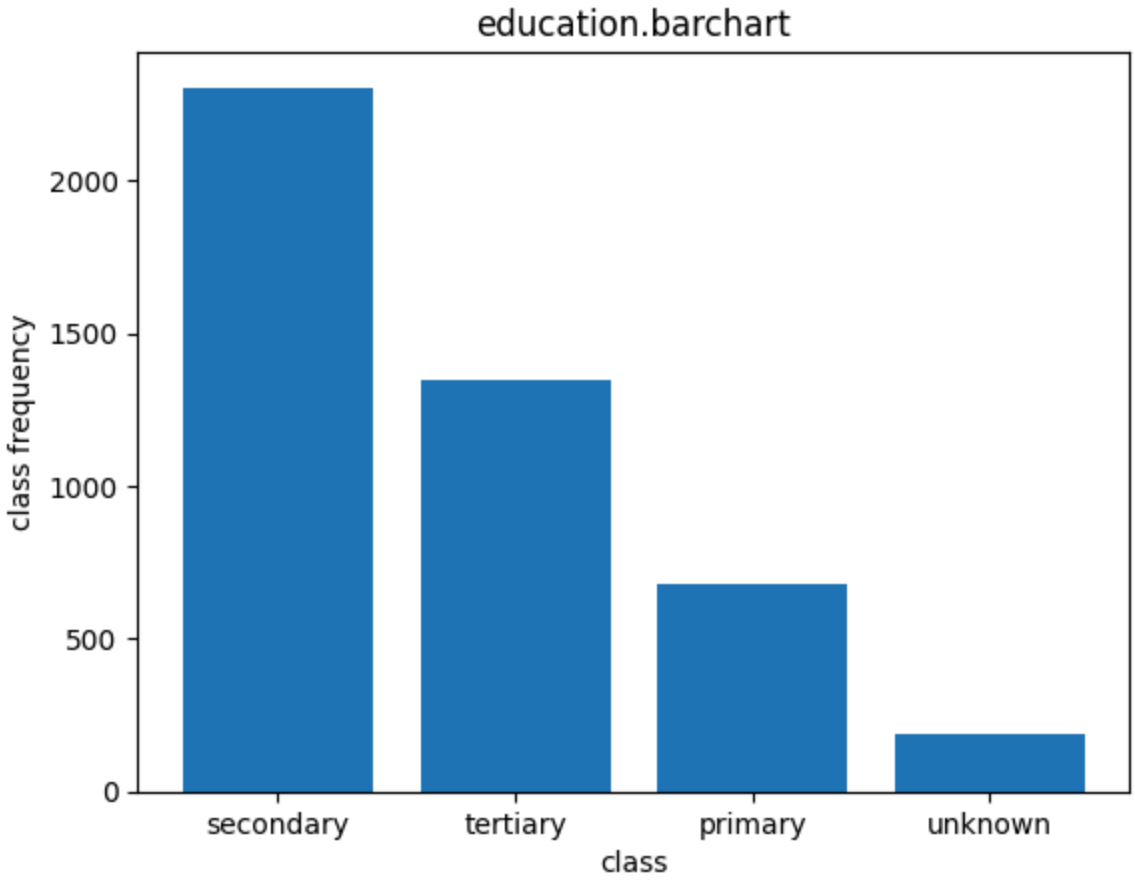
In [97]:
```
categorical
```

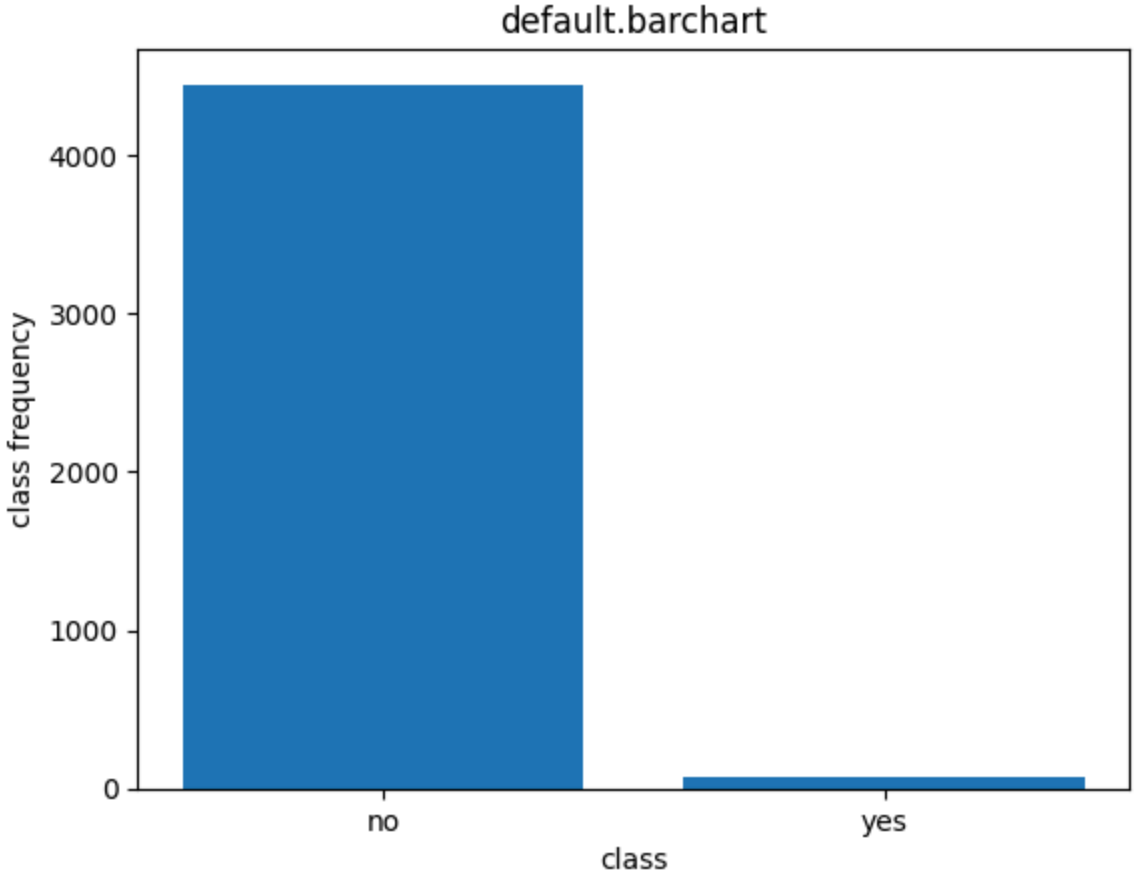
Out[97]:
```
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
       'month', 'poutcome', 'y'],
      dtype='object')
```

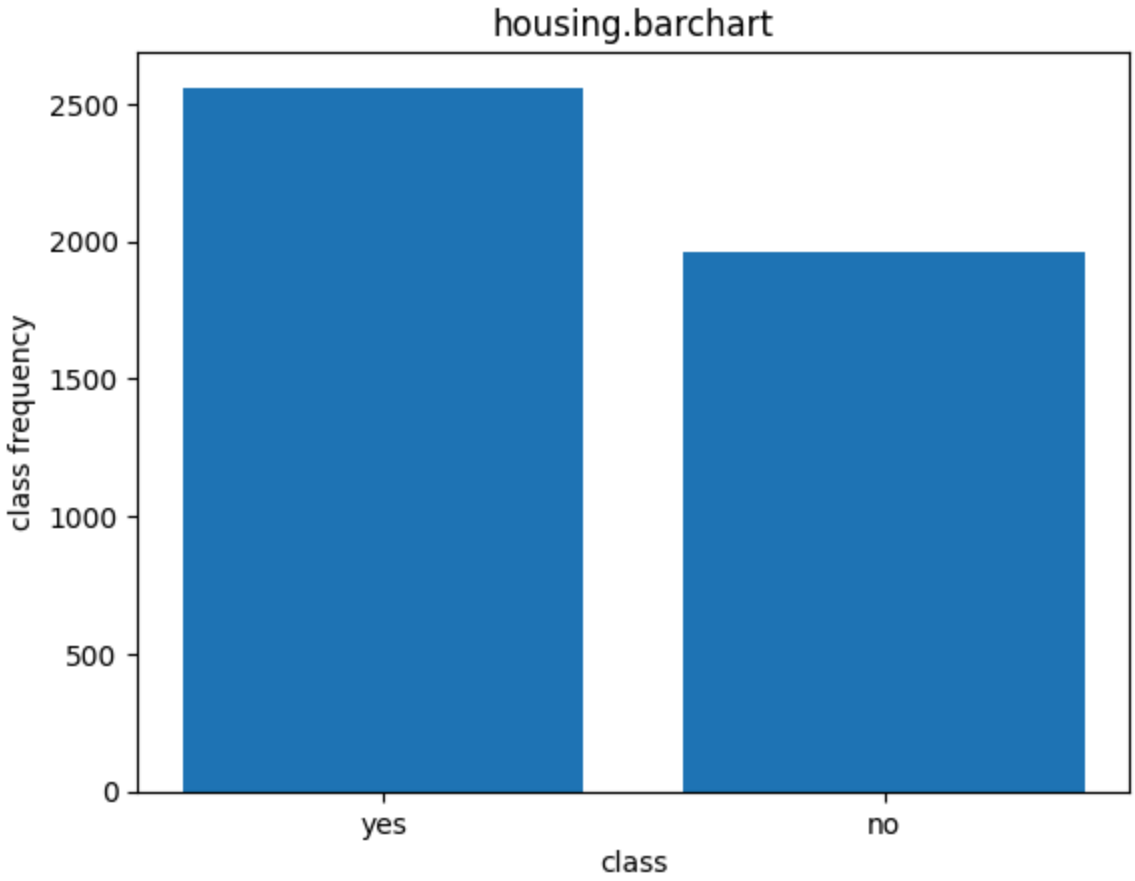
In [98]:
```
for i in categorical:
    keys=bank_df[i].value_counts().keys()
    values=bank_df[i].value_counts().values
    plt.bar(keys, values)
    plt.title(f'{i}.barchart')
    plt.xlabel('class')
    plt.ylabel('class frequency')
    plt.savefig('i.png')
    plt.show()
```

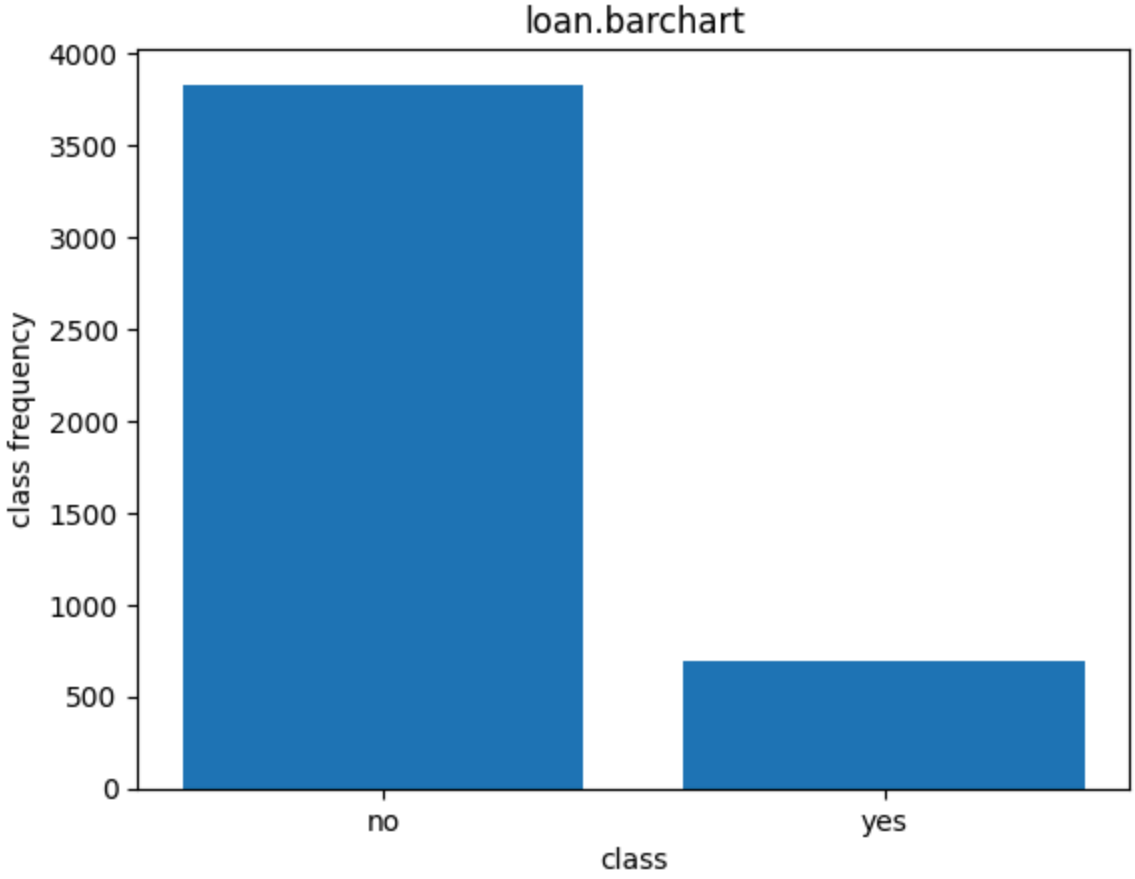

job.barchart
class frequency
1000
800
600
400
200
0
class

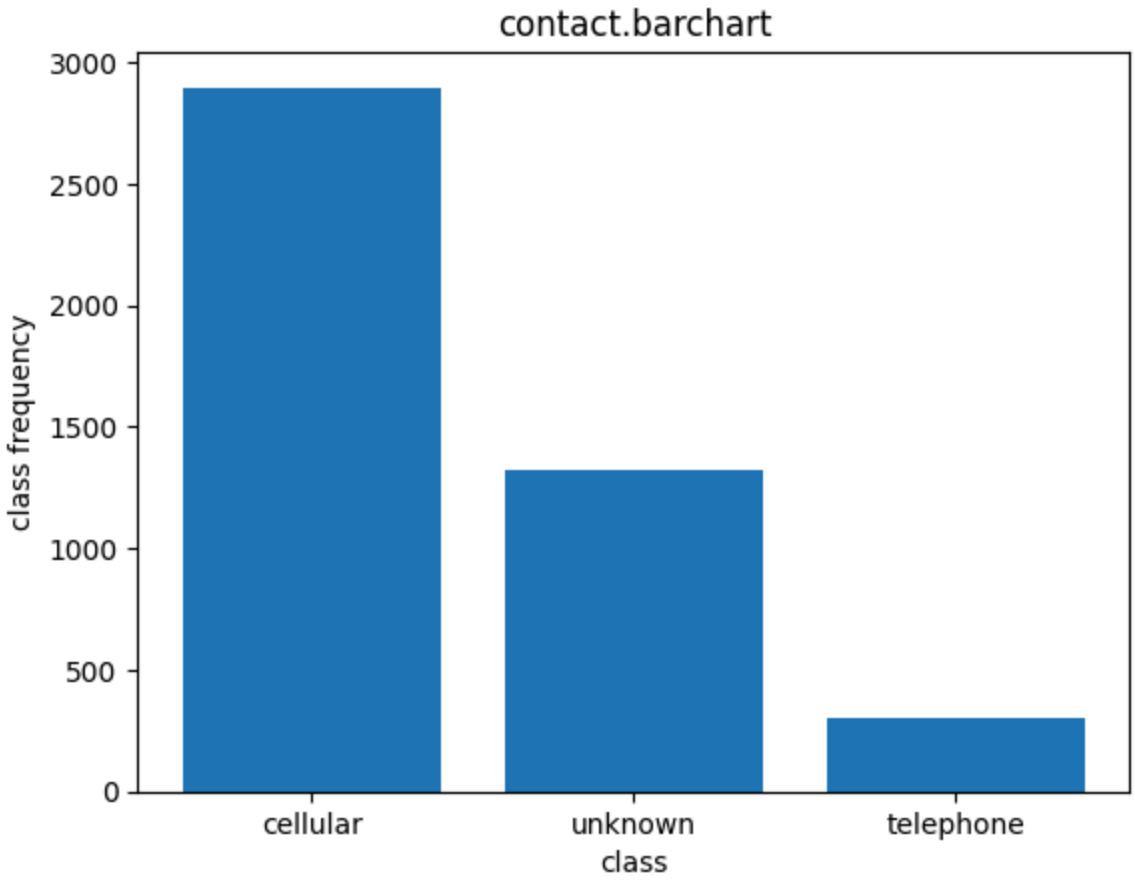

marital.barchart
class frequency
2500
2000
1500
1000
500
0
married
single
divorced
class

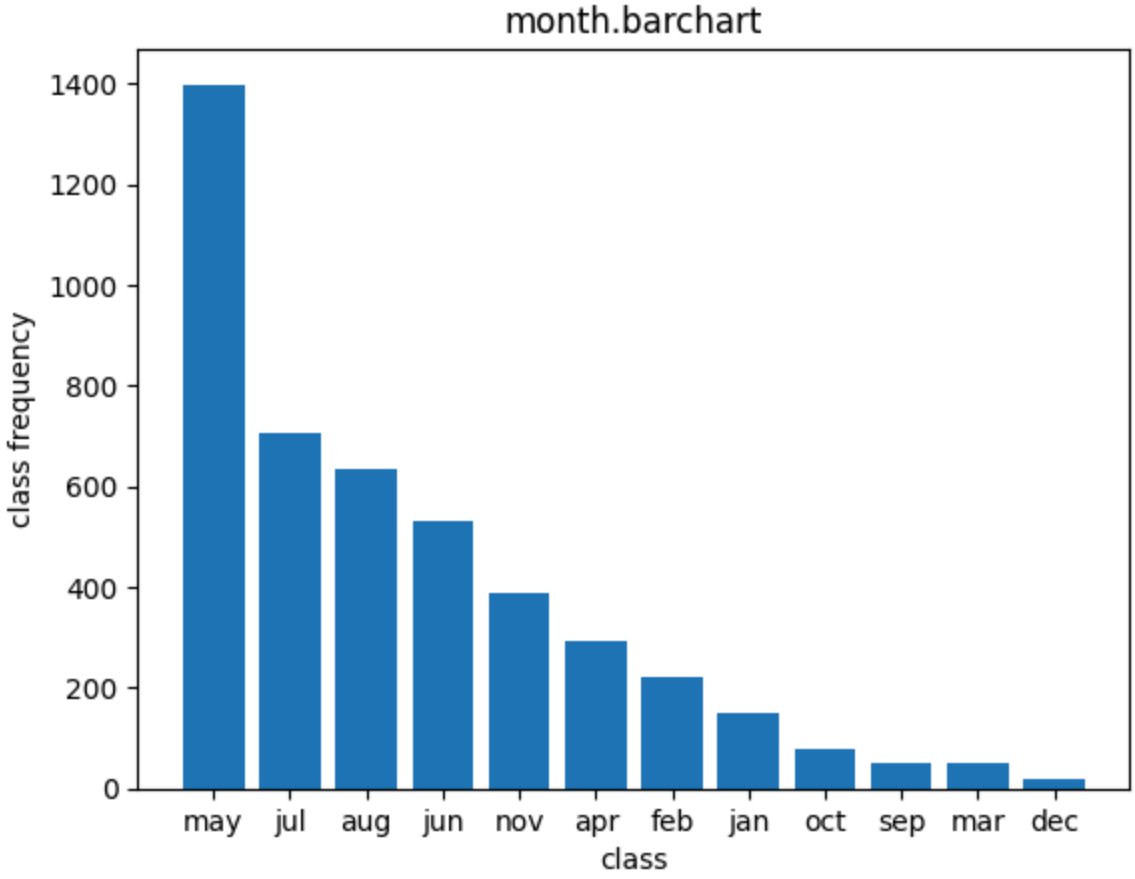

education.barchart
class frequency
2000
1500
1000
500
0
secondary
tertiary
primary
unknown
class

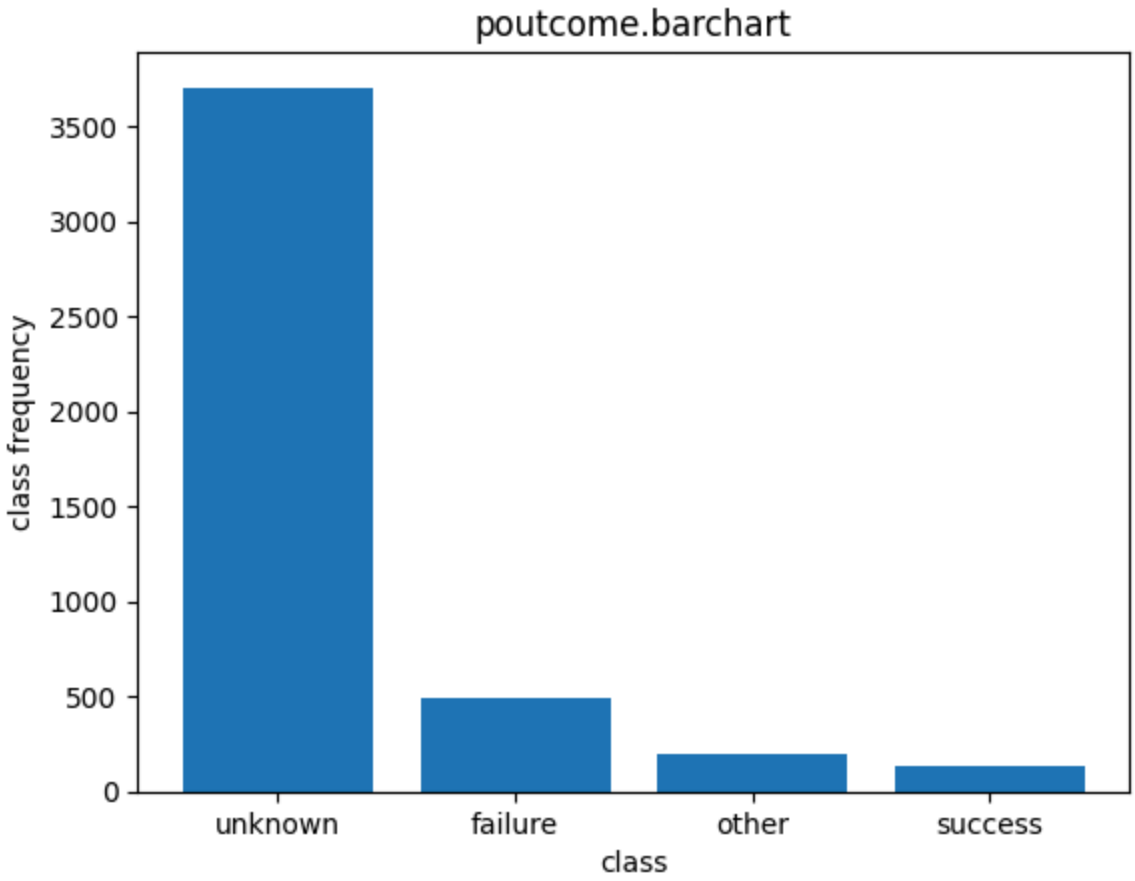

default.barchart
class frequency
4000
3000
2000
1000
0
no
yes
class

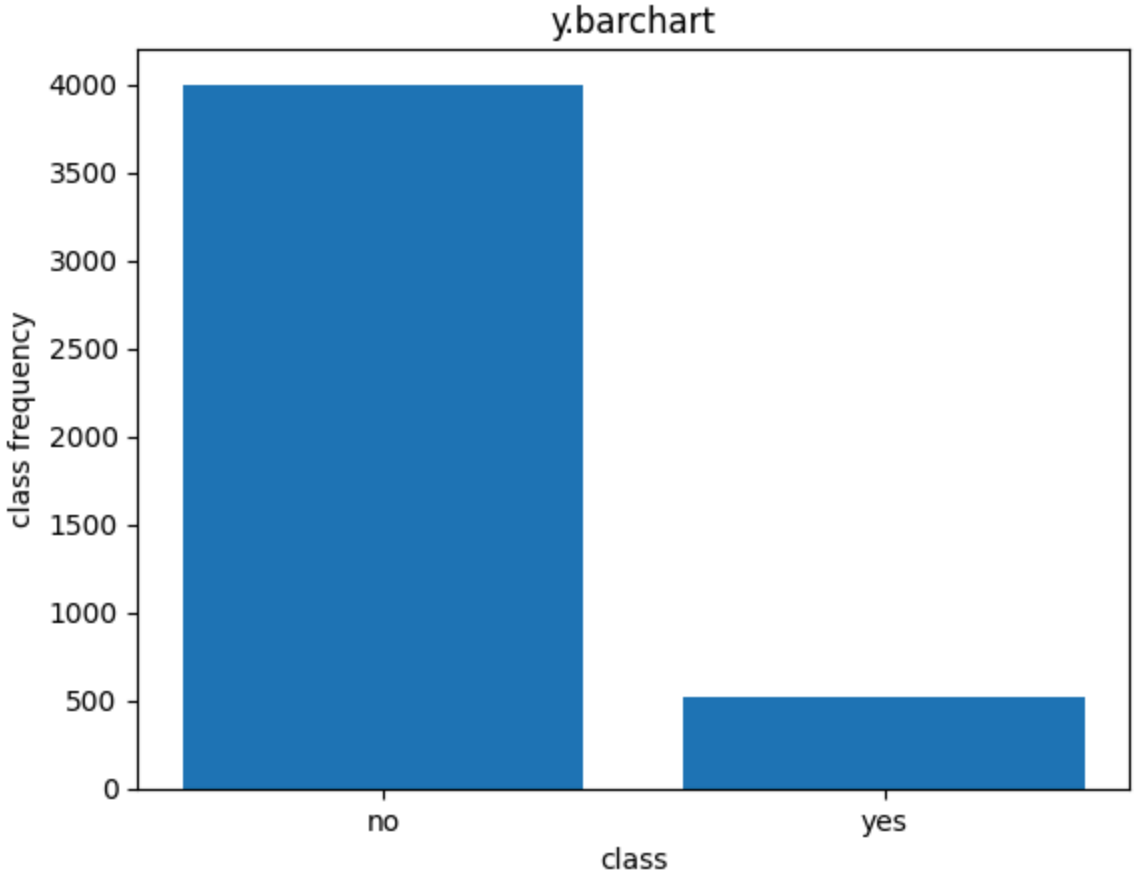

housing.barchart
2500
2000
1500
1000
500
0
class frequency
yes
no
class


loan.barchart
4000
3500
3000
2500
2000
1500
1000
500
0
class frequency
no
yes
class


contact.barchart
3000
2500
2000
1500
1000
500
0
class frequency
cellular
unknown
telephone
class

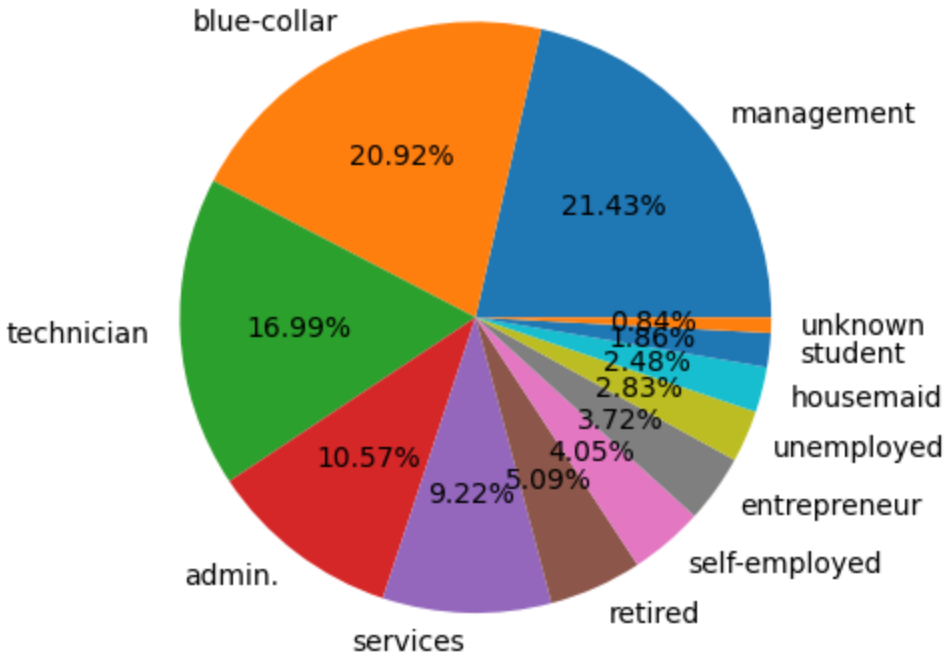

month.barchart
1400
1200
1000
800
600
400
200
0
class frequency
may
jul
aug
jun
nov
apr
feb
jan
oct
sep
mar
dec
class

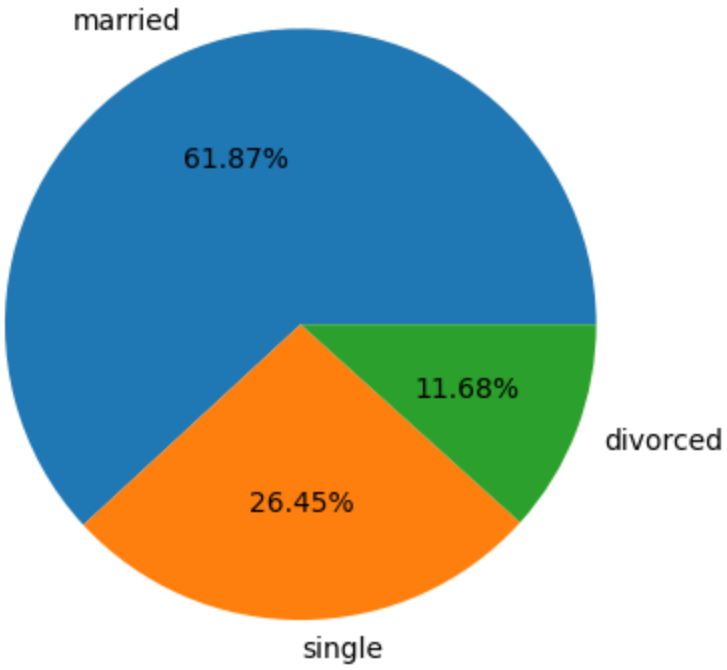

poutcome.barchart
class frequency
0
500
1000
1500
2000
2500
3000
3500
unknown
failure
other
success
class

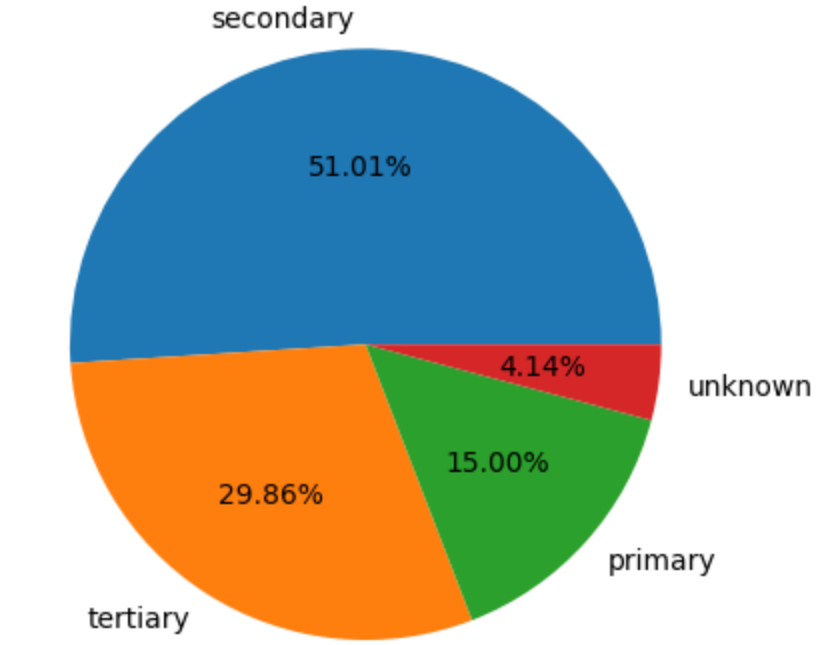

y.barchart
class frequency
0
500
1000
1500
2000
2500
3000
3500
4000
no
yes
class

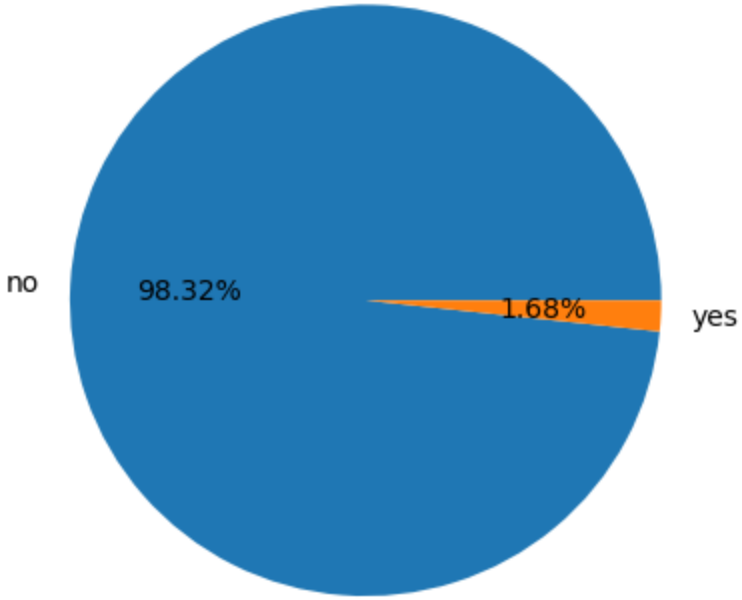
Piechart

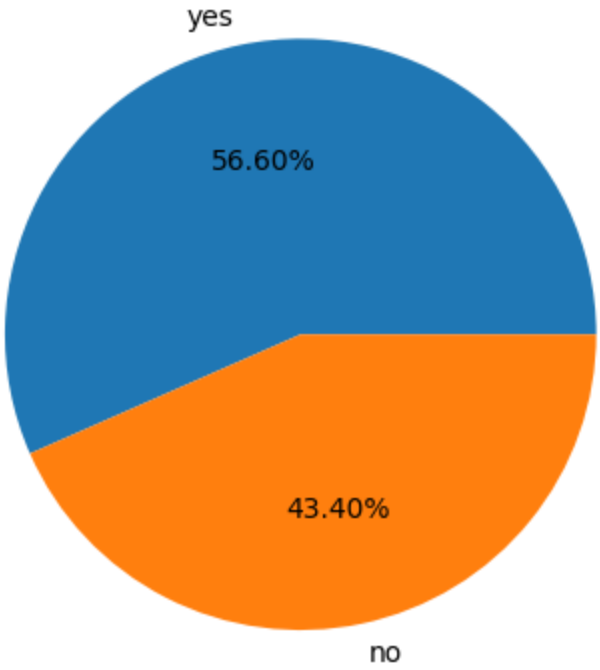
In [99]:
```
for i in categorical:
    keys=bank_df[i].value_counts().keys()
    values=bank_df[i].value_counts().values
    plt.pie(x=values, labels=keys, autopct='%0.2f%%')
    plt.savefig('i.png')
    plt.show()
```

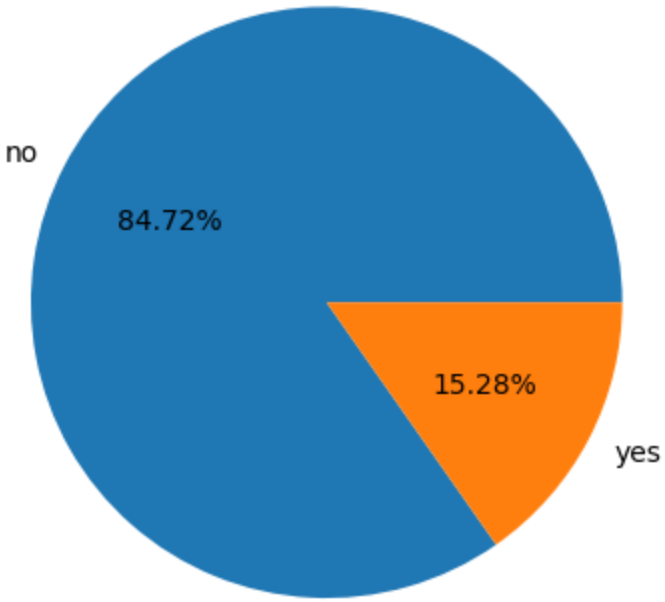


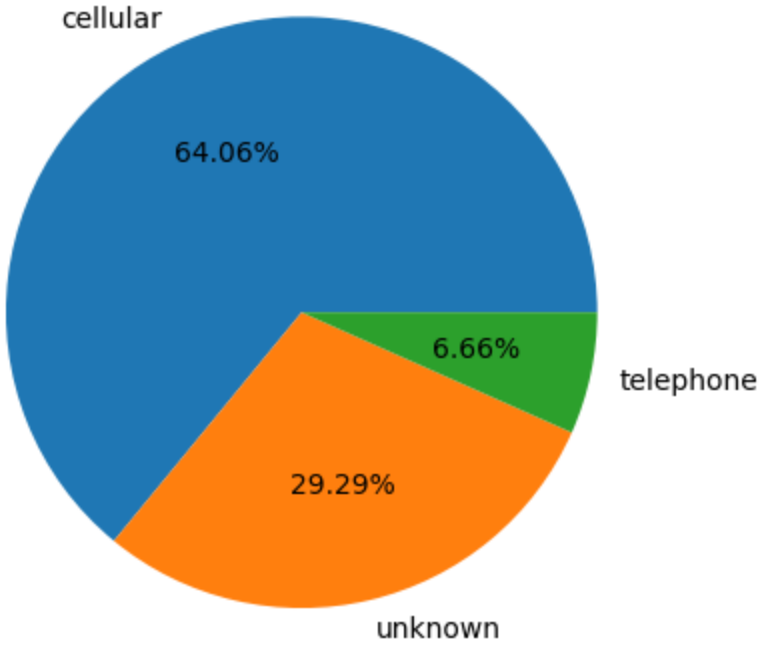


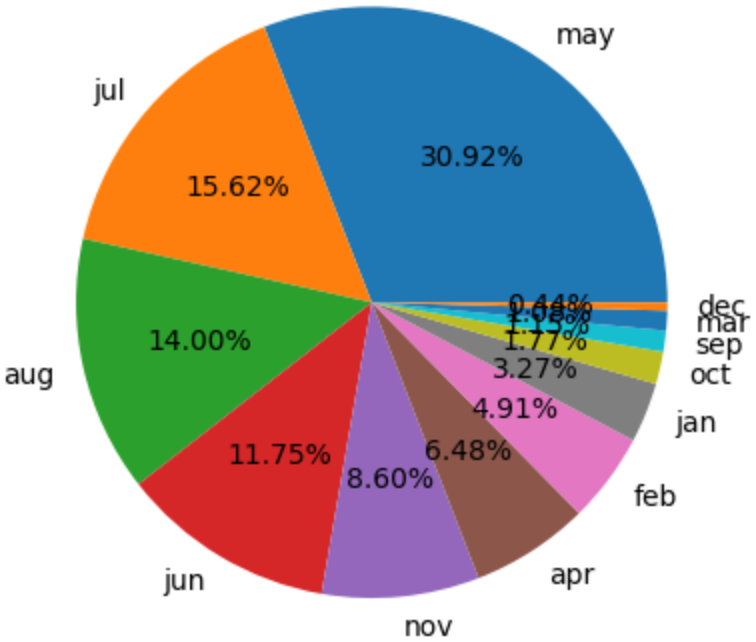

secondary
51.01%
4.14%
unknown
15.00%
29.86%
primary
tertiary

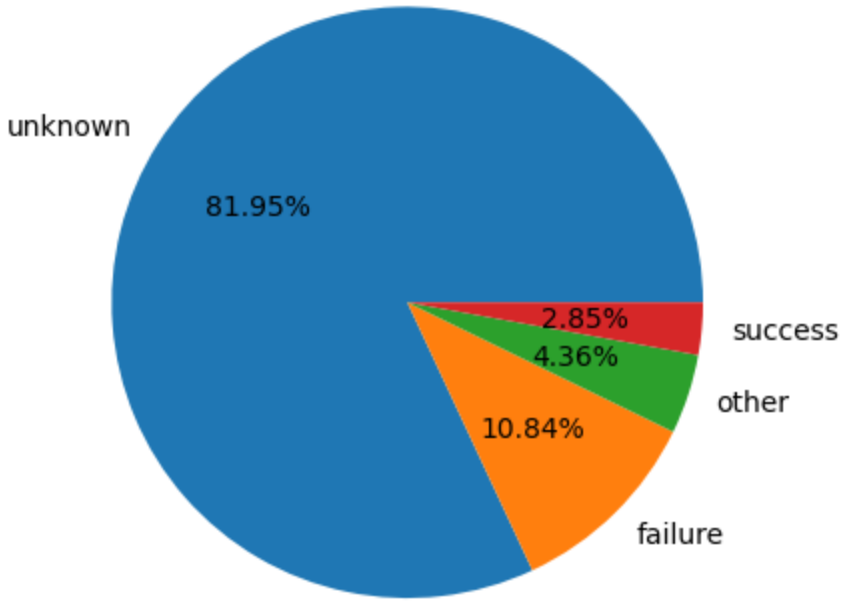

no
98.32%
1.68%
yes

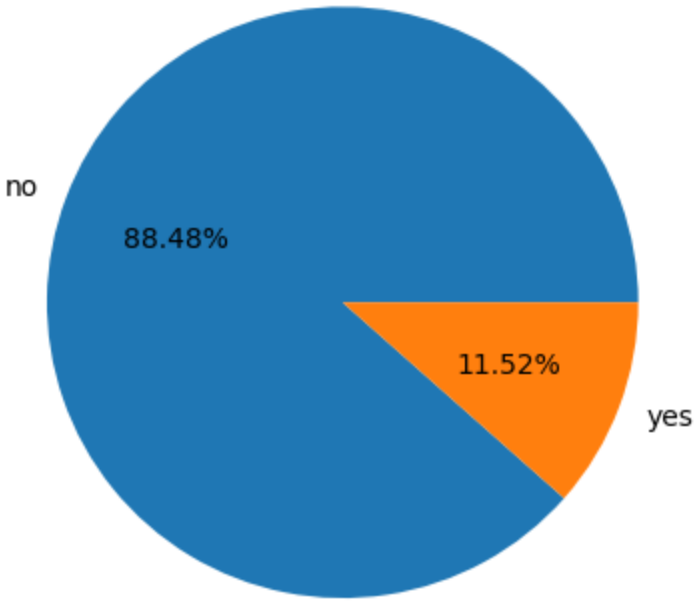

yes
56.60%
43.40%
no


no
84.72%
15.28%
yes


cellular
64.06%
6.66%
telephone
29.29%
unknown


may
30.92%
jul
15.62%
aug
14.00%
jun
11.75%
nov
8.60%
apr
6.48%
feb
4.91%
jan
3.27%
oct
1.77%
sep
mar
dec





Step-8: Numerical column Analysis

In [100…
```
numerical=bank_df.select_dtypes(exclude='object').columns
numerical
```

Out[100]:
```
Index(['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous'], dtype
='object')
```

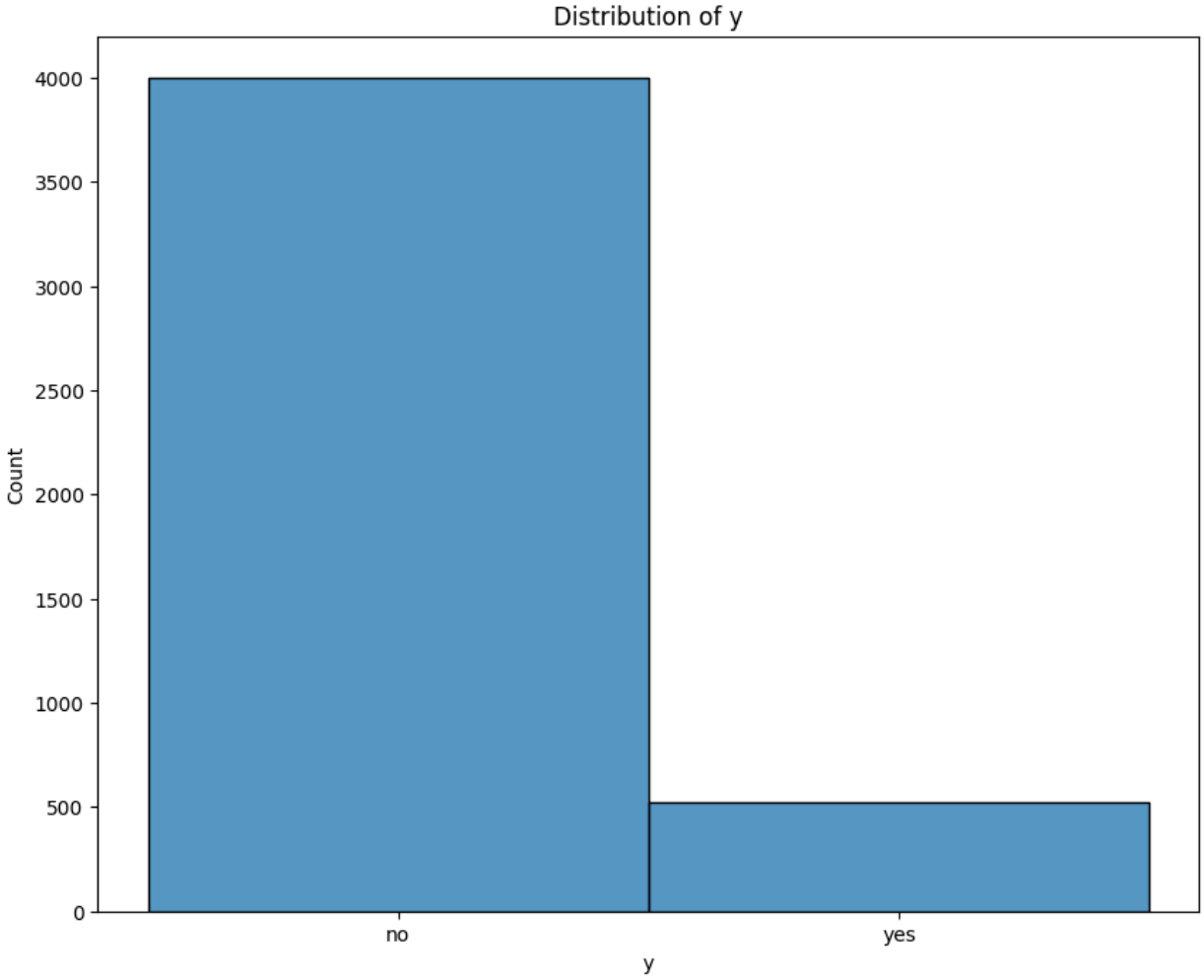
In [101…
```
bank_df.describe()
```

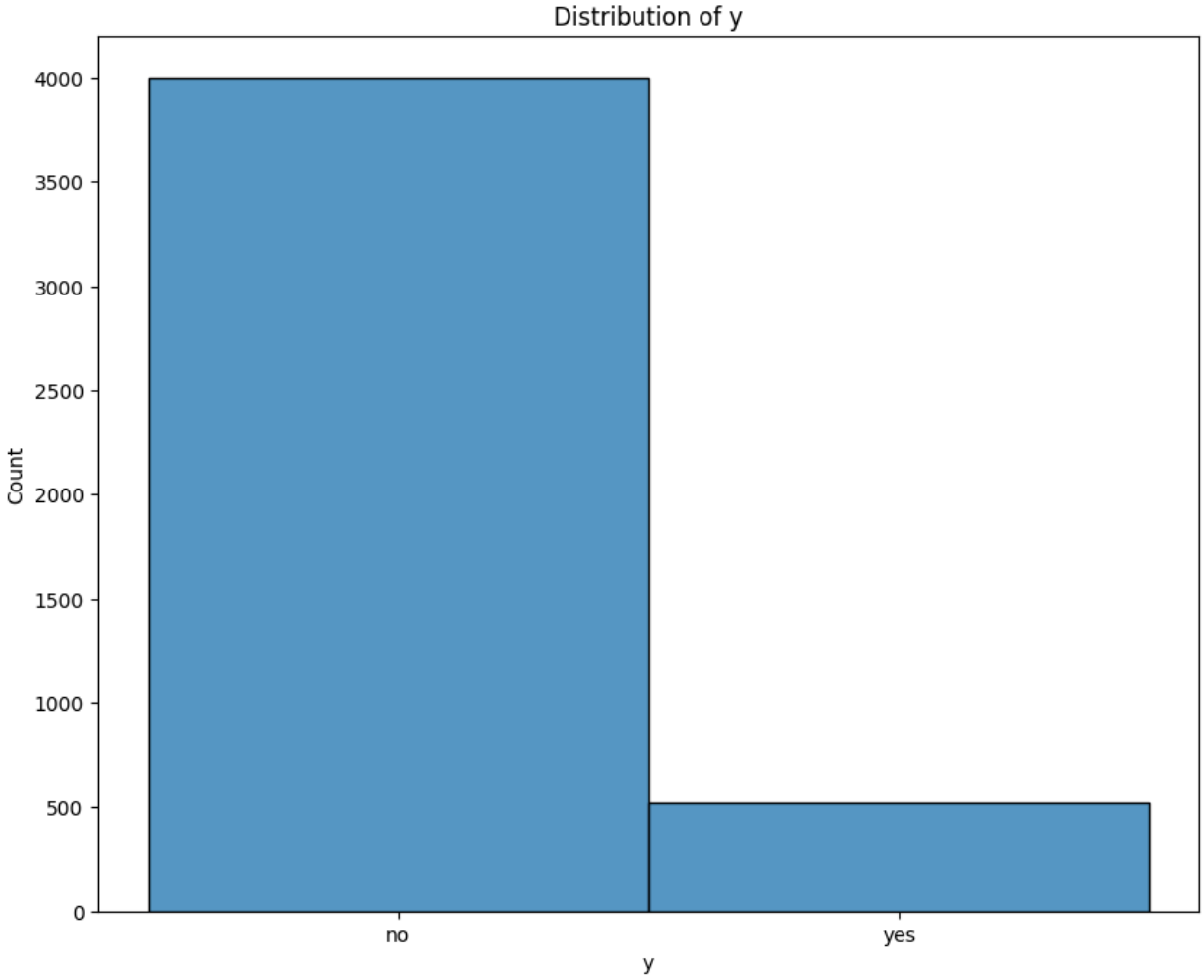
Out[101]:

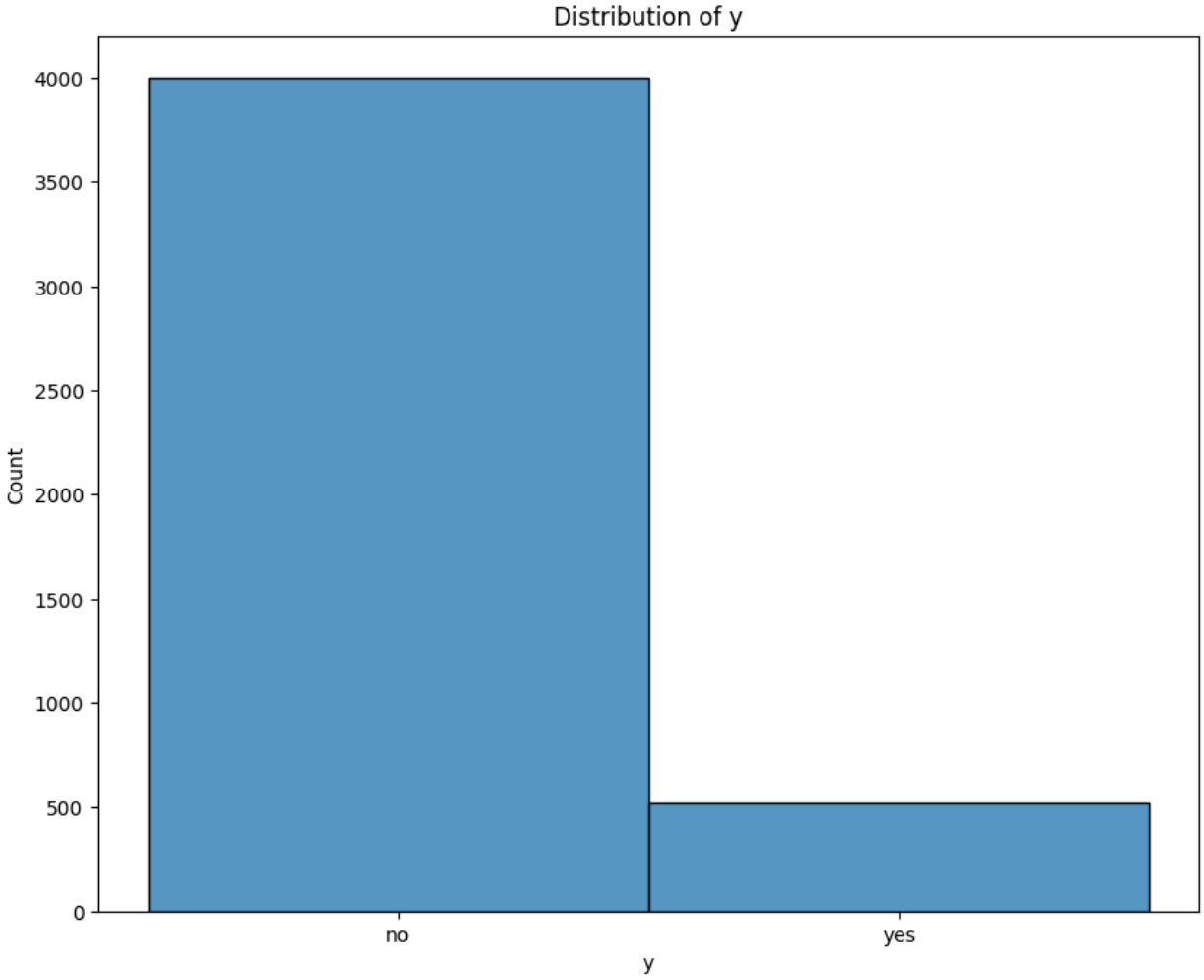
	age	balance	day	duration	campaign	pdays	previous
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000
mean	41.170095	1422.657819	15.915284	263.961292	2.793630	39.766645	0.542579
std	10.576211	3009.638142	8.247667	259.856633	3.109807	100.121124	1.693562
min	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.000000
25%	33.000000	69.000000	9.000000	104.000000	1.000000	-1.000000	0.000000
50%	39.000000	444.000000	16.000000	185.000000	2.000000	-1.000000	0.000000
75%	49.000000	1480.000000	21.000000	329.000000	3.000000	-1.000000	0.000000
max	87.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.000000

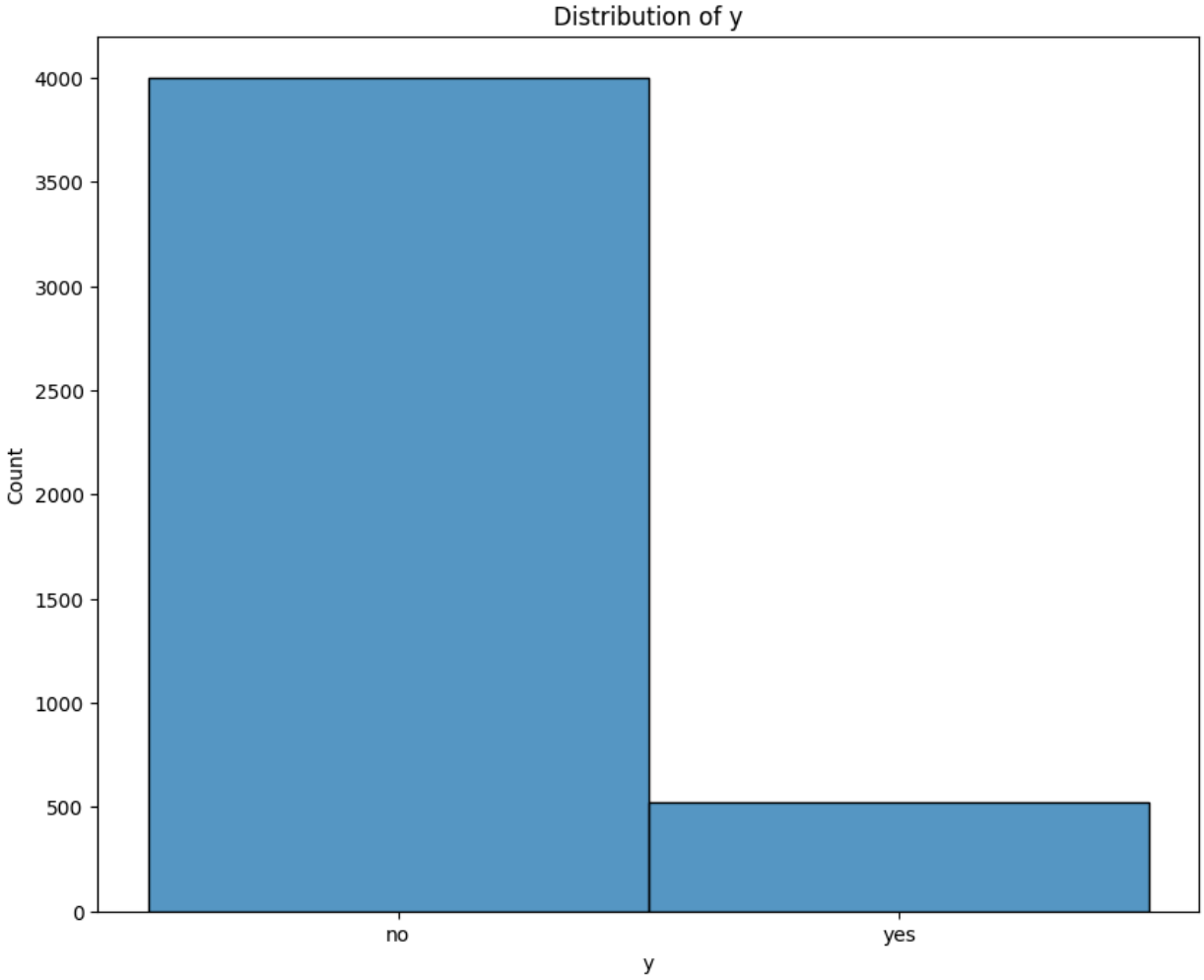
Histplot

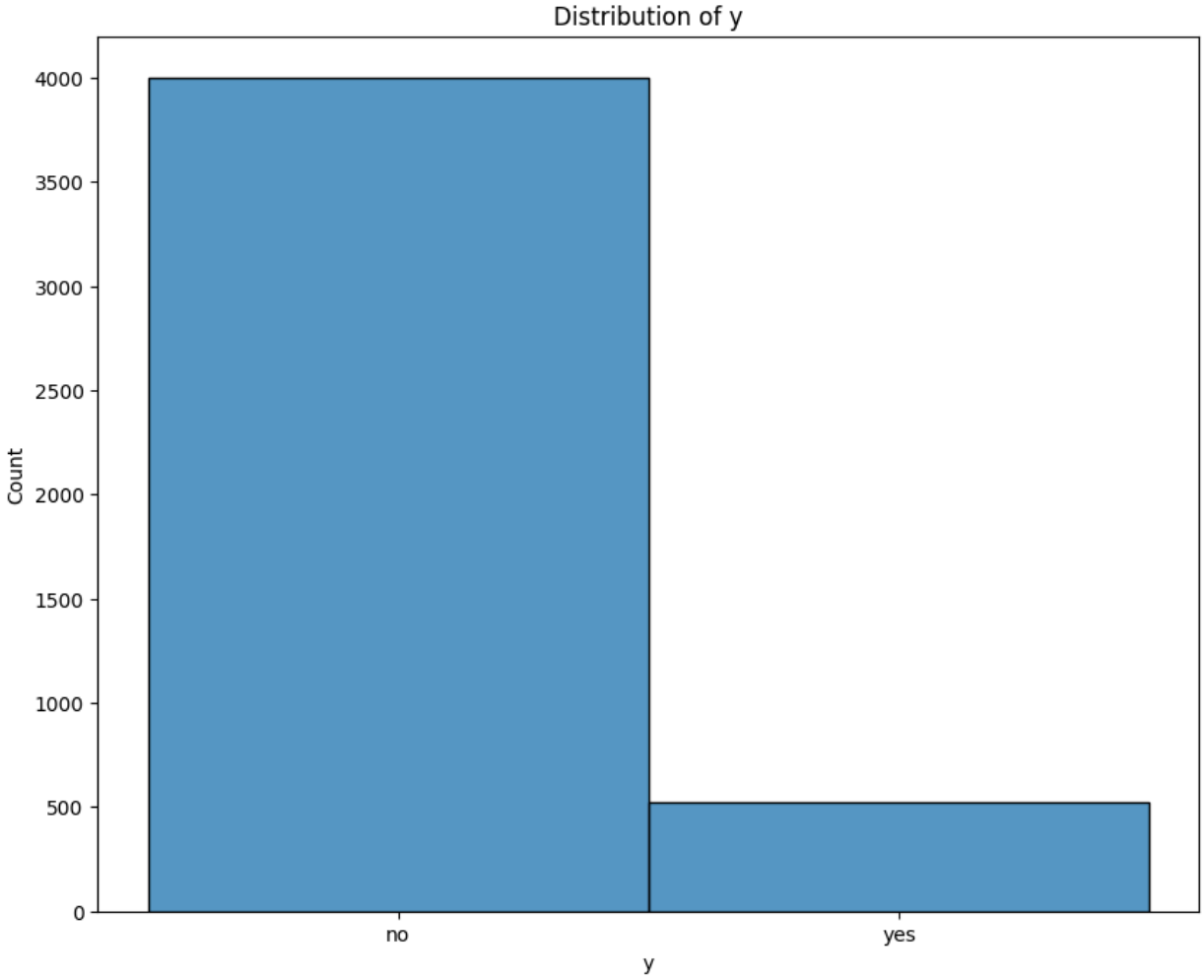
In [102…
```
for col in numerical:
    plt.figure(figsize=(10,8))
    sns.histplot(bank_df[i], bins=10, kde=False)
    plt.title(f'Distribution of {i}')
    plt.savefig(f'{i}.png')
    plt.show()
```

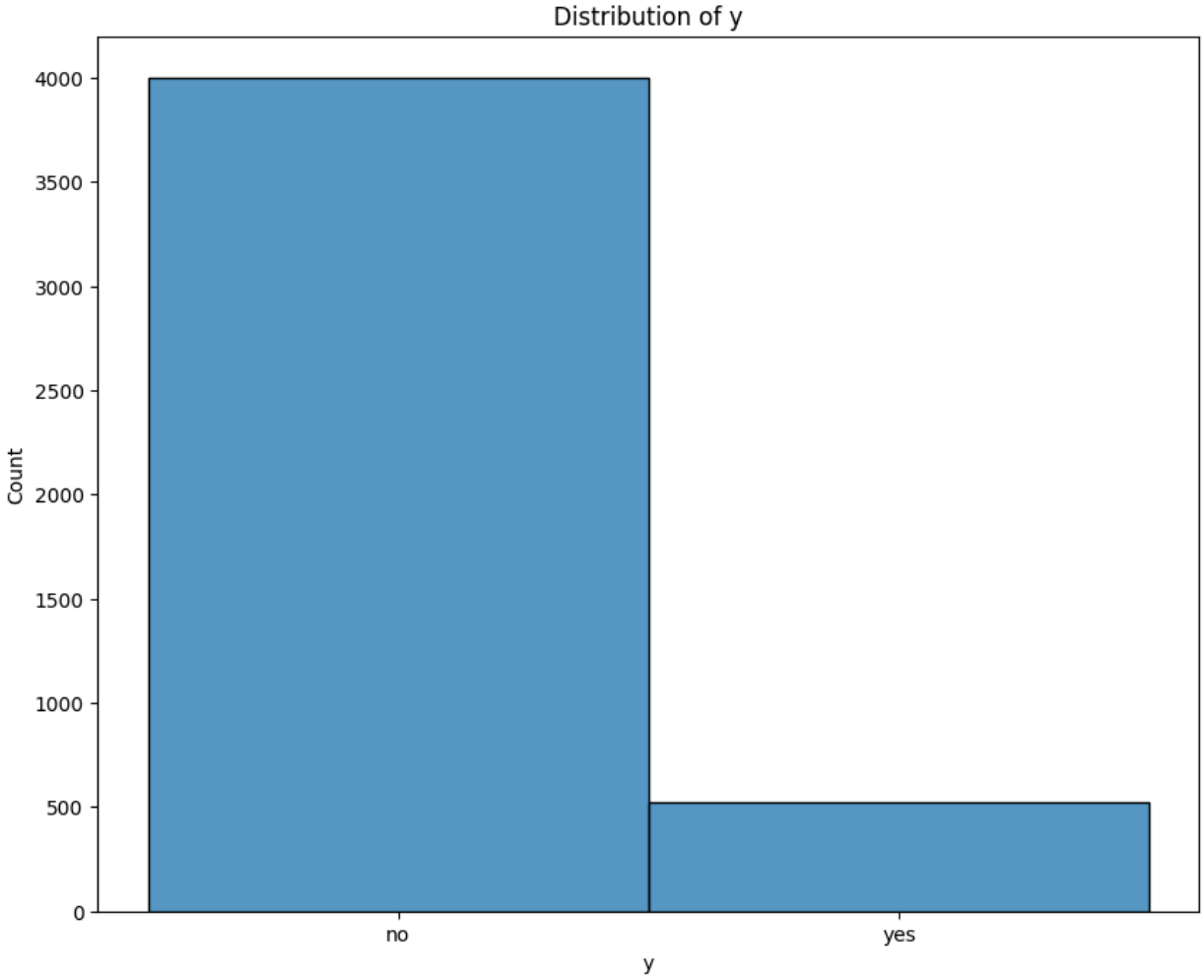

Distribution of y
Count
4000
3500
3000
2500
2000
1500
1000
500
0
no
yes
y

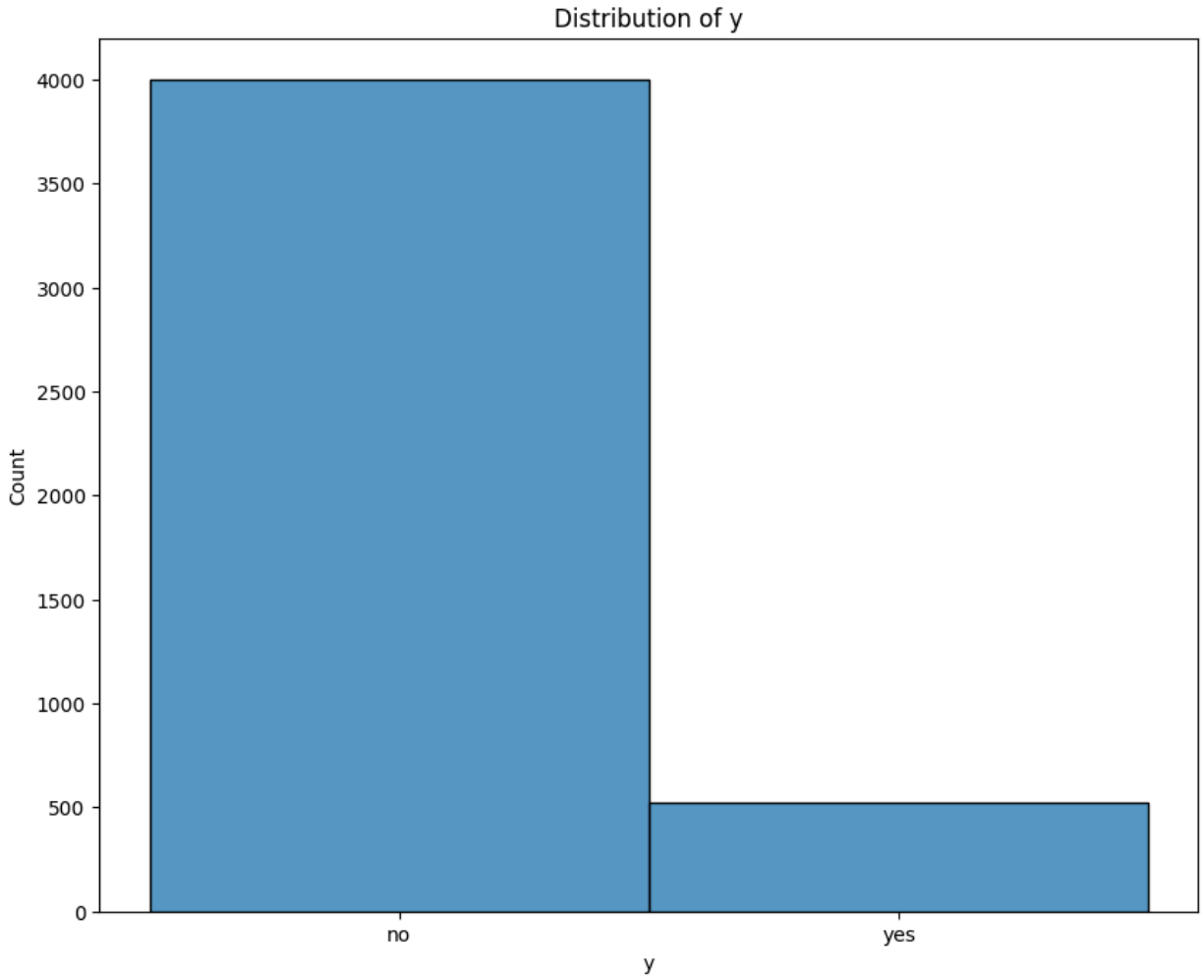

Distribution of y
Count
4000
3500
3000
2500
2000
1500
1000
500
0
no
yes
y


Distribution of y
Count
4000
3500
3000
2500
2000
1500
1000
500
0
no
yes
y


Distribution of y
Count
4000
3500
3000
2500
2000
1500
1000
500
0
no
yes
y


Distribution of y
Count
4000
3500
3000
2500
2000
1500
1000
500
0
no
yes
y

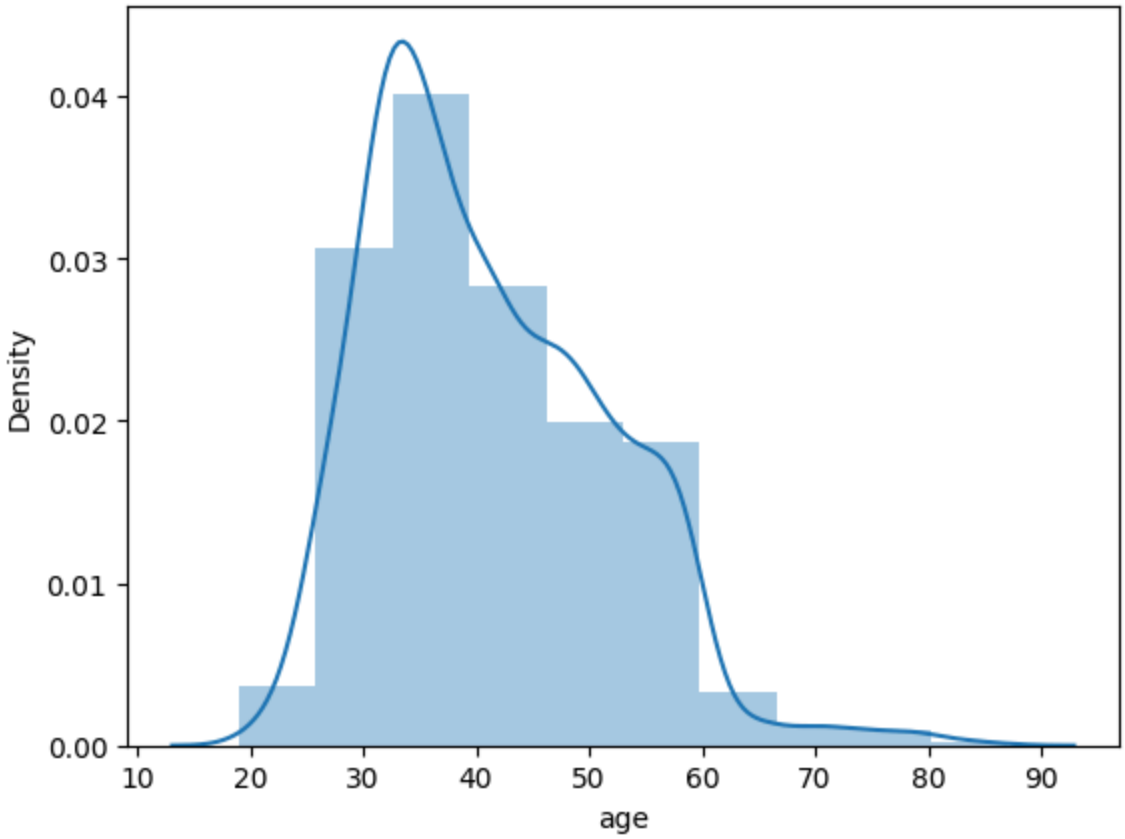

Distribution of y
Count
4000
3500
3000
2500
2000
1500
1000
500
0
no
yes
y



Distplot

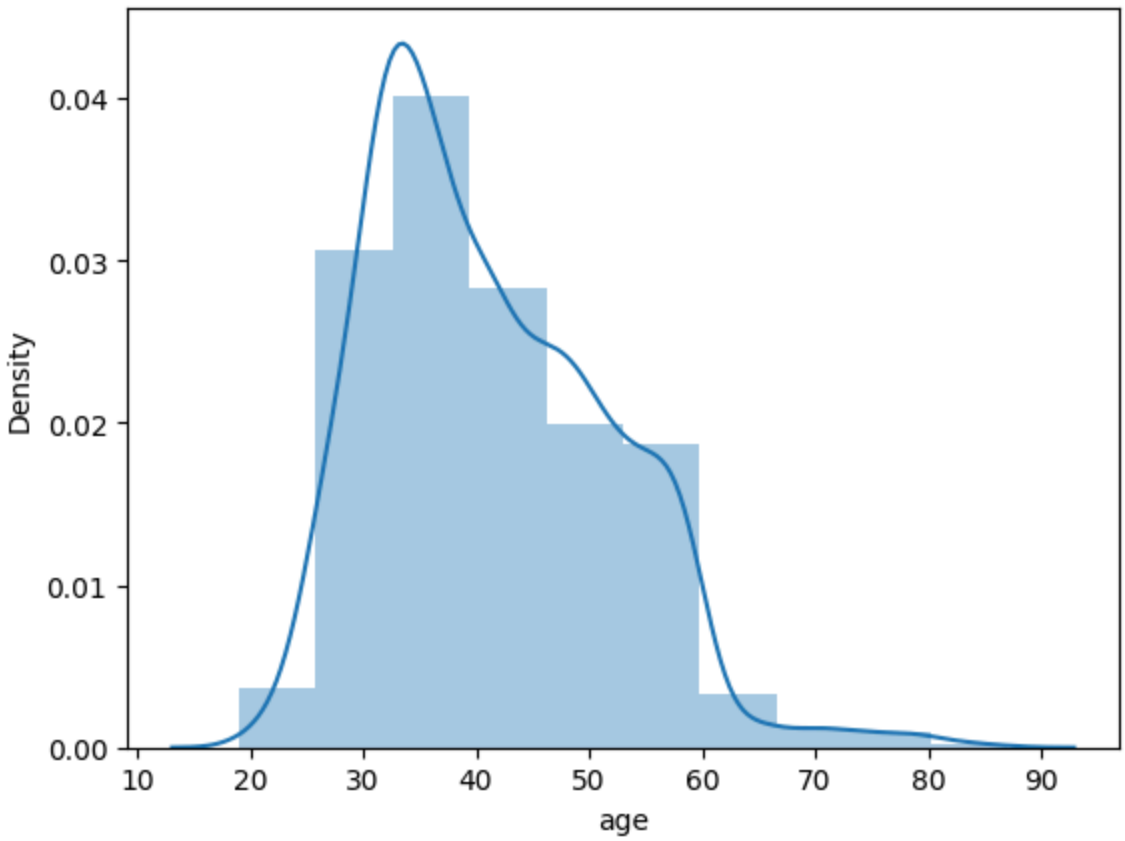
In [103…

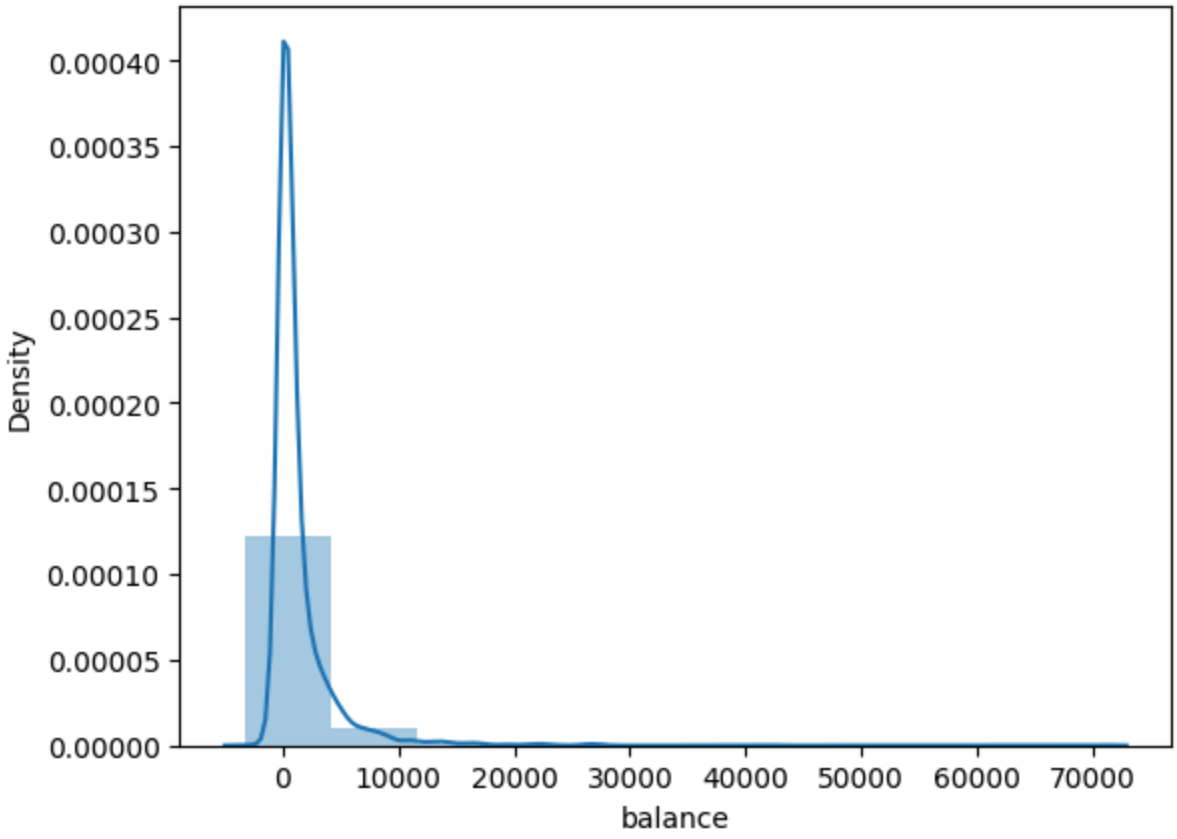
```
import warnings
warnings.filterwarnings('ignore')

sns.distplot(bank_df['age'], bins=10)
plt.show()
```

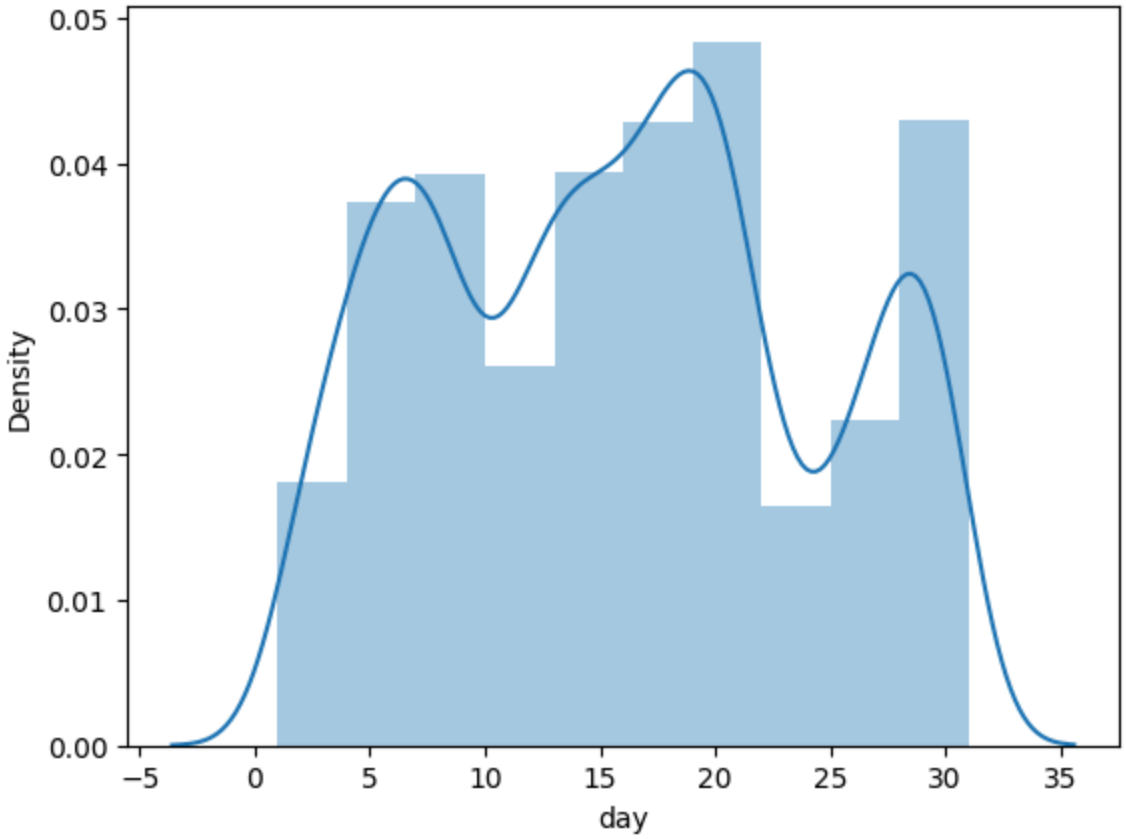


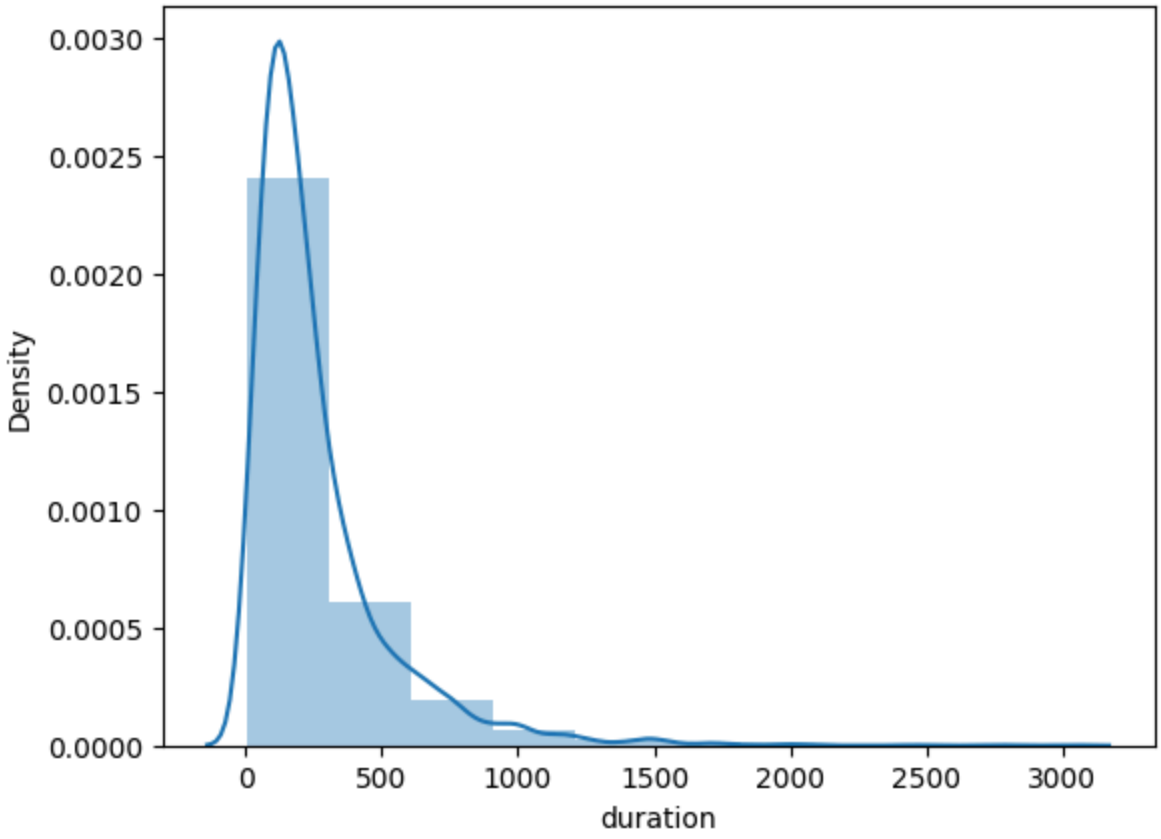
In [104…

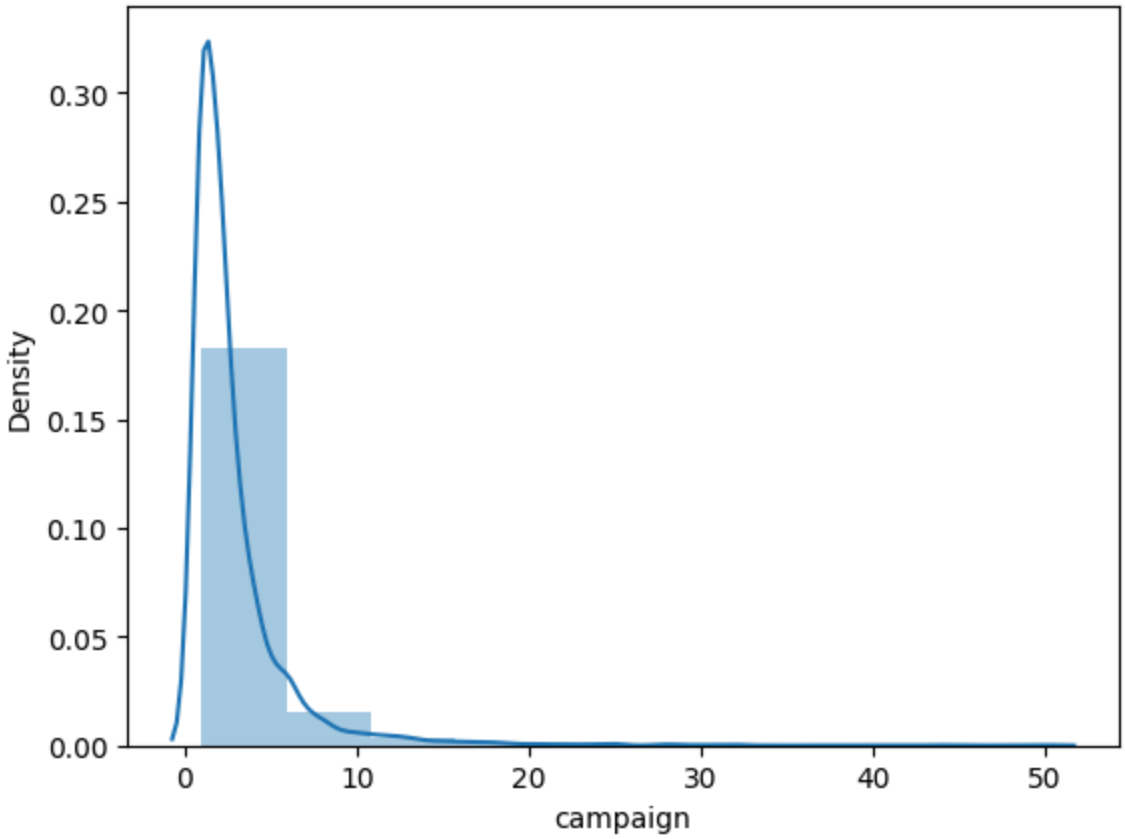
```
for i in numerical:
    sns.distplot(bank_df[i], bins=10)
    plt.show()
```

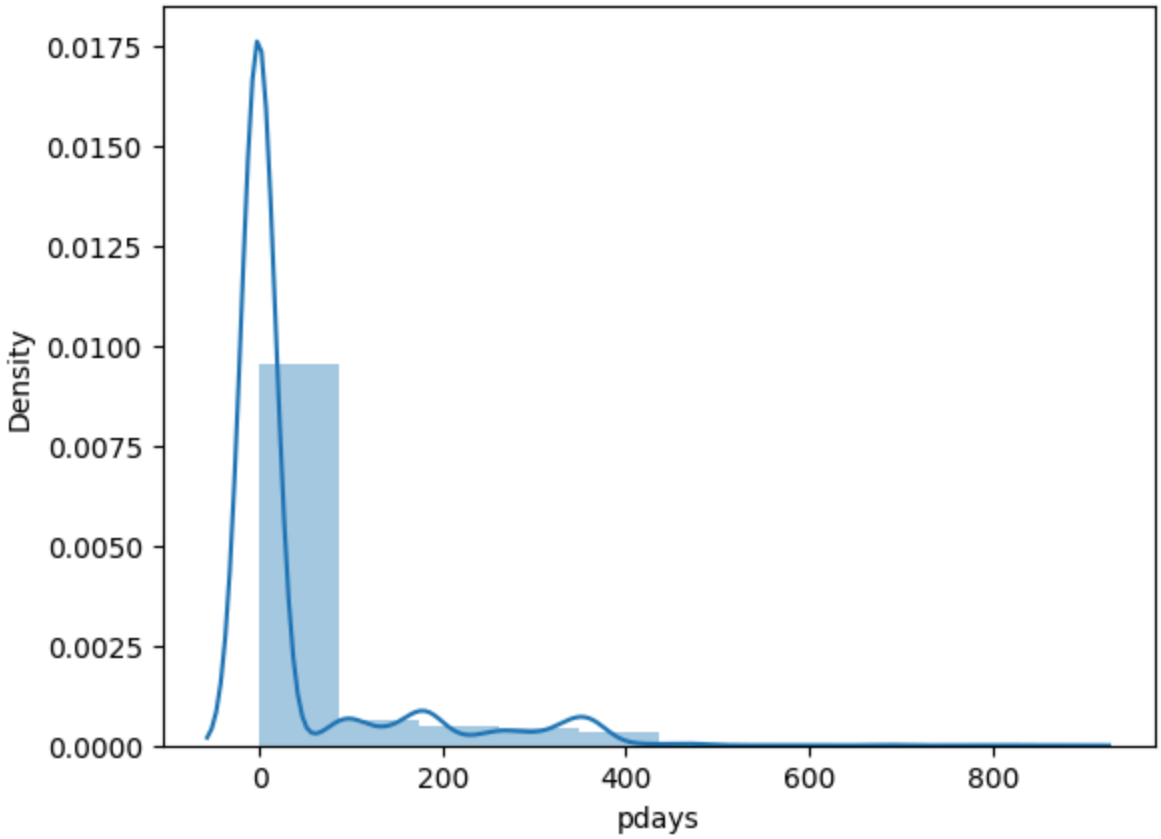


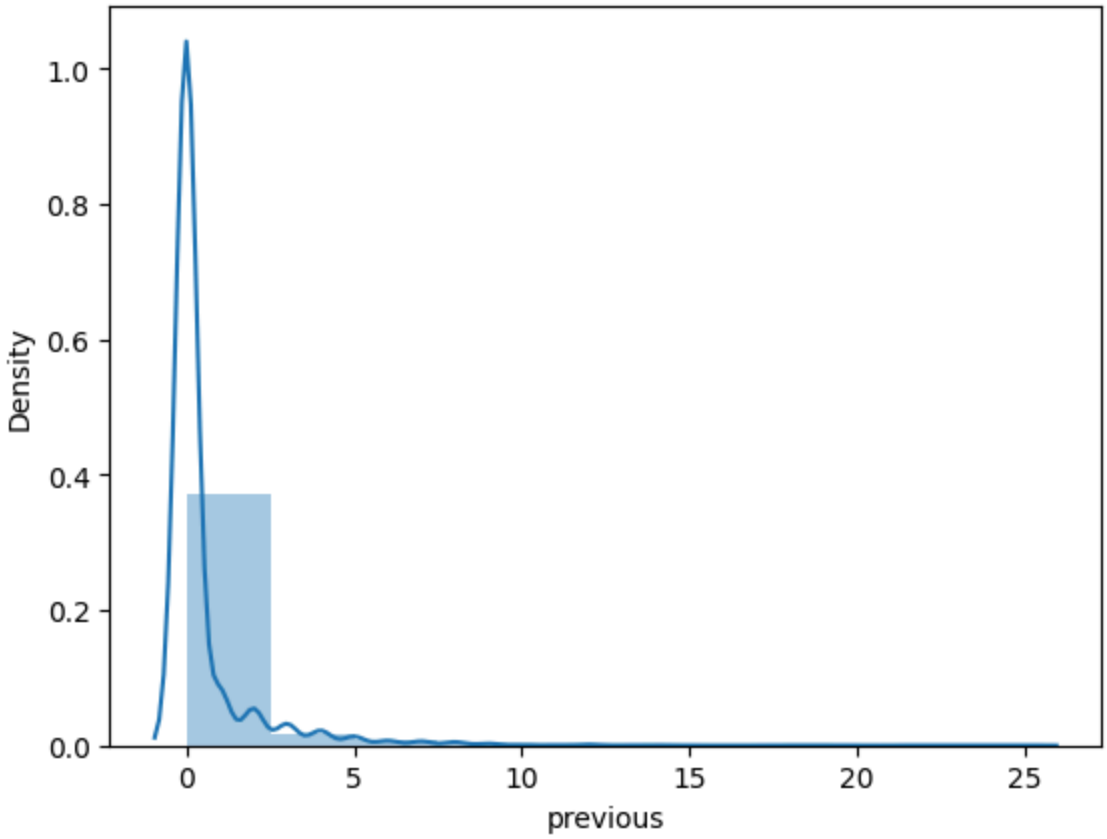

0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Density
0
10000
20000
30000
40000
50000
60000
70000
balance


0.05
0.04
0.03
0.02
0.01
0.00
Density
−5
0
5
10
15
20
25
30
35
day


0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Density
0
500
1000
1500
2000
2500
3000
duration


0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
0
10
20
30
40
50
campaign





Step-9: Outlier Analysis

In [122…

```
numerical
```

Out[122]:
```
Index(['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous'], dtype
='object')
```

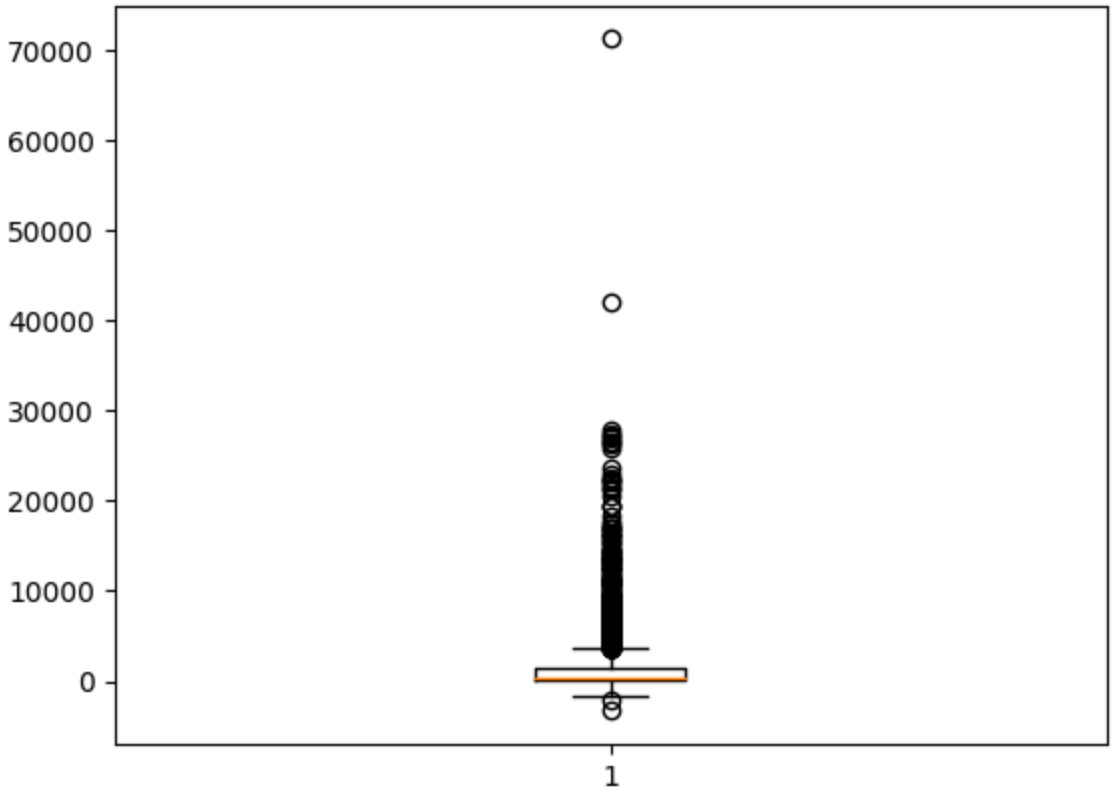
In [125…
```
bal_data=bank_df['balance']

# calculate the 1st and 3rd quartile
q1=round(np.quantile(bal_data,0.25),2)
q3=round(np.quantile(bal_data, 0.75),2)

# Compute the IQR and the lower and upper bounds
IQR=q3-q1

lb=q1-1.5*IQR
ub=q3+1.5*IQR
con1=bank_df['balance']>lb
con2=bank_df['balance']<ub
con3=con1&con2
count=len(bank_df[con3])
non_outliers_data=bank_df[con3]
non_outliers_data
```

Out[125]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may
5	35	management	single	tertiary	no	747	no	no	cellular	23	feb
…	…	…	…	…	…	…	…	…	…	…	…
4515	32	services	single	secondary	no	473	yes	no	cellular	7	jul
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr

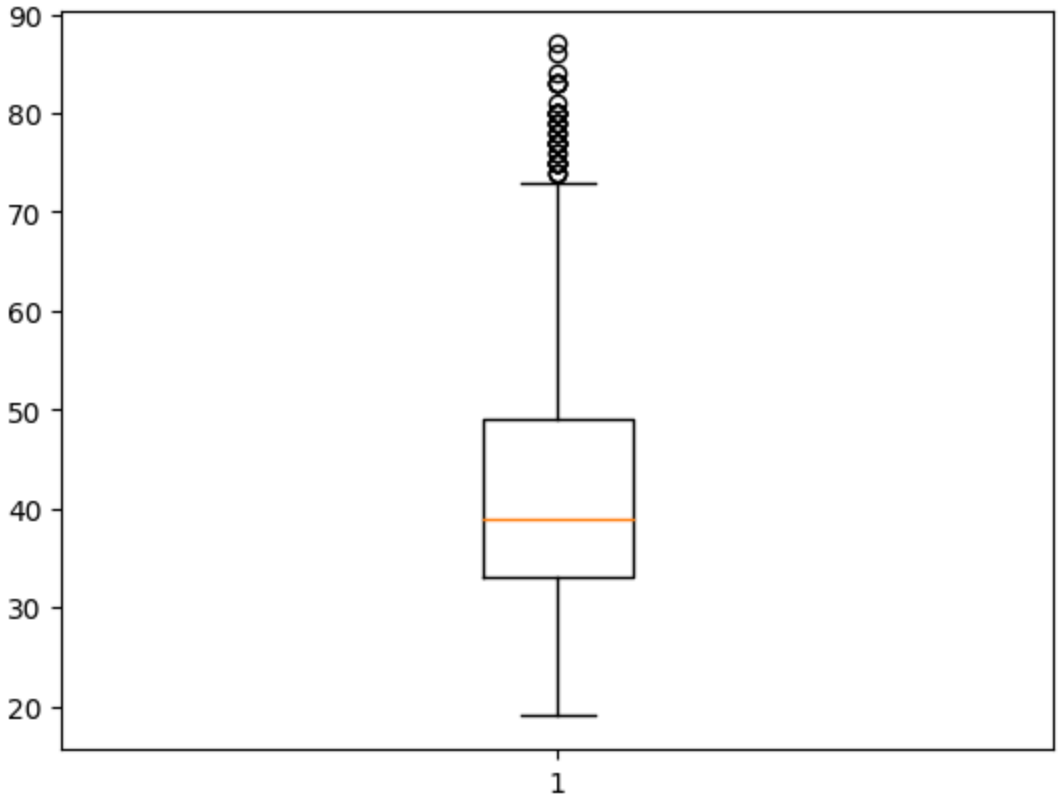
4015 rows × 19 columns

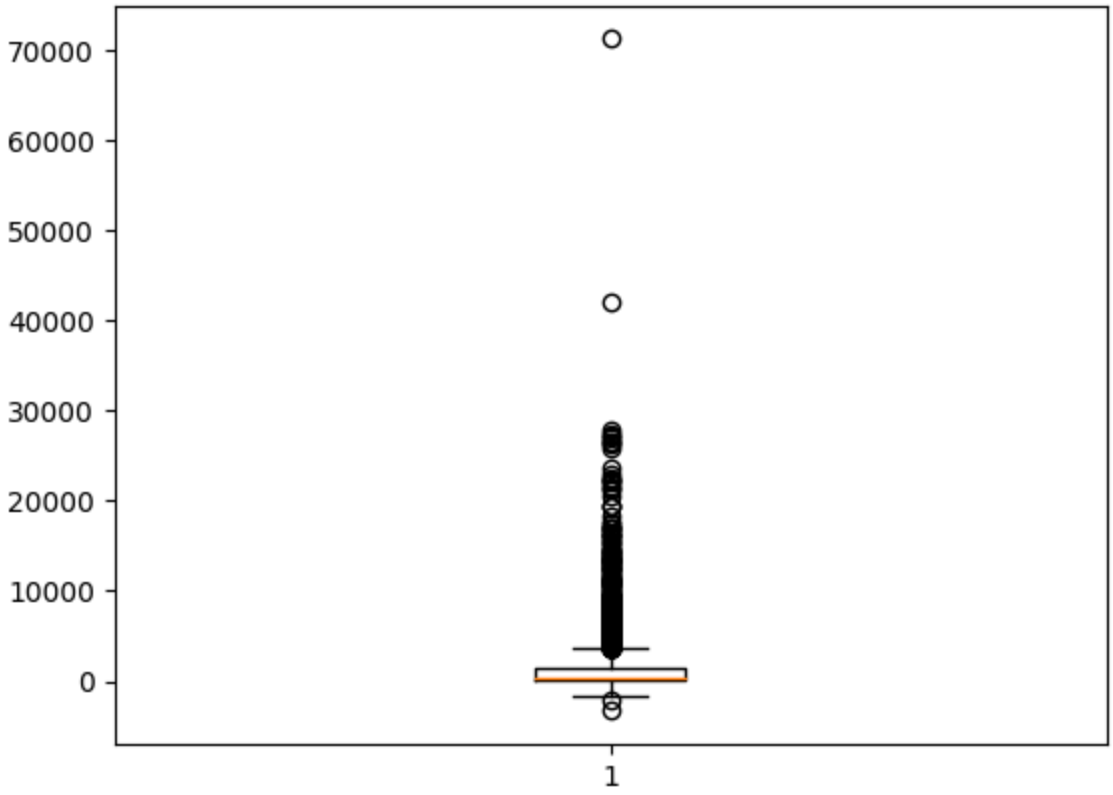
In [108…
```
plt.boxplot(bank_df['balance'])
plt.show()
```

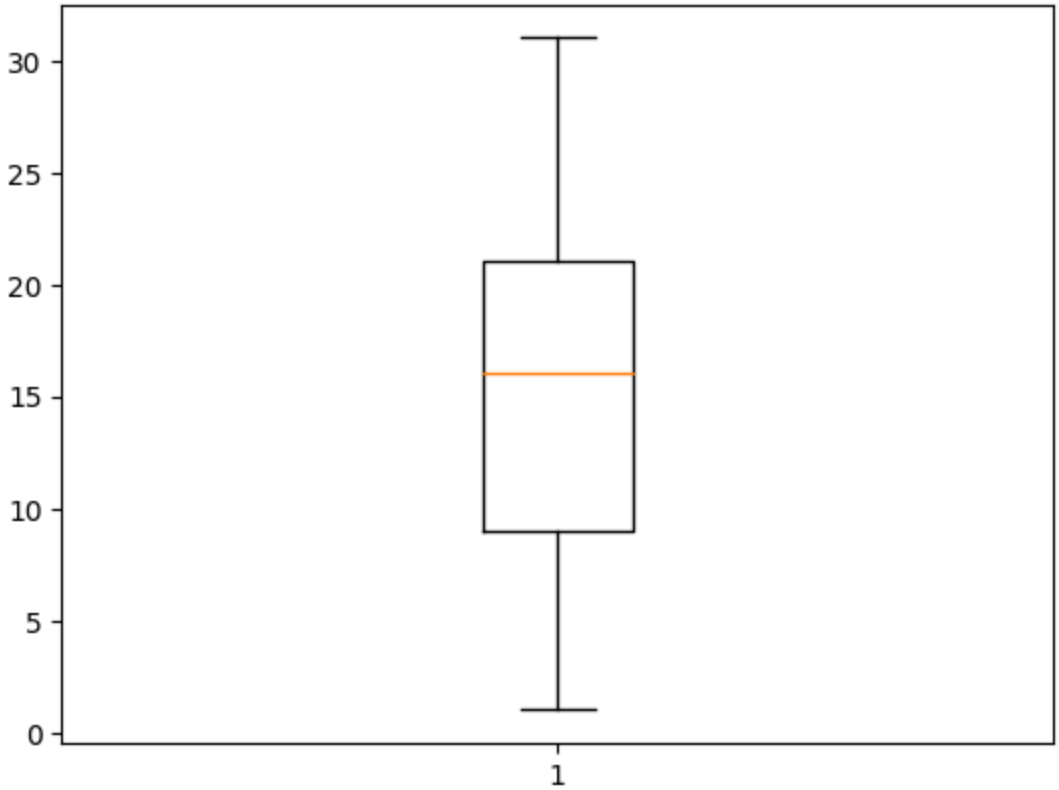


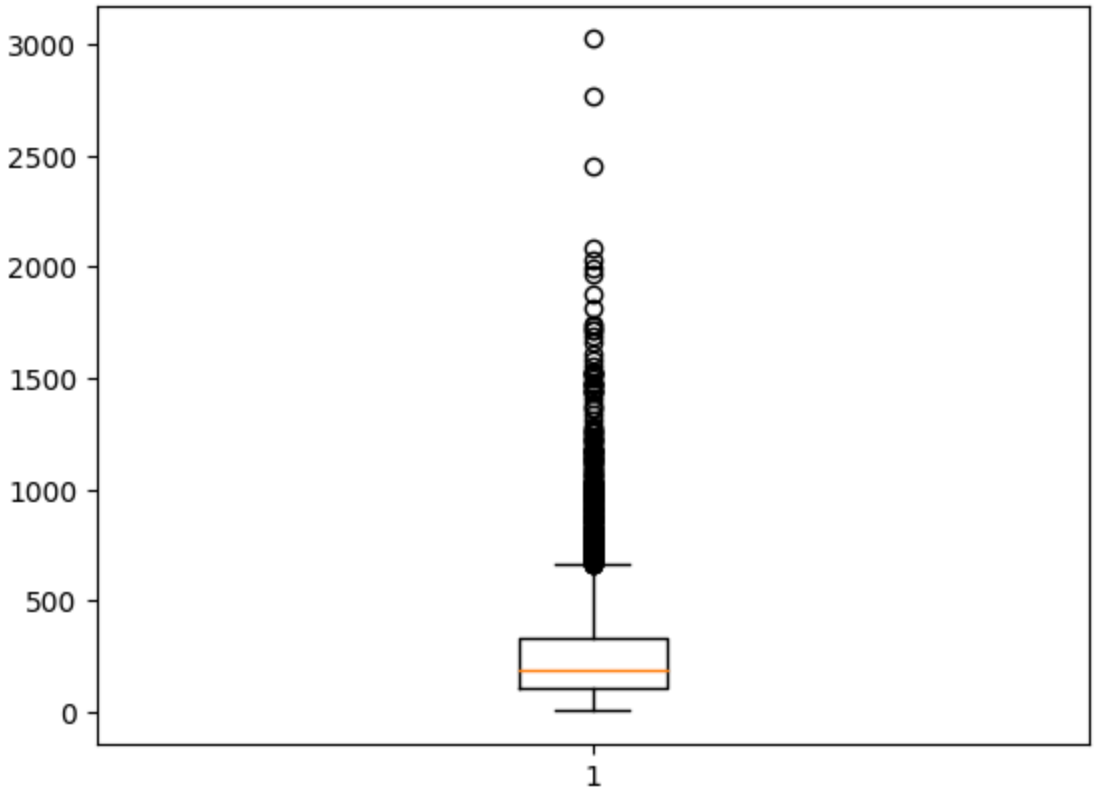
In [109…

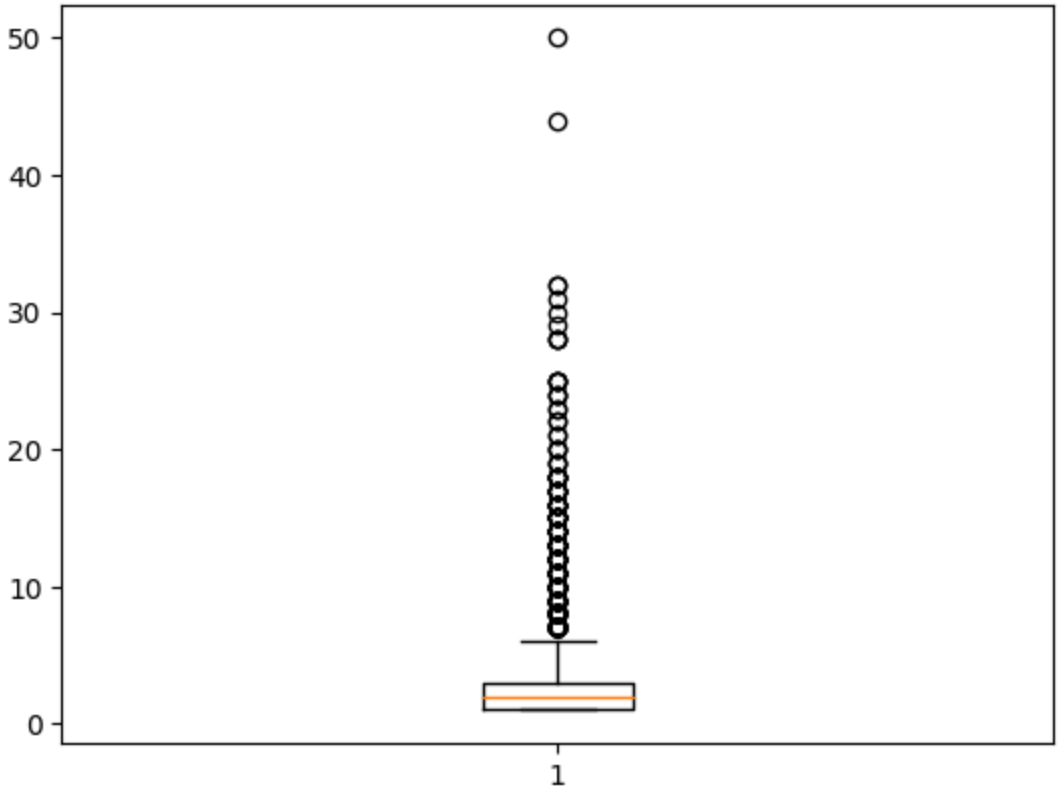
```
for i in numerical:
    plt.boxplot(bank_df[i])
    plt.show()
```

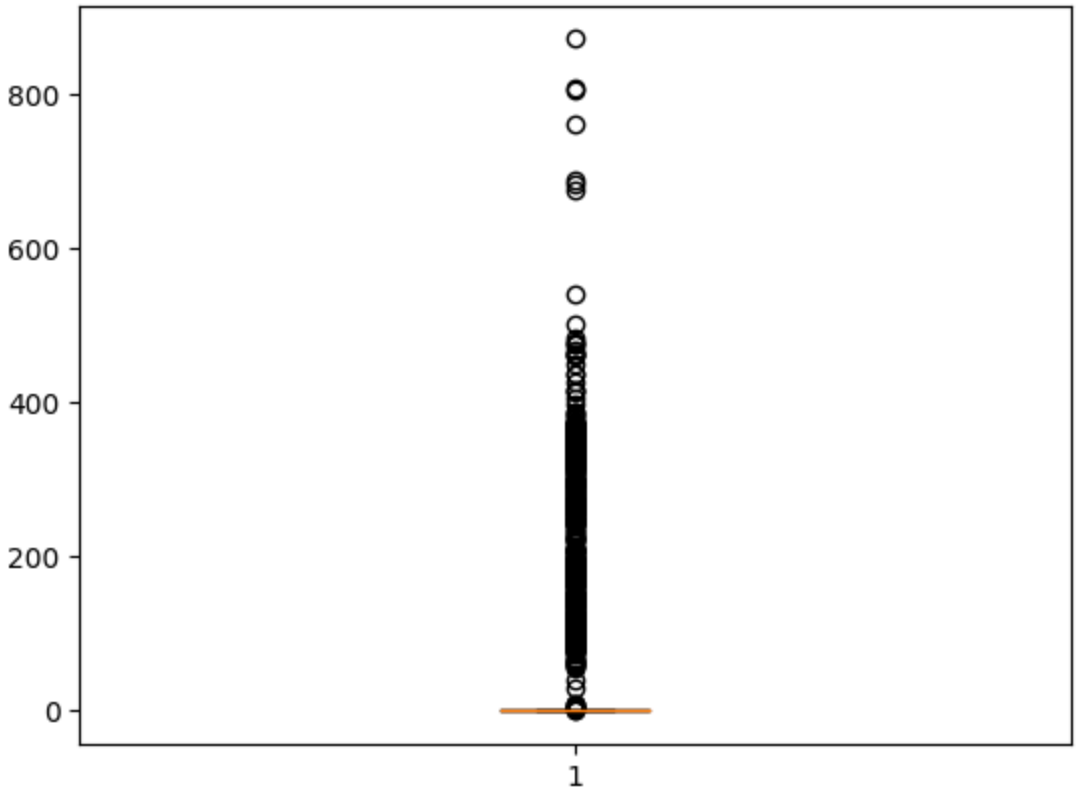


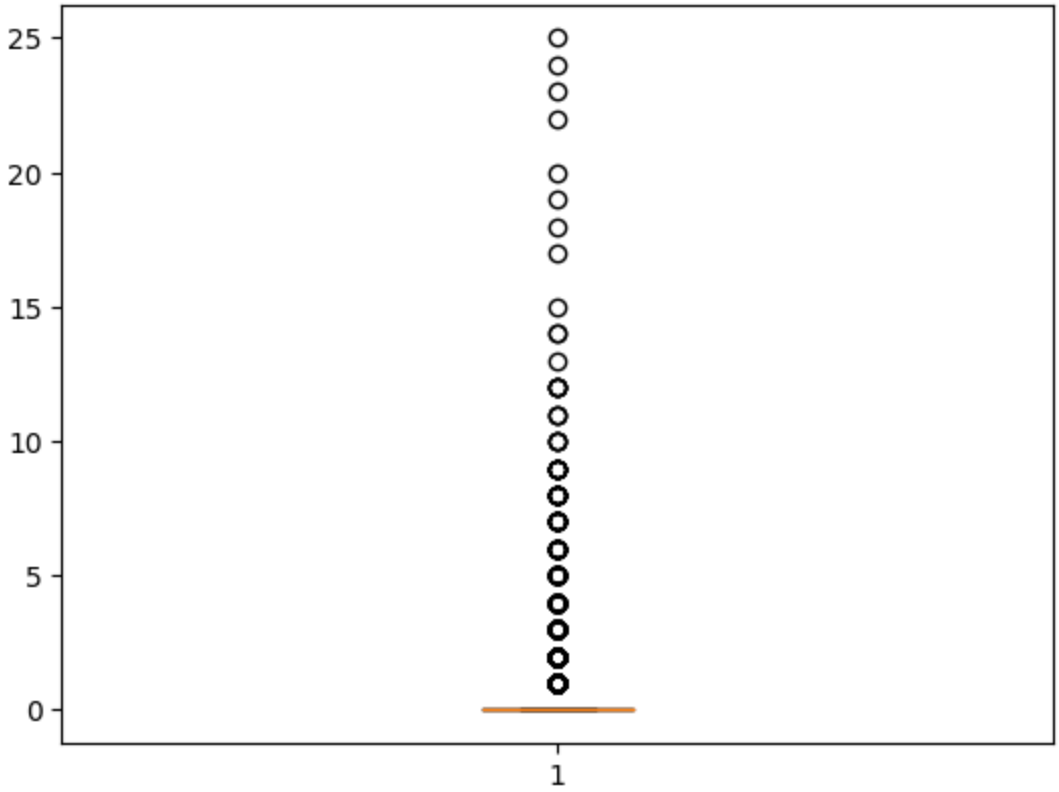

70000
60000
50000
40000
30000
20000
10000
0
1


30
25
20
15
10
5
0
1


3000
2500
2000
1500
1000
500
0
1


50
40
30
20
10
0
1





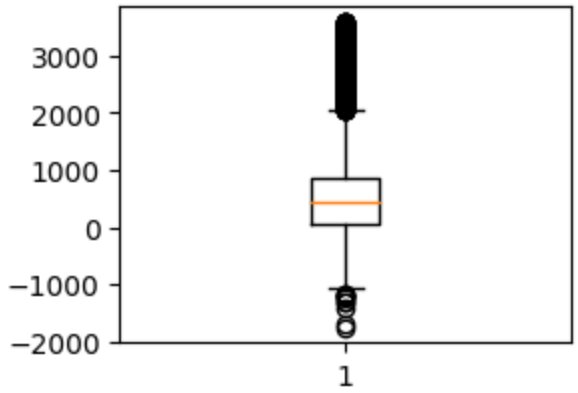
- how to treat the outliers
  - 1. fill the outliers with median
  - 2. cap the outliers

In [113…

```
bal_data=bank_df['balance']
# calculate the 1st and 3rd quartile
q1=round(np.quantile(bal_data,0.25),2)
q3=round(np.quantile(bal_data,0.75),2)

# Compute the IQR and the lower and upper bounds
IQR=q3-q1
lb=q1-1.5*IQR
ub=q3+1.5*IQR

# calculate the median
median=bal_data.median()
new_data=[]
for i in bal_data:
    if i<lb or i>ub:
        new_data.append(median)
    else:
        new_data.append(i)


bank_df['balance_1'] = new_data
```

In [115…

```
plt.subplot(2,2,1).boxplot(bank_df['balance_1'])
plt.show()
```

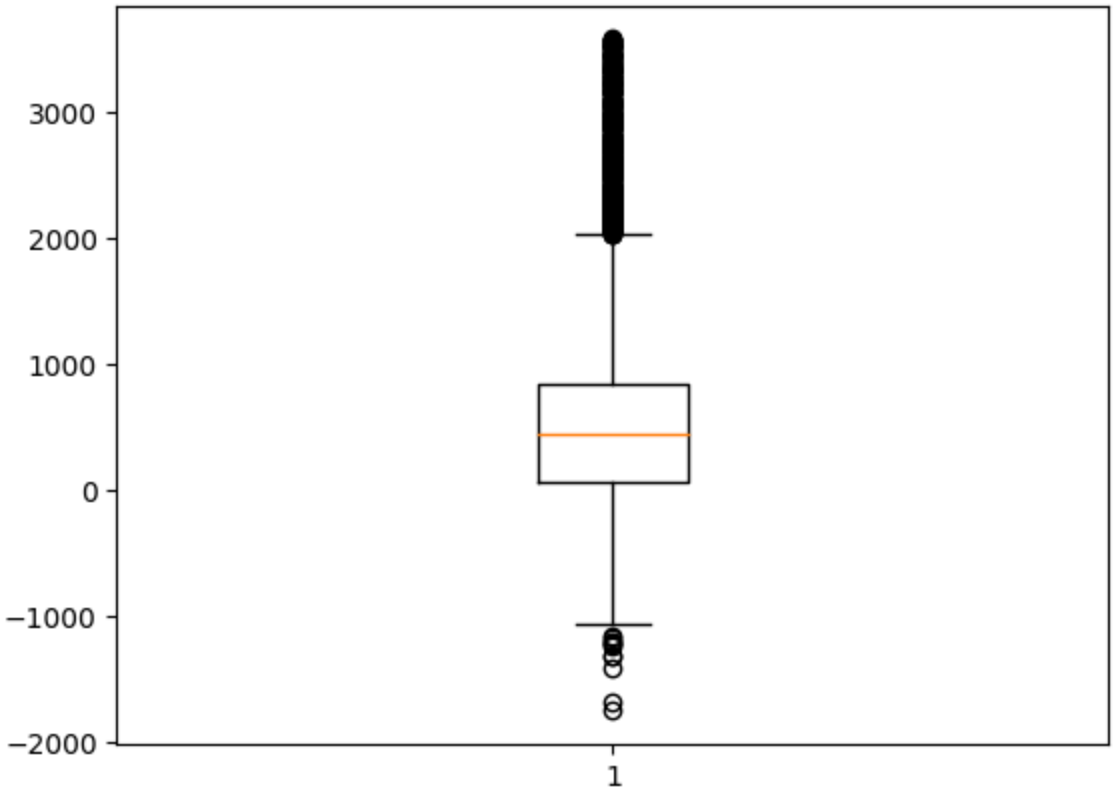


In [120…

```
bal_data=bank_df['balance']
q1=round(np.quantile(bal_data,0.25),2)
q3=round(np.quantile(bal_data,0.75),2)
IQR=q3-q1
lb=q1-1.5*IQR
ub=q3+1.5*IQR
median=bal_data.median()
# replace the outliers with median using np.where
con=(bank_df['balance']<lb) | (bank_df['balance']>ub)
true=median
false=bank_df['balance']
bank_df['balance1']=np.where(con, true,false)
```

In [121…

```
plt.boxplot(bank_df['balance_1'])
plt.show()
```



Step-10: Correlation

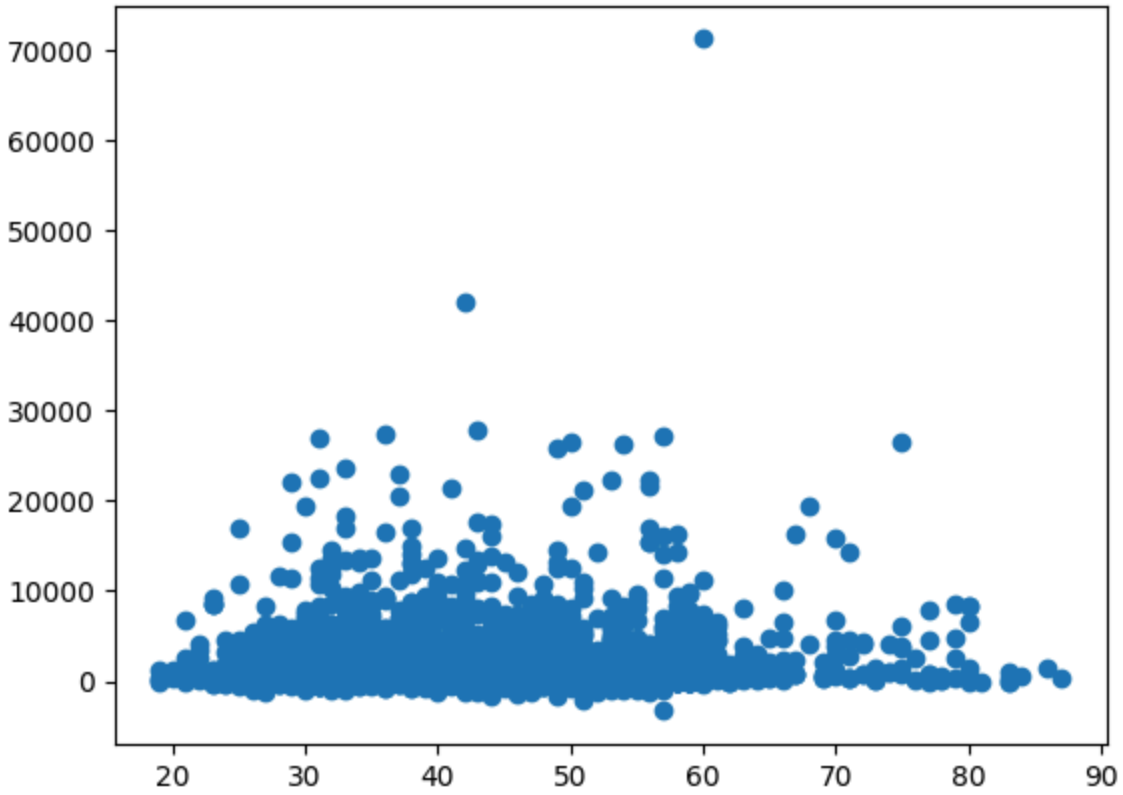
Find the Correlation Between Numerical columns

In [127…
```
numerical
```

Out[127]:
```
Index(['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous'], dtype
='object')
```

In [128…
```
col1=bank_df['age']
col2=bank_df['balance']
col3=bank_df['day']
plt.scatter(col1,col2)
```

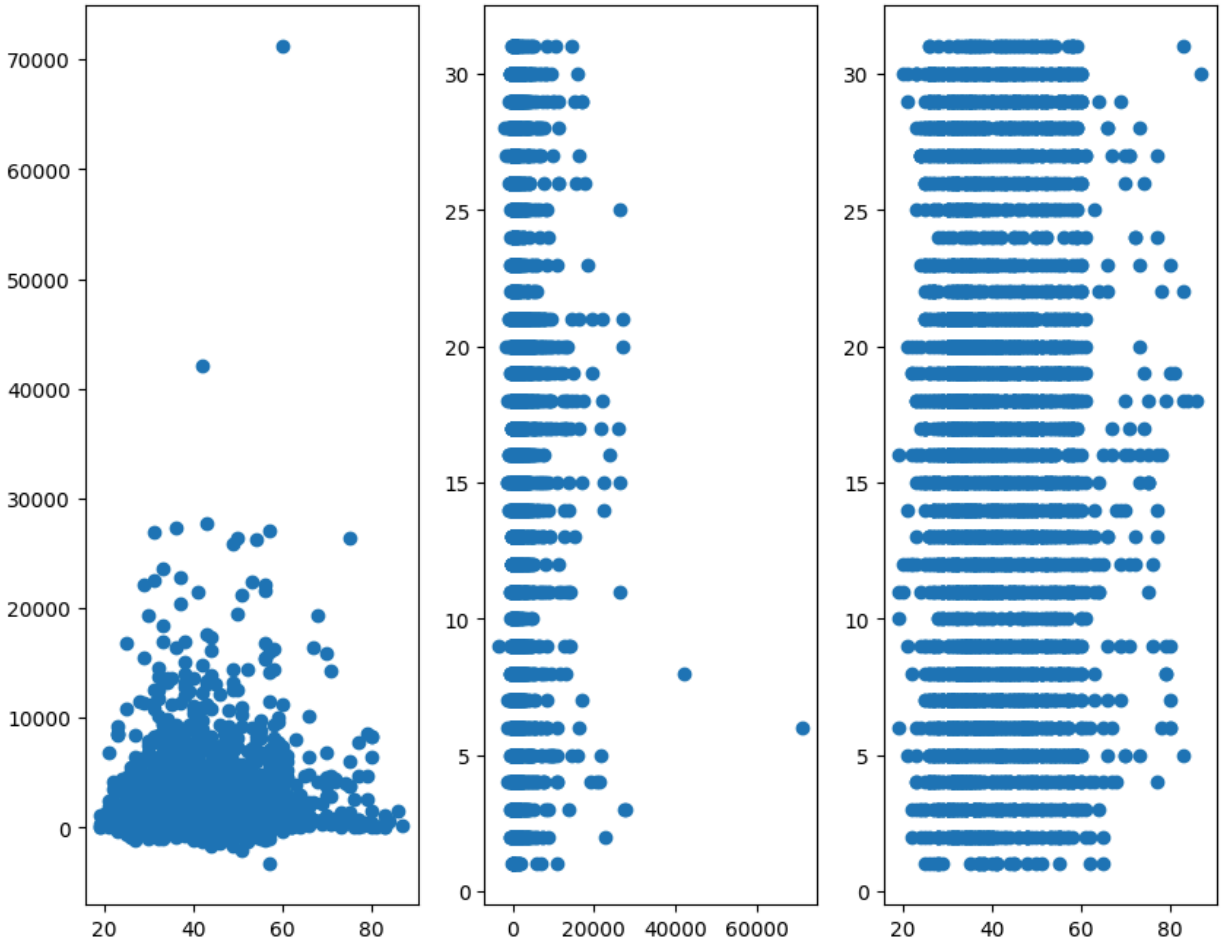
Out[128]:
```
<matplotlib.collections.PathCollection at 0x2b419e17ca0>
```



In [129…

```
plt.figure(figsize=(10,8))
plt.subplot(1,3,1).scatter(col1,col2)
plt.subplot(1,3,2).scatter(col2,col3)
plt.subplot(1,3,3).scatter(col1,col3)
```

Out[129]: <matplotlib.collections.PathCollection at 0x2b41a06dd80>



In [130…
```
bank_df.corr(numeric_only=True)
```

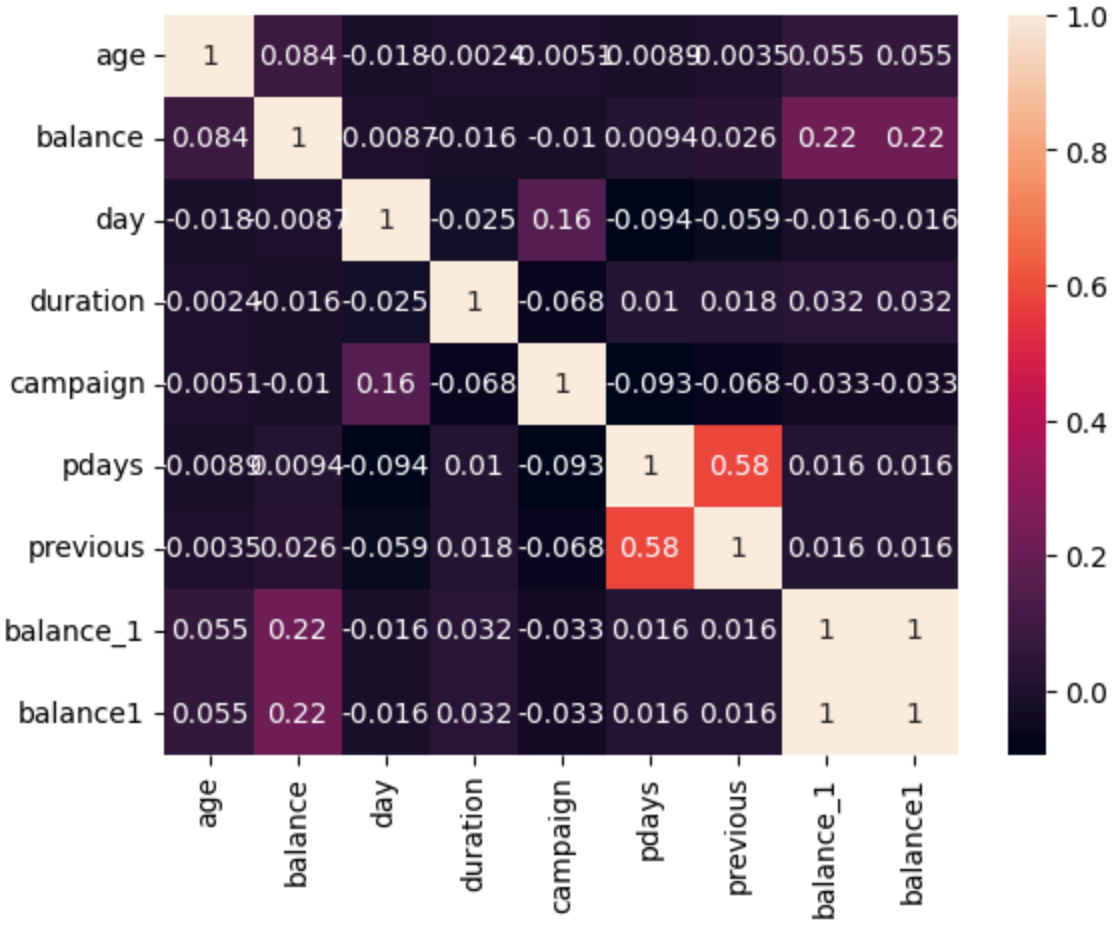
Out[130]:

	age	balance	day	duration	campaign	pdays	previous	balance_1	bala
age	1.000000	0.083820	-0.017853	-0.002367	-0.005148	-0.008894	-0.003511	0.055307	0.05
balance	0.083820	1.000000	-0.008677	-0.015950	-0.009976	0.009437	0.026196	0.215264	0.2
day	-0.017853	-0.008677	1.000000	-0.024629	0.160706	-0.094352	-0.059114	-0.015530	-0.0
duration	-0.002367	-0.015950	-0.024629	1.000000	-0.068382	0.010380	0.018080	0.031843	0.03
campaign	-0.005148	-0.009976	0.160706	-0.068382	1.000000	-0.093137	-0.067833	-0.033043	-0.03
pdays	-0.008894	0.009437	-0.094352	0.010380	-0.093137	1.000000	0.577562	0.015697	0.0
previous	-0.003511	0.026196	-0.059114	0.018080	-0.067833	0.577562	1.000000	0.015868	0.0
balance_1	0.055307	0.215264	-0.015530	0.031843	-0.033043	0.015697	0.015868	1.000000	1.00
balance1	0.055307	0.215264	-0.015530	0.031843	-0.033043	0.015697	0.015868	1.000000	1.00

Heatmap

In [131…
```
corr=bank_df.corr(numeric_only=True)
sns.heatmap(corr, annot=True)
```

Out[131]: <AxesSubplot: >



Step-11: Convert Categorical to Numerical

LableEncoder

In [132…
```
categorical
```

Out[132]:
```
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
       'month', 'poutcome', 'y'],
      dtype='object')
```

In [133…
```
from sklearn.preprocessing import LabelEncoder
```

In [134…
```
le=LabelEncoder()
```

In [135…
```
bank_df['job']=le.fit_transform(bank_df['job'])
```

In [136…
```
bank_df.head()
```

Out[136]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	c
0	30	10	married	primary	no	1787	no	no	cellular	19	oct	79	
1	33	7	married	secondary	no	4789	yes	yes	cellular	11	may	220	
2	35	4	single	tertiary	no	1350	yes	no	cellular	16	apr	185	
3	30	4	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	
4	59	1	married	secondary	no	0	yes	no	unknown	5	may	226	

In [137…

```
for i in categorical:
    bank_df[i]=le.fit_transform(bank_df[i])
```

In [138…

```
bank_df
```

Out[138]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration
0	30	10	1	0	0	1787	0	0	0	19	10	79
1	33	7	1	1	0	4789	1	1	0	11	8	220
2	35	4	2	2	0	1350	1	0	0	16	0	185
3	30	4	1	2	0	1476	1	1	2	3	6	199
4	59	1	1	1	0	0	1	0	2	5	8	226
…	…	…	…	…	…	…	…	…	…	…	…	…
4516	33	7	1	1	0	-333	1	0	0	30	5	329
4517	57	6	1	2	1	-3313	1	1	2	9	8	153
4518	57	9	1	1	0	295	0	0	0	19	1	151
4519	28	1	1	1	0	1137	0	0	0	6	3	129
4520	44	2	2	2	0	1136	1	1	0	3	0	345

4521 rows × 19 columns

One hot-Encoder

In [139…

```
bank_df=pd.read_csv(r"F:\FSDS\Data Files\bank.csv", sep=';')
pd.get_dummies(bank_df, dtype='int')
```

Out[139]:

	age	balance	day	duration	campaign	pdays	previous	job_admin.	job_blue-collar	job_entrepren
0	30	1787	19	79	1	-1	0	0	0	
1	33	4789	11	220	1	339	4	0	0	
2	35	1350	16	185	1	330	1	0	0	
3	30	1476	3	199	4	-1	0	0	0	
4	59	0	5	226	1	-1	0	0	1	
...	...	...	...	...	...	...	...	...	...	
4516	33	-333	30	329	5	-1	0	0	0	
4517	57	-3313	9	153	1	-1	0	0	0	
4518	57	295	19	151	11	-1	0	0	0	
4519	28	1137	6	129	4	211	3	0	1	
4520	44	1136	3	345	2	249	7	0	0	

4521 rows × 53 columns

Step-12: Scale the Data

Standarization

In [140…

```
bal_data=bank_df['balance']
mean=bal_data.mean()
std=bal_data.std()
data=(bal_data-mean)/std
data
```

Out[140]:

```
0       0.121058
1       1.118521
2      -0.024142
3       0.017724
4      -0.472701
          ...
4516   -0.583345
4517   -1.573497
4518   -0.374682
4519   -0.094914
4520   -0.095247
Name: balance, Length: 4521, dtype: float64
```

In [141…

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
ss.fit_transform(bank_df[['balance']])
```

Out[141]:
```
array([[ 0.12107186],
       [ 1.1186443 ],
       [-0.02414438],
       ...,
       [-0.37472364],
       [-0.09492484],
       [-0.09525714]])
```

In [142…
```
bank_df[['balance']]
```

Out[142]:

	balance
0	1787
1	4789
2	1350
3	1476
4	0
…	…
4516	-333
4517	-3313
4518	295
4519	1137
4520	1136

4521 rows × 1 columns

In [143…
```
bank_df=pd.read_csv(r"F:\FSDS\Data Files\bank.csv", sep=';')
bal_data=bank_df['balance']
mean=bal_data.mean()
std=bal_data.std()
bank_df['balance_1']=(bal_data-mean)/std
bank_df
```

Out[143]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may
...	...	...	...	...	...	...	...	...	...	...	...
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	may
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr

4521 rows × 18 columns

In [144…

```
bank_df[['balance','balance_1']]
```

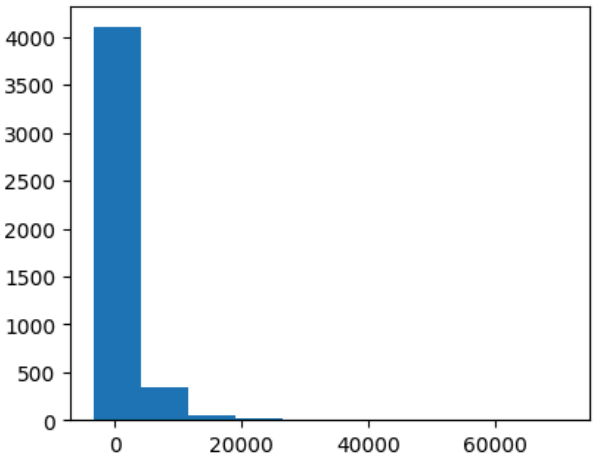
Out[144]:

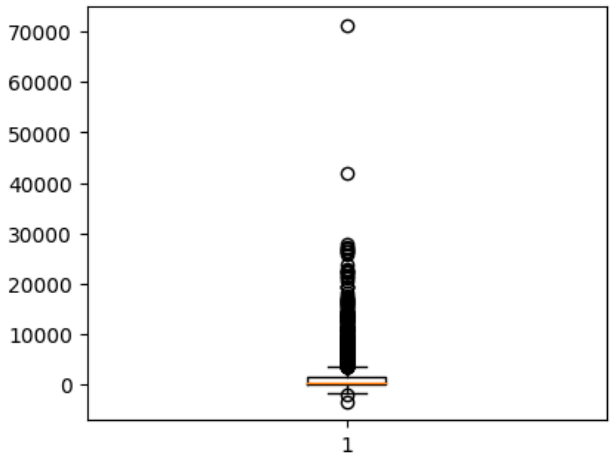
	balance	balance_1
0	1787	0.121058
1	4789	1.118521
2	1350	-0.024142
3	1476	0.017724
4	0	-0.472701
...	...	...
4516	-333	-0.583345
4517	-3313	-1.573497
4518	295	-0.374682
4519	1137	-0.094914
4520	1136	-0.095247

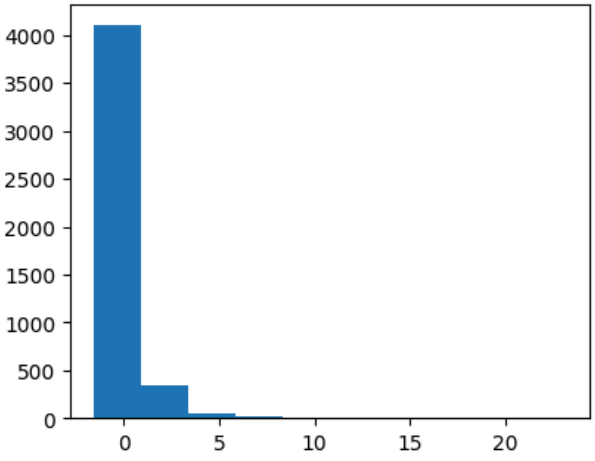
4521 rows × 2 columns

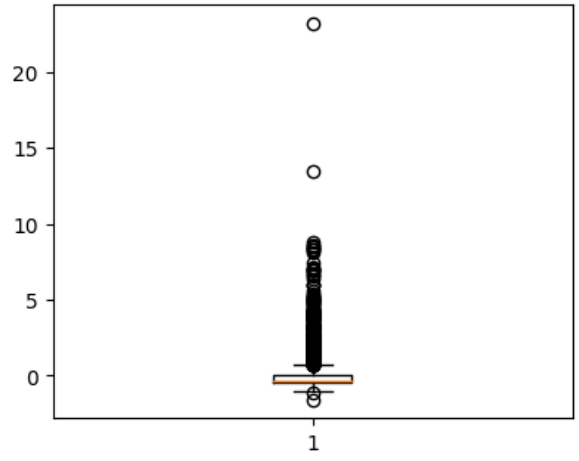
In [145…

```
plt.figure(figsize=(10,8))
plt.subplot(2,2,1).hist(bank_df['balance'])
plt.subplot(2,2,2).boxplot(bank_df['balance'])
plt.subplot(2,2,3).hist(bank_df['balance_1'])
plt.subplot(2,2,4).boxplot(bank_df['balance_1'])
plt.show()
```


4000
3500
3000
2500
2000
1500
1000
500
0
0
20000
40000
60000


70000
60000
50000
40000
30000
20000
10000
0
1


4000
3500
3000
2500
2000
1500
1000
500
0
0
5
10
15
20


20
15
10
5
0
1

In [ ]: